UNIVERSITY OF CALIFORNIA SAN DIEGO

Augmented Translation Models for Sequential Recommendation

A Thesis submitted in partial satisfaction of the requirements
for the degree Master of Science

in

Computer Science

by

Rajiv Pasricha

Committee in charge:

Professor Julian McAuley, Chair
Professor Ndapa Nakashole
Professor Lawrence Saul

2018

The Thesis of Rajiv Pasricha as it is listed on UC San Diego Academic Records is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California San Diego

2018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Julian McAuley for his support over the past four years, and for his helpful guidance as my research advisor.

Special thanks to my family, whose encouragement and support made this thesis a reality.

Various chapters in this thesis include material that has been submitted for publication as it may appear in RecSys 2018, Pasricha, Rajiv; McAuley, Julian, ACM, 2018. The thesis author was the primary investigator and author of this paper.

ABSTRACT OF THE THESIS

Augmented Translation Models for Sequential Recommendation

by

Rajiv Pasricha

Master of Science in Computer Science

University of California San Diego, 2018

Professor Julian McAuley, Chair

Sequential recommendation algorithms aim to predict users' future behavior given their historical interactions. In particular, a recent line of work has achieved state-of-the-art performance on sequential recommendation tasks by adapting ideas from metric learning and knowledge-graph completion. These algorithms replace inner products with low-dimensional embeddings and distance functions, employing a simple translation dynamic to model user behavior over time.

In this thesis, we analyze the task of sequential recommendation and discuss TransRec, a recent algorithm that models users with linear translation vectors over low-dimensional item

embeddings. We present a variety of extensions to this model, increasing complexity via additional features, neural networks, and session-based approaches. We evaluate these extensions on a variety of datasets and also present relevant qualitative analyses. These extensions provide insights into the translation framework and effectively inform future research directions.

We also propose *TransFM*, a model that combines translation and metric-based approaches for sequential recommendation with Factorization Machines (FMs). Doing so allows us to reap the benefits of FMs (in particular, the ability to straightforwardly incorporate content-based features) while enhancing the state-of-the-art performance of translation-based models. We learn an embedding and translation space for each feature, replacing the inner product with the squared Euclidean distance to measure interaction strength. Like FMs, the model equation can be computed in linear time and optimized using classical techniques. As *TransFM* operates on arbitrary feature vectors, content features can be easily incorporated without significant changes to the model itself. Empirically, the performance of *TransFM* significantly increases when taking content features into account, outperforming state-of-the-art models on sequential recommendation tasks.

Chapter 1

Introduction

Recommender systems play a key role in helping individuals manage the proliferation of available choices in a variety of domains, while facilitating a more targeted and personalized experience. They have seen significant success in the domains of media consumption and e-commerce systems, where efficient and effective recommender systems help optimize user experience. With a recommender system, Netflix is able to select the movies or shows that most closely align with each user's personalized interests and tailor their offerings to viewers' evolving viewing habits over time [6]. Similarly, Amazon employs recommender systems to develop accurate profiles of user interests and match them with the most relevant products [31].

Additional applications of recommender systems include point-of-interest recommendations involving systems that suggest activities and destinations for tourists visiting a new area [55, 15]. Under the umbrella of media recommendation, music services such as Spotify and Pandora select the songs most likely to be enjoyed by each user from their massive catalog. Spotify even creates personalized weekly playlists for their users, providing them with a constantly evolving selection of new songs they are likely to enjoy. In the realm of advertising, the ability to effectively match users with relevant advertisements is critical to the success of online advertising companies such as Google and Facebook. Finally, recommender systems have also

been successfully applied to news, social networks, healthcare, and many other domains.

1.1 Recommender System Approaches

Formally, a recommender system is an algorithm that seeks to predict the rating or preference between a user and corresponding item. Recommender systems take a variety of forms, from content-based systems that rely on manually specified features, to collaborative filtering algorithms that rely on observed purchases of similar users and items to make recommendations. Lastly are model-based approaches, which have seen significant success in recent years. Model-based systems use a large corpus of training data to learn low-dimensional representations of each user and item. These user and item representations are then used to make future recommendations.

Within the overall category of model-based recommender systems, there have been several successful approaches. Matrix Factorization or latent factor models learn a low-rank approximation to the user-item rating or purchase matrix, learning useful dimensions along which to categorize users and items in the process [29]. Many improvements to matrix factorization models make use of additional sources of information, such as temporal, geographical, and social data to improve recommendation quality.

While matrix factorization models aim to directly model the predicted rating between a given user and item, sequential recommender systems add an additional dynamic: taking the order of previous interactions into account. Intuitively, this means that the likelihood of purchasing an item depends on a user's previous interactions. Successfully modeling these third order interactions (between a user, an item under consideration, and the previous item consumed) facilitates a more engaging user experience, resulting in recommendations that are

more responsive to recent user and item dynamics.

1.1.1 TransRec

In 2017, He et al., introduced TransRec, which is a novel framework for performing translation-based recommendation [19]. In traditional recommender systems, algorithms model user preferences and sequential dynamics separately, impacting performance and generalizability. As these two dynamics are inherently interconnected, they should be learned jointly to enable the insights of one part of the model to benefit the other. TransRec addresses this problem by defining a joint "translation space" that enables both preference and sequential dynamics to be jointly learned. Within this space, the model learns item embeddings as points in the space, similar to most traditional recommender systems. The key insight comes from representing users as translation vectors, explicitly modeling translations between their observed purchases. This follows a line of work that adapts ideas from metric learning [34] and knowledge-graph completion [48, 57] into recommender systems, which has led to state-of-the-art performance on a variety of tasks.

The intuition behind TransRec is as follows: items are embedded in different locations throughout the translation space, and each user follows an independent trajectory through the space. Representing each user as an independent translation vector allows the model to learn the dynamic nature behind a user's viewing or purchase habits, extending these patterns to new locations in the item space.

TransRec was successful as it modeled this complex translation dynamic with a simple formulation, enabling the model to perform well on very sparse datasets and scale to very large datasets. In addition, by using metric distances in the translation space rather than the standard inner products, TransRec is able to better generalize to new item sequences. In many

recent models [15, 11], employing metric distances has led to significant improvements in recommendation performance by taking advantage of the transitive property of the triangle inequality. For example, if items i and j and j and k are related, then the triangle inequality encourages i and k to be brought closer together in the space, assisting generalization behavior.

1.2 Proposed Extensions to TransRec

In this thesis, we discuss a variety of proposed extensions to the TransRec model. While TransRec was able to outperform many existing recommender system models on a variety of datasets, its simple form could hurt its performance.

Some examples of our proposed extensions are as follows. We learn more complex translations by adding an item translation component to the model. We also incorporate temporal features, specifically the time delta between previous and next item purchases. This allows us to model translations through the embedding space as a function of time.

Additionally, we test a variety of models that use neural networks in order to introduce a nonlinear component into the TransRec model. The last few years have seen a rapid rise in neural networks and deep learning, and these models are now the state-of-the-art for a variety of tasks including image, speech, and video processing. Neural networks have also been applied to the field of recommender systems, and a variety of models in recent years have taken advantage of deep learning capabilities to improve recommendation performance [9, 21, 56]. We employ neural network models to learn nonlinear translation vectors, allowing us to model more complex translation dynamics for each user through the item embedding space. Neural networks also excel at incorporating multiple sources of information into a single model in the form of extended feature vectors. Finally, we develop explicit session-based models that employ the generality

of neural networks to automatically partition a user's consumption sequence into "sessions" of related items purchased in a short time period.

1.2.1 Content Features

Another natural avenue to extending TransRec is to adapt its metric and translation intuition to incorporate content features. A few approaches have been attempted to integrate content features into metric learning and knowledge graph algorithms, such as using a specialized convolutional neural network to incorporate audio features [51] or using a variational Bayes technique for playlist generation with both collaborative and content features [5]. However, offering a general-purpose technique that incorporates content features into metric-based algorithms remains open.

Factorization Machines (FMs) achieve this goal in inner-product spaces, incorporating additional features without sacrificing model simplicity [41]. FMs operate on arbitrary real-valued feature vectors, and model higher-order interactions between pairs of features via factorized parameters. They can be applied to general prediction tasks and are able to replicate a variety of common recommender system models, such as matrix factorization and FPMC, simply by selecting appropriate feature representations.

We propose *TransFM*, which adopts ideas from FMs into translation-based sequential recommenders. Doing so allows us to straightforwardly model complex interactions between features (as in FMs) while extending the state-of-the-art performance of metric / translation-based approaches.

Specifically, we replace the inner product in the FM translation term with a translation component between feature embeddings, employing the squared Euclidean distance to compare compatibility between pairs of feature dimensions (see Figure 1.1). As with Factorization

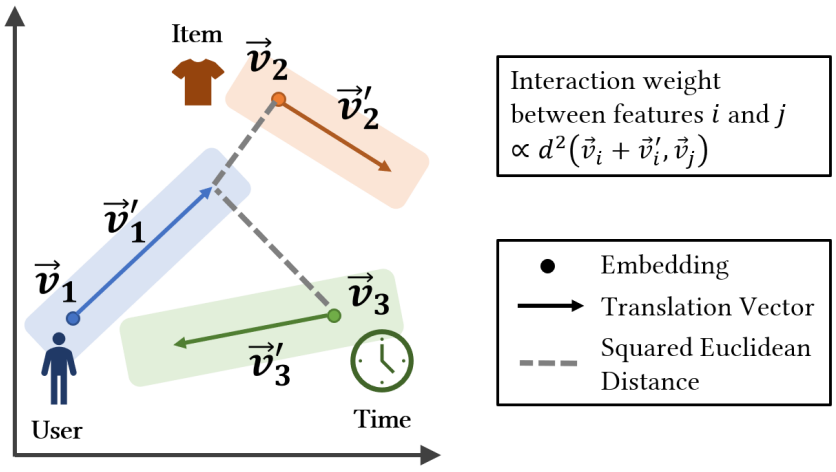


Figure 1.1. The general-purpose *TransFM* model. Unlike standard sequential and metric-based algorithms, *TransFM* models interactions between all observed features. For each feature i, the model learns two entities: a low-dimensional embedding $\vec{v}_i$ and a translation vector $\vec{v}'_i$. The interaction strength between pairs of features is then measured using the squared Euclidean distance $d^2(\cdot,\cdot)$. In the example above, we plot the embeddings and translation vectors for a user (feature 1), an item e.g. a movie or book (feature 2), and a temporal feature (feature 3). Interaction weights are given by the distance between the ending and starting points of the respective features.

Machines, we show that the *TransFM* model equation can be computed in linear time in both the feature and parameter dimensions, making it efficient to implement for large-scale sequential recommendation datasets.

The translation component of *TransFM* effectively learns relationships among collaborative and content-based features with minimal preprocessing and feature engineering. We empirically evaluate *TransFM* on a variety of datasets and find that *TransFM* with content features provides significant improvements over state-of-the-art baselines with and without additional features included.

We present a generalization of this approach and derive related models by merging FMs with similar baseline models. This leads to general-purpose recommendation approaches that incorporate the intuitions of other baseline algorithms, consistently outperforming vanilla

Factorization Machines.

1.3 Structure and Scope of the Thesis

In this thesis, we first provide an overview of the field of recommender systems followed by a discussion of the essential algorithms and advancements in Chapter 2. In particular, we trace the history of recommender systems, highlighting the major categories of algorithms, including content-based, collaborative filtering, and model-based approaches. We also discuss the main types of recommender systems, including implicit feedback, sequential, and ranking models.

In Chapter 3, we review related literature to highlight recent advances in the field. This includes algorithms that incorporate additional sources of information to improve recommendation performance (e.g. temporal, social, and geographical dynamics), as well as the rise of models that apply the advantages of neural networks and deep learning.

Chapter 4 expands the discussion on TransRec, presenting the formal model and highlighting potential areas for improvement. We introduce our proposed extensions in Chapter 5, presenting the intuition behind each model and its relationship to TransRec.

Quantitative and qualitative results are presented in Chapter 6, where we discuss each model's performance and analyze the learned item embeddings and translation vectors. In Chapter 7, we apply the intuition behind *TransFM* to related baselines, applying the FM framework to derive similar general-purpose recommendation models. Finally, we discuss conclusions and potential future work in Chapter 8.

Various chapters in this thesis include material that has been submitted for publication as it may appear in RecSys 2018, Pasricha, Rajiv; McAuley, Julian, ACM, 2018. The thesis author was the primary investigator and author of this paper.

Chapter 2

Background

In this chapter, we discuss relevant background for the field of recommender systems. Considering the variety of algorithms that have been successfully applied, we also provide a description of some of the most influential ones, that have inspired the development of many recent approaches.

The main purpose of a recommender system is to analyze observed interactions between users and items in a system, and to use this data to predict items that a user will rate highly or purchase in the future. Recommender systems can be applied to settings that involve *explicit* or *implicit* feedback. In an explicit feedback system, a user provides a positive or negative assessment of the items they have interacted with, traditionally on a 1-5 or thumbs-up thumbs-down scale. The goal of the recommendation algorithm then becomes to predict future ratings that users will give to unseen items. In the case of a ranking task, where the algorithm must return e.g. the top k items to display as recommendations, the system can simply return the k items with the highest predicted ratings.

This is in contrast to the implicit feedback setting, where users do not directly specify their level of interest or engagement with the platform. Instead, in an implicit feedback system, a user's level of interest with individual items must be inferred from their general interactions.

Purchases on Amazon or views on YouTube are examples of implicit feedback scenarios. As users are not directly specifying their likes and dislikes, each interaction carries a weaker positive or negative signal than in the case of explicit feedback. A user may watch a video because of genuine interest, or because of a recommendation from an acquaintance, or for a class assignment; the system cannot tell the difference. It is also more difficult to identify which items a user explicitly dislikes. In most cases, all interactions in an implicit feedback system are treated as positive signals, and the system is designed to predict the items that the user will purchase or interact with in the future. The benefit of implicit feedback scenarios is that they typically contain much more data, as users generate data simply by interacting with the system, rather than submitting a rating or review. Having this surplus of data enables personalized recommendations to be made to all users in the system, even those who have not provided explicit feedback.

A specific area of concern when designing and implementing recommender system algorithms is the *cold-start problem* [44]. Recommender system settings are commonly associated with extremely high sparsity; "cold-start users" are those with little or no associated data in a given training set. Given the large number of items available in a system, a typical user only purchases or rates a tiny fraction of possible items. Similarly, given the large number of users in the system, any particular item is likely to be rated or purchased by only a small fraction of users. As a result, recommender systems must be designed to accurately handle cases where new users or items enter the system with little or no associated data. There are multiple approaches to handling this problem, including only recommending the most popular items as a baseline or reverting to content-based approaches where no rating or purchase data is available. Algorithms and practical implementations must be able to appropriately handle sparsity, as users should be able to expect to receive quality recommendations from a system regardless of their level of

involvement.

There are two primary categories of recommender systems, specifically *content-based* and *collaborative filtering* approaches. Content-based recommender systems rely on features associated with individual users and items. Pandora is one of the most famous examples of a content-based recommender system. It is based on the Music Genome Project, which endeavors to assign scores to songs based on hundreds of individual characteristics [39]. Some of these characteristics include song genre, instrumental vs. vocal, and artist gender. Given a user's listening history, Pandora creates a profile of preferred musical characteristics and creates a personalized radio station centered around songs with similar attributes. While comprehensive content-based approaches can perform well, they tend to suffer from scalability issues and are limited by the creativity of feature designers and accuracy of the assigned scores.

In contrast, collaborative filtering recommender systems do not rely on features of the users and items themselves. Instead, they automatically analyze dependencies in the provided dataset and use these similarities to make recommendations. *Neighborhood-based* collaborative filtering approaches aggregate observed rating and purchase data and explicitly compute similarities between users and items. This enables a collaborative filtering algorithm to, for example, automatically "learn" that an individual is a fan of action movies, and provide movies watched by other action fans as recommendations. The two predominant types of neighborhood-based collaborative filtering algorithms are user-based and item-based collaborative filtering. In a *user-based* collaborative filtering algorithm, recommendations are made based on items purchased by similar users [8]. This is in contrast to *item-based* collaborative filtering, where recommendations are based on similar items in a user's purchase history. [31].

A *model-based* collaborative filtering approach uses a machine learning algorithm to

develop an internal representation of users and items in the system. This model is trained with observed ratings or purchases and predicts the best items to recommend. A variety of machine learning algorithms have been successfully applied to the recommender systems task, including Markov chains [43], neural networks [9], and dimensionality reduction techniques such as matrix factorization [29].

A significant advantage of neighborhood and model-based collaborative filtering over content-based approaches is that the system can automatically extract the most important characteristics of users and items from a recommendation perspective, evolving to changing characteristics of the dataset over time. As they do not rely on manually assigned feature values, collaborative filtering algorithms are also considerably more scalable, as long as a sufficient number of interactions are present in the training dataset.

2.1 The Netflix Prize

Announced in 2006, the Netflix Prize was a challenge posed by video streaming and rental service Netflix, to improve the performance of its commercial recommender system by 10% [6]. The prize generated significant publicity and also spurred considerable interest in the field with a $1 million grand prize and the release of the largest training dataset to date, consisting of over 100 million Netflix ratings, with associated users, items, and timestamps. As the first recommendation dataset of this magnitude to be released, the Netflix Prize dataset became a popular test bed for novel recommendation algorithms and highlighted the mutual importance of algorithms and data in designing successful recommender systems.

However, the dataset also highlighted the importance of privacy, especially in commercial settings. Although users and items were anonymized, it was shown to be possible to deanonymize

the dataset and identify individual users and movies by their corresponding reviews on related sites, such as the Internet Movie Database [36]. Unfortunately, by discovering individual records in the anonymized dataset, researchers were also able to identify other potentially sensitive information about Netflix users, such as their interests and political beliefs. As a result, the Netflix Prize dataset is no longer publicly available.

The winning submission to the Netflix Prize Competition, "BellKor's Pragmatic Chaos," developed a complex ensemble model consisting of hundreds of constituent models developed by three of the leading teams [27, 47, 38]. These component models included a variety of neighborhood and model-based collaborative filtering algorithms, as well as additional models such as Restricted Boltzmann Machines. One of the most effective approaches in the ensemble was matrix factorization, which learned low-dimensional user and item representations via stochastic gradient descent over the training set. The effectiveness of matrix factorization is based on its scalability and extensibility, and many of the component models in the winning solution used an updated form of this basic algorithm, augmented with neighborhood or temporal components. In addition, similar extensions to matrix factorization have since been proposed that incorporate social, geographical, or text-based data to improve performance. Along with the tremendous increase in available data and processing power in recent years, these improved models have led to successful recommender systems being applied in almost every conceivable domain, and they are now critical to the success of any commercial system.

2.2 Neighborhood-based Collaborative Filtering

We first discuss neighborhood-based algorithms for collaborative filtering, which do not learn a prespecified model to make predictions of future ratings. Instead, neighborhood-based

approaches calculate the similarity between related users and items and make recommendations based on corresponding similarity values. The two predominant types of neighborhood-based algorithms are user-based and item-based collaborative filtering. Both approaches are compatible with explicit and implicit feedback settings, where implicit feedback can be represented as a binary rating value.

We first discuss user-based collaborative filtering. Let u be the target user for whom we are trying to provide recommendations. The user-based algorithm starts by computing the similarity between u and all other users in the dataset. The similarity is defined over the observed rating history, so two items u and v who have provided similar ratings to overlapping items should have a high similarity value. One commonly used measure of similarity is the Pearson Correlation Coefficient, defined below [2]. I_u represents the set of items which have received ratings by user u, and μ_u is the average of user u's provided ratings. Finally, r_{uk} is the rating given by user u to item k.

$$S(u,v) = \frac{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} (r_{vk} - \mu_v)^2}} \tag{2.1}$$

The similarity computed by the Pearson Correlation Coefficient is normalized between -1 and 1 and has the advantage that it takes each user's average rating into account. This is important in practical systems, especially when one user provides much higher ratings than another on average. Another common similarity function between users is the Cosine Similarity, which is the cosine of the angle between two users' rating vectors [2]:

$$S(u,v) = \frac{\sum_{k \in I_u \cap I_v} r_{uk} r_{vk}}{\sqrt{\sum_{k \in I_u \cap I_v} r_{uk}^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} r_{vk}^2}} \tag{2.2}$$

Once these similarity values have been computed for all pairs of users, we can then predict the rating that user u will give a new item j as follows:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in N_k(u,j)} S(u,v)(r_{vj} - \mu_v)}{\sum_{v \in N_k(u,j)} |S(u,v)|} \tag{2.3}$$

Where $N_k(u,j)$ is the neighborhood of the k most similar users to u, who have provided ratings for item j.

While the user-based collaborative filtering approach recommends products purchased by similar users, item-based collaborative filtering simply recommends products that are similar to those that the user has already rated or purchased [31]. The computation proceeds similarly to the user-based model. We first compute the similarity between all pairs of items i and j, using the ratings observed in the dataset. Let U_i be the set of users who have given ratings for item i. As in the user-based case, we can use the Pearson Correlation Coefficient to compute the similarity between items i and j [2].

$$S(i,j) = \frac{\sum_{u \in U_i \cap U_j} (r_{ui} - \mu_i) \cdot (r_{uj} - \mu_j)}{\sqrt{\sum_{u \in U_i \cap U_j} (r_{ui} - \mu_i)^2} \cdot \sqrt{\sum_{u \in U_i \cap U_j} (r_{uj} - \mu_j)^2}} \tag{2.4}$$

If we are simply generating a ranked list of recommendations, we can find the top k most similar items to a user's historical interactions and return these as recommendations. This is similar to the method used by Amazon in the "Customers who bought this item also bought" section of each item page. It is also possible to retrieve explicit rating predictions using the item-based approach. This can be done by taking a weighted average of the target user's ratings for similar items. Specifically, let $N_k(u,i)$ be the neighborhood of the k most similar items to the

target item i which have been reviewed by user u. Then, the predicted rating for i is then:

$$\hat{r}_{ui} = \frac{\sum_{t \in N_k(u,i)} S(i,t) \cdot r_{ut}}{\sum_{t \in N_k(u,i)} |S(i,t)|} \tag{2.5}$$

While the item-based approach is less personalized than the user-based neighborhood model, it is significantly faster and more scalable. It was one of the preferred methods of choice for Amazon to provide recommendations on its item pages, as detailed in [31]. Once item-similarities have been computed offline, it is possible to compute a ranked list of recommendations very quickly. The computation scales only with the number of items rated by a user, rather than the total number of users and items in the dataset, which would be prohibitive for a website of Amazons scale.

2.3 Matrix Factorization

While neighborhood-based collaborative filtering approaches are simple to understand and implement, they are less competitive when it comes to recommendation performance. Model-based techniques consistently outperform neighborhood approaches, with matrix factorization methods being one of the most common techniques.

Popularized by Koren et al. in their winning Netflix prize submission [29], the standard matrix factorization approach learns a vector of latent factors for each user and item in the dataset. A high value of the inner product between corresponding user and item latent factors leads to a high rating prediction, and vice versa. This is equivalent to embedding users and items into a shared low-dimensional space, where similar users and items are located closer together if they share similar characteristics in the observed training dataset.

The matrix factorization model essentially aims to factorize the large but extremely sparse

user-item ratings matrix, and reconstruct it using a low-rank approximation. This is similar to the *Singular Value Decomposition* (SVD), a matrix decomposition technique with a wide variety of applications. However, the SVD operates on dense matrices and is prone to overfitting, so matrix factorization uses a gradient descent-based approach with regularization to learn an effective and generalizable model.

The model equation for matrix factorization is as follows:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u \tag{2.6}$$

μ denotes the average rating over all observed ratings in the dataset. For each item i or user u, b_i and b_u denote the average deviations for i and u from μ. These three parameters encapsulate the linear effects for the overall dataset, as well as the independent effects of individual users and items. For example, if $\mu = 2.9$ and $b_i = 0.6$, this means that the particular item i tends to receive ratings that are 0.6 stars higher than average, or 3.5 stars. If $b_u = -0.2$, this means that user u tends to rate movies on average 0.2 stars lower than average, or 2.7 stars. These bias terms help separate the effects of each predicted rating into independent parameters, so that the more complex interaction parameters q_i and p_u do not need to account for easily-modeled effects.

Finally, the interaction parameters $q_i \in \mathbb{R}^f$ and $p_u \in \mathbb{R}^f$ model the latent factors associated with user u and item i. f is the dimensionality of the latent factors and is a hyperparameter. Each of the f dimensions in each latent factor can be interpreted as representing a specific dimension associated with the dataset, and the value in that dimension corresponds to the extent to which the item exhibits the factor, or the user enjoys items that possess the factor. For example, one

possible dimension could encapsulate the extent to which a movie displays characteristics of romance vs. action, or whether it is geared towards male or female audiences. The strength of the matrix factorization model is that it learns these dimensions from the data itself, rather than requiring predetermined content features for each user and item. However, this comes at a cost of model interpretability. In most cases, it is possible to analyze the learned factor values for individual users and items to determine an interpretation of the factor dimensions, but this is not always feasible.

This model is trained using a squared error loss function along with an $\mathcal{L}_2$ regularization term, and the parameter values are optimized using stochastic gradient descent. The matrix factorization model is scalable and easily extensible, and thus forms the basis for many later state-of-the-art recommender system approaches. Even in the winning Netflix prize submission, Koren et al. do not rely solely on the basic matrix factorization model described above. Instead, they extend the model by adding implicit feedback data, varying confidence levels of observed ratings, and incorporating temporal dynamics [27]. For temporal dynamics, some of the bias and factor parameters are updated to evolve as a function of time, either by learning a parameterized form or by bucketizing the input data and learning multiple independent values. The resulting model equation thus becomes:

$$\hat{r}_{ui} = \mu + b_i(t) + b_u(t) + q_i^T p_u(t) \tag{2.7}$$

In most cases, adding additional data to matrix factorization helps to improve performance, as the model is able to rely on the additional signals provided by this new data to learn a more generalizable function. In [29], the best performing models were those that incorporated

temporal dynamics, demonstrating the importance of appropriately modeling time when creating a successful recommender system model.

2.4 Markov Chains

The next overall category of recommender systems is Markov chain models. Markov chains are stochastic models in mathematics that describe probabilistic sequences of events. They consist of a series of states with a matrix describing the probability of transitioning between any pair of states. The "Markov" of Markov chains comes from the Markov property, which states that the probability of transitioning to the next state only depends on the current state, and not any previous historical state sequence. This is why we can describe transition probabilities as a matrix from previous to next states, rather than as a higher order tensor starting from a longer sequence of previous states.

With regards to recommender systems, Markov chain models operate on the intuition that sequential dynamics are important for making recommendations. In the matrix factorization model discussed above, the predicted rating between a user and item depends only on the identities of the user and item, and not on which items a user has recently interacted with. This is inconsistent with most intuition, as we should expect the item that will be purchased or rated next by a particular user to be related to their previous interaction sequence. Markov chain and related models are often applied in implicit feedback settings, where the goal of the recommender system is to determine which items a user will view or purchase next, rather than the rating that will be given to that item. For example, in the Netflix case, if the five most recent movies that a user watched are in the "Action" genre, then we should expect an increase in the probability that the next movie will be an action movie.

The Markov property comes into play with recommender systems as well, representing the simplifying assumption that the next interaction only depends on the most recent interaction, rather than a longer sequence of previous events. This assumption considerably simplifies Markov chain-based recommendation models and also improves their scalability and generalizability. In particular, a model that relies on a longer sequence of previous items has a considerably higher risk of overfitting to the sequences in the provided training set, and appropriate precautions must be observed to maintain adequate generalization performance.

In 2010, Rendle et al. proposed the *Factorized Personalized Markov Chain* (FPMC) model for recommendation [43]. This model combines aspects of Markov chain and matrix factorization models, learning latent user and item factors of low dimensionality while also considering sequential behavior by taking the previous item into account. Empirically, combining these models enables FPMC to outperform both raw matrix factorization and Markov chain approaches.

In [43], Rendle et al. apply FPMC to the task of next-basket recommendation, e.g. recommending a set of items that a user will purchase next, such as a shopping cart on an e-commerce site. Formally, given a sequence of user baskets $B^u = (B^u_1, \ldots, B^u_{t_u-1})$, the task is to recommend $B^u_{t_u}$, the entire basket of items that will be purchased at the next time step. The model aims to learn a probability that each item will appear in the next basket, given the user and the set of previous items. This leads to a three-dimensional probability tensor A, with independent dimensions for users, previous items, and next items. In particular, the entry $a_{u,l,i}$ represents the probability that item i will be in the next basket for user u, given that item l was in the previous basket.

$$a_{u,l,i} = P(i \in B^u_{t_u} | l \in B^u_{t_u-1}) \tag{2.8}$$

It is infeasible to learn a cubic number of transition probabilities, so FPMC uses the Tucker Decomposition to convert the overall tensor into a combination of factorized matrices. In particular, FPMC defines six parameter matrices: $V^{U,I}$ and $V^{I,U}$ to model the interactions between the user and the next item i; $V^{U,L}$ and $V^{L,U}$ to model the interactions between the user and the previous item l; and $V^{I,L}$ and $V^{L,I}$ to model the interactions between the previous item l and the next item i. For the item recommendation task, the user-previous item (U,L) decomposition is redundant, and can be removed from the prediction equation. Using these factorized matrices, FPMC computes predictions as follows:

$$P(i \in B_t^u | B_{t-1}^u) = \langle v_u^{U,I}, v_i^{I,U} \rangle + \frac{1}{|B_{t-1}^u|} \sum_{l \in B_{t-1}^u} \langle v_i^{I,L}, v_l^{L,I} \rangle \tag{2.9}$$

In order to recommend a specific basket, these values are calculated for each proposed item i, and the items with the highest predicted probabilities are returned. These probabilities are calculated by summing over the respective transition probabilities for all previous items l in the most recent basket. The model is learned using stochastic gradient descent with the S-BPR optimization criterion. S-BPR, or *Sequential Bayesian Personalized Ranking*, is a comparison-based optimization approach which maximizes the difference between correct and incorrect predictions and is widely used for ranking tasks [43]. S-BPR will be discussed in detail in the next section.

We next discuss *Personalized Ranking Metric Embedding* (PRME), proposed by Feng et al. in 2015 [15]. The main insight of this model is that it moves beyond the inner product to model interactions between users and item factors, instead relying on the Euclidean distances between locations in a shared embedding space. The model was proposed to address the next

point of interest (POI) recommendation task, where a sequence of user visits to various points of interest is observed and the recommender system provides a new POI to visit next. In order to determine which POI to recommend, all POIs are embedded in a shared latent space and the probability of a sequential transition between two objects is given by the Euclidean distance between them. Items located closer together in the space are more closely related, so a higher translation likelihood is expected.

Formally, PRME is defined as follows. The model defines a user preference space X^P which represents the personalized recommendation component. A user and item located close together in this space represents a higher user preference for the item. A sequential transition space X^S is also defined, in which the Euclidean distance between two items represents the likelihood of a sequential transition between them. The model computes the "distance" between the previous POI l^c and the next POI l for user u as follows:

$$\mathcal{D}_{u,l^c,l} = \alpha||X^P(u) - X^P(l)||^2 + (1-\alpha)||X^S(l^c) - X^S(l)||^2 \tag{2.10}$$

Where α is a hyperparameter determined using a validation set. As with FPMC, PRME is trained using the BPR optimization technique using the maximum likelihood approach.

Intuitively, the metric embedding approach outperforms traditional inner products due to the additional generalization performance gained from the triangle inequality. By representing users and items as points in metric spaces, the model is able to cluster related users and items together in the preference space, and related sequential transitions in the sequence space. The triangle inequality, a property of the Euclidean distance metric used by the model, states that if two pairs of items a, b and b, c are located close together, then the pair a, c should be located close

together as well. This facilitates generalization, as if the original two translations are observed, then it is highly likely that items a and c are closely related, even if the specific transition a, c is not observed in the training set. Therefore, by enforcing the triangle inequality, the model ensures that these transitive relationships are maintained, which leads to more accurate predictions if these unobserved transitions appear in future test examples.

2.5 Bayesian Personalized Ranking

Next, we discuss the BPR optimization technique for learning recommender system models that rely on implicit feedback [42]. In most rating prediction recommender systems, such as the standard matrix factorization model discussed above, the squared error loss function is employed. This loss function aims to minimize the squared difference between predicted and observed ratings and is one of the most commonly used loss functions in all of machine learning. It works well when a real number is to be predicted, as is the case when the recommender system is trained to output a predicted rating value as a real number. In the implicit feedback case, such as where the input data to the algorithm consists of binary interactions or purchases, the model can be trained to output a probability of a future interaction between user u and item i, using the Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$. In such cases the logistic loss is commonly used, as defined below:

$$L = \sum_{y_i=1} \log \Pr(y_i|X_i) + \sum_{y_i=0} \log(1 - \Pr(y_i|x_i)) \tag{2.11}$$

The above two loss functions, and many commonly used alternatives, are intended to optimize the model's predicted values directly. In the squared error case, the loss function attempts to tune the model such that the output rating most closely matches the observed rating, and in the logistic case, predictions for observed examples are tuned to have high probability,

while unobserved examples are trained to have predicted probabilities close to zero.

However, these loss functions do not adequately model most recommendation scenarios. In the majority of practical cases, a recommender system is not intended to directly predict an output rating or probability of purchase, but rather is applied to a ranking task. Specifically, in most cases where recommendations are visible, the system is expected to provide a list of the top n items that the user will be most likely to purchase. The exact order of the items within this list is less relevant, as long as the items contained within the list are more relevant to those which are excluded.

Bayesian Personalized Ranking (BPR), introduced by Rendle et al. [42], is intended to address this problem. BPR specifies an optimization function tailored to the ranking task, so that recommender systems can be appropriately tailored to the task that they are intended to solve. BPR is also model agnostic. It is defined as an optimization step that takes as input the output of a rating or purchase prediction model, and so excels as a general-purpose optimization criterion. Intuitively, rather than learning the output values of the constituent model directly, BPR aims to learn a ranking over all items. This means that for any given positive example, the model output should be greater than the corresponding output for a negative example. By using such an optimization criterion, the model learns to rank relevant items ahead of irrelevant items, a much more natural technique for ranking-oriented recommender systems.

Formally, the BPR model aims to learn a personalized total ranking $>_u \subset I^2$ over all pairs of items. The model defines the probability that user u prefers item i over item j as follows:

$$p(i >_u j|\Theta) = \sigma(\hat{x}_{uij}(\Theta)) \tag{2.12}$$

Where $\hat{x}_{uij}(\Theta)$ is a real-valued scalar that captures the relationship between u, i, and j. The computation of $\hat{x}_{uij}$ is delegated to the underlying model class. Using a maximum a-posteriori estimator, the BPR optimization criterion can be defined:

$$\text{BPR} - \text{Opt} = \sum_{(u,i,j) \in D_s} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta ||\Theta||^2 \tag{2.13}$$

In most cases, BPR is used with algorithms that model the interaction between a user and a single item. In such cases, the relationship term $\hat{x}_{uij}$ can be represented as the difference between the raw predicted values: $\hat{x}_{uij} = \hat{x}_{ui} - \hat{x}_{uj}$.

In order to optimize a model according to BPR, [42] presents the LearnBPR algorithm, which is a combination of random sampling and stochastic gradient descent. In order to perform an optimization step, the algorithm requires a user u, a positive item i, and a negative item j. The users and positive items are typically included along with standard training sets. However, as the vast majority of possible interactions are unobserved, it is infeasible to calculate the gradient over all user-positive item-negative item triplets. So, LearnBPR uniformly samples a (u,i,j) triplet at each training step, followed by an SGD optimization step.

Finally, we note that BPR is analogous to optimizing the AUC, or area under the ROC curve. The AUC is a commonly used ranking metric that aggregates the relationship between the true positive and false positive rate of a classifier. It can also be interpreted as measuring the fraction of triplets in the dataset for which the positive item is ranked ahead of the negative item. Formally:

$$AUC = \frac{1}{C} \sum_{(u,i,j) \in D_s} \delta(\hat{x}_{uij} > 0) \tag{2.14}$$

Where C is a normalization constant enforcing the AUC value to be between 0 and

1. This is analogous to BPR, except the AUC uses the step function $\delta(x)$ whereas BPR uses the differentiable Sigmoid function. This means that models trained using BPR are trained to maximize the AUC, making the BPR method particularly well suited to the ranking task.

2.5.1 Sequential BPR

The above BPR approach optimizes a personalized item ranking independently of the sequence of user interactions. For sequential data, a related approach can be derived called *Sequential Bayesian Personalized Ranking* (S-BPR) [43]. This approach is analogous to BPR but rather than optimizing a global total ordering $>_u$ for each user, a total order $>_{u,t}$ is optimized for user u at time t, taking the sequential nature of their consumptions into account.

In the S-BPR case, the probability that user u prefers item i over item j at time t is given by:

$$P(i >_{u,t} j|\Theta) = \sigma(\hat{x}_{u,t,i} - \hat{x}_{u,t,j}) \tag{2.15}$$

The derivation proceeds similarly, but rather than summing over all triplets (u,i,j), S-BPR also incorporates a temporal component. For FPMC in particular, each user u is associated with a set of item baskets $\mathcal{B}^u$, giving the following optimization criterion:

$$\text{argmax}_\Theta \sum_{u \in U} \sum_{B_t \in \mathcal{B}^u} \sum_{i \in B_t} \sum_{j \notin B_t} \ln \sigma\left(\hat{x}_{u,t,i} - \hat{x}_{u,t,j}\right) - \lambda_\Theta ||\Theta||_2^2 \tag{2.16}$$

We apply the S-BPR approach to optimize TransRec and our proposed extensions.

2.6 Factorization Machines

Most recommender system algorithms, including those discussed above, operate on user and item interactions, which are then associated with corresponding latent feature vectors. For

example, in matrix factorization, each user is associated with a latent vector q_u and each item is associated with a latent vector p_i. These models are designed for a training dataset that simply consists of a history of user-item interactions, without any additional data.

Additional sources of data can be incorporated into matrix factorization and other recommender system models, such as temporal data, social network connections, sequential data, and geographical information. However, the format of the model must be updated to take these data sources into account, e.g. by adding new bias, interaction, or regularization terms. This contrasts with most machine learning algorithms in other domains, which operate on arbitrary feature vectors and output values. When training a neural network or support vector machine, for example, the model is agnostic to the specific application, and instead aims to learn the function that transforms input feature vectors into desired output values.

Factorization Machines (FMs) [41], aim to bridge the gap between factorization methods for recommender systems and models that work with arbitrary feature vectors; a Factorization Machine is a general predictive model that operates on arbitrary feature vectors and outputs. FMs are designed to work well in problems with high sparsity, such as recommender systems, by modelling higher order interactions between features with factorized parameter vectors. They are also efficient, with the model equation requiring only linear time computation and storage requirements. This allows them to scale to massive datasets similar to those commonly seen in recommendation tasks. Finally, Factorization Machines are a general-purpose model that can subsume many commonly used recommender system algorithms by selecting appropriate feature vectors, including Matrix Factorization [29], SVD++ [26], and FPMC [43].

Formally, the model equation for a Factorization Machine is specified as follows. Given

an input feature vector $\vec{x}$, the predicted output value $\hat{y}$ is:

$$\hat{y}(\vec{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \vec{v}_i \vec{v}_j \rangle x_i x_j \tag{2.17}$$

Where the learned model parameters are the global bias $w_0 \in \mathbb{R}$, linear factors $\vec{w} \in \mathbb{R}^n$, and factorization terms $\mathbf{V} \in \mathbb{R}^{n \times k}$. n is the feature dimension of the x_i, and k is a prespecified latent dimension.

As compared to SVMs with a quadratic polynomial kernel, FMs do not learn a single parameter for every pair of input features. Instead, they learn a set of factors for each feature and use the inner product to weight higher order interactions between respective features. With a properly chosen feature dimension k, this facilitates improved generalization performance by reusing parameters across the model and reducing the number of parameters that must be learned.

As they simply specify a model prediction function, FMs can be trained using any loss function or training method, but stochastic gradient descent using the square or logistic loss is commonly used. BPR is also used to optimize factorization machines when applied to ranking problems in recommender systems. Their flexibility and ability to work with arbitrary feature vectors make Factorization Machines a method of choice for a wide variety of applications, in recommender systems and related domains.

Various chapters in this thesis include material that has been submitted for publication as it may appear in RecSys 2018, Pasricha, Rajiv; McAuley, Julian, ACM, 2018. The thesis author was the primary investigator and author of this paper.

Chapter 3

Literature Review

In this chapter, we summarize and review relevant literature in a variety of subfields of recommender systems. In particular, we describe many extensions to the basic models previously discussed that draw upon additional sources of data commonly found in recommendation settings, as well as additional methods in the field of machine learning as a whole that have been successfully applied to improve recommendation performance.

3.1 Matrix Factorization

We first discuss some extensions to and applications of the standard matrix factorization recommender system model. As previously discussed, matrix factorization models exploded in popularity due to their success in the Netflix Prize competition and have remained among the best performing recommender system models ever since. However, the strength of the matrix factorization framework is not in the performance of the original model. Instead, matrix factorization is one of the most extensible recommendation frameworks, and significant performance gains have been achieved by adding additional features or complexity to the model. For reference, the standard matrix factorization model includes user and item biases, and an interaction component between low-dimensional factors associated with each user and item and

derived from the provided rating or purchase history.

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \tag{3.1}$$

In [45], Sembium et al. apply the matrix factorization framework to recommending products of appropriate sizes to customers on Amazon.com. This model is applied to shoe sizes in particular, a challenging problem as shoe sizes tend to differ significantly between manufacturers, forcing users to compensate for manufacturer bias when making online purchases. In particular, [45] defines latent factors that encode the predicted true size of each user, irrespective of a particular shoe manufacturer. This enables size recommendations to be made by comparing a user's true size with the direct measurements of each provided manufacturer size option. Formally, this is done by defining a loss function that imposes different penalty on the true size value, based on whether the user reported a small, large, or precise fit.

Probabilistic Matrix Factorization (PMF) is a probabilistic version of the matrix factorization technique [35]. PMF aims to solve the same problem as matrix factorization but takes a probabilistic approach. A probabilistic model with Gaussian noise is adopted, placing Gaussian priors on user and item feature vectors. Given these distributions and the desired outputs, the model maximizes the log-posterior distribution over movie and user features, which is equivalent to minimizing the mean squared error with $\mathcal{L}_2$ regularization. Thus, PMF becomes nearly equivalent to matrix factorization, but the probabilistic derivation allows the model to be more straightforwardly extended probabilistically. Some examples are as follows: (1) introducing prior distributions on the regularization hyperparameters allows model complexity to be automatically derived from the training data, and (2) enforcing similar prior means for similar users facilitates

improved recommendation performance on infrequently observed users.

An additional example that shows the impressive extensibility of matrix factorization is [26], where matrix factorization models are combined with neighborhood-based collaborative filtering approaches. This allows the combined model to draw upon the advantages of each constituent approach, improving performance beyond each model independently.

By extending user factors with neighborhood terms, the model is able to explicitly incorporate the relationships between each user and the explicit and implicit signals in his or her neighborhood of items. These relationships might not be extracted as clearly if only the matrix factorization component were used, and the performance gains afforded by learning implicit user and item factors are not incorporated into a purely neighborhood-based model.

Finally, we discuss CoFactor, proposed by Liang et al. [30]. Starting with the original matrix factorization model, CoFactor adds an additional optimization term that takes into account the item co-occurrence matrix. In particular, the item co-occurrence matrix encodes the frequency with which every pair of items i and j is purchased or rated together by the same user, forming another measure of the similarity between two items. This information is incorporated into matrix factorization by factorizing the shifted positive pointwise mutual information (SPPMI) matrix M.

The item factors β_i are shared between the matrix factorization and co-occurrence components of the objective function, leading to the desired regularization behavior where β_i must account for both user-item and item-item interactions. This helps regularize similar items, which tend to be purchased together, to have similar embeddings, improving the model's generalization behavior. One distinct advantage of CoFactor is that item co-occurrence information is present in all recommender systems datasets, making the model applicable in any situation.

While we have only provided a short list of matrix factorization extensions above, a large number of models that will be discussed in future sections incorporate elements of this framework.

3.2 Markov Chains and Sequential Recommendation

Next, we discuss recommender system approaches that adopt the Markov chain approach to making recommendations. Markov chain recommendation models are especially well suited to making sequential recommendations, where the next item to be rated or purchased by a given user depends on the previous items in the user's consumption sequence. Non-sequential models such as matrix factorization only take a single user and item into account and predict a rating or probability of consumption regardless of the order in which the items were consumed. Two influential sequential recommender systems are FPMC and PRME, previously discussed in Chapter 2. In this section, we present a selection of extensions to these models as well as some additional sequential recommender systems.

In [7], Benson et al. analyze user consumption sequences in the context of recommender systems. In particular, they discuss two common phenomena observed in consumption sequences, namely boredom and abandonment. Specifically, they observe that the majority of repeated consumptions do not occur frequently throughout the entire sequence. Instead, they occur sporadically from the start of their observed occurrences, decreasing in frequency over the course of their lifetime as the user becomes bored of consuming the same item. The boredom phenomenon is characterized by steadily increasing temporal gaps between the user's repeated consumptions of the item. Eventually, the user purchases the item for the final time, a phenomenon termed abandonment. [7] analyzes these phenomena in a variety of real-world datasets, observing

that the phenomena of boredom and abandonment occur consistently and predictably across sequences in all datasets.

A combination of three models is proposed to take the above effects into account, first, a "temporal model" predicts the inter-arrival times in a user's sequence $\Delta_{i+1} = t_{i+1} - t_i$. Next, a "novelty model" determines whether the user will purchase a novel item in the next time step, or repeat a purchase previously made earlier in the sequence. Finally, the "choice" model determines the identity of the item to predict, chosen from either the set of non-purchased items for a novel consumption, or the set of previously purchased items for a repeat consumption. Although boredom and abandonment are not explicitly encoded into the three models, the authors find that the parameters of the trained models cause these trends to be observed in item predictions, highlighting their significance in user consumption sequences.

In [11], Chen et al. propose the Latent Markov Embedding (LME) model to generate music playlists. This model combines aspects of Markov chain models for sequential recommendation with metric embeddings. The first variant of LME represents each song by a single point in a latent embedding space and models the transition probabilities between adjacent songs in a playlist by the distance between their respective embedding locations.

A dual-point model is also proposed, where each song is represented by two points $U(s)$ and $V(s)$ in the embedding space, representing the "entry" point and "exit" points of the song respectively. This is to model unique characteristics associated with songs that come before and after a particular song in a playlist. Songs that come before a given song might have significantly different characteristics from songs that come after and allowing for two points per song takes this effect into account.

The LME model does not require content features for each song in a playlist, and also

facilitates playlist generation and discovery. The authors also highlight the extensibility of their proposed model framework, demonstrating the ability to add additional features such as song popularity, user personalization, and semantic tags.

Wang et al., in [53], propose a Hierarchical Representation Model (HRM) for next-basket recommendation. As was the case with FPMC, the model aims to predict the next basket of items that will be purchased, such as a shopping cart or a playlist with multiple items. HRM combines both sequential and personalized dynamics to improve predictive performance. The model relies on aggregating multiple factors to form a concise hybrid representation encapsulating both the user and previous items. Formally, the HRM model learns an embedding of users and items in a low-dimensional latent space and aggregates these latent representations to construct a hybrid representation of a user's last transaction.

To construct the hybrid representation, HRM applies aggregation operations on the user and previous item representations. The two aggregation operations discussed are average and max pooling over the constituent vectors. The max pooling operation introduces a nonlinear component into the model, and thus achieves better performance on the datasets considered. Given aggregation functions f_1 and f_2, and the previous basket T^u_{t-1}, the hybrid representation is constructed as follows:

$$\vec{v}^{Hybrid}_{u,t-1} = f_2\left(\vec{v}_u,\ f_1\left(\vec{v}_l \in T^u_{t-1}\right)\right) \tag{3.2}$$

The model computes the probability of a proposed item being contained within the next basket according to the inner product between the proposed item and the hybrid representation. The model can be trained by applying stochastic gradient descent to maximize the log-likelihood over the observed baskets in the training set. The authors of [53] also show that by selecting specific aggregation functions, the HRM model can reduce to the Markov Chain,

Matrix Factorization, or FPMC algorithms for recommendation, highlighting its generality and extensibility.

Finally, we discuss Fossil, a model proposed by He and McAuley that combines Markov chain and similarity-based recommendation models [20]. While Markov chains excel at sequential recommendation tasks, similarity-based models facilitate improved prediction performance on datasets exhibiting high sparsity, by enforcing the constraint that recommendations for a given user u should be similar to those items previously consumed by u. Similarity-based models are related to the neighborhood-based collaborative filtering models previously discussed and improve generalization performance by reducing the number of parameters that must be learned and by taking transitive item-to-item relationships into account.

Fossil builds on FPMC and a state-of-the-art similarity recommendation model, called Factored Item Similarity Models (FISM). FISM applies the matrix factorization approach to the item-to-item similarity matrix W, factorizing it into two low-rank matrices $W = PQ^T$, which significantly improves performance. To reduce the number of parameters that must be learned, Fossil enforces equality among the low-rank item matrices, encoding the intuition that similar items should also be sequentially related.

This model can be trained using sequential Bayesian Personalized Ranking (S-BPR), optimizing the maximum a posteriori (MAP) estimation. Evaluating the results, the authors observed that the Fossil model outperformed existing baselines and was able to successfully capture both sequential and personalized dynamics, relying more on sequential dynamics for users with few observed actions and vice versa.

3.3 Temporal Dynamics

Next, we discuss two recommender system models that extend basic algorithms by incorporating temporal dynamics. From a practical perspective, In addition, temporal information is included in the majority of recommender system datasets, as every rating, purchase, or interaction in a commercial system happens at an associated timestamp. As a result, most commercial recommender systems rely on temporal dynamics to improve recommendation performance, and temporal data is incorporated into a large number of proposed academic models as well.

We first discuss Time Weight Collaborative Filtering, proposed by Ding et al. [14]. This model extends basic item-to-item collaborative filtering, a simple neighborhood-based collaborative filtering model which makes recommendations by selecting items similar to those a user has already purchased. While the item-to-item model sacrifices elements of complexity found in matrix factorization and other state-of-the-art models, it is extremely scalable [31]. In particular, the expensive item-to-item similarity table is computed offline and requires quadratic complexity. The online recommendation algorithm scales linearly with the number of items rated or purchased by a customer, rather than the total number of users or items in the dataset. As shown on Amazon.com, given a massive dataset, this algorithm scales effectively and provides quality recommendations.

Time Weight Collaborative Filtering introduces a temporal scaling parameter to the prediction equation. The temporal function proposed in the paper is the simple exponential function $f(t) = e^{-\lambda t}$. This means that items rated long ago have an exponentially decaying effect on the predicted rating for future items. This additional temporal component significantly

improves the performance of the item-based collaborative filtering algorithm, measured in Mean Absolute Error, while maintaining the desired scalability properties of the original algorithm.

For temporal extensions to the matrix factorization model, we turn to Bell et al.'s 2008 progress report for the Netflix Prize competition [4]. During this year, Bell, Koren, and Volinsky had the best performing ensemble model of the competition, and so published a report detailing the algorithms used to achieve that benchmark. Their model was a combination of over one hundred individual recommendation algorithms, each contributing to the final prediction. While not all of these algorithms make use of temporal data, the most significant improvement in performance resulted from the incorporation of temporal dynamics into the matrix factorization framework. There are a variety of temporal models discussed in [4], and we discuss one example formulation here.

The goal of a temporal matrix factorization algorithm is to define a framework for learning parameters as a function of time, resulting in a model form of:

$$\hat{r}_{ui} = \mu + b_u(t) + b_i(t) + q_i^T p_u(t) \tag{3.3}$$

In particular, user biases, item biases, and user factors are learned to be temporally-dependent. These have the following intuitive interpretations. Temporal user biases account for users changing their baseline biases over time, for example users becoming more generous in their rating behavior. A time-dependent movie bias accounts for the changing popularity of movies over time. For example, a movie may be very highly regarded soon after it is released but decreases in the ratings it receives over time. Finally, time-dependent user factors account for changing user preferences, such as a user gradually developing a taste for action movies and

losing interest in horror films.

The following temporal parameters are proposed for the above time-dependent parameters. For item biases, the dataset is split into 30 bins, and an individual item bias parameter is learned for each bin. This is feasible given the large size of the Netflix data, and the repetition observed in the majority of item ratings.

User biases are parameterized using two approaches. The first introduces very few additional parameters but is also less flexible, scaling the parameter value by the distance from the mean time of all user interactions. Given the mean rating date t_u, the temporal bias term becomes:

$$b_u^{(1)}(t) = b_u + \alpha_u \cdot sign(t - t_u) \cdot |t - t_u|^{0.4} \tag{3.4}$$

An alternative approach is to bin the training set in a similar manner to the item biases, learning an independent bias term for every user and using a bin size of one day, giving:

$$b_u^{(2)}(t) = b_{u,t} \tag{3.5}$$

In the overall model, these two parameterizations are combined together, giving:

$$b_u^{(3)}(t) = b_u^{(1)}(t) + b_u^{(2)}(t) \tag{3.6}$$

An equivalent parameterization is applied to each component of the user factors, giving a parameterized form for index k of user vector p_u as:

$$p_{uk}^{(3)}(t) = p_{uk} + \alpha_{uk} \cdot sign(t - t_u) \cdot |t - t_u|^{0.4} + p_{uk,t} \tag{3.7}$$

These temporal parameters can be substituted into the matrix factorization prediction equation, but in [4] are substituted into the combined matrix factorization-neighborhood model SVD++. The above temporal formulations introduce a significant number of additional parameters to be trained from the data. However, given the large size of the Netflix prize dataset, the temporal model was able to be successfully trained using stochastic gradient descent and achieved significantly improved RMSE performance than comparable models that did not take temporal information into account.

3.4 Neural Networks

Given the rapid rise in popularity of neural networks and deep learning algorithms in recent years, many algorithms have been proposed which apply neural network techniques to the recommendation task. Well-designed neural network models excel at learning to encode relevant features that assist in improving prediction performance. They also encode a level of nonlinearity typically not present in the standard matrix factorization model. As a result, this facilitates more complex models but also increases the risk of overfitting due to the high sparsity in most recommender system datasets. We will cover two recommender system models that incorporate neural networks to improve their prediction performance.

First, in [21], He et al. propose a model called Neural Collaborative Filtering (NCF), which extends the matrix factorization model by replacing the inner product term with an arbitrary feedforward neural network, also known as a multilayer perceptron (MLP). This adds nonlinearities to the matrix factorization framework and leads to significant performance improvements on a variety of datasets.

The model starts by learning a latent factor matrix P for users and Q for items, as is the

case in the standard matrix factorization model. However, rather than simply taking the inner product of these two latent vectors, in the NCF models they are used as inputs to an MLP. Each layer in the MLP is able to learn a higher level feature representation building upon the weights in lower layers, with the most complex features used to calculate the output $\hat{y}_{ui}$. To train the NCF model, [21] employs the Adam [25] optimization algorithm to optimize the binary cross-entropy loss.

NCF provides significant improvements in recommendation performance over matrix factorization models without a neural network component. Interestingly, adding additional hidden layers to the MLP leads to increased recommendation performance, and the authors leave additional room for investigation on deeper or more sophisticated neural network components. Another advantage of the model is that it takes as input arbitrary feature vectors for users and items, facilitating the easy inclusion of content or related features.

We next discuss [22], which proposes a sequential recommendation algorithm based on recurrent neural networks. Similar to Markov chains as recommendation algorithms, recurrent neural networks (RNNs) are the sequential equivalent of neural networks. However, unlike Markov chains, RNNs can encode much longer-term dependencies via internal states.

Rather than a standard RNN, a Gated Recurrent Unit (GRU)-based network is used as a more elaborate model that attempts to mitigate the vanishing gradient phenomenon commonly observed with vanilla RNNs [13]. A GRU consists of a linear update equation for the hidden state of each recurrent unit, which depends on the values of an "update" and "reset" gate computed as functions of the input features and hidden state at the previous time step.

Within the context of recommendation, the GRU RNN model is applied to session-based recommendations, where a sequence of closely-occurring recommendations for each user is

grouped together to form a session. Intuitively, a session typically corresponds to a single browsing or shopping session on an e-commerce website, and the interactions within a single session are typically assumed to be closely related. In particular, the input feature for each item in the session is passed through an embedding layer to a sequence of GRU layers, which update their hidden states and alter the output values for the layer based on previously observed items in the session. Finally, the output of the final GRU layer is passed through standard feedforward layers to compute output probabilities over all items in the corpus.

The authors of [22] apply ranking-based loss functions to train the model, including the standard BPR loss and a new proposed loss function TOP1, which directly optimizes the rank of relevant items. A variety of experiments are performed, testing a variety of architectures, loss functions, recurrent units, hidden layer sizes, etc. The best performing models are able to achieve accuracy gains over the best performing baseline models of around 20-30%. This result was further improved upon by [40], which added a second recurrent network to handle inter-session dynamics. The significant performance gains by these neural network models, along with a variety of additional recently proposed models that incorporate neural networks with recommendation algorithms, are promising for the development of future algorithms that combine state-of-the-art approaches in the fields of both recommender systems and deep learning.

3.5 Factorization Machines

Given their simplicity and applicability to a variety of machine learning tasks, FMs have been extended in a variety of ways since their introduction. While traditionally applied in the recommender systems domain, FMs are an arbitrary predictive model that operate on arbitrary feature vectors. This facilitates the easy incorporation of content features, without requiring

significant changes to the form of the model itself. [54] incorporates features of skip-gram and other text mining algorithms to apply FMs to sentiment classification, and [46] includes FMs in a multi-stage predictive model to extract relevant reviews for recommendation. Finally, [23, 24] propose domain-specific models applying FMs to content modeling on Twitter and CTR prediction in advertising.

FMs and related hybrid algorithms aim to improve performance and make useful recommendations to "cold-start" users and items (with few observed interactions) by incorporating additional sources of information. These include temporal [28, 16], social [10, 17], and geographical [37, 52] features. Recent hybrid approaches have incorporated image features to improve content-based or next POI recommendation [55], and applied deep learning techniques to automatically generate useful content features [49] or introduce additional modeling flexibility [21]. However, while these hybrid models achieve state-of-the-art performance compared to baseline approaches, they all rely on specialized models and techniques to incorporate additional features.

Various chapters in this thesis include material that has been submitted for publication as it may appear in RecSys 2018, Pasricha, Rajiv; McAuley, Julian, ACM, 2018. The thesis author was the primary investigator and author of this paper.

Chapter 4

Translation-based Recommendation

In this chapter, we will discuss TransRec, proposed in 2017 by He et al., and extended in this thesis [19]. Compared to the various approaches to recommender system algorithms previously discussed, TransRec is a sequential recommendation algorithm that models interactions between a user and a sequence of interactions, e.g. purchases. Given a user and their most recently purchased item, the model predicts which item is most likely to be purchased next.

This contrasts with many commonly used algorithms, such as matrix factorization, that do not take sequential data into account. These algorithms learn separate latent user and item factors to develop a "profile" about each entity in the dataset. However, these models only take short term information into account, specifically the current user and item, building a global profile that does not account for the user's recent behavior. On the other hand, sequential recommender algorithms model third-order interactions, taking into account the user, previous item, and next item when making recommendations.

The TransRec algorithm models these third-order interactions through a translation approach. As with latent factor models, in the TransRec setting, users and items are embedded into a common "translation space," a unique interpretation of the recommendation task. Rather than simply measuring interactions between individual users and items, the translation space

facilitates modeling the entire sequence of items consumed by each user, interpreting user factors as translation vectors moving through this shared space. By combining the sequential nature of Markov chain algorithms with the latent factors common in matrix factorization approaches, the proposed algorithm is able to achieve improved performance,

Translation-based recommendation is a novel framework that addresses the sequential recommendation task by interpreting recommendations as "translations" through a latent space. Intuitively, we have $\vec{\gamma}_i + \vec{t}_u \approx \vec{\gamma}_j$, where $\vec{\gamma}_i$ and $\vec{\gamma}_j$ are item embeddings for the previous and next item respectively, and $\vec{t}_u$ is user u's personalized translation vector. This enables the authors to create a model that successfully learns third-order interactions while remaining easily interpretable. A distance function, such as Euclidean distance, is employed to determine compatibility between the translation destination and the next item, i.e. $d(\vec{\gamma}_i + \vec{t}_u,\ \vec{\gamma}_j)$, maintaining the advantages provided by recommender system algorithms that employ metric distance functions, specifically improved generalization characteristics provided by the triangle inequality. Finally, in order to make predictions at testing time, e.g. for a deployed model, a nearest neighbor search is employed, centered at the translation destination $\vec{\gamma}_i + \vec{t}_u$.

One significant strength of this model is that it is easy to visualize a sequence of items embedded in the latent space, with one user represented as a vector traveling along his or her own personalized sequence through this space. Although this does not seem to be the case when analyzing the learned embeddings, as will be discussed in future chapters, this is a useful interpretation and devising a model along these lines leads to successful results.

Adopting a translation approach for sequential recommendation provides the following advantages: TransRec can model complex third-order interactions with fewer parameters compared to other techniques, and it employs a metric distance function to achieve improved

Table 4.1. Notation for TransRec and proposed extensions

Notation	Explanation
$\mathcal{U}, \mathcal{I}$	user set, item set
u, i, j	user $u \in \mathcal{U}$, items $i, j \in \mathcal{I}$
$\mathcal{S}^u$	historical sequence of user u: $(\mathcal{S}^u_1, \mathcal{S}^u_2, \ldots, \mathcal{S}^u_{\lvert\mathcal{S}^u\rvert})$
Φ	transition space; $\Phi = \mathbb{R}^k$
Ψ	a subspace in Φ; $\Psi \subseteq \Phi$
$\vec{\gamma}_i$	embedding vector associated with item i; $\vec{\gamma}_i \in \Psi$
$\vec{t}$	(global) translation vector $\vec{t} \in \Phi$
$\vec{t}_u$	translation vector associated with user u; $\vec{t}_u \in \Phi$
$\vec{t}_i$	translation vector associated with item i; $\vec{t}_i \in \Phi$
$\vec{T}_u$	$\vec{T}_u = \vec{t} + \vec{t}_u$; $\vec{T}_u \in \Phi$
$\vec{T}_{ui}$	$\vec{T}_{ui} = \vec{t} + \vec{t}_u + \vec{t}_i$; $\vec{T}_{ui} \in \Phi$
β_i	bias term associated with item i; $\beta_i \in \mathbb{R}$
$d(x, y)$	distance between x and y
δ_{uij}	processed time delta between user u's purchase of item i and item j
τ_{ui}	time associated with user u's purchase of item i
θ_u	translation vector scaling factor for user u
f, g	arbitrary multilayer perceptrons (MLPs) used in model predictions

generalization. Finally, its simple form enables it to be quite scalable, and it is easily able to handle large datasets with millions of instances.

4.1 The Formal TransRec Model

The TransRec model introduced in [19] learns embeddings for each user and item in the training dataset. Each user is associated with a personalized translation vector $\vec{T}_u$, and each item with latent factors $\vec{\gamma}_i$. The intuitive equation is $\vec{\gamma}_i + \vec{T}_u \approx \vec{\gamma}_j$, encoding the desired personalized translation relationship in the embedding space.

Table 4.1 provides the mathematical notation used in the original paper, as well as in our extensions described in future chapters. In the original model, each user $u \in \mathcal{U}$ has a sequence of items $\mathcal{S} = \left(\mathcal{S}^u_1,\ \mathcal{S}^u_2, \ldots, \mathcal{S}^u_{|\mathcal{S}_u|}\right)$ that the user has interacted with (e.g. rated or purchased). Given this sequence of items, the algorithm attempts to predict the item most likely to be consumed

next by the user.

Formally, given the latent space $\Phi = \mathbb{R}^K$, TransRec learns a representation for each item i, specifically $\vec{\gamma}_i \in \Phi$. The user offsets $\vec{t}_u$ are also learned. Due to the sparsity inherent in the majority of recommender systems datasets, a global translation vector $\vec{t}$ is learned along with user-specific offsets $\vec{t}_u$. This allows $\vec{t}$ to capture global transition dynamics across users, allowing "cold-start" users without many ratings to be regularized towards this average translation behavior. On the other hand, $\vec{t}_u$ represents specific offsets from this average behavior for each user in the dataset.

The probability that a user u transitions from previous item i to next item j is given by the following expression:

$$\begin{gathered} P(j|u,i) \propto \beta_j - d\left(\vec{\gamma}_i + \vec{T}_u, \vec{\gamma}_j\right) \\ \text{subject to } \vec{\gamma} \in \Psi \subseteq \Phi, \text{ for } i \in \mathcal{I} \end{gathered} \tag{4.1}$$

The subspace $\Psi \in \Phi$ is also a form of regularization for the model. By restricting the $\vec{\gamma}_i$ terms to this subspace, this ensures that the item representations stay within a maximum level of complexity, which is especially useful for items with less data present.

The loss function for the model is given by Sequential Bayesian Personalized Ranking [43]. This means that the model is optimized according to the pairwise ranking with regards to a positive item j and a negative item j'. As previously discussed, the goal of S-BPR is to learn a model that ranks all examples with positive feedback over all examples with negative or missing feedback. As the model relies on implicit feedback (purchases) rather than explicit feedback (e.g. ratings out of five stars), the negative items j' are items that the user has not interacted with in the dataset.

The S-BPR loss function is as follows. Given a user and previous item i, we would like the positive item that follows i in the training set to be ranked ahead of all other items. $\Omega(\Theta)$ is a general $\mathcal{L}_2$ regularization term for the model's parameters. In addition, $\hat{p}_{u,i,j}$ is a shorthand for $P(j|u,i)$ described above.

$$\hat{\Theta} = \text{argmax}_{\Theta} \ln \prod_{u \in \mathcal{U}} \prod_{j \in \mathcal{S}^u} \prod_{j' \notin \mathcal{S}^u} P\left(j >_{u,i} j' \middle| \Theta\right) P(\Theta) \tag{4.2}$$

$$= \text{argmax}_{\Theta} \sum_{u \in \mathcal{U}} \sum_{j \in \mathcal{S}^u} \sum_{j' \notin \mathcal{S}^u} \ln \sigma\left(\hat{p}_{u,i,j} - \hat{p}_{u,i,j'}\right) - \Omega(\Theta) \tag{4.3}$$

In order to learn the parameters of this model, the authors use stochastic gradient descent with random sampling. First, a user $u \in \mathcal{U}$ is randomly sampled. Next, a positive item j and negative item $j\prime$ are sampled from $\mathcal{S}^u \backslash \mathcal{S}^u_1$ and $\mathcal{I} \backslash \mathcal{S}^u$ respectively. Let i be the item that comes immediately before j in user u's sequence. Given u, i, j, and $j\prime$, the parameters of the model are updated according to stochastic gradient descent.

Chapter 5

Proposed Models and Extensions

In this chapter, we present a variety of extensions to the original TransRec model. The overall idea behind these extensions is to introduce additional sources of information into the model, in order to evaluate whether the model is able to incorporate this additional information to improve prediction quality. We also present a variety of models that add increased complexity and nonlinear dynamics, in order to facilitate the learning of more complex sequential relationships. Finally, we present a joint model that combines aspects of both TransRec and Factorization Machines. This model combines the advantages of both prediction frameworks, specifically the generality and extensibility of FMs along with the translation intuition and metricity assumption of TransRec. Notation for our extensions is presented in Table 4.1, with *TransFM*-specific notation in Table 5.1.

5.1 Item Offset Model

Our first proposed model incorporates item offsets into the learned translation vectors of TransRec. Currently, TransRec learns a global translation term $\vec{t}$ and a user-specific translation offset $\vec{T}_u = \vec{t} + \vec{t}_u$. Intuitively, this means that each user is assigned a characteristic "translation" in the derived item space, and that user will always move in that single direction through their

sequence of interactions. This model works well when users tend to have similar translations between adjacent items in their consumption sequences, regardless of the identities of those items.

However, this user-specific translation model does not encapsulate the intuition of item-to-item translation items. For example, if a given user watches the first item in a television series, then they are move likely to watch the next item in the series as their next interaction. This effect is independent of the user and is instead a characteristic of the item itself. Certain items will have certain "next items" which tend to follow them in sequence regardless of the user performing the interactions, and the TransRec model does not capture that notion.

To incorporate these item-specific effects, we define a translation term for each item, $\vec{t}_i$. We then update the translation vector to include the item-specific term, along with the global and user translation terms: $\vec{T}_{ui} = \vec{t} + \vec{t}_u + \vec{t}_i$. This leads to the following model formulation:

$$P(j|u,i) \propto \beta_j - d\left(\vec{\gamma}_i + \vec{T}_{ui},\ \vec{\gamma}_j\right) \tag{5.1}$$

5.2 Temporal Models

The next set of proposed translation models incorporate temporal dynamics into TransRec. As previously discussed, recommender systems that incorporate temporal dynamics often exhibit significantly improved performance over their simpler counterparts, and temporal information is included in the majority of all recommendation datasets. This was the case even in the Netflix Prize competition, which spurred significant interest in the field. Although latent factor models performed well on the Netflix dataset, the most significant improvements to the models'

performance came from incorporating temporal dynamics and implicit feedback into the matrix factorization framework. We incorporated temporal dynamics into TransRec in a variety of ways, some of which will be discussed later along with more complex model formulations.

5.2.1 Time Delta Model

The first proposed temporal model adds a scaling factor to the user-specific translation vector. Intuitively, this corresponds to scaling the translation vector by the time delay between the previous and next item in a user's consumption sequence. If we imagine a user as moving through the latent item space through a sequence of translations, then having a larger time delay between two items in the sequence gives them additional time to perform a larger translation, and so their translation vector can be larger. On the other hand, if the next interaction happens immediately following the previous one, then their position in the space should not change very much from the previous to the next item.

This also allows the model to learn the concept of "user sessions," a sequence of user interactions that occur in short succession, such as purchasing the items in a single shopping cart or listening to a playlist of songs in one sitting. Items that tend to occur within single sessions will have shorter time deltas between them, and thus should be located closer together in the item space. Similarly, items that do not frequently occur simultaneously in observed sessions will have a larger time delta, and thus a larger translation vector separating them, allowing them to be located further apart.

Formally, we add a time delay term δ_{uij} to the translation model, representing the difference between the observed timestamps for user u's purchases of item i and item j. This means that we now represent the probability of a user u interacting with item j after item i as

follows:

$$P(j|u,i) \propto \beta_j - d\left(\vec{\gamma}_i + \delta_{uij}\vec{T}_u, \vec{\gamma}_j\right) \tag{5.2}$$

The δ_{uij} terms are not learned from the model but are instead computed as follows. Let τ_{ui} and τ_{uj} be the timestamps associated with the previous and next item interactions, respectively. Based on the observed timestamps in our datasets, many pairs of items have very large time deltas between them, on the order of years. So, we compute the difference between the two times, after normalizing all timestamps in the dataset to have zero mean and unit standard deviation. Next, we add a constant one-day offset so the translation vectors will never vanish. This gives:

$$\delta_{uij} = \tau_{uj} - \tau_{ui} + 1 \text{ day} \tag{5.3}$$

5.2.2 Personalized Time Delta Model

The above time delta model assumes that all user translation vectors are affected equally by the observed time deltas in their corresponding consumption sequences. However, it may be the case that some users are affected differently by these temporal effects than others. For example, one user's translation vector may show significant scaling properties as they traverse their sequence of items, while another may follow a sequence relatively unaffected by temporal considerations.

In order to handle these potential differences, we introduce a time delta scaling factor θ_u for each user u. This parameter is learned from the data during the model training process and determines the effect that the time delta factor has on scaling the translation vector. Given this

additional term, we update the probability of user u interacting with item j after item i as:

$$P(j|u,i) \propto \beta_j - d\left(\vec{\gamma}_i + \theta_u \delta_{uij} \vec{T}_u, \vec{\gamma}_j\right) \tag{5.4}$$

5.3 Neural Network Models

The next set of proposed models make use of neural networks to add nonlinear components to the TransRec framework. Specifically, we use multilayer perceptrons (MLPs) to learn arbitrary relationships between users and consecutive items in their consumption sequences. One of the most significant limitations of TransRec is that each user is associated with a linear translation vector which remains the same regardless of the user's location in space or where they are in their consumption sequence. This is likely an oversimplification of the true relationships between the items in the dataset, especially when the positions of items in the latent space are determined by all users that interact with them. The linear translation approach leads to enforcing a rigid structure in the resulting item embeddings, encouraging item embeddings for items that occur in a consumption sequence to be an equal distance away, even though it might be more reasonable for certain related items to be located close together.

Another limitation of the linear translation structure is that it does not support repeated transactions. For example, say a user u purchases item i, followed by item j, and then purchases item i again. This is a common occurrence in many practical settings, and many datasets consist of items that are purchased many times by a single user. However, in the TransRec framework, once a user has purchased an item and followed his translation vector to the next item in the sequence, it becomes impossible for his location in the embedding space to return back to the

previous item. Adding nonlinear components facilitates more complex translation dynamics, such as repeated transactions, translations of different magnitudes, or translations traveling in different directions through the embedding space.

While it is possible to introduce nonlinearities through custom nonlinear model formulations, this often involves a significant amount of trial and error to derive the model that is best able to account for the nonlinearities present in the data. An alternative approach is to introduce nonlinear dynamics via a more general model that is able to learn the relationships between the input parameters from the data itself, without having to rely on a predetermined formulation. Neural networks are ideal for this task, and there is currently significant activity related to adding neural network and deep learning dynamics to recommender systems models.

5.3.1 Neural Network Translation Model

The first nonlinear model replaces the translation calculation with the output of a multilayer perceptron (MLP). In the original TransRec model, translations through the item space are calculated according to $\vec{\gamma}_i + \vec{T}_u$, and the model attempts to bring this value as close as possible to the latent representation of the next item $\vec{\gamma}_j$.

Let the function $f(\cdot)$ be a shorthand for the MLP added to our model. The input parameters to f are the input features provided to the MLP, and the output value is the output of the network's final layer. The Neural Network Translation model calculates the output position in the latent space as $f\left(\vec{\gamma}_i, \vec{T}_u\right)$. This means that the probability that user u transitions from item i to item j is given by:

$$P(j|u,i) \propto \beta_j - d\left(f\left(\vec{\gamma}_i, \vec{T}_u\right),\ \vec{\gamma}_j\right) \tag{5.5}$$

Even though we are adding an MLP to TransRec to represent complex nonlinear calcu-

lations, the total number of parameters added to the model does not increase by a significant amount. This is because the MLP is a global component of the model, and the same parameters are used to compute the translations for all users and items in the dataset. Because of the global nature of this component, we do not regularize the MLP parameters as the amount of data used to learn the weights in the network vastly exceeds the number of weights in the network itself.

5.3.2 Neural Network Distance Model

An additional method of incorporating nonlinear characteristics is to directly estimate the probability of transitioning from one item to the next in a user's consumption sequence. Rather than returning a point in the latent space which is then compared to the next item in the sequence via Euclidean distance, as is done in the Neural Network Translation model, we can instead estimate the probability of transitioning to the next item in the user's sequence directly using the MLP.

By replacing the Euclidean distance calculation with the output of the MLP, we are essentially learning a custom distance function on the latent item space which is optimized for the recommendation task. One downside of the model is that the output of the MLP does not necessarily obey the triangle inequality, so it is not a true distance metric. While obeying the triangle inequality allows information to propagate effectively to unseen user-item pairs, the additional complexity of the MLP model might offset the performance penalty from not obeying it.

Under the Neural Network Distance model, the probability of user u transitioning from item i to item j is given by:

$$P(j|u,i) \propto \beta_j - f\left(\vec{\gamma}_i, \vec{T}_u, \vec{\gamma}_j\right) \tag{5.6}$$

5.3.3 Neural Network Time Delta Models

The above two models train a neural network to either predict translations in the item space, or directly predict the transition probabilities. We next experiment with adding temporal information to the neural network models. As opposed to the explicit temporal models above, the presence of an MLP will allow the model to learn how to most effectively utilize the provided temporal information. For example, depending on the weights of the MLP, the model can learn a positive, negative, or thresholding relationship from the interaction times, or it can learn a relationship that is not immediately interpretable.

First, we propose adding the time delta from previous to next item to the MLP models. These time deltas are timed using the same formulation as above, specifically $\delta_{uij} = \tau_{uj} - \tau_{ui} +$ 1 day. Adding these time deltas, the Neural Network Translation model becomes:

$$P(j|u,i) \propto \beta_j - d\left(f\left(\vec{\gamma}_i, \vec{T}_u, \delta_{uij}\right),\ \vec{\gamma}_j\right) \tag{5.7}$$

Similarly, the Neural Network Distance model becomes:

$$P(j|u,i) \propto \beta_j - f\left(\vec{\gamma}_i, \vec{T}_u, \vec{\gamma}_j \delta_{uij}\right) \tag{5.8}$$

5.3.4 Neural Network Timestamp Models

In the models above, we added temporal information to the neural network models in the form of deltas between the previous and next items in a user's sequence. However, given an MLP, we can instead add the raw timestamps and allow the network itself to learn the most useful relationships between the temporal data and provided consumption sequences.

Given that tanh is used as the MLP activation function for each layer, we see that adding the raw timestamps is a strict generalization of the above models, which used the time deltas between the previous and next item. In particular, setting the network's weights and biases appropriately can recover the exact time delta given input timestamps. This means that if the time deltas are the more useful feature for making recommendations, this feature should be derived from the raw input timestamps.

With raw timestamps, the Neural Network Translation model becomes:

$$P(j|u,i) \propto \beta_j - d\left(f\left(\vec{\gamma}_i,\ \vec{T}_u, \tau_{ui}, \tau_{uj}\right),\ \vec{\gamma}_j\right) \tag{5.9}$$

Similarly, the Neural Network Distance model becomes:

$$P(j|u,i) \propto \beta_j - f\left(\vec{\gamma}_i, \vec{T}_u, \vec{\gamma}_j, \tau_{ui}, \tau_{uj}\right) \tag{5.10}$$

5.4 Session-based Models

The neural network models discussed above are effective at deriving the appropriate translation dynamics by optimizing weights of the network from the data itself. This contrasts with more explicit model formulations, which take additional sources of data or more complex dynamics into account by adding them directly into the prediction equation. While not as flexible as general models that employ MLPs to model arbitrary translations, these explicit models can be effective in scenarios where the model's assumptions are present in the observed datasets. They also tend to perform well with sparse datasets, where there might not be enough data to learn the complex dynamics afforded by a neural network model, hindering generalizability and

performance.

In a previous section, we proposed extensions to the TransRec model that explicitly modeled temporal dynamics. In this section, we extend TransRec to more explicitly model session dynamics, while still maintaining some aspects of the flexibility afforded by the complex neural network models.

5.4.1 Default-Origin Session Model

Our approach to modeling session dynamics in the TransRec framework is similar to the MLP models discussed above, in that we would like to provide a high level of flexibility to the model, allowing it to learn the most appropriate dynamics from the data itself. The "session" dynamic enforced by this model is that each user has a customized trajectory that starts from the origin, and travels along their linear trajectories for the duration of the session. At the end of the session, the user's location in the embedding space jumps back to the origin, allowing the user to begin their translation process again, perhaps through a consumption sequence of items related to those consumed in the first session.

Contrary to traditional session-based models, this model does not rely on explicit sessions computed by thresholding the elapsed time between successive item transitions. It instead learns the concept of sessions from the data itself, essentially splitting the observed consumption sequences into a series of reduced sequences based on the user and identities of the items themselves. We use a neural network g to predict the probability of continuing along the current session, with $1-g$ being the probability of transitioning back to the origin to begin a new session. Contrary to the neural networks proposed in the previous models, we only add one output unit to this network, which uses the Sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$ to enforce that the network output is a probability between 0 and 1. The probability of user u transitioning from

item i to item j is thus as follows:

$$P(j|u,i) \propto \beta_j - d\left(g\left(\vec{\gamma}_i,\ \vec{T}_u\right)\left(\vec{\gamma}_i + \vec{T}_u\right),\ \vec{\gamma}_j\right) \tag{5.11}$$

The probability of transitioning back to the origin is associated with a translation location of $\vec{0}$, so it is not included in the model expression above. We see that this session-based model is equivalent to simply scaling the translation vector by the output of the neural network g, which takes the previous item embedding and user translation vector as input parameters.

5.4.2 Personalized-Origin Session Model

The next iteration of the explicit session based TransRec model updates the location to where the user will transition upon the commencement of a new session. In the previous session model, the user would transition back to the origin if the output of the neural network was small and continue along their trajectory if the neural network output was large. The actual translation destination was computed as a weighted average of these two locations.

However, simply transitioning back to the origin of the embedding space to begin a new session may not be the most appropriate action for each user in the dataset. On the contrary, each user may have a separate location where they transition to in order to commence their future sessions. These personalized "home" locations for each user could be located near items that the user tends to purchase frequently or aligned with frequently observed trajectories through the space. Simply having all users transition back to the origin when starting a new session does not take these additional dynamics into account, instead assuming that all users behave identically to one another when they start a new session.

In the personalized-origin session model, we define a "home" location $\vec{h}_u$ for each user

u. This is the translation destination for user u when the neural network output determines that the user should begin a new session. As before, the probability of beginning a new session or continuing along the current session is determined by an MLP g, with a single output unit that employs a sigmoid activation function. We also take the weighted average of both possible translation destinations using the network's output as the weight, in order to maintain the smoothness and differentiability of our model. Formally, this gives:

$$P(j|u,i) \propto \beta_j - d\left(g\left(\vec{\gamma}_i,\ \vec{T}_u\right)\left(\vec{\gamma}_i + \vec{T}_u\right) + \left(1 - g\left(\vec{\gamma}_i, \vec{T}_u\right)\right)\vec{h}_u,\ \vec{\gamma}_j\right) \tag{5.12}$$

5.4.3 Nonlinear Personalized-Origin Session Model

Our final explicit session-based model is a combination of the intuition behind our two previous models. As discussed in the previous section, we use a neural network to determine whether the user should begin a new session or continue along their current trajectory. We also define explicit home locations for each user to account for differences in sequential characteristics, leading to different starting locations for user sessions.

In this nonlinear model, we also add the nonlinear component proposed in our neural network translation model, discussed above. The neural network takes as input the user's vector $\vec{T}_u$ and the embedding of the previous item in the sequence $\vec{\gamma}_i$, and outputs the translation vector to be followed by the user to their next item. This is the most flexible session-based model, introducing the highest level of nonlinearities into the original TransRec framework. In particular, the translation vectors for each user are a nonlinear function of the user's identity and previous item, as is the probability of transitioning back to a personalized home location.

This means that for the nonlinear personalized-origin model, there are two neural net-

works determining different aspects of the translation dynamics for each user. In order to prevent overfitting, we share the input and hidden layers of the two networks. This means that we define a single network that takes as input the user vector $\vec{T}_u$ and the previous item embedding $\vec{\gamma}_i$, and has a single hidden layer. This hidden layer is connected to two output layers, one computing a vector in $\mathbb{R}^k$ to be the translation vector to the next item j, and the other computing the probability of continuing along the current trajectory or transitioning back to the user's home location. In order to differentiate between these two output layers, we refer to the translation and session networks as $g^{(1)}$ and $g^{(2)}$, respectively.

So, we can define the probability of user u transitioning from item i to item j as follows:

$$P(j|u,i) \propto \beta_j - d\left(g^{(2)}\left(\vec{\gamma}_i, \vec{T}_u\right) g^{(1)}\left(\vec{\gamma}_i,\ \vec{T}_u\right) + \left(1 - g^{(2)}\left(\vec{\gamma}_i, \vec{T}_u\right)\right)\vec{h}_u,\ \vec{\gamma}_j\right) \tag{5.13}$$

5.5 Category Cosine

Our next model follows the theme discussed at the beginning of this chapter, specifically making use of additional sources of data to improve recommendation performance. This model was derived specifically for the Amazon datasets but is easily extended to other datasets which contain similar additional sources of information.

In addition to the item purchase history of individual users, the Amazon datasets also contain additional metadata associated with each item. This rich metadata includes the description, price history, sales ranking, brand information, and co-purchase links for each item in the entire datasets. It also includes the list of categories assigned to each item on its Amazon webpage. This model makes use of this category information as a form of regularization, to improve the

embeddings learned by TransRec and thus lead to improved performance.

The categories included in the Amazon datasets form a hierarchical list, from the highest-level representing the overarching genre the item belongs to, down to the most specific describing its particular use case. As an example, a window motor in the Amazon Automotive dataset has the following list of categories: "Automotive," "Replacement Parts," "Window Regulators & Motors," "Power Window Motors."

An additional characteristic of the Amazon datasets is that the number of possible items to select is quite large, but most items have relatively few purchases. So, this means that the item categories become a potentially useful source of data to improve recommendation performance. In particular, given the granularity observed in the recorded item categories, we would expect that items with similar category vectors should have similar observed purchase sequences in the dataset, and should have similar embeddings in the learned embedding space. However, this may not actually be observed in the TransRec model due to a lack of data associated with each particular item.

We define a model that takes into account this intuition, that items with similar categories should have similar locations in the resulting embedding space. In order to enforce this, we define feature vectors for each item $\vec{c}_i$, where for feature k, $\vec{c}_{ik} = 1$ if item i has category k, and $\vec{c}_{ik} = 0$ otherwise. In order to compare category feature vectors of different items, we use the cosine similarity metric. This metric has been commonly used as a similarity measure in neighborhood-based recommender system models and has the convenient property of being 1 when two items have all categories in common, and 0 when two items have no categories in

common. Formally, the cosine similarity between vectors $\vec{v}$ and $\vec{w}$ is:

$$s(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{||\vec{v}|| \times ||\vec{w}||} \tag{5.14}$$

This allows us to define a regularization term for the TransRec model to enforce the aforementioned intuition. In order to enforce that similar items should have similar embeddings, we add a weighted sum of the magnitude of consecutive pairs of item embeddings, weighted by the cosine similarity of their category vectors. This means that if two items do not share any categories in common, the difference in their item embeddings will not be penalized, and the model will not encourage their embeddings to lie close together in the embedding space. However, if two items have identical categories, then there will be a high weighting term added to their embedding difference, resulting in the model encouraging similar embeddings for these items.

Formally, we define the additional regularization term as follows:

$$L(\Theta) = \mathcal{R} = \sum_{u\in\mathcal{U}} \sum_{j\in\mathcal{S}^u} s(\vec{c}_{u,i},\vec{c}_{u,j})(\vec{\gamma}_i - \vec{\gamma}_j)^2 \tag{5.15}$$

This term is added to the TransRec loss function, along with the standard $\mathcal{L}_2$ regularization term. The prediction function remains the same as the TransRec model.

5.6 Translation-based Factorization Machines

We propose a joint model, *TransFM*, which combines aspects of both TransRec and Factorization Machines. As previously discussed, FMs are a general-purpose prediction model that take as input arbitrary feature vectors, and model the output variable via linear and second-

Table 5.1. Notation for *TransFM*

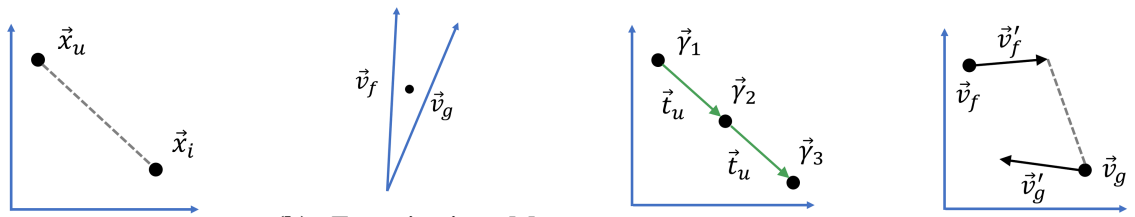
Notation	Explanation
$\mathcal{U}, \mathcal{I}$	user set, item set
$\mathcal{S}^u$	historical interaction sequence for user u
$\vec{x}_{u,i,j}$	feature vector for user u, previous item i, and next item j
k	dimensionality of embedding and translation spaces
n	dimensionality of $\vec{x}_{u,i,j}$
w_0	global bias term
$\vec{w}$	linear terms; $\vec{w} \in \mathbb{R}^n$
$\mathbf{V}$	feature embedding space; $\mathbf{V} \in \mathbb{R}^{n \times k}$
$\mathbf{V}'$	feature translation space; $\mathbf{V}' \in \mathbb{R}^{n \times k}$
$d^2(\vec{a}, \vec{b})$	squared Euclidean distance between $\vec{a}$ and $\vec{b}$

order feature interactions using factorized parameters. TransRec also excels at making high-quality recommendations from sparse data, employing a metricity assumption to improve its generalization performance. However, one drawback of TransRec is that it does not accept arbitrary feature vectors as inputs, instead only accepting user and item indices in the form of user consumption sequences.

5.6.1 The TransFM Model

Table 5.1 presents relevant notation for *TransFM*. As with Factorization Machines, *TransFM* operates on real-valued feature vectors $\vec{x}$. In the sequential recommendation setting, $\vec{x}$ includes feature representations for the user u, the previous item i, and next item j, along with any additional content features.

Each dimension in $\vec{x}$ is associated with both an embedding and a translation vector. Formally, for feature x_i, we learn two vectors: an embedding vector $\vec{v}_i \in \mathbb{R}^k$ and a translation vector $\vec{v}'_i \in \mathbb{R}^k$. We apply the translation operation to the previous item embedding and measure the distance to the next item embedding $\vec{v}_j$ by the squared Euclidean distance. The resulting distance gives the weight assigned to the corresponding feature interaction.



(a) PRME **(b)** Factorization Machines **(c)** TransRec **(d)** *TransFM*

Figure 5.1. A visual comparison of translation, metric, and factorization-based recommender system models.

The model equation of *TransFM* is given by:

$$\hat{y}(\vec{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} d^2(\vec{v}_i + \vec{v}'_i, \vec{v}_j) x_i x_j \tag{5.16}$$

where w_0 is a global bias term and w_i is the linear term for feature x_i. $\vec{v}_i$ and $\vec{v}'_i$ are the embedding and translation vectors (resp.) for feature x_i, and $d^2(\vec{a}, \vec{b})$ represents the squared Euclidean distance between the vectors $\vec{a}$ and $\vec{b}$:

$$d^2(\vec{a}, \vec{b}) = (\vec{a} - \vec{b}) \cdot (\vec{a} - \vec{b}) = \sum_{f=1}^{k} (a_f - b_f)^2 \tag{5.17}$$

Like other metric-based models, *TransFM* replaces the inner product term with the (squared) Euclidean distance. This leads to improved generalization performance, and more effectively captures the transitive property between feature embeddings.

Figure 5.1 provides a comparison of the prediction methods used by *TransFM* and various baseline models. PRME (5.1a) learns a personalized metric space in which the distance between embeddings measures user-item compatibility (the corresponding item-item sequential space is not shown); Factorization Machines (5.1b) measure interactions between arbitrary features by computing the inner product of their corresponding factorized parameters; TransRec (5.1c)

learns embeddings $\vec{\gamma}_i$ for each item, and a translation vector $\vec{t}_u$ for each user that traverses their interaction sequence. Finally *TransFM* (5.1d) learns an embedding $\vec{v}_i$ and translation vector $\vec{v}'_i$ for each feature, using the squared Euclidean distance to measure feature interactions.

5.6.2 Computation

The model equation for Factorization Machines can be computed in linear time $O(kn)$, where k is the dimensionality of the model parameter vectors and n is the dimensionality of the input feature vectors [41]. In this section, we show that the same result applies to the *TransFM* model.

In order to simplify the squared Euclidean distance d^2, we take advantage of the ability to write d^2 in terms of inner products:

$$d^2(\vec{v}_i + \vec{v}'_i, \vec{v}_j) = (\vec{v}_i + \vec{v}'_i - \vec{v}_j) \cdot (\vec{v}_i + \vec{v}'_i - \vec{v}_j). \tag{5.18}$$

This allows us to rewrite the interaction term as follows:

$$\begin{aligned}
&\sum_{i=1}^{n} \sum_{j=i+1}^{n} d^2(\vec{v}_i + \vec{v}'_i, \vec{v}_j) x_i x_j \\
&= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} d^2(\vec{v}_i + \vec{v}'_i, \vec{v}_j) x_i x_j - \frac{1}{2} \sum_{i=1}^{n} d^2(\vec{v}_i + \vec{v}'_i, \vec{v}_i) x_i x_i \\
&= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \left((\vec{v}_i + \vec{v}'_i - \vec{v}_j) \cdot (\vec{v}_i + \vec{v}'_i - \vec{v}_j) x_i x_j \right) - \frac{1}{2} \sum_{i=1}^{n} (\vec{v}'_i \cdot \vec{v}'_i) x_i x_i \\
&= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} (x_i x_j)(\vec{v}_i \cdot \vec{v}_i + \vec{v}'_i \cdot \vec{v}'_i + \vec{v}_j \cdot \vec{v}_j + 2\vec{v}_i \cdot \vec{v}'_i - 2\vec{v}_i \cdot \vec{v}_j - 2\vec{v}'_i \cdot \vec{v}_j) \\
&\qquad - \frac{1}{2} \sum_{i=1}^{n} (\vec{v}'_i \cdot \vec{v}'_i) x_i x_i
\end{aligned}$$

The first sum above can be split into six individual sums, each of which multiplies the

feature product $x_i x_j$ with one of the corresponding inner products. We present a simplified version of one of the six sums below:

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\vec{v}_i \cdot \vec{v}_i)x_i x_j = \frac{1}{2}\left(\sum_{i=1}^{n}(\vec{v}_i \cdot \vec{v}_i)x_i\right)\left(\sum_{j=1}^{n} x_j\right) \tag{5.19}$$

(others are similar and omitted for brevity).

Thus we see that all terms in Equation 5.16 can be computed with at most two sums over the input features, and at most one inner product between corresponding parameter vectors. Given input features of dimensionality n and parameters of dimensionality k, this shows that the *TransFM* model can be computed in linear complexity in both k and n, or $O(kn)$.

As with FMs, the above feature vectors are sparse (e.g. one-hot user/item encodings), so the above sums need to be computed only over the nonzero elements of the input feature vectors, further improving performance.

Various chapters in this thesis include material that has been submitted for publication as it may appear in RecSys 2018, Pasricha, Rajiv; McAuley, Julian, ACM, 2018. The thesis author was the primary investigator and author of this paper.

Chapter 6

Results

6.1 Datasets and Statistics

In order to evaluate the quantitative performance of our proposed models, we perform experiments using a variety of publicly available datasets that vary significantly in terms of size and sparsity. As we are concerned with learning models from implicit feedback, we first convert all observed ratings to be positive feedback, discarding observed ratings if present. We then remove all users and items with fewer than five observed interactions. Statistics of the datasets under consideration are included in Table 6.1.

Amazon[1]: This dataset, originally introduced by [33], contains a large corpus of product ratings, reviews, and metadata, collected from Amazon.com from May 1996 to July 2014. The full dataset consists of 83 million ratings and reviews collected during this period, along with additional features including item metadata and visual features. Notable for its high sparsity, the Amazon dataset provides a useful benchmark to evaluate recommender system algorithms on sparse input data. The additional available metadata also makes it an appealing choice to evaluate algorithms combining collaborative filtering techniques with additional sources of information. We use purchases from five top-level categories covering a variety of distinct purchase domains.

[1] http://jmcauley.ucsd.edu/data/amazon/

Table 6.1. Dataset Statistics (after preprocessing)

Dataset	**# Users** ($\|\mathcal{U}\|$)	**# Items** ($\|\mathcal{I}\|$)	**# Actions**	**Avg. # actions / user**	**Avg. # actions / item**
Office	16,716	22,357	128,070	7.66	5.73
Automotive	34,316	40,287	138,573	5.35	4.56
Video Games	31,013	23,715	287,107	9.26	12.11
Toys and Games	57,617	69,147	410,920	7.13	5.94
Cell Phones	68,330	60,083	429,231	6.28	7.14
North Carolina	4,573	7,846	31,167	6.82	3.97
Colorado	4,586	7,989	34,880	7.61	4.37
Washington	4,453	7,196	39,316	8.83	5.46
Florida	12,096	21,388	77,145	6.38	3.61
Texas	16,066	24,729	136,930	8.52	5.54
California	23,644	35,252	237,051	10.03	6.72
MovieLens	943	1,349	99,287	105.29	73.60
Total	274k	321k	2.05M	-	-

Google Local[2]: This dataset contains a large collection of business reviews and ratings and was originally introduced by [19]. The dataset also includes many associated content features, including user demographics and business locations. The availability of GPS coordinates facilitates evaluating *TransFM* in a geographical recommendation setting. In this work, we evaluate datasets containing businesses and reviews from six U.S. states of varying sizes and populations.

MovieLens[3]: The MovieLens dataset has been used for many years to evaluate a large variety of recommendation algorithms [18]. Created by the GroupLens research group at the University of Minnesota, MovieLens allows its users to submit ratings and reviews for movies they have watched and recommends movies that those users may enjoy. From its inception, MovieLens and its associated datasets have been vital to the development of improved

[2]http://jmcauley.ucsd.edu/data/googlelocal/
[3]https://grouplens.org/datasets/movielens/

recommendation algorithms as well as related studies in psychology and other domains [3, 12, 58, 32]. In this work, we use the MovieLens-100k benchmark dataset. Compared to the Amazon datasets, this dataset exhibits a much higher degree of user and item density.

6.2 Features

We have discussed a variety of extensions to TransRec that incorporate temporal information as well as additional content features. *TransFM* in particular is intended to be a 'feature-agnostic' general-purpose model that can yield significant performance improvements when incorporating additional features, with no other changes to the model format. Thus, our focus is not on complex feature design techniques, but rather to show that significant performance improvements can be achieved with minimal feature preprocessing. To that end, we extract the following content-based features from each dataset to evaluate our models:

Temporal Features: Temporal data has been widely used to improve recommendation performance [28, 14, 50]. Each of our datasets contain temporal information, specifically the time τ_{ui} for each rating between user u and item i.

For the implicit feedback recommendation task with a ranking loss, each training example consists of a triplet (u, i, j) of a user, previous item, and next item. As a result, we add two additional features to $\vec{x}_{u,i,j}$: the time τ_{ui} of user u's rating of previous item i, and the time τ_{uj} of u's rating of next item j. All temporal values for each dataset are first normalized to have zero mean and unit variance.

During training, corresponding positive and negative instances are both associated with the same timestamp, so that we are optimizing a time-specific ranking loss.

Item Category Features: The Amazon datasets also provide a list of categories for each

item. These categories form a hierarchical list of labels, which are useful as item features to improve recommendation performance and generalizability, especially in the sparse setting. We convert the observed category labels into binary indicator vectors for previous and next items and add them to their respective feature vectors.

User and Item Content Features: MovieLens provides a variety of content features for both users and items. We use the following features in our models: *User age, User gender, User occupation, User zip code,* and *Movie genre*. Movie genre, user occupation, and user zip code features are encoded into binary features. We convert the user gender to a single binary feature and apply no processing steps to the user age.

Geographical Features: As the Google Local datasets capture ratings and reviews for various businesses, each "item" is associated with its corresponding latitude and longitude coordinates. We add these GPS coordinates to *TransFM* to evaluate the model in a geographical setting. For each state, we first round the latitude and longitude coordinates to a single decimal place, and then create binary feature vectors $\vec{x}^{c}$ with one feature for each observed bin. For the Google Washington dataset, we observe 38 and 78 latitude and longitude features respectively.

6.3 Optimization

We consider the sequential recommendation setting with implicit feedback. i.e., rather than optimizing the precise output value of our model equation, we instead aim to rank the observed next item j ahead of all other items $j' \in \mathcal{I} \backslash j$ in the dataset. To this end, we adopt the Sequential Bayesian Personalized Ranking (S-BPR) optimization criterion [43].

Applying S-BPR, we optimize the total order $>_{u,i}$ given a user u and previous item i:

$$\begin{aligned}\hat{\Theta} &= \arg\max_{\Theta} \ln \prod_{u \in \mathcal{U}} \prod_{j \in \mathcal{S}^u} \prod_{j' \notin \mathcal{S}^u} \Pr(j >_{u,i} j' | \Theta) \Pr(\Theta) \\ &= \arg\max_{\Theta} \sum_{u \in \mathcal{U}} \sum_{j \in \mathcal{S}^u} \sum_{j' \notin \mathcal{S}^u} \ln \sigma(\hat{y}(\vec{x}_{u,i,j}) - \hat{y}(\vec{x}_{u,i,j'})) - \Omega(\Theta)\end{aligned} \tag{6.1}$$

where i is the item immediately preceding j in the consumption sequence. Accordingly, we also restrict j from being the first item in the sequence as it has no associated previous item. $\hat{y}(\vec{x})$ is the *TransFM* model described in Equation 5.16, Θ is the set of parameters $\{w_0, \vec{w}, \mathbf{V}, \mathbf{V}'\}$ to be learned by the model and $\Omega(\Theta)$ is a standard $\mathcal{L}_2$ regularization term. Finally, we denote by $\vec{x}_{u,i,j}$ the feature vector for the (u,i,j) user, previous item, next item triplet.

6.4 Implementation Details

We implement our models in TensorFlow [1] and use mini-batch gradient descent with Adam Optimization to train each model [25]. Adam is effective for learning models with many parameters on sparse datasets and was the most effective optimization algorithm in our experiments.

We apply the standard BPR optimization process, based on stochastic gradient descent with bootstrap sampling [42]. For every positive training triple (u,i,j), we randomly sample a negative item $j' \in \mathcal{I}$ on every iteration to add to our mini batch. This set of positive and negative triples is then used to update the parameters of the model. In our implementation, we do not enforce that j' is an item that has not been previously observed, which differs from the standard BPR optimization algorithm and related implementations. However, in our experiments, we observed that removing this additional check greatly reduced the execution time of our

algorithms, allowing us to vectorize the sampling process with sparse matrices when generating each training, validation, and test batch. As the length of each user's sequence is small compared to the total number of observed items, sampling negative items uniformly at random did not have a significant impact on the models' performance compared to enforcing that j' is an unobserved item.

All model parameters are randomly initialized within the interval $[-0.1, 0.1]$, and regularization parameter values are optimized using a grid search over the values $\{0.0, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0\}$. We iterate until convergence, as measured by performance on a held-out validation set.

The provided implementation of TransRec enforces that all item embeddings $\vec{\gamma}_i$ lie within a restricted subspace Ψ, for example a unit ball. This prevents the item embeddings from reaching extreme values, acting as a secondary form of regularization on the model. However, this constraint is implemented as a conditional parameter update after every training step, specifically $\vec{\gamma} \leftarrow \vec{\gamma} / \max(1, ||\vec{\gamma}||)$. This prevents the loss function from being continuously differentiable at all points and also increases the complexity of the model's implementation. In our implementation, we update the normalization step to have the following continuous form:

$$\vec{\gamma} \leftarrow \frac{\vec{\gamma}}{1 + ||\vec{\gamma}||} \tag{6.2}$$

This has the same effect as the original normalization approach of preventing the magnitude of $\vec{\gamma}$ from exceeding 1 but is a continuous and differentiable update that can be unconditionally applied. We replace the conditional update rule with this continuous approach for all models except TransFM, as we empirically find that it leads to improved performance. The TransFM

model does not apply any normalization on its parameters, relying solely on an $\mathcal{L}_2$ regularization term to reduce their magnitudes.

6.5 Evaluation Methodology

In the sequential recommendation setting, the prediction for each item depends on the previous items in the user's consumption sequence. As a result, we first partition the consumption sequence for user u into three sub-sequences. The most recent item $\mathcal{S}^u_{|\mathcal{S}^u|}$ is added to the test set, the previous item $\mathcal{S}^u_{|\mathcal{S}^u|-1}$ to the validation set, and the remaining $|\mathcal{S}^u|-2$ items are kept in the training set.

We report the performance of each model according to the **Area Under the ROC Curve**, or AUC, defined below.

$$\text{AUC} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{I} \setminus \mathcal{S}^u|} \sum_{j' \in \mathcal{I} \setminus \mathcal{S}^u} \mathbb{1}(R_{u,g_u} < R_{u,j'}) \tag{6.3}$$

Where g_u is the ground truth item for user u in the test set, and $R_{u,i}$ is the rank of item i for user u in the output list of recommendations. Finally, $\mathbb{1}(\cdot)$ is the indicator function that returns 1 if the ground truth item is ranked ahead of the unobserved item j'.

6.6 Baselines

We compare our proposed models against the following baselines:

PopRec: This is a naive popularity baseline that ranks items in order of their overall popularity in the dataset. It is not personalized, so it provides the same list of recommendations to all users.

BPR-MF: This model uses the Bayesian Personalized Ranking (BPR) framework with

Matrix Factorization (MF) as the underlying model [42]. It learns global personalized user-item dynamics but does not take sequential signals into account.

Factorized Markov Chain (FMC): This is a non-personalized sequential model that factorizes the global item-to-item transition matrix. It does not take personalized user interactions into account.

Factorized Personalized Markov Chain (FPMC): A combination of the MF and FMC models, FPMC factorizes the three dimensional sequential interaction tensor [43]. Predictions are computed by taking inner products between factorized parameter vectors:

$$P(j|u,i) \propto \langle \vec{v}_u^{U,J}, \vec{v}_j^{J,U} \rangle + \langle \vec{v}_i^{I,J}, \vec{v}_j^{J,I} \rangle, \tag{6.4}$$

where $V^{U,J}$, $V^{J,U}$, $V^{I,J}$, and $V^{J,I}$ are the four embedding spaces learned by the model.

Personalized Ranking Metric Embedding (PRME): This model replaces the inner products in FPMC with Euclidean distances, embedding users and items into two latent spaces to model personalized and sequential dynamics respectively [15]. The hyperparameter α modulates the relative importance between these two spaces:

$$P(j|u,i) \propto -(\alpha \cdot d(\vec{v}_u, \vec{v}_j) + (1-\alpha) \cdot d(\vec{w}_i, \vec{w}_j)). \tag{6.5}$$

Hierarchical Representation Model (HRM): This model introduces an aggregation component to FPMC to allow more flexibility in modeling interactions between users and items [53]:

$$P(j|u,i) \propto \langle f(\vec{v}_u, \vec{v}_i), \vec{v}_j \rangle \tag{6.6}$$

We test average and max pooling for the aggregation function f.

TransRec: This is the model proposed in [19], which embeds each item in a shared embedding space and learns personalized translation vectors through this space for each user. This allows the TransRec model to achieve state-of-the-art performance, excelling on datasets with the highest levels of sparsity. TransRec has the following prediction equation:

$$P(j|u,i) \propto \beta_j - d\left(\vec{\gamma}_i + \vec{T}_u, \vec{\gamma}_j\right) \tag{6.7}$$

FM: This is the standard Factorization Machine model, which models interactions between all pairs of features by using an inner product between corresponding parameter vectors:

$$\hat{y}(\vec{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \vec{v}_i, \vec{v}_j \rangle x_i x_j. \tag{6.8}$$

We evaluate the FM model in three cases: (1) without additional features, (2) with temporal features, and (3) with category / content features. These are represented in the results as "FM", "FM_{time}", and "$\text{FM}_{\text{content}}$" respectively.

The goal of our baselines is to compare (1) standard recommendation baselines, (2) specialized models for sequential recommendation (TransRec), and (3) general-purpose models with inner-product interaction terms. These models are evaluated against the models proposed in this thesis, which add additional complexity to the TransRec framework, incorporate content features by adding additional constraints, or model arbitrary feature vectors using metric/translation based interactions.

6.7 Models

We evaluate the following proposed models.

Item Offset Model: This model extends the translation vectors of TransRec to incorporate an item offset term, giving the following translation vector: $\vec{T}_{ui} = \vec{t} + \vec{t}_u + \vec{t}_i$

Time Delta Model: This model adds a time-dependent scaling factor to the model's translation vector. The translation vector $\vec{T}_u$ is multiplied by the time delta between the previous and next item consumption δ_{uij}.

Personalized Time Delta Model: This model extends the time delta approach, adding an additional coefficient θ_u that is personalized for each user. This results in the translation vector $\vec{T}_u$ being multiplied by the scaled time delta $\theta_u \delta_{uij}$.

Neural Network Translation Model: This model introduces an MLP to model the translation operation of the TransRec model, replacing the translation $\vec{\gamma}_i + \vec{T}_u$ by the output of a neural network: $f\left(\vec{\gamma}_i,\ \vec{T}_u\right)$.

Neural Network Translation Model (with Time Deltas): We add the time delta term as an input to the above MLP, computing the translation operation as $f\left(\vec{\gamma}_i,\ \vec{T}_u,\ \delta_{uij}\right)$.

Neural Network Translation Model (with Timestamps): Rather than using the preprocessed time delta to introduce temporal dynamics, we use the raw timestamps of the previous and next consumptions as input features. This gives the following translation operation: $f\left(\vec{\gamma}_i, \vec{T}_u, \tau_{ui}, \tau_{uj}\right)$.

Neural Network Distance Model: This model replaces the entire Euclidean distance calculation with an MLP, incorporating parameters from the user, previous item, and next item. It proposes the following interaction term $f\left(\vec{\gamma}_i, \vec{T}_u, \vec{\gamma}_j\right)$.

Neural Network Distance Model (with Time Deltas): Similar to the translation model above, we add the time delta between the previous and next consumption for each user into the MLP prediction equation, giving the following interaction term: $f\left(\vec{\gamma}_i,\ \vec{T}_u, \vec{\gamma}_j, \delta_{uij}\right)$.

Neural Network Distance Model (with Timestamps): In this model, we update the interaction component of the Neural Network Distance model to also incorporate the raw timestamps of the previous and next interactions for the respective user. This gives the following interaction term: $f\left(\vec{\gamma}_i,\ \vec{T}_u,\ \vec{\gamma}_j,\ \tau_{ui},\ \tau_{uj}\right)$.

Default-Origin Session Model: For this model, we define a neural network that approximates the probability that the user begins a new session with their next interaction. If a new session is started, the user transitions back to the origin instead of continuing on their personalized trajectory. With a soft threshold, this leads to the following expression for the translation vector: $g\left(\vec{\gamma}_i,\ \vec{T}_u\right)\left(\vec{\gamma}_i + \vec{T}_u\right)$.

Personalized-Origin Session Model: This extends the default origin session model above to learn a personalized 'home' location for each user. When the user begins a new session, rather than translating back to the origin, they transition back to their personalized home location. This gives the following expression for the translation vector: $g\left(\vec{\gamma}_i,\ \vec{T}_u\right)\left(\vec{\gamma}_i + \vec{T}_u\right) + \left(1 - g\left(\vec{\gamma}_i, \vec{T}_u\right)\right)\left(\vec{h}_u\right)$.

Nonlinear Personalized-Origin Session Model: We replace the linear translation term in the personalized-origin session model with the output of a neural network. This combines two of the models discussed above: the neural network translation model and the personalized origin session model. Each user follows a nonlinear trajectory and transitions back to a personalized home location on a new session, giving the following translation expression: $g^{(2)}\left(\vec{\gamma}_i, \vec{T}_u\right) g^{(1)}\left(\vec{\gamma}_i, \vec{T}_u\right) + \left(1 - g^{(2)}\left(\vec{\gamma}_i, \vec{T}_u\right)\right)\vec{h}_u$.

Category Cosine: This is a nave extension to TransRec that incorporates content features. We follow the intuition that items with similar content features should have similar embeddings. This is enforced by adding a regularization term that computes the distance between consecutive pairs of item embeddings $\vec{\gamma}_i$ and $\vec{\gamma}_j$, weighted by the cosine similarity of their corresponding content vectors:

$$\mathcal{R} = \sum_{u \in \mathcal{U}} \sum_{j \in \mathcal{S}^u} s\left(\vec{x}^c_{u,i}, \vec{x}^c_{u,j}\right) \left(\vec{\gamma}_i - \vec{\gamma}_j\right)^2 \tag{6.9}$$

TransFM: This model replaces the inner product of Factorization Machines with a translation operation followed by the computation of the squared Euclidean distance:

$$\hat{y}\left(\vec{x}\right) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} d^2\left(\vec{v}_i + \vec{v}'_i,\ \vec{v}_j\right) x_i x_j \tag{6.10}$$

As with FMs, we evaluate *TransFM* in three cases: (1) without additional features, (2) with temporal features, and (3) with content features. These are represented in the results as "*TransFM*," "$\mathit{TransFM}_{\text{time}}$," and "$\mathit{TransFM}_{\text{content}}$" respectively.

6.8 Performance and Quantitative Analysis

Results from our experiments are collected below. The number of factor dimensions k for all models is set to 10; we analyze the impact of changing the dimensionality in a future section.

6.8.1 Baseline Models

Results for our baseline models are summarized in Table 6.2. As expected, all models outperform the simple popularity-based baseline. BPR-MF and FMC respectively model personalized and sequential components, and their performance compared to the simple popularity baseline shows that both components play a significant role in making successful recommenda-

Table 6.2. Results of our baseline models with respect to the AUC (higher is better). The best performing model for each dataset is bolded.

Dataset	PopRec	BPR-MF	FMC	FPMC	PRME	HRM_{avg}	HRM_{max}	TransRec
Office Products	0.6427	0.6979	0.6865	0.6859	0.7006	0.6985	0.6983	**0.7383**
Automotive	0.5870	0.6307	0.6442	0.6415	0.6473	0.6703	0.6560	**0.6953**
Video Games	0.7497	0.8551	0.8423	0.8523	0.8601	0.8779	0.8566	**0.8885**
Toys and Games	0.6240	0.7289	0.6948	0.7198	0.7264	0.7581	0.7263	**0.7643**
Cell Phones	0.6959	0.7611	0.7548	0.7376	0.7887	0.7891	0.7656	**0.8080**
Amazon Average	0.6599	0.7347	0.7245	0.7274	0.7446	0.7588	0.7406	**0.7789**
North Carolina	0.4888	0.7096	0.6542	0.6698	0.7064	**0.7691**	0.7067	0.7507
Colorado	0.5085	0.6826	0.6164	0.6463	0.6602	**0.7219**	0.6666	0.7161
Washington	0.5123	0.6994	0.6491	0.6662	0.6837	**0.7440**	0.6941	0.7313
Florida	0.4722	0.7275	0.6432	0.6619	0.7107	**0.7812**	0.7109	0.7685
Texas	0.5612	0.7657	0.7153	0.7239	0.7532	**0.8207**	0.7506	0.8030
California	0.5785	0.7969	0.7284	0.7462	0.7750	**0.8346**	0.7692	0.8215
Google Average	0.5203	0.7303	0.6673	0.6857	0.7149	**0.7786**	0.7164	0.7652
MovieLens	0.7413	0.8602	0.8515	0.8858	0.8851	0.8856	0.8844	**0.8873**

tions. By adding a personalization component, FPMC outperforms FMC for all datasets and is among the best baselines for the (dense) MovieLens dataset. However, it loses to BPR-MF for most datasets, suggesting that learning multiple independent embeddings is not well-suited to sparse domains.

By replacing inner products with metric distances, PRME outperforms FPMC for all Amazon and Google datasets. HRM_{avg} outperforms PRME in most cases, demonstrating the effectiveness of an appropriate aggregation term. This contrasts with [53], in which the nonlinear max pooling operation performed best. We expect that the increased sparsity of our data inhibits the ability of HRM_{max} to uncover appropriate nonlinear dynamics.

TransRec is the best performing content-agnostic method for Amazon and MovieLens but loses to HRM_{avg} for all Google Local datasets. This suggests that the translation vector intuition of TransRec does not effectively model interactions in Google Local as well as the simpler HRM model.

Table 6.3. Results of our proposed item offset and temporal extensions with respect to the AUC (higher is better). The first column contains the best performing baseline model (from Table 6.2). The best performing model for each dataset is bolded.

Dataset	Best Baseline	Item Offset	Time Delta	Personalized Time Delta
Office Products	**0.7383**	0.7369	0.7286	0.7233
Automotive	0.6953	**0.6954**	0.6900	0.6780
Video Games	**0.8885**	0.8839	0.8823	0.8573
Toys and Games	0.7643	**0.7732**	0.7713	0.7547
Cell Phones	0.8080	**0.8137**	0.8083	0.7927
Amazon Average	0.7789	**0.7806**	0.7761	0.7612
North Carolina	**0.7691**	0.7380	0.7467	0.7228
Colorado	**0.7219**	0.7103	0.6887	0.6882
Washington	**0.7440**	0.7125	0.7242	0.7247
Florida	**0.7812**	0.7557	0.7389	0.7192
Texas	**0.8207**	0.7902	0.7963	0.7820
California	**0.8346**	0.8112	0.8197	0.8041
Google Average	**0.7786**	0.7530	0.7525	0.7402
MovieLens	**0.8873**	0.8707	0.8539	0.8624

6.8.2 Proposed TransRec Extensions

Item Offset Model: Results for the item offset and time delta models are summarized in Table 6.3. Adding an item offset term to the translation vector improved the performance for a few datasets. However, most improvements were minor and for most datasets, the performance decreased compared to the original TransRec model. This could be due to the characteristics of the datasets themselves, in which items are less conducive than users to being modeled with a translation-based approach. In addition, simply adding linear translation component for both users and items could lead to interference between vectors that point in varying directions.

Time Delta Models: Both the time delta and personalized time delta models result in a decrease in performance for most datasets. We also observe that the personalized time delta model tends to perform worse compared to the non-personalized version. Overall, scaling the translation vector by the time delta between the previous and next interactions does not lead to

Table 6.4. Results of our proposed neural network models with respect to the AUC (higher is better). The first column contains the best performing baseline model (from Table 6.2). The best performing model for each dataset is bolded.

Dataset	Best Baseline	NN Translation	NN Distance
Office Products	**0.7383**	0.7015	0.7033
Automotive	**0.6953**	0.6466	0.6522
Video Games	**0.8885**	0.8560	0.8601
Toys and Games	**0.7643**	0.7245	0.7216
Cell Phones	**0.8080**	0.7707	0.7772
Amazon Average	**0.7789**	0.7399	0.7429
North Carolina	**0.7691**	0.6991	0.6949
Colorado	**0.7219**	0.6552	0.6745
Washington	**0.7440**	0.6882	0.6966
Florida	**0.7812**	0.6981	0.7053
Texas	**0.8207**	0.7548	0.7628
California	**0.8346**	0.7727	0.7892
Google Average	**0.7786**	0.7114	0.7206
MovieLens	**0.8873**	0.8645	0.8523

better predictions, even when this scaling is determined for each user.

There are a variety of reasons for this decline in performance. This is likely due to the fact that this particular temporal model does not match the true temporal effects observed in the data. A slightly different or more complex temporal model could more closely approximate the true interactions. The performance drop could also be due to the wide range of possible time delta values. We observe time deltas varying on a scale of minutes to years. As we normalize all time deltas to have zero mean and unit standard deviation, this leads to many deltas having approximately the same processed value, reducing the effective degrees of freedom of the model.

This result supports the notion that in order to appropriately model arbitrary temporal effects, a more complex model formulation is required. Models that learn the temporal effects from the data or model arbitrary interactions between features (e.g. Factorization Machines) could provide improved performance over enforcing a custom form for temporal dynamics through data inspection or trial and error.

Neural Network Models: Results for our proposed neural network models are summarized in Table 6.4. Both the neural network translation and neural network distance models perform worse than TransRec on all datasets. This suggests that the additional degrees of freedom afforded by incorporating neural networks into the TransRec framework do not lead to improved prediction performance. Due to the significant sparsity of our datasets, the linear structure of TransRec more closely approximates the true interactions between users and items, despite its simple form. Attempting to learn a personalized nonlinear translation vector that depends on a user, previous item, and next item is not feasible when for most datasets, fewer than 10 interactions are observed for each user and item.

The neural network distance model relaxes the metricity assumption of TransRec, replacing the Euclidean distance with an MLP that directly computes the probability of interacting with the next item in a user's sequence. The reduced performance of this model supports the notion that the constraints enforced by the Euclidean distance metric facilitate improved generalization behavior. By replacing the distance metric with an arbitrary neural network, the triangle inequality guarantee is lost, and performance suffers as a result, especially for sparse datasets.

Neural Network Temporal Models: Results for our proposed neural network temporal models are summarized in Table 6.5. Adding temporal dynamics to the neural network models discussed above leads to improved AUC performance for some datasets, but these improvements are not consistent. For MovieLens and most of the Google datasets, the neural network temporal models achieve similar performance to their non-temporal counterparts. In contrast, for the Amazon datasets, adding timestamps to the neural networks lead to more significant improvements over the standard and time delta-based approaches. However, even the best MLP approaches do not exceed the original TransRec model in terms of performance.

Table 6.5. Results of our proposed neural network temporal models with respect to the AUC (higher is better). The first column contains the best performing baseline model (from Table 6.2). The best performing model for each dataset is bolded.

Dataset	Best Baseline	NN Translation Time Deltas	NN Translation Timestamps	NN Distance Time Deltas	NN Distance Timestamps
Office Products	**0.7383**	0.6986	0.7368	0.7034	0.7276
Automotive	**0.6953**	0.6532	0.6639	0.6519	0.6562
Video Games	**0.8885**	0.8521	0.8573	0.8678	0.8854
Toys and Games	**0.7643**	0.6923	0.7460	0.7228	0.7609
Cell Phones	**0.8080**	0.7450	0.8066	0.7794	0.8035
Amazon Average	**0.7789**	0.7282	0.7621	0.7451	0.7667
North Carolina	**0.7691**	0.6794	0.6987	0.6989	0.6929
Colorado	**0.7219**	0.6527	0.6449	0.6679	0.6598
Washington	**0.7440**	0.6915	0.6914	0.6932	0.6965
Florida	**0.7812**	0.6920	0.6873	0.7058	0.7087
Texas	**0.8207**	0.7580	0.7592	0.7569	0.7590
California	**0.8346**	0.7754	0.7718	0.7862	0.7833
Google Average	**0.7786**	0.7082	0.7089	0.7182	0.7167
MovieLens	**0.8873**	0.8695	0.8617	0.8685	0.8622

Although the MLP approach allows the model to automatically uncover the appropriate temporal dynamics from the data itself, the temporal models suffer from the same problems as the MLP approaches described above. In particular, the sparsity of the data makes it difficult for the model to extract appropriate nonlinear translation dynamics and incorporating temporal dynamics does not overcome this lack of appropriate interaction data.

In addition, we only evaluate one architecture for our MLP, with a single hidden layer with 10 units and a tanh activation function. Additional performance improvements could be achieved by incorporating more complex network structures (e.g. more hidden units or additional hidden layers), different activation functions (e.g. sigmoid or ReLU), or more sophisticated regularization techniques (e.g. dropout). There are also a variety of additional neural network architectures that have been successfully been applied to recommendation and related settings, such as convolutional and recurrent networks. As previously discussed, many recent state-of-

Table 6.6. Results of our proposed session-based models with respect to the AUC (higher is better). The first column contains the best performing baseline model (from Table 6.2). The best performing model for each dataset is bolded.

Dataset	Best Baseline	Default Origin	Personalized Origin	Nonlinear Personalized
Office Products	0.7383	**0.7396**	0.7183	0.6961
Automotive	0.6953	**0.7012**	0.6776	0.6449
Video Games	**0.8885**	0.8851	0.8642	0.8560
Toys and Games	0.7643	**0.7725**	0.7402	0.7089
Cell Phones	0.8080	**0.8125**	0.7801	0.7492
Amazon Average	0.7789	**0.7822**	0.7561	0.7310
North Carolina	**0.7691**	0.7365	0.7280	0.6949
Colorado	**0.7219**	0.7030	0.7030	0.6619
Washington	**0.7440**	0.7221	0.7053	0.6826
Florida	**0.7812**	0.7509	0.7305	0.6979
Texas	**0.8207**	0.7875	0.7652	0.7558
California	**0.8346**	0.8075	0.7860	0.7779
Google Average	**0.7786**	0.7513	0.7363	0.7118
MovieLens	**0.8873**	0.8694	0.8596	0.8663

the-art recommendation approaches incorporate various types of neural networks, so a more sophisticated neural network approach could lead to more significant performance improvements.

Session-based Models: Results from our proposed session-based models are collected in Table 6.6. The default-origin session-based model outperforms the original TransRec baseline for most Amazon datasets. These improvements are very minor but suggest that session dynamics in user transaction sequences are a useful signal to include. The simple nature of the default-origin approach also leads to improved performance in the sparse setting, with the complex neural network only used to determine the probability of a new session, and the translation vector being updated accordingly.

We observe that the more complex personalized and nonlinear session-based models both display worse performance than the default-origin approach. Both models aim to learn more complex session or translation dynamics, including a personalized home location per user along

Table 6.7. Results of baselines and proposed models that incorporate content features. Results are presented with respect to the AUC (higher is better). The first column contains the best performing model (from Table 6.2). The best performing model for each dataset is bolded. The final column shows the percent improvement of $TransFM_{\text{content}}$ over the best baseline.

Dataset	Best Baseline	CatCos	FM	FM_{time}	$\text{FM}_{\text{content}}$	TransFM	$\text{TransFM}_{\text{time}}$	$\text{TransFM}_{\text{content}}$	Improv. vs. Best Baseline
Office Products	0.7383	0.7402	0.7075	0.7426	0.7586	0.7169	0.7430	**0.8463**	11.6%
Automotive	0.6953	0.7048	0.6572	0.6671	0.7328	0.6675	0.6776	**0.8319**	13.5%
Video Games	0.8885	0.8878	0.8523	0.8866	0.8912	0.8584	0.8778	**0.9587**	7.6%
Toys and Games	0.7643	0.7762	0.6994	0.7488	0.7761	0.7203	0.7583	**0.8673**	11.7%
Cell Phones	0.8080	0.8099	0.7558	0.8153	0.7611	0.7767	0.8209	**0.8406**	3.1%
Amazon Average	0.7789	0.7838	0.7344	0.7721	0.7840	0.7480	0.7755	**0.8690**	10.8%
North Carolina	0.7691	0.7524	0.6787	0.6554	0.7673	0.7454	0.6257	**0.7947**	3.3%
Colorado	0.7219	0.7177	0.6504	0.6392	0.7345	0.6327	0.6203	**0.7535**	2.6%
Washington	0.7440	0.7352	0.6812	0.6761	0.7352	0.6498	0.6289	**0.7586**	2.0%
Florida	0.7812	0.7639	0.7057	0.6757	0.7821	0.6507	0.6233	**0.8095**	3.5%
Texas	0.8207	0.8021	0.7435	0.7251	0.8025	0.7072	0.6857	**0.8371**	2.0%
California	0.8346	0.8221	0.7732	0.7608	0.8107	0.7341	0.7157	**0.8379**	0.4%
Google Average	0.7786	0.7656	0.7055	0.6887	0.7721	0.6700	0.6499	**0.7986**	2.6%
MovieLens	0.8873	0.8678	0.8575	0.8617	0.8660	0.8611	0.8722	**0.9381**	5.7%

with potential nonlinear translation dynamics. The reduced performance of these models matches what we have previously observed, with the more complex model extensions often resulting in reduced performance. Given so few observed actions per user and item, it is infeasible for a model to accurately learn multiple quantities per user and item. The personalized origin approach learns a translation vector and home location for every user, with the nonlinear model adding custom translation vectors that are dependent on the user, previous item, and next item. These models might provide improved performance in more dense settings, where there is sufficient data to learn more complex dynamics.

6.8.3 Extensions with Content Features

We next discuss models that incorporate content-based features into the prediction equation. This also includes general-purpose models that operate on arbitrary feature vectors.

Results for content-based models are summarized in Table 6.7.

CatCos: The CatCos baseline model outperforms the non-content aware baselines for Amazon but loses to solely collaborative approaches for the other datasets, despite the addition of useful content features. This indicates that more specialized models or feature representations would be necessary to fully incorporate content information into the TransRec framework.

FMs: The standard FM model performs worse than more specialized sequential baseline models for all datasets. As opposed to many other baseline approaches, FMs do not explicitly model personalized sequential dynamics, and use inner products to model arbitrary feature interactions. Compared to metric-based approaches, these inner products are less effectively able to extract useful dynamics from extremely sparse datasets.

FMs with Features: Factorization Machines are effectively able to incorporate content features and achieve significant performance benefits, without requiring any changes to the model format itself. Adding temporal data to FMs leads to significant performance improvements for Amazon and MovieLens, despite only adding two additional features to $\vec{x}_{u,i,j}$.

This highlights the importance of effectively modeling temporal data to improve recommendation performance and shows that strong temporal effects are present in these datasets. However, adding temporal features causes performance for Google Local to decline, as temporal dynamics do not play as significant a role in modeling review sequences for local businesses.

Adding content features also results in substantial improvements, especially for datasets with the highest sparsity. $\text{FM}_{\text{content}}$ outperformed FM_{time} in most cases, demonstrating the importance of content features to compensate for insufficient interaction data.

TransFM: Although *TransFM* (without features) does not outperform all baseline models, it does exceed standard FMs for the Amazon and MovieLens datasets. However, FMs

perform better for the Google Local datasets, suggesting that without any additional features, inner products more effectively model interactions in this setting. This matches our observations of TransRec, which is outperformed by the inner product-based HRM_{avg} baseline.

TransFM with Features: Adding temporal data to *TransFM* has a similar effect as the corresponding FM_{time} baseline. When temporal features play a significant role in the datasets (e.g. for Amazon and MovieLens), *TransFM* is able to extract these dynamics.

The $\textit{TransFM}_{\text{content}}$ approach achieves the highest AUC for all datasets. The translation technique is effective at modeling both content and collaborative feature interactions, resulting in more significant improvements over vanilla *TransFM* than the corresponding $\text{FM}_{\text{content}}$ approach. These improvements hold for all datasets: Amazon (with category features), Google Local (with geographical features), and MovieLens (with user/item content features). Despite the increased density of MovieLens, *TransFM* is still able to extract additional value from user and item content features to improve recommendation performance.

For the Google Local dataset, geographical features play a more significant role in user-item interactions than temporal data. The performance of $\textit{TransFM}_{\text{content}}$ on this dataset indicates the translation component is effectively able to model interactions between arbitrary user, item, and geographical features.

6.8.4 Sign of the TransFM Interaction Term

Like FMs, *TransFM* adds the interaction term in the prediction equation (see Equation 5.16). This assigns features that are farther apart a higher interaction strength. In order to more closely match the intuition of standard metric-based models, where smaller distances correspond to higher interaction weights, we also tested a variant of *TransFM* with a negative interaction term. The resulting model displayed similar performance with no additional fea-

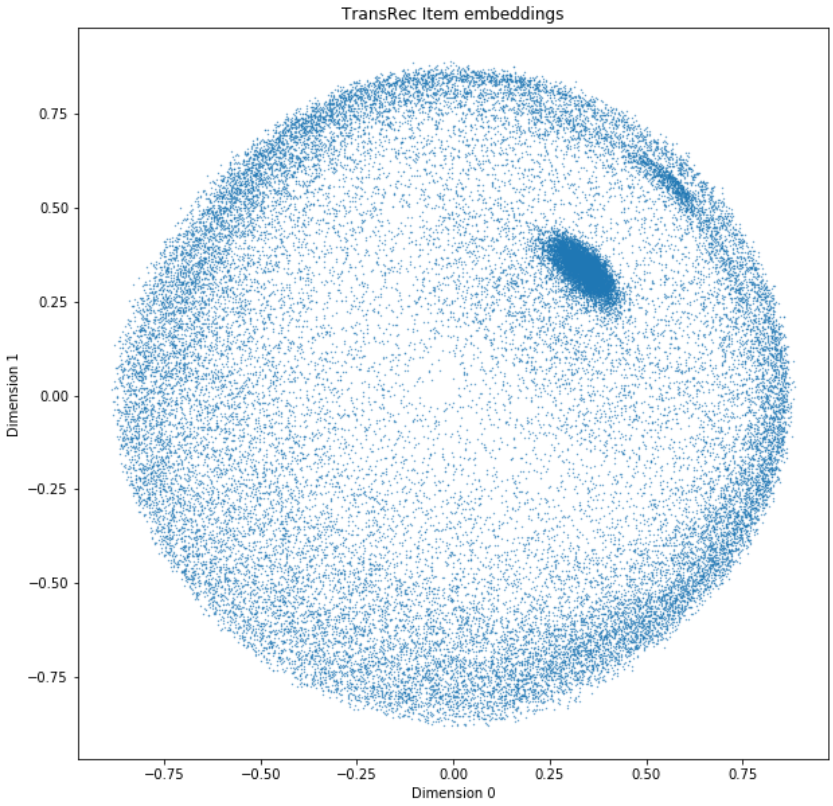


Figure 6.1. Plot of the item embeddings learned by TransRec. The model is trained with item factor dimensionality $k = 2$, e.g. $\vec{\gamma}_i \in \mathbb{R}^2$.

tures or with temporal data but had significantly reduced performance with content features. This suggests that an additive distance term increases the model's flexibility to appropriately model interactions between users, items, and content features, with the $\mathcal{L}_2$ regularization term constraining the feasible set of embedding locations.

6.9 Qualitative Analysis

6.9.1 TransRec Embeddings and Trajectories

In this section, we present some qualitative results of the original TransRec model. We first train TransRec using two latent factors, learning a latent representation $\vec{\gamma}_i \in \mathbb{R}^2$ for each item. These latent representations are plotted in Figure 6.1. We see that the normalization constraint is evident in the plot of item factors, constraining the item embeddings to be in a ball around the origin. The normalization step of TransRec acts as a regularization constraint for the model, and we observed that including this step led to significantly improved performance compared to not normalizing the item representations at all.

For the most part, the item embeddings look fairly evenly distributed throughout the embedding space. Interestingly, this highlights the lack of any straightforward sequential dynamics in the derived model embeddings. However, we do see a tight cluster of embeddings in the north east section of the diagram. The most likely explanation for this cluster is that there are a large number of item embeddings with few observed interactions that all share similar gradients through the training process. We also observe a cluster of embeddings located around the edge of the circle. This demonstrates the impact of the normalization approach applied to the item embeddings of the model. Given the smooth normalization technique previously discussed, where we divide each embedding by 1 plus the embedding norm on each iteration, this leads to a smooth boundary around the observed item embeddings. When applying the embedding approach of the original TransRec model, we observed a rigid boundary at the boundary of the unit circle, where item embeddings beyond the boundary would be normalized to be exactly on the boundary, and items within the unit ball would be unaffected.

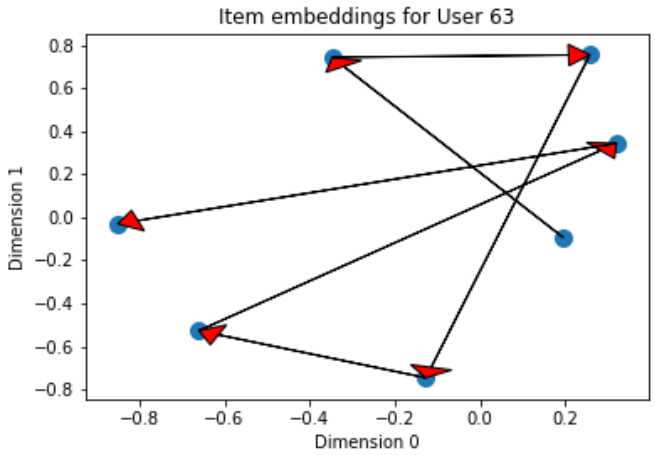


Figure 6.2. Item embedding locations and sequential transitions for an example user in the Amazon Automotive dataset. Item embeddings are learned by TransRec with item embedding dimensionality $k = 2$.

We also plot the sequence of item factors for a single user in the Amazon Automotive dataset, in Figure 6.2. Unlike the model assumption of TransRec, we do not observe a direct relationship among adjacent items in the user's consumption sequence. This matches what we observed in the global item factor plot above. Given the precise sequential format of the model, we would have expected many items to be arranged in visible sequences in the global embedding plot. We would also have expected the items for a single user to lie approximately linearly within the space, to allow the user to effectively transition from one item to the next. However, this is not the case, and the overall items and sequences for each user seem to be relatively arbitrarily distributed.

These visualizations of TransRec suggest that despite the model's simple form and translation structure, the embeddings learned by the model do not necessarily have an explicit "translation" structure as might be expected. Perhaps the translation structure becomes more evident in higher dimensions, and these visualizations do not take into account item biases or rankings of predicted items. However, the lack of explicit structure among the embeddings and

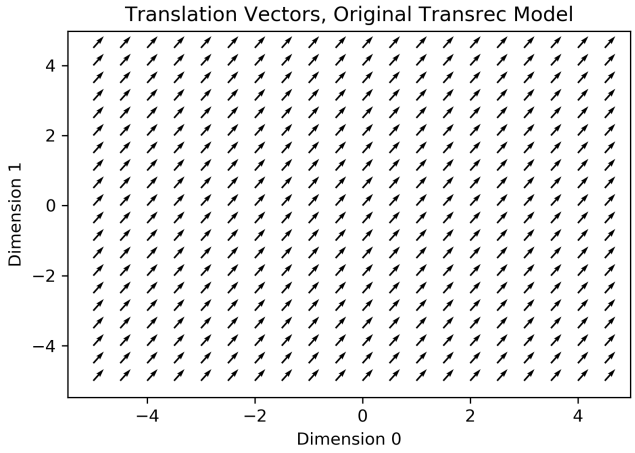


Figure 6.3. Vector field of the translation vectors learned by TransRec. The model was trained with embedding dimensionality $k = 2$, and each arrow represents the average translation vector for the given previous item embedding, averaged across all users.

trajectories suggests that extended models that learn more appropriate embeddings and sequences could provide improved recommendation performance.

6.9.2 Vector Fields

We plot vector field diagrams to explicitly visualize the translation vectors that are learned by TransRec and extended models. Figure 6.3 shows the average translation vector learned by TransRec on the Amazon Automotive dataset. This is the average translation vector $\vec{T}_u = \vec{t} + \vec{t}_u$ for every user u. As the model learns a linear translation vector that does not depend on the location in the item embedding space, plotting a vector field of the average translation vectors in the item embedding space results in all translation vectors pointing in the same direction.

By contrast, Figure 6.4 shows the translation vectors learned by the neural network translation model. The vectors are plotted in a two-dimensional item embedding space, where the location of each arrow corresponds to the location of the previous item embedding $\vec{\gamma}_i$. We compute the average translation vectors at various locations in the embedding space for the first

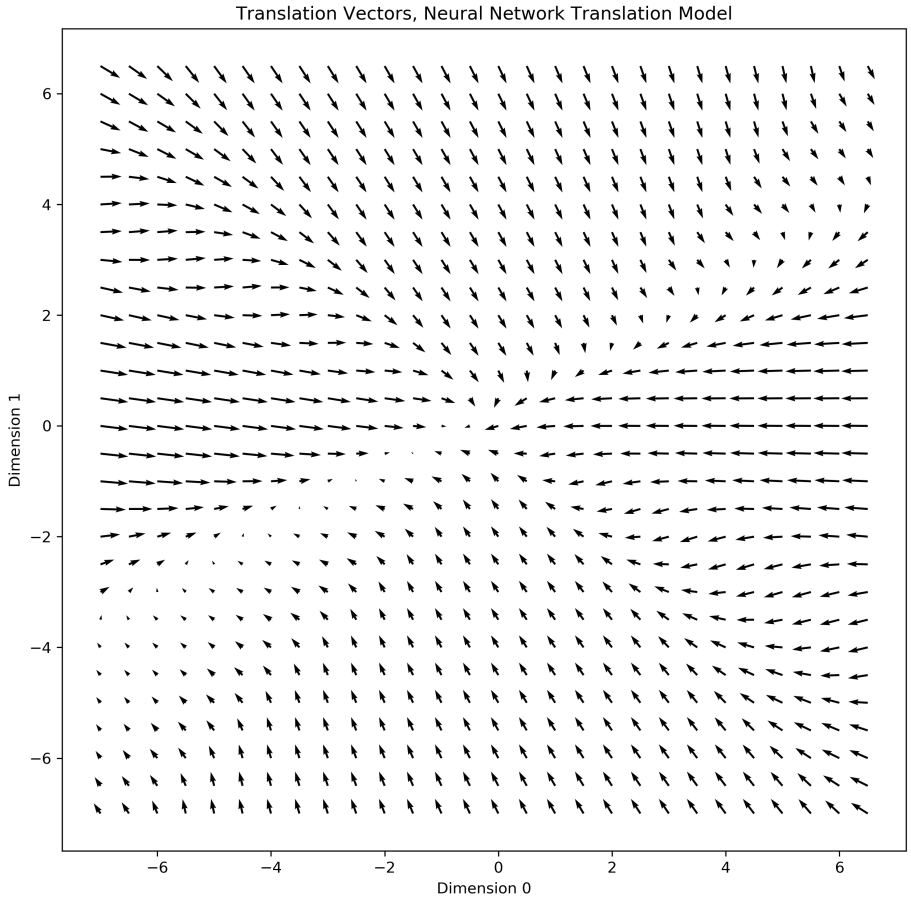


Figure 6.4. Vector field of the translation vectors learned by the NN translation model. The model was trained with an embedding dimensionality $k = 2$, and each arrow represents the average translation vector for the given previous item embedding, averaged across the first 100 users in the dataset.

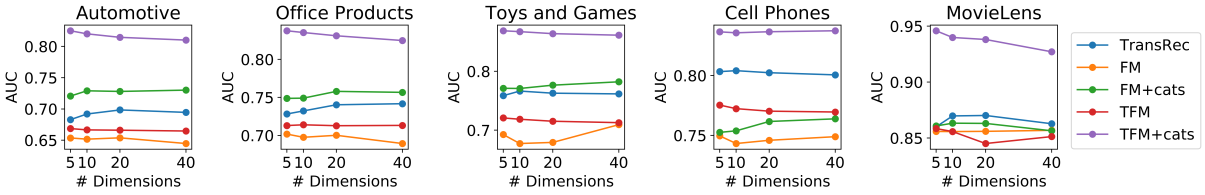


Figure 6.5. AUC vs. Dimensionality

100 users in the Amazon Automotive dataset. We observe that the neural network translation model, which uses an MLP to compute translation vectors, displays a much more complex vector field diagram, with the direction of the translation vector varying significantly depending on the location in the item embedding space. This demonstrates that the MLP is learning complex nonlinear translation dynamics from observed interaction sequences.

However, the resulting complexity of the translation vectors, compared to the significant sparsity in the dataset, suggests that the neural network is significantly overfitting to the observed interactions. This matches the reduced performance of the neural network models observed in our results and suggests that the arbitrary modeling capability provided by the MLP is not well suited to the sparse sequential recommendation task. A simpler model more finely optimized to the specific recommendation task could provide improved performance by more closely aligning with the observed sequential dynamics in the data itself.

6.9.3 Sensitivity to Dimensionality

We analyze the sensitivity of *TransFM* and various baseline models to the dimensionality of the learned parameter vectors. Specifically, we analyze TransRec as well as FMs and *TransFM* (without additional features and with content features). We adjust k in the set $k \in 5, 10, 20, 40$ and plot the resulting AUC values for five datasets in Figure 6.5 (other datasets exhibited similar performance). We observe that in most cases, performance does not increase significantly with

dimensionality. However, $TransFM_{\text{content}}$ significantly outperforms all other models for all values of k.

Various chapters in this thesis include material that has been submitted for publication as it may appear in RecSys 2018, Pasricha, Rajiv; McAuley, Julian, ACM, 2018. The thesis author was the primary investigator and author of this paper.

Chapter 7

FMs Applied to Related Recommendation Approaches

TransFM is a general-purpose model which adds elements from translation and metric-based sequential algorithms to the FM framework. TransRec, a similar translation-based model, was applied in [19] to the sequential recommendation task but lacked the ability to be natively extended with content features. In this section, we present related extensions of FMs that draw inspiration from similar baseline algorithms to achieve improved performance while retaining compatibility with arbitrary feature vectors.

We apply a similar approach to two baseline models: PRME and HRM, specialized sequential recommendation models similar to TransRec. PRME models sequential recommendations with embeddings in a metric space, using single embedding locations rather than translation vectors. HRM, specifically HRM_{avg}, models a similar vector addition operation but relies on inner products rather than metric spaces. By incorporating both translation and metric-space intuitions, TransRec is able to outperform PRME for all datasets and HRM_{avg} for Amazon and MovieLens. We observe similar results for their FM-inspired counterparts, with the translation and distance components of *TransFM* providing improved performance over related models.

7.1 Personalized Ranking Metric Embedding (PRME)

PRME [15] extends FPMC by learning personalized and sequential embeddings and replacing inner products with Euclidean distances (see Equation 6.5). To apply the PRME approach to FMs, we replace the inner product with the squared Euclidean distance between corresponding features. This gives the following *PRME-FM* model:

$$\hat{y}(\vec{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} d^2(\vec{v}_i, \vec{v}_j) x_i x_j. \tag{7.1}$$

Note that *PRME-FM* is simply *TransFM* without the translation space. The model learns a single embedding space and computes interaction weights according to the (squared) distance between embeddings. This model also retains the general-purpose nature of FMs and *TransFM* and can be simplified (similar to Section 5.6.2) to be computed in linear time.

7.2 Hierarchical Representation Model (HRM)

We next present a combined model between FMs and HRM [53]. HRM aggregates user and item representations (see Equation 6.6) prior to taking the inner product. We found above that average pooling is more effective, and HRM_{avg} is the best performing baseline for most Google Local datasets. We apply a similar intuition to the FM framework. Specifically, we adapt the first term of the inner product to take the sum of both embeddings, giving the following *HRM-FM* model:

$$\hat{y}(\vec{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \vec{v}_i + \vec{v}_j, \vec{v}_i \rangle x_i x_j. \tag{7.2}$$

The sum $\vec{v}_i + \vec{v}_j$ is the aggregation term that combines the learned representations for features i and j prior to taking the inner product. This model is also a general-purpose algorithm

Table 7.1. Results for alternative FM-derived approaches. Models are evaluated according to the AUC (higher is better).

Model	Amazon Automotive	Google Florida	MovieLens
FM	0.6572	0.7057	0.8575
FM_{time}	0.6671	0.6757	0.8617
$FM_{content}$	0.7328	0.7821	0.8660
TransFM	0.6675	0.6507	0.8611
$TransFM_{time}$	0.6776	0.6233	0.8722
$TransFM_{content}$	0.8319	0.8095	0.9381
PRME-FM	0.6674	0.6501	0.8639
$PRME\text{-}FM_{time}$	0.6749	0.6240	0.8701
$PRME\text{-}FM_{content}$	0.7422	0.8115	0.8557
HRM-FM	0.6662	0.6521	0.8581
$HRM\text{-}FM_{time}$	0.6720	0.6281	0.8744
$HRM\text{-}FM_{content}$	0.7411	0.8160	0.8606

and can be computed in linear time with a similar simplification as in Section 5.6.2.

7.3 Experiments

We compare *PRME-FM* and *HRM-FM* against standard FMs and *TransFM*. We evaluate these models against three datasets: "Amazon Automotive," "Google Florida," and "MovieLens". As in our previous experiments, models are evaluated according to the AUC in the following settings: (1) without features, (2) with temporal features, and (3) with content features. Results are presented in Table 7.1.

The overall trends for *PRME-FM* and *HRM-FM* are similar to FMs and *TransFM*. *TransFM* outperforms both *PRME-FM* and *HRM-FM* on Automotive and MovieLens, indicating that the increased expressiveness of the translation space more effectively models feature interactions. The models are similar in performance on the Florida dataset, potentially indicating a simpler relationship between features that is captured by all three approaches.

We do not observe a significant difference between *PRME-FM* and *HRM-FM* in terms of AUC. The sum and distance operations both improve on the inner product operation of standard

FMs, and both capture a similar amount of signal in all evaluated datasets.

Compared to standard Factorization Machines, *HRM-FM*, *PRME-FM*, and *TransFM* all provide significantly improved AUC performance with content features. This demonstrates that merging FMs with specialized sequential algorithms can consistently lead to effective general-purpose recommendation models.

Various chapters in this thesis include material that has been submitted for publication as it may appear in RecSys 2018, Pasricha, Rajiv; McAuley, Julian, ACM, 2018. The thesis author was the primary investigator and author of this paper.

Chapter 8

Conclusions and Future Work

In this thesis, we have presented a variety of extensions to the TransRec model for sequential recommendation. Many of these extensions augment the TransRec model formulation by adding increased complexity, e.g. by employing neural networks to model nonlinear translations, incorporating temporal or content information, modeling "sessions" in user interaction sequences, or relaxing the metricity assumption present in the original model. We evaluate our proposed extensions on a variety of real-world datasets and find that in most cases, performance decreases compared to the original TransRec model. Despite the reduced effectiveness of most of our proposed models, they provide a useful framework to which additional extensions of TransRec and related translation-based approaches can be compared. Additionally, we provide qualitative analysis of the learned item embeddings, user consumption sequences, and translation vectors of TransRec and our nonlinear approaches. These qualitative discussions provide an intuitive analysis of the linear and nonlinear dynamics learned by these models.

Finally, we introduced *TransFM*, which combines translation and metric-based approaches for sequential recommendation with Factorization Machines. This model learns an embedding and translation space for each feature and replaces the inner product of FMs with a translation term and distance metric. This general-purpose model natively supports the addition

of content features without requiring specialized constraints or adjustments. We evaluated *TransFM* on a variety of datasets and found that it achieves state-of-the-art performance when incorporating content features. We also found that applying a similar intuition, combining FMs with other baselines, consistently leads to improved general-purpose models.

Future research directions include (1) further extending TransRec by applying additional concepts from recommender systems and machine learning as a whole, (2) applying *TransFM* to arbitrary machine learning tasks besides sequential recommendation, and (3) determining the impact of additional features or feature representations on the models' performance.

Various chapters in this thesis include material that has been submitted for publication as it may appear in RecSys 2018, Pasricha, Rajiv; McAuley, Julian, ACM, 2018. The thesis author was the primary investigator and author of this paper.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.

[2] Charu C Aggarwal. *Recommender Systems*. Springer, 2016.

[3] Gerard Beenen, Kimberly Ling, Xiaoqing Wang, Klarissa Chang, Dan Frankowski, Paul Resnick, and Robert E Kraut. Using social psychology to motivate contributions to online communities. In *CSCW*, 2004.

[4] Robert M Bell, Yehuda Koren, and Chris Volinsky. The bellkor 2008 solution to the netflix prize. *Statistics Research Department at AT&T Research*, 1, 2008.

[5] Shay Ben-Elazar, Gal Lavee, Noam Koenigstein, Oren Barkan, Hilik Berezin, Ulrich Paquet, and Tal Zaccai. Groove radio: A bayesian hierarchical model for personalized playlist generation. In *WSDN*, 2017.

[6] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, 2007.

[7] Austin R Benson, Ravi Kumar, and Andrew Tomkins. Modeling user consumption sequences. In *WWW*. International World Wide Web Conferences Steering Committee, 2016.

[8] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI*. Morgan Kaufmann Publishers Inc., 1998.

[9] Rose Catherine and William Cohen. Transnets: Learning to transform for recommendation. *arXiv preprint arXiv:1704.02298*, 2017.

[10] Allison JB Chaney, David M Blei, and Tina Eliassi-Rad. A probabilistic model for using social networks in personalized item recommendation. In *RecSys*, 2015.

[11] Shuo Chen, Josh L Moore, Douglas Turnbull, and Thorsten Joachims. Playlist prediction via metric embedding. In *KDD*. ACM, 2012.

[12] Yan Chen, F Maxwell Harper, Joseph Konstan, and Sherry Xin Li. Social comparisons and contributions to online communities: A field experiment on movielens. *American Economic Review*, 2010.

[13] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[14] Yi Ding and Xue Li. Time weight collaborative filtering. In *CIKM*. ACM, 2005.

[15] Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. Personalized ranking metric embedding for next new poi recommendation. In *IJCAI*, 2015.

[16] Flavio Figueiredo, Bruno Ribeiro, Jussara M Almeida, and Christos Faloutsos. Tribeflow: Mining & predicting user trajectories. In *WWW*, 2016.

[17] Peixin Gao, Hui Miao, John S Baras, and Jennifer Golbeck. Star: Semiring trust inference for trust-aware social recommenders. In *RecSys*, 2016.

[18] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *TiiS*, 2016.

[19] Ruining He, Wang-Cheng Kang, and Julian McAuley. Translation-based recommendation. In *RecSys*, 2017.

[20] Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *ICDM*. IEEE, 2016.

[21] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *WWW*, 2017.

[22] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

[23] Liangjie Hong, Aziz S Doumith, and Brian D Davison. Co-factorization machines: Modeling user interests and predicting individual decisions in twitter. In *WSDM*, 2013.

[24] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *RecSys*, 2016.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*. ACM, 2008.

[27] Yehuda Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81, 2009.

[28] Yehuda Koren. Collaborative filtering with temporal dynamics. *Communications of the ACM*, 2010.

[29] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 2009.

[30] Dawen Liang, Jaan Altosaar, Laurent Charlin, and David M Blei. Factorization meets the item embedding: Regularizing matrix factorization with item co-occurrence. In *RecSys*. ACM, 2016.

[31] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 2003.

[32] Jian-Guo Liu, Tao Zhou, and Qiang Guo. Information filtering via biased heat conduction. *Physical Review E*, 2011.

[33] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *SIGIR*, 2015.

[34] Brian McFee and Gert R Lanckriet. Metric learning to rank. In *ICML*, 2010.

[35] Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In *NIPS*, 2008.

[36] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *SP*. IEEE, 2008.

[37] Tuan-Anh Nguyen Pham, Xutao Li, and Gao Cong. A general model for out-of-town region recommendation. In *WWW*, 2017.

[38] Martin Piotte and Martin Chabbert. The pragmatic theory solution to the netflix grand prize. *Netflix prize documentation*, 2009.

[39] Matthew Prockup, Andreas F Ehmann, Fabien Gouyon, Erik M Schmidt, and Youngmoo E Kim. Modeling musical rhythm at scale with the music genome project. In *WASPAA*. IEEE, 2015.

[40] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *RecSys*. ACM, 2017.

[41] Steffen Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[42] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *UAI*, 2009.

[43] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *WWW*, 2010.

[44] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and metrics for cold-start recommendations. In *SIGIR*. ACM, 2002.

[45] Vivek Sembium, Rajeev Rastogi, Atul Saroop, and Srujana Merugu. Recommending product sizes to customers. In *RecSys*. ACM, 2017.

[46] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Multi-pointer co-attention networks for recommendation. *arXiv preprint arXiv:1801.09251*, 2018.

[47] Andreas Töscher, Michael Jahrer, and Robert M Bell. The bigchaos solution to the netflix grand prize. *Netflix prize documentation*, 2009.

[48] Théo Trouillon, Christopher R Dance, Éric Gaussier, Johannes Welbl, Sebastian Riedel, and Guillaume Bouchard. Knowledge graph completion via complex tensor factorization. *JMLR*, 2017.

[49] Trinh Xuan Tuan and Tu Minh Phuong. 3d convolutional networks for session-based recommendation with content features. In *RecSys*, 2017.

[50] Farman Ullah, Ghulam Sarwar, Sung Chang Lee, Yun Kyung Park, Kyeong Deok Moon, and Jin Tae Kim. Hybrid recommender system with temporal information. In *ICOIN*. IEEE, 2012.

[51] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *NIPS*, 2013.

[52] Hao Wang, Yanmei Fu, Qinyong Wang, Hongzhi Yin, Changying Du, and Hui Xiong. A location-sentiment-aware recommender system for both home-town and out-of-town users. In *KDD*, 2017.

[53] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. Learning hierarchical representation model for next basket recommendation. In *SIGIR*, 2015.

[54] Shuai Wang, Mianwei Zhou, Geli Fei, Yi Chang, and Bing Liu. Contextual and position-aware factorization machines for sentiment classification. *arXiv preprint arXiv:1801.06172*, 2018.

[55] Suhang Wang, Yilin Wang, Jiliang Tang, Kai Shu, Suhas Ranganath, and Huan Liu. What your images reveal: Exploiting visual contents for point-of-interest recommendation. In *WWW*, 2017.

[56] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. Recurrent recommender networks. In *WSDM*. ACM, 2017.

[57] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *KDD*, 2016.

[58] Yan-Bo Zhou, Ting Lei, and Tao Zhou. A robust ranking algorithm to spamming. *EPL*, 2011.