Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

A Methodological Study of Factorization Machines

Tesis de Licenciatura en Ciencias de la Computación

Sebastián Prillo

Director: Leandro Lombardi

Buenos Aires, 2019

UN ESTUDIO METODOLÓGICO SOBRE MÁQUINAS DE FACTORIZACIÓN

La tarea de un sistema de recomendación es recomendarle a usuarios (tales como un usuario de Netflix, un usuario navegando Amazon, o un usuario escuchando música en Spotify) productos nuevos e interesantes (tales como películas nuevas para mirar, productos para comprar, ó música para escuchar). Con la llegada del e-commerce, los sistemas de recomendación se han vuelto ubicuos y la investigación en sistemas de recomendación ha prosperado.

En este trabajo estudiamos las Máquinas de Factorización (FMs), un modelo de aprendizaje automático que forma parte del estado del arte en problemas de recomendación. Nuestro objetivo es estudiar dos aspectos de las FMs: la *asimetría* de la ecuación del modelo FM, y la viabilidad de parametrizar de forma continua el orden de interacción de las FMs como propuesto en las FMs de orden intermedio. Más en general, en este trabajo tratamos de desarollar una comprensión más profunda sobre el sesgo inductivo de las FMs. Este trabajo se basa en [1] y complementa la propuesta original con experimentación extensiva y análisis.

Con respecto a la asimetría de las FMs, mostramos que esto tiene un impact no trivial sobre las predicciones de las FMs. Por este motivo, mostramos que ensamblar dos FMs, una entrenada sobre los datos originales y otra entrenada luego de negar el valor de la variable respuesta, puede llevar a mejoras inusualmente grandes de performance (cuando se lo compara con la mejora de performance típicamente observada al ensamblar dos modelos de la *misma* clase). También consideramos una formulación simétrica de FMs para la cual la estrategia de ensamblado puede ser vista como una estrategia efectiva de regularización.

Con respecto a parametrizar de forma continua el orden de interacción de FMs, mostramos que esto puede ser efectivo cuando las FMs se usan como predictor general, en algunos casos obteniendo performance comparable a la de predictores generales estado del arte como los *gradient boosted decision trees* (GBDTs). Por otra parte, cuando lo utilizamos en el contexto de problemas de recomendación, obtenemos poca o ninguna mejora.

Palabras claves: Aprendizaje Automático, Sistemas de Recomendación, Máquinas de Factorización, Factorización de Matrices, Kernel ANOVA

A METHODOLOGICAL STUDY OF FACTORIZATION MACHINES

The task of a recommender system is to recommend to users (such as a Netflix user, a user browsing on Amazon, or a user listening to music on Spotify) interesting, novel products (such as new movies to watch, products to purchase, or music to listen to). With the advent of e-commerce, recommender systems have become ubiquitous and research on recommender systems has thrived.

In this work we study Factorization Machines (FMs), a state-of-the-art machine learning model which excels at recommendation problems. Our goal is to study two aspects of FMs: the *asymmetry* of the FM model equation, and the viability of continuously parameterizing the interaction order in FMs as proposed by Intermediate-Order FMs. More generally, in this work we try to develop a deeper understanding about the inductive bias of FMs. This work is based on [1] and complements the original proposal with thorough experimentation and analysis.

Regarding the asymmetry of FMs, we show that it has a non trivial impact on the predictions of FMs. Because of this, we show that ensembling two FMs, one trained on the original data and another trained after reversing the value of the response variable, can lead to unusually large performance improvements (when compared to the performance improvement typically observed by ensembling two models from the *same* model family). We consider a symmetric formulation of FMs for which the ensembling strategy can be seen as an effective regularization technique.

Regarding continuously parameterizing the interaction order in FMs, we show that this can be effective when FMs are used as a general predictor, in some cases nearing the performance of state-of-the-art general predictors such as gradient boosted decision trees. On the other hand, when used in the context of recommendation problems, we obtain little to no improvement.

Keywords: Machine Learning, Recommender Systems, Factorization Machines, Matrix Factorization, ANOVA Kernel

CONTENTS

1. Introduction 1
   1.1 Motivation 1
   1.2 Organization of the Thesis 2

2. Types of Recommender Systems 5
   2.1 Content-Based Recommender Systems 5
   2.2 Collaborative Filtering 7
      2.2.1 User-Based Collaborative Filtering 7
      2.2.2 Item-Based Collaborative Filtering 8
   2.3 Latent-Factor Models 9
      2.3.1 The Matrix Factorization Model 10

3. Learning Pairwise Feature Interactions 15
   3.1 The Degree 2 Polynomial Model 15
   3.2 The Factorization Machine Model 16
   3.3 The Field-Aware Factorization Machine Model 17

4. Asymmetry of Factorization Machines 19
   4.1 Motivation 19
   4.2 Reformulating FMs 19
   4.3 Asymmetry of FMs 20
   4.4 Exploiting or Addressing Asymmetry 23

5. Intermediate-Order Factorization Machines 25
   5.1 Motivation 25
   5.2 Implementation 26

6. Experimental Results 27
   6.1 Small Datasets 28
      6.1.1 phishing dataset 30
      6.1.2 adult dataset 32
      6.1.3 rna dataset 34
      6.1.4 ijcnn dataset 35
      6.1.5 movielens-100K dataset 36
      6.1.6 movielens-100K-all dataset 37
   6.2 Criteo Dataset 39
      6.2.1 Table of Results 39
      6.2.2 Correlation Between Predictions 41
      6.2.3 Varying Symmetry Type and Order 42
   6.3 Avazu Dataset 43
      6.3.1 Table of Results 43
      6.3.2 Correlation Between Predictions 44
      6.3.3 Varying Symmetry Type and Order 45

6.4 Summary 46

7. Conclusion and Future Work 47

8. Appendix 49
8.1 Results with Linear Terms 49
8.1.1 Results on Criteo Dataset with Linear Terms 49
8.1.2 Results on Avazu Dataset with Linear Terms 49
8.2 A Remark on L2-Regularization used for FMs 50

1. INTRODUCTION

1.1 Motivation

If you have ever viewed videos on YouTube, watched movies on Netflix, listened to music on Spotify, purchased products on Amazon, or used Facebook (among so many other examples), then you have already interacted with recommender systems. The task of a recommender system, broadly speaking, is to recommend novel, relevant **items** to **users**. Depending on the context, items could be videos, movies, songs, books, or even other people (as in Facebook friend recommendations).

Paired with the increasing growth of e-commerce in recent years, recommender systems research has thrived and recommender systems have become one of the most successful applications of machine learning. Unlike physical stores which are constrained by physical space, e-commerce websites can host a plethora of different products, giving rise to what is known as the *long-tail* phenomenon: many relevant products are unknown to users. Recommender systems solve the problem of bubbling up these relevant items to users. The following image[1] illustrates the long-tail phenomenon:

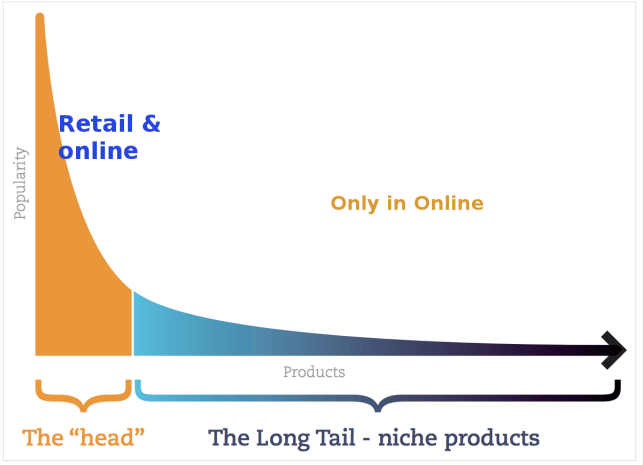


Fig. 1.1: Long-tail phenomenon

Here is a good example of how recommender systems help solve the long-tail problem: In 1988 a book called *Touching the Void* was published but did not receive much attention. A decade later, in 1997, a similar book called *Into Thin Air* was released and people started buying it. Amazon's recommender system noticed that some users who had bought *Into Thin Air* had also bought *Touching the Void*, and so started to recommend *Touching the Void* to users who had bought or considered *Into Thin Air*. In the end, *Touching the Void* became quite popular, even more that *Into Thin Air*. Amazon's recommender system allowed *Touching the Void* reach its intended audience, benefiting both the book and the audience.

[1] Borrowed from http://dataaspirant.com/2015/05/25/collaborative-filtering-recommendation-engine-implementation-in-python/

When it comes down to the end-goal of a recommender system, this might be to increase purchases, user engagement, or the customer base. These are called the *business metrics*. Unfortunately, business metrics are hard to *measure* and *optimize* without running the recommender system in production (known as the *online* setting). For this reason, simpler metrics that can be measured *offline* are used, such as precision and recall of the recommender system when evaluated against past user activity. The hope is that offline metrics will correlate well with online metrics, so that we can now focus on optimizing our offline metrics. Even then, offline metrics are not *differentiable* and so still hard to optimize efficiently. For this reason, differentiable surrogates called *loss functions* are used to train the machine learning models, such as root-mean-squared-error (RMSE) of cross-entropy loss (logloss).

Examples of problems recommender systems directly try to solve are: minimizing the RMSE when predicting the rating that a user would give to a movie, or minimizing the logloss when predicting the probability that a user would click on an ad or sponsored product (called click-through-rate prediction or CTR prediction for short). To make such predictions, recommender systems typically rely on information (called *observed variables*) such as past user activity, the user's age, gender, country, and attributes of the item such as (in the context of movie recommendations) genre, director, and starred actors.

The difficulty of making predictions lies in the fact that the target variable, such as the movie rating in the case of movie recommendations, depends heavily on *combinations* or *interactions* among the observed variables. This is obvious because the rating a user gives to a movie cannot be explained *independently* by just who the user is and what the movie is. For example, a user that tends to give high ratings might rate a highly rated movie low because it does not belong to the genre he is interested in. For this reason, classical machine learning techniques such as linear models fall short at the recommendation problem and specialized techniques have been developed.

In this work we focus on Factorization Machines (FMs), a class of models which excel at learning pairwise feature interactions from data and thus at the recommendation task. We explore two technical aspects of FMs: the *asymmetry* of the FM model equation, and continuously parameterizing the interaction order of FMs. Our goal is to determine if these two aspects can be used to improve the performance of FMs as measured by loss functions. More generally, we try to develop a deeper understanding about the inductive bias of FMs.

1.2 Organization of the Thesis

The thesis is organized as follows:

- Chapter 2 introduces content-based recommender systems, followed by collaborative filtering techniques, and finally latent-factor models, specifically Matrix Factorization (MF).

- Chapter 3 discusses the problem of learning pairwise feature interactions from data, and in this context introduces the degree 2 polynomial model (Poly2), and finally the Factorization Machine (FM) model, as well as the Field-Aware Factorization Machine (FFM) model.

- Chapter 4 introduces and discusses the asymmetry of the FM model equation from

a theoretical point of view. We trace this back to the inductive bias of FMs, which we try to characterize in more detail.

- Chapter 5 discusses how the interaction order of FMs could be continuously parameterized, deriving in what we call *Intermediate-Order Factorization Machines*.
- Chapter 6 contains the experimental results where we assess the impact of asymmetry on the performance of FMs, and the viability of Intermediate-Order FMs. We analyze these aspects in the context of both recommendation and non-recommendation datasets.
- Chapter 7 presents the conclusions of our work, and discusses several areas that could be explored in the future.

2. TYPES OF RECOMMENDER SYSTEMS

There are several approaches to try to solve the recommendation problem. Broadly speaking, recommender systems models can be categorized into three main groups: content-based models, collaborative filtering models, and latent-factor models. In this chapter we discuss them in turn.

2.1 Content-Based Recommender Systems

The simplest kind of recommender system builds a user profile based on the items consumed by the user, and then uses this profile to generate recommendations. For example, in the movie recommendation domain, the profile of a user might indicate how much that user likes action, romance, comedy, and so on, in a movie. With these user profiles, the recommender systems can recommend to users movies that match their interests.

Formally, for each item i, let $v_i \in \mathbb{R}^k$ be a vector quantifying the characteristics of the item. In the case of movies, the first entry of v_i might contain a 1 if the movie is an action movie and 0 if not, the second entry of v_i might contain a 1 if the movie is a romance movie and 0 if not, and so on. Continuous values between 0 and 1 could be used instead to specify the degree to which the movie meets the criteria.

Regardless of the choice of the item representation, based on these item representations we can build a user profile $v_u \in \mathbb{R}^k$ (also called a *user representation*) by averaging the representations of the items she liked. To recommend new items, we find those items whose representation is most similar to the user's representation, for example as measured by cosine similarity, which is the angle between the user and the item representations:

$$sim(v_u, v_i) = \frac{v_u \cdot v_i}{||v_u||_2||v_i||_2}$$

As a concrete example, suppose we are trying to recommend movies to users, and that we have the following *ratings matrix* at our disposal:

	Star Trek	Star Wars	Titanic	Notting Hill	Alien
Alice		1	4		1
Bob	5	4		3	
Charlie		5		3	4
Daniel	1		5	4	

Fig. 2.1: Observed ratings matrix. Unknown entries are highlighted in yellow.

Furthermore, we know that the content of the movies is as follows:

	Sci-Fi	Action	Romance	Comedy
Star Trek	1	1	0	0
Star Wars	1	1	0	0
Titanic	0	0.5	1	0
Notting Hill	0	0	1	1
Alien	1	0.5	0	0

Fig. 2.2: Movie content matrix

Then, if we define a user to like a movie if his rating is at least 4, then the user representations would be the following:

	Sci-Fi	Action	Romance	Comedy
Alice	0	0.5	1	0
Bob	1	1	0	0
Charlie	1	0.75	0	0
Daniel	0	0.25	1	0.5

Fig. 2.3: User representations

For example, to find the representation of Daniel we must average the representations of Titanic and Notting Hill, which are $(0, 0.5, 1, 0)$ and $(0, 0, 1, 1)$ respectively (we do not average Star Trek because he did not like it according to our arbitrary cutoff score of 4). This way we obtain $(0, 0.25, 1, 0.5)$.

Now, to make recommendations for all users, we first compute the cosine similarity between all user and item representations:

	Star Trek	Star Wars	Titanic	Notting Hill	Alien
Alice	0.32	0.32	1.00	0.63	0.20
Bob	1.00	1.00	0.32	0.00	0.95
Charlie	0.99	0.99	0.27	0.00	0.98
Daniel	0.15	0.15	0.88	0.93	0.10

Fig. 2.4: Cosine similarity matrix between user and item representations

For example, the cosine similarity between Alice and Star Trek is:

$$sim(\text{Alice}, \text{Star Trek}) = \frac{\langle (0, 0.5, 1, 0), (1, 1, 0, 0) \rangle}{||(0, 0.5, 1, 0)||_2 ||(1, 1, 0, 0)||_2} = \frac{0.5}{\sqrt{1.25}\sqrt{2}} \cong 0.32$$

With these similarities, we can now recommend Notting Hill to Alice, Alien to Bob, Star Trek to Charlie, and nothing (because the similarities are too low!) to Daniel.

The main problem with content-based recommendation is that the representation of each item has to be hand-crafted and there is no principled way to get this right. Also, while the content-based approach could be viable for movie recommendations, it is totally unusable once the set of items is heterogeneous. For example, Amazon sells all kinds of

different items, and knowing that a user bought a Harry Potter movie is a strong indicator that he might enjoy reading the book. But movies and books are from different domains, and comparing item content across domains is a daunting task.

Finally, item content can be noisy or missing completely. For example, if users upload video content to YouTube, they might not specify an accurate title or video description. All these issues make content-based recommendation only viable when content and similarity can be easily and robustly defined. Even then, as soon as data gets large, approaches based on collaborative filtering outperform content-based approaches by a large margin.

2.2 Collaborative Filtering

Collaborative filtering (CF) solves the problem of making product recommendations without the need of *any* item content information. The key insight behind collaborative filtering is that if two users have a similar opinion about an item, then they are more likely to share a similar opinion about another item than two randomly chosen users. For example, if user U likes products A and B, and user V likes product A, we could recommend product B to user V, even if we do not know anything about the content of items A or B. The point is that even though we do not know this, the fact that user U liked both A and B, *implicitly* tells us that they have something in common. For example, products A and B might both be from the Harry Potter franchise. This allows collaborative filtering approaches to be applicable to homogeneous sets of items. For example, we can now easily recommend the Harry Potter books to users who have bought the Harry Potter movies.

2.2.1 User-Based Collaborative Filtering

User-based CF makes recommendations by finding *users similar to a given user.*

Let us start by introducing some definitions; later we will give a concrete example. Let I_u be the set of items user u has rated, $r_{u,i}$ be the rating that user u has given to item i, and $\overline{r_u}$ be the average rating of user u. We can then define the similarity of two users u_1 and u_2 as the centered cosine similarity between their ratings:

$$sim(u_1, u_2) = \frac{\sum_{i \in I_{u_1} \cap I_{u_2}} (r_{u_1,i} - \overline{r_{u_1}})(r_{u_2,i} - \overline{r_{u_2}})}{\sqrt{\sum_{i \in I_{u_1}} (r_{u_1,i} - \overline{r_{u_1}})^2} \sqrt{\sum_{i \in I_{u_2}} (r_{u_2,i} - \overline{r_{u_2}})^2}}$$

This is the same as computing the Pearson correlation between the ratings of users u_1 and u_2 after their ratings have been centered and missing ratings treated as 0.

Now, to predict the rating that user u would give to item i, we can consider:

$$\hat{r}_{u,i} = \frac{1}{k} \sum_{v \in U} r_{v,i}$$

where U is the set of k most similar users to u which have rated item i. More sophisticated approaches such as a weighted sum are possible:

$$\hat{r}_{u,i} = \frac{\sum_{v \in U} sim(u, v) r_{v,i}}{\sum_{v \in U} sim(u, v)}$$

Let us go back to our movie ratings example. How would we predict the rating that Bob gives to Alien? In this small-scale example, let us use user-based CF with $k = 1$, so

that we are looking for the most similar user to Bob which has rated Alien. Since Alice and Charlie have both rated Alien, we must determine which of the two is most similar to Bob. The mean user ratings are $\overline{r_{\text{Alice}}} = 2, \overline{r_{\text{Bob}}} = 4, \overline{r_{\text{Charlie}}} = 4$ and $\overline{r_{\text{Daniel}}} = 3.33$. After subtracting these mean ratings from each user, we obtain the following *user-centered ratings matrix*:

	Star Trek	Star Wars	Titanic	Notting Hill	Alien
Alice		-1	2		-1
Bob	1	0		-1	
Charlie		1		-1	0
Daniel	-2.33		1.67	0.67	

Fig. 2.5: (User-) Centered ratings matrix. Unknown entries are highlighted in yellow.

Now we can compute the centered cosine similarity between Bob and Alice as:

$$sim(\text{Bob}, \text{Alice}) = \frac{\langle (1,0,0,-1,0), (0,-1,2,0,-1) \rangle}{||(1,0,0,-1,0)||_2 ||(0,-1,2,0,-1)||_2} = \frac{0}{\sqrt{2}\sqrt{6}} = 0$$

And similarly between Bob and Charlie as:

$$sim(\text{Bob}, \text{Charlie}) = \frac{\langle (1,0,0,-1,0), (0,1,0,-1,0) \rangle}{||(1,0,0,-1,0)||_2 ||(0,1,0,-1,0)||_2} = \frac{1}{\sqrt{2}\sqrt{2}} = 0.5$$

This way, Charlie is the most similar user to Bob which has rated Alien, and hence we can now estimate the rating of Bob for Alien using the rating of Charlie: a 4.

The problem with user-based collaborative filtering is that when there are many users in the system, finding the users similar to a given user is time-consuming. Therefore, user-based CF systems do not scale well. The first recommender systems were of this kind, but where soon replaced by item-based CF upon their inception.

2.2.2 Item-Based Collaborative Filtering

Instead of trying to find users similar to a given user, item-based collaborative filtering tries to find *items similar to a given item.*

Once again we start with some definitions and follow with a concrete example. Let U_i be the set of users that have rated item i and let $\overline{r_i}$ be the mean rating for item i. Then, we can define similarity between items in the same way as we did for users, with the centered cosine:

$$sim(i_1, i_2) = \frac{\sum_{u \in U_{i_1} \cap U_{i_2}} (r_{u,i_1} - \overline{r_{i_1}})(r_{u,i_2} - \overline{r_{i_2}})}{\sqrt{\sum_{u \in U_{i_1}} (r_{u,i_1} - \overline{r_{i_1}})^2} \sqrt{\sum_{u \in U_{i_2}} (r_{u,i_2} - \overline{r_{i_2}})^2}}$$

To predict the rating that user u would give to item i, let I be the set of k most similar items to i which user u has rated. Then, we can predict with the sum:

$$\hat{r}_{u,i} = \frac{1}{k} \sum_{j \in I} r_{u,j}$$

In our example, let us determine the rating that Bob would give to Alien, but this time using item-based CF. Again, let us choose $k = 1$, so that we are looking for the movie most similar to Alien which Bob has rated. The candidates are Star Trek, Star Wars, and Notting Hill. We start by centering the item ratings, subtracting $\overline{r_{\text{Star Trek}}} = 3, \overline{r_{\text{Star Wars}}} = 3.33, \overline{r_{\text{Titanic}}} = 4.5, \overline{r_{\text{Notting Hill}}} = 3.33$ and $\overline{r_{\text{Alien}}} = 2.5$ from each item column respectively:

	Star Trek	Star Wars	Titanic	Notting Hill	Alien
Alice		-2.33	-0.5		-1.5
Bob	2	0.67		-0.33	
Charlie		1.67		-0.33	1.5
Daniel	-2		0.5	0.67	

Fig. 2.6: (Item-) Centered ratings matrix. Unknown entries are highlighted in yellow.

We can now compute the centered cosine similarities as follows:

$$sim(\text{Alien}, \text{Star Trek}) = \frac{\langle(-1.5, 0, 1.5, 0), (0, 2, 0, 2)\rangle}{||(-1.5, 0, 1.5, 0)||_2||(0, 2, 0, 2)||_2} = 0$$

$$sim(\text{Alien}, \text{Star Wars}) = \frac{\langle(-1.5, 0, 1.5, 0), (-2.33, 0.67, 1.67, 0)\rangle}{||(-1.5, 0, 1.5, 0)||_2||(-2.33, 0.67, 1.67, 0)||_2} = 0.96$$

$$sim(\text{Alien}, \text{Notting Hill}) = \frac{\langle(-1.5, 0, 1.5, 0), (0, -0.33, -0.33, 0.67)\rangle}{||(-1.5, 0, 1.5, 0)||_2||(0, -0.33, -0.33, 0.67)||_2} = -0.29$$

Therefore, we estimate Bob's rating for Alien by using his rating on Star Wars: a 4.

Since items are typically much less numbered than users, item-based CF scales better to more users. Also, using similarity across items tends to empirically give better results, possibly because it is more meaningful to compare items to items than to compare users to users. Indeed, items are a lot simpler to characterize than users. For example, a movie can typically be categorized into one or a few genres, whereas a user might like several, unrelated genres, leading to more diversity among users than among items.

Item-based CF was introduced by Amazon in 2003 [2] and represented one of the major breakthroughs in recommender systems. They still enjoy widespread use, which earned Amazon an IEEE test-of-time award in 2017 [3] for their original work. The simplicity and strong performance of item-based CF makes the method very attractive, even when faced against latent-factor models, which we discuss next. For example, the YouTube recommender system used item-based CF methods until as late as 2010 [4] before moving on to latent-factor models [5].

2.3 Latent-Factor Models

The development of latent-factor models was propelled by the Netflix Prize competition, launched in October 2006. The Netflix Prize challenged teams to predict the ratings of users on movies. The available training data consisted of 100M ratings of 480K users on 18K movies. Ratings were on a 1 to 5 scale, and the goal was to minimize the RMSE on 3M withheld test ratings dated posterior to the training data. The winning team would be the first to improve the RMSE of Netflix's recommender system by 10%. The competition ended in September 2009, when the 10% improvement was achieved.

The best approaches to the Netflix Prize competition relied on what are known as *latent-factor* models. These can be thought of as machine-learnt versions of content-based models, and are actually a more involved form of CF (however, since they work in a very different way from the traditional CF-based approaches we discussed earlier, we keep them out of the CF taxonomy).

In latent-factor models we pose (in the simplest setting) that each item and user can be represented by k latent (that is to say, unobserved) *factors* or *dimensions*, and that the preference of a user for an item is given by a function of the latent vector of the user and the item, such as the similarity between the user and item latent vectors. These latent dimensions could represent the action, romance, etc. contents of a movie, and how much a user likes action, romance, etc. respectively. This is very similar to the content-based methods. However, unlike content-based methods, where the dimensions are explicitly specified, in latent-factor models the dimensions are *automatically learnt from data*. We now discuss the Matrix Factorization model, the first latent-factor model introduced for recommendation.

2.3.1 The Matrix Factorization Model

The *Matrix Factorization* (MF) model was introduced during the Netflix Prize competition, motivated by the SVD matrix decomposition[1]. The original Netflix recommender system relied on item-based CF [6]. MF outperformed Netflix's item-based CF approach by around 4%; the following figure from [6] shows this, as well as the effectiveness of different methods on the Netflix Prize competition (note that SVD corresponds to MF):

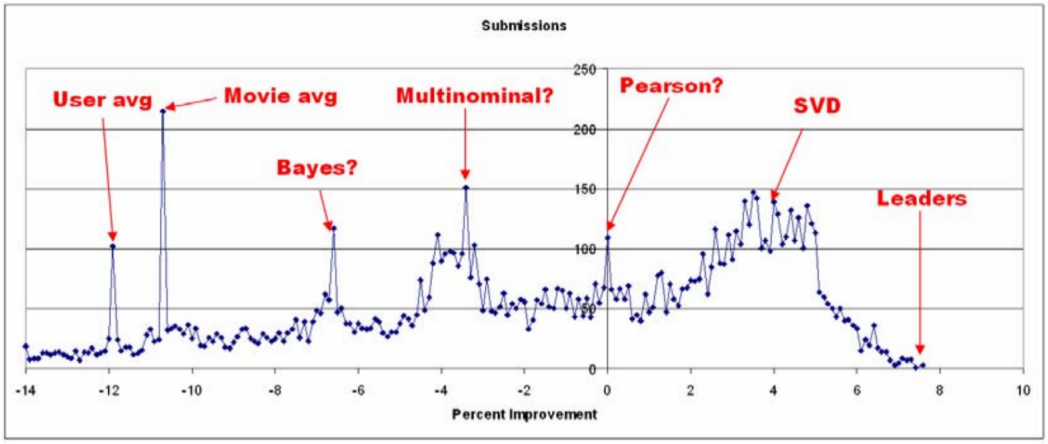


Fig. 2.7: Detail of distribution of leading submissions indicating possible techniques

In MF the central object of study is the partially observed matrix M of user-item interactions. Each row of M corresponds to a user and each column of M corresponds to an item. The interaction of user i with item j correspond to the entry (i, j) of this matrix. For example, in the context of movie recommendations, entry (i, j) would contain the rating of user i on movie j, or be empty if it is unknown. This is the same as the ratings matrix in our toy example 2.1. This way, the problem of recommendation can be

[1] https://sifter.org/simon/journal/20061211.html

framed as a matrix completion problem: given the partially observed user-item interaction matrix, we must infer the unobserved entries. If we can do this effectively, then we can use the predictions of our model on the unobserved entries to generate recommendations.

It is in general impossible to predict accurately (by which we mean better than randomly) the unobserved entries of an arbitrary matrix. However, the user-item interaction matrix is far from random. The existence of such structure is suggested by empirical evidence that *users whose opinions agree on a set of items usually agree on the remaining set of items*, which we already discussed as the basis for collaborative-filtering methods; this is the reason why latent-factor models are a form of collaborative filtering.

A successful approach to modeling the user-item interaction matrix is with the *matrix factorization model*: Let $M \in \mathbb{R}^{m \times n}$ be the matrix of user-item interactions. Then, we pose that M can be well approximated by a low rank matrix. That is, we pose that $M \cong UV$ for some $U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{k \times n}$, where k is significantly smaller than m and n.

Why does this make sense? Recall content-based methods, where each user and item had representations v_u and v_i respectively, and we ranked recommendations using the cosine similarity between user and item representations. If instead of using cosine similarity (which results in a value between -1 and 1) we take the inner product $\langle v_u, v_i \rangle$, and pose that this is exactly the rating that user u gives to item i, then we obtain the MF model equation. Indeed, if we let U be the matrix of user representations and V be the matrix of movie representations, what we are stating is precisely that $M = UV^t$.

MF can be seen as a machine-learnt version of content-based methods. In both models each item has its set of attributes, each user is represented by how much he likes each attribute, and ratings are the result of the compatibility between the representations of the item and the user. However, in MF we do not specify *what* the dimensions of the latent factors are. We just specify *how* they are related to the rating, and statistical methods are used to learn the dimensions and the representations of the users and items by trying to match the observed ratings. These dimensions need not make intuitive sense at all. In fact, note that $UV = (UW^{-1})(VW^t)^t$ for any invertible $W \in \mathbb{R}^{k \times k}$, so that if U is the matrix of user representations, then UW^{-1} could also be the matrix of user representations, and similarly for items.

Back to our previous example where *Bob* (B) has rated *Star Wars* (SW) a 4, *Star Trek* (ST) a 5, and *Charlie* (C) has rated *Star Wars* a 5, let us see what MF would predict about the rating of Charlie for Star Trek.

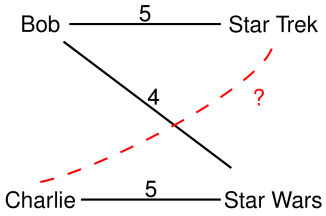


Fig. 2.8: Rating prediction example

There are four latent vectors involved: v_B, v_C, v_{SW}, v_{ST}. Since *Bob* rated *Star Trek* highly, then $\langle v_B, v_{ST} \rangle$ must be large, which encourages v_B and v_{ST} to be close in embedding space. Similarly v_B and v_{SW} must be close, and finally also v_C and v_{SW}. But then, by

transitivity, v_C are v_{ST} are likely to be close too, and so $\langle v_C, v_{ST} \rangle$ must be large. This way MF predicts a high rating for *Charlie* on *Star Trek*.

In general, MF embeds all users and items into the same k-dimensional space $\mathbb{R}^k$, trying to put users close to movies they rated highly and far from movies they disliked. In our example, a 2-dimensional embedding space might (roughly) look like this:

Fig. 2.9: Embedding users and items together with Matrix Factorization

To learn the low rank representation of M when M contains continuous values (such as movie ratings), we pose the learning problem as minimizing the (root mean) squared error between the observed entries of the matrix and the predicted values (plus a regularization term to avoid overfitting):

$$\arg\min_{U,V} \lambda(||U||_2^2 + ||V||_2^2) + \sum_{(i,j)\in I} (u_i v_j - m_{i,j})^2$$

Here I is the set of observed entries of the matrix M and $\lambda \geq 0$ is a regularization hyperparameter. Stochastic gradient descent (SGD) can be used to find a local minimum, as well as alternating least squares since the objective is separately convex in U and V. When the matrix M is binary, say with values in $\{-1, 1\}$ (for example in a CTR-prediction task), we can use the logarithmic loss instead:

$$\arg\min_{U,V} \lambda(||U||_2^2 + ||V||_2^2) + \sum_{(i,j)\in I} \log(1 + \exp(-m_{i,j} u_i v_j))$$

The choice of this loss arises from posing the problem as maximum likelihood estimation under the probabilistic model given by

$$u_i, v_j \sim \mathcal{N}(0, \lambda^{-1})$$

$$m_{i,j} | u_i, v_j \sim Be(\sigma(u_i v_j))$$

where $\sigma(t) = \frac{1}{1+e^{-t}}$ is the logistic function. The general probabilistic framework gives rise to probabilistic matrix factorization [7] and point estimates can be replaced by actual (intractable) probabilistic inference, which typically outperforms maximum likelihood methods, at the expense of computational time.

Other loss functions such as Bayesian Personalized Ranking (BPR) Loss [8] can be used to train the model, and the choice of loss function actually has a big impact on the model's generalization performance and on online metrics, but this dimension is orthogonal to our work so we do not discuss it.

The matrix factorization model can also be enriched with user and item biases b_i^{user} and b_j^{item} respectively, as well as a global bias b. In this case, we would predict entry (i, j) of the ratings matrix as $b + b_i^{user} + b_j^{item} + u_i v_j$.

The main drawback of MF is that it cannot deal with data beyond user and item. For example, what if we wanted to leverage other data like the gender of the user? In other words, MF is not a *general predictor*. Research in this area of recommender systems has led to the proposal of several variants of MF which can take into account side information, the most successful and encompassing of which are *Factorization Machines*, and which will be the center of our discussion.

3. LEARNING PAIRWISE FEATURE INTERACTIONS

In this chapter we discuss models that are able to capture pairwise feature interactions in data beyond user and item, overcoming the limitations of MF. We start by introducing the degree 2 polynomial model (Poly2), and then introduce Factorization Machines (FMs) and their variant Field-Aware Factorization Machines (FFMs).

3.1 The Degree 2 Polynomial Model

The simplest way to model pairwise feature interactions in data is to consider a degree 2 polynomial mapping. If x is the vector of explanatory features, then:

$$\hat{y}_{Poly2}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} x_i x_j \tag{3.1}$$

where $w_i, w_{ij} \in \mathbb{R}$. This model is able to capture *any* set of pairwise feature interactions because it has one weight w_{ij} per feature pair. In fact, the model is so expressive that it is hardly useful in practice. Indeed, imagine a recommendation scenario where we have a million users and a million items. In this case, we would need on the order of 10^{12} parameters for such a model, which is inviable. Also, we would not be able to learn anything about interactions we have not observed in the data: In our example, imagine trying to predict the rating of Charlie on Star Trek. In that case, the weight $w_{Charlie,Star\ Trek}$ will have no training signal because the product $x_{Charlie} x_{Star\ Trek}$ is always 0, and so the prediction of the model is meaningless. In the best case we have a prior on $w_{Charlie,Star\ Trek}$ which drives it to 0 during training and then we predict a 'mean-type' rating of $w_0 + w_{Charlie} + w_{Star\ Trek}$.

One possible way to bound the expressiveness of the Poly2 model is by using what is known as the *hashing trick*. Consider instead the mapping:

$$\hat{y}_{Poly2}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{h(i,j)} x_i x_j \tag{3.2}$$

where h is a hash function, for example $h(i,j) = (\frac{1}{2}(i+j)(i+j+1)+j) \bmod B$ for some positive integer B. This effectively cuts the number of pairwise interaction weights from $\binom{n}{2}$ to B, tying together the weights of all feature pairs which happen to get hashed into the same bucket.

This model, however, is still unable to learn anything meaningful about unobserved interactions. Back to our example, what would we predict about the rating Charlie gives to Star Trek? In this case, the interaction weight for Charlie and Star Trek is $w_{h(Charlie,Star\ Trek)}$, which has been learned from all other feature pairs whose hash collided with that of $(Charlie, Star\ Trek)$. But the rating Alice gives to Titanic is certainly not immediately useful for predicting the rating Charlie gives to Star Trek, so even though we have learned something about $w_{h(Charlie,Star\ Trek)}$, it is meaningless for predicting the value of unobserved interactions.

Factorization Machines take the best from both worlds: like the Poly2 model, they are able to model pairwise feature interactions across any set of variables. However, they also

leverage the inductive bias of MF, which allows them to learn meaningful interactions for pairs of variables that have never been observed to co-occur in the data.

3.2 The Factorization Machine Model

The Factorization Machine (FM) [9] model generalizes MF by embedding *every* feature into $\mathbb{R}^k$, not just user and item. This way, if we are trying to predict the rating that user u of gender g would give to movie i, we embed user u with embedding v_u, item i with embedding v_i, and *also* gender g with embedding v_g. FMs finally pose that the rating is given by:

$$\langle v_u, v_i \rangle + \langle v_u, v_g \rangle + \langle v_i, v_g \rangle$$

This generalizes MF because setting $v_g = 0$ for all g falls back to MF.

In general, given a vector of explanatory features $x \in \mathbb{R}^n$ (where categorical variables such as user, item and gender have been one-hot encoded), the FM model equation (without linear terms) is given by:

$$\hat{y}_{FM}(x) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle \, x_i x_j \tag{3.3}$$

where $V \in \mathbb{R}^{n \times k}$ are parameters. The term $x_i x_j$ is used to scale the inner product $\langle v_i, v_j \rangle$ by x_i and x_j, allowing us to use numeric features. k plays the role of the latent representation dimension in MF. Let us formally show (based on equation 3.3) that FMs generalize MF: For a MF model with parameters $U_{MF} \in \mathbb{R}^{m \times k}$ and $V_{MF} \in \mathbb{R}^{k \times n}$, let us represent the feature vector x for FMs as the concatenation of the one-hot encoding of the user and the item. Then $x \in \mathbb{R}^{m+n}$ has exactly two coordinates equal to 1, one within the first m positions and one within the last n. We let $V_{FM} \in \mathbb{R}^{(m+n) \times k}$ be the concatenation of U_{MF} and V_{MF}^t along the first dimension. Then this FM reproduces MF exactly.

The following example from the paper on FMs [9] shows how FMs could extend the MF model with more features beyond user and item:

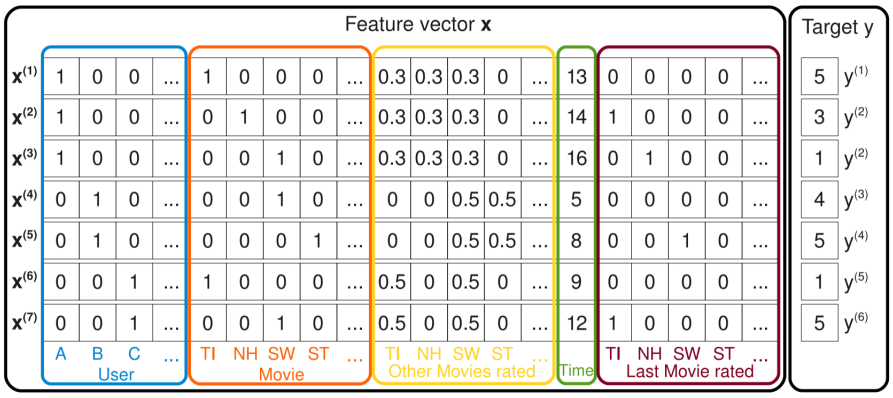


Fig. 3.1: Feature engineering example for Factorization Machines

Just like with MF, we can enrich FMs with linear terms $w_i \in \mathbb{R}$ to obtain:

$$\hat{y}_{FM}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle \, x_i x_j \tag{3.4}$$

This is the original formulation of FMs as in [9].

Note the stark similarity between the FM model equation (3.4) and the Poly2 model equation (3.1): the FM model can be thought of as a polynomial model of degree 2 where the pairwise interaction weights w_{ij} have been factored out as the inner product of two latent vectors v_i and v_j. In other words, if $W = (w_{ij})$ is the pairwise interaction weight matrix, then FMs pose that $W = VV^t$. This allows FMs to model pairwise feature interactions with nk parameters rather than $\binom{n}{2}$, yet unlike the Poly2 model with hashed feature interactions, it is able to infer unobserved interactions (such as user-item interactions) by leveraging the inductive bias of the MF model.

As shown in [9], FMs also generalize a number of previous attempts to extend MF with metadata, such as SVD++ [10], PITF [11], and FPMC [12].

3.3 The Field-Aware Factorization Machine Model

A successful variant of FMs, called *Field-Aware Factorization Machines* (FFM) [13], considers several small embedding spaces instead of a large unique one.

Let the *features* $x_1, x_2, \ldots, x_n \in \mathbb{R}$ in the dataset be grouped into *fields*. For example, in the context of predicting movie ratings, let U and I be the set of users and items respectively. Let $x_1, x_2, \ldots, x_{|U|}$ constitute the one-hot representation of the user (so that $x_u \in \{0, 1\}$ indicates the presence of user u), and let $x_{|U|+1}, \ldots, x_{|U|+|I|}$ constitute the one-hot representation of the item. Then there are n features but only two fields: *user* and *item*; $x_1, x_2, \ldots, x_{|U|}$ are features corresponding to the *user* field, and $x_{|U|+1}, \ldots, x_{|U|+|I|}$ are features corresponding to the *item* field.

To each unordered pair of possibly repeated fields we associate an embedding space, obtaining $\binom{f+1}{2}$ embedding spaces. Each feature has its own embedding in each of the f embedding spaces corresponding to its field. This results in a total of fk parameters per feature. Note that this means that the value of k in FFMs is typically much smaller than the value of k in FMs: to achieve the same number of parameters, we must have $k^{FFM} = \frac{k^{FM}}{f}$.

To compute the FFM model equation, we add up all pairs of similarities between features present in the dataset, where the similarity between features i and j is defined as the inner product of their embeddings in the embedding space corresponding to their two fields. When there are only two fields, such as in the user-item case, only one embedding space is used (interactions between two users or two movies are never considered), and so FFMs are equivalent to FMs. Similarly, when all features are grouped under the same field, there is only one embedding space and FFMs are equal to FMs. However, when there are more than two fields, FFMs and FMs start to differ.

With the same example as before, where we had a user u, item i, and user's gender g, the FFM model equation would be:

$$\left\langle v_u^{\text{user-item}}, v_i^{\text{user-item}} \right\rangle + \left\langle v_u^{\text{user-gender}}, v_g^{\text{user-gender}} \right\rangle + \left\langle v_i^{\text{item-gender}}, v_g^{\text{item-gender}} \right\rangle$$

In general, rearranging notation slightly and scaling by $x_i x_j$ to allow for numeric features, the FFM model equation is given by:

$$\hat{y}_{FFM}(x) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left\langle v_{i,f(j)}, v_{j,f(i)} \right\rangle x_i x_j \tag{3.5}$$

where the $v_{i,f(j)} \in \mathbb{R}^k$ and $f(i)$ and $f(j)$ are the respective fields of features i and j. Here the intuition is that $v_{i,f(j)}$ stands for the embedding of feature i in the embedding space corresponding to i's field and j's field. We can extend the FFM model equation with linear terms to obtain:

$$\hat{y}_{FFM}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left\langle v_{i,f(j)}, v_{j,f(i)} \right\rangle x_i x_j \tag{3.6}$$

FFMs have been shown to outperform FMs on a number of recommendation datasets: they have been the key model in the winning team's approach in the Avazu CTR Prediction competition[1] and in the Criteo Display Advertising Challenge[2], both in Kaggle. FFMs will serve as our state-of-the-art to compare against when we experiment with our variants of FMs.

[1] https://www.kaggle.com/c/avazu-ctr-prediction
[2] https://www.kaggle.com/c/criteo-display-ad-challenge

4. ASYMMETRY OF FACTORIZATION MACHINES

4.1 Motivation

In this work we explore an interesting phenomenon underlying the FM model equation, which is that it is not symmetric with respect to the response variable. To state this precisely, let

$$FM_k^+ = \{\hat{y} : \mathbb{R}^n \to \mathbb{R} \mid \exists w \in \mathbb{R}^{n+1}, V \in \mathbb{R}^{n\times k} : \hat{y}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n}\sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j\} \tag{4.1}$$

be the FM hypothesis space, and let:

$$FM_k^- = \{-\hat{y} : \hat{y} \in FM_k^+\} \tag{4.2}$$

be the result of negating each hypothesis. Then, as we will see, it is in general not true that $FM_k^+ = FM_k^-$. In particular, this means that something as simple as changing the label of the response variable in a binary classification task might affect the performance of FMs. For example, when training a FM classifier to distinguish cats from dogs, labeling cat as 0 and dog as 1 *makes a difference* compared to labeling cat as 1 and dog as 0. Similarly, in a regression setting, multiplying the target variable by -1 might inherently affect the way that FMs learn and make predictions.

This might come as a bit of a surprise because we do not expect the ability of FMs to learn pairwise feature interactions to be related in any way to the encoding or sign of the response variable. There is a gap between how we expect the model to behave, and how it behaves in practice. In the following we explain why this asymmetry arises and relate it to the inductive bias of FMs, which we try to shed some light on.

4.2 Reformulating FMs

Recall that the FM model equation is given by:

$$\hat{y}_{FM}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n}\sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j \tag{4.3}$$

where v_i are k-dimensional vectors. Note that naive computation would require $\mathcal{O}(n^2 k)$ time. Compare this against the $\mathcal{O}(n)$ computational cost of linear models and the fact that in typical recommender system applications n is in the millions, and it looks like FMs are too expensive for any practical purpose. However, the FM model equation can be rewritten as follows, as shown in [9]:

$$\hat{y}_{FM}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2}\sum_{j=1}^{k}\Big(\Big(\sum_{i=1}^{n} v_{i,j} x_i\Big)^2 - \sum_{i=1}^{n} v_{i,j}^2 x_i^2\Big) \tag{4.4}$$

This, instead, can easily be computed in linear time $\mathcal{O}(nk)$. In fact, this can be computed in time $\mathcal{O}(\tilde{n}k)$ where $\tilde{n}$ is the number of non-zero entries in x (and which is typically equal

to f, the number of fields of the dataset). Similarly, each gradient $\frac{\partial}{\partial v_{i,j}}\hat{y}_{FM}(x)$ can be computed in time $\mathcal{O}(\tilde{n})$:

$$\frac{\partial}{\partial v_{i,j}}\hat{y}_{FM}(x) = x_i\Big(\sum_{i=1}^{n} v_{i,j}x_i\Big) - v_{i,j}x_i^2 \tag{4.5}$$

Moreover, since for a fixed j the sum $\sum_{i=1}^{n} v_{i,j}x_i$ is shared across all gradients $\frac{\partial}{\partial v_{i,j}}\hat{y}_{FM}$ for $1 \leq j \leq n$, then computation can be arranged to compute *all* gradients in linear time $\mathcal{O}(\tilde{n}k)$. Hence both evaluation and gradient computation of FMs (for a single training example) can be carried out in time $\mathcal{O}(\tilde{n}k)$. The following pseudocode shows how we would train FMs this way:

Algorithm 1 SGD training of FMs under logarithmic loss (without regularization)

Initialize $w_i \in \mathbb{R}, v_i \in \mathbb{R}^k$ randomly, and let η be the learning rate (which in general can be adaptive and per-coordinate)
Repeat until convergence:
for $(x, y) \in$ Training Set **do**
    $\text{loss} = \log(1 + \exp(-y\hat{y}(x)))$
    $\frac{d\text{loss}}{d\hat{y}} = \frac{-y}{1+\exp(y\hat{y}(x))}$
    $w_0 \leftarrow w_0 - \eta \, \frac{d\text{loss}}{d\hat{y}}$
    for $i \in \{1, 2, \ldots, n\}$ such that $x_i \neq 0$ **do**
        $w_i \leftarrow w_i - \eta \, \frac{d\text{loss}}{d\hat{y}} \, x_i$
    end for
    for $j \in \{1, 2, \ldots, k\}$ **do**
        $s = \sum_{i=1}^{n} v_{i,j}x_i$
        for $i \in \{1, 2, \ldots, n\}$ such that $x_i \neq 0$ **do**
            $v_{i,j} \leftarrow v_{i,j} - \eta \, \frac{d\text{loss}}{d\hat{y}} \, x_i(s - v_{i,j}x_i)$
        end for
    end for
end for

4.3 Asymmetry of FMs

The reformulation of FMs is interesting in its own right, but not only because it allows efficient learning of FMs, but because it also reveals the asymmetric nature of the FM model equation. Indeed, the pairwise interaction terms got regrouped as a *sum of k squared hyperplanes* (modulo the self-interaction terms $\sum_{i=1}^{n} v_{i,j}^2 x_i^2$). These squared hyperplanes are all convex and so impose a convex shape to the FM model equation. One (brutal) way to think about this is that FMs are to a linear model as the function $f(x) = x^2$ is to $f(x) = x$. In fact, if we omit the linear term $w_0 + \sum_{i=1}^{n} w_i x_i$ from the FM model equation and drop the self-interaction terms $\sum_{i=1}^{n} v_{i,j}^2 x_i^2$ in the inner sum, then FMs are simply a sum of squared hyperplanes and cannot make negative predictions at all:

$$\hat{y}_{FM}(x) = \frac{1}{2}\sum_{j=1}^{k}\Big(\sum_{i=1}^{n} v_{i,j}x_i\Big)^2 \tag{4.6}$$

This perspective, where we reason about FMs in terms of sums of squared hyperplanes as in reformulation 4.4 rather than as a polynomial model of degree 2 with factored interaction

weights as in 4.3, is what is referred to as the *elementary view on FMs* in [1]. It is meant to be elementary in the sense that much of the original intuition based on inner products is dropped in favour of what is arguably an easier way to reason about the actual shape of the FM model equation. Indeed, the asymmetry of the FM model equation has now become apparent.

The natural question now is: *how does this asymmetry relate to the inductive bias of Factorization Machines? How are FMs learning pairwise feature interactions?* The next is an good example to reason about this.

Consider the following dataset, which portrays the impact of several factors on life expectancy:

smokes	sedentary life	unhealthy diet	life expectancy
0	0	0	10
1	0	0	8
0	1	0	8
0	0	1	8
1	1	0	2
1	0	1	2

Fig. 4.1: Toy dataset

We purposefully omitted the observation $(0, 1, 1, 2)$ from the dataset to use it as our test case. What will FMs learn about the interaction between *sedentary life* and *unhealthy diet* from this dataset? Out intuition demands they interact in a harmful way. However, for FMs, note that *smokes* (S) has similar interactions with *sedentary life* (SL) and *unhealthy diet* (UD) for predicting *life expectancy*. In other words, $\langle v_S, v_{SL}\rangle \sim \langle v_S, v_{UD}\rangle << 0$. This puts v_{SL} and v_{UD} diametrically opposite from v_S. But then v_{SL} and v_{UD} will be close in the embedding space. Hence $\langle v_{SL}, v_{UD}\rangle > 0$ and FMs will infer that *sedentary life* and *unhealthy diet* must have a joint positive contribution to *life expectancy*. This is the opposite of what we were expecting! What went wrong?

Let us explain the previous example with concrete numbers. To make our point we will consider $k = 1$, but as we will prove, a similar result holds in higher dimensions. The FM model equation for $k = 1$ is:

$$\hat{y}_{FM}(x) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + v_1v_2x_1x_2 + v_1v_3x_1x_3 + v_2v_3x_2x_3 \tag{4.7}$$

where $w_i, v_i \in \mathbb{R}$. Let us fit this FM to the data and see what it learns about the pairwise interaction between the features *sedentary life* and *unhealthy diet*; our intuition demands this weight to be negative. Note that the weights for the pairwise interactions are v_1v_2, v_1v_3 and v_2v_3, whose product is a perfect square. Thus, if the FM infers from the data that *smoking* and having a *sedentary life* should have a negative impact on *life expectancy*, and similarly that *smoking* and having an *unhealthy diet* should have a negative impact on *life expectancy*, then the FM **must** conclude that having a *sedentary life* and an *unhealthy diet* should have a positive impact on life expectancy. For example, the FM could learn the weights

$$(w_0, w_1, w_2, w_3, v_1, v_2, v_3) = (10, -2, -2, -2, -2, 2, 2)$$

which fit the data, but will surprisingly predict a *life expectancy* of 10 for point $(0, 1, 1)$. Note that if we reverse the effect of the independent variables on the dependent variable, say by multiplying the *life expectancy* by -1 (call this *death expectancy* due to lack of a better term) as in figure 4.2, then the FM can learn weights

$$(w_0, w_1, w_2, w_3, v_1, v_2, v_3) = (-10, 2, 2, 2, 2, 2, 2)$$

and correctly predict a value of -2 for point $(0, 1, 1)$.

smokes	sedentary life	unhealthy diet	life expectancy
0	0	0	10
1	0	0	8
0	1	0	8
0	0	1	8
1	1	0	2
1	0	1	2
0	1	1	10

(a) FMs fail to infer the interaction as desired

smokes	sedentary life	unhealthy diet	'death' expectancy
0	0	0	-10
1	0	0	-8
0	1	0	-8
0	0	1	-8
1	1	0	-2
1	0	1	-2
0	1	1	-2

(b) FMs infer the interaction as desired

Fig. 4.2: Effect of reversing the response variable on FM's inductive bias. Toy dataset.

In the general case ($k \geq 1$), it is not necessarily true that if $\langle v_1, v_2 \rangle$ and $\langle v_1, v_3 \rangle$ are negative, then $\langle v_2, v_3 \rangle$ is positive. For larger values of k, the high-dimensionality of the embedding space allows the vectors to accommodate in more sophisticated ways. However, the following result holds:

Theorem. Given $n + 2$ vectors in $\mathbb{R}^n$ ($n \geq 1$), there are two whose inner product is non-negative.

Proof. We prove the proposition by induction on n. For $n = 1$ it is trivial. Suppose it is true for some $n - 1 \geq 1$ and let us prove it for n. Let $v_1, v_2, \ldots, v_{n+2}$ be $n + 2$ vectors in $\mathbb{R}^n$. If $v_1 = 0$ we are done, so suppose otherwise. If any of the other v_i ($i \neq 1$) forms a non-obtuse angle with v_1 we are also done, so suppose $\langle v_i, v_1 \rangle < 0$ for all $i \neq 1$. Write $v_i = \langle v_i, v_1 \rangle \frac{v_1}{|v_1|^2} + u_i$ where $u_i \in \langle v_1 \rangle^{\perp}$. Then $u_2, u_3, \ldots, u_{n+2}$ are $n + 1$ vectors in an $n - 1$-dimensional space isomorphic to $\mathbb{R}^{n-1}$, so by the inductive hypothesis there exist $2 \leq s < t \leq n+2$ such that $\langle u_s, u_t \rangle \geq 0$. Since $\langle v_s, v_1 \rangle, \langle v_t, v_1 \rangle$ are both negative it follows that $\langle v_s, v_t \rangle = \left\langle \langle v_s, v_1 \rangle \frac{v_1}{|v_1|^2} + u_s, \langle v_t, v_1 \rangle \frac{v_1}{|v_1|^2} + u_t \right\rangle = \frac{\langle v_s, v_1 \rangle \langle v_t, v_1 \rangle}{|v_1|^2} + \langle u_s, u_t \rangle > \langle u_s, u_t \rangle \geq 0$, and we are done. ■

As a corollary, given a FM with embedding of dimension k, for any $k+2$ chosen features there will always be two whose pairwise interaction weight is non-negative. This is an easy to grasp result showing a limitation of FMs. Of course, the result is false if 'non-negative' is changed by 'non-positive', which displays the asymmetric nature of FMs.

Our analysis raises some questions regarding the inductive bias of FMs: we would like to know intuitively what criterion FMs are using to infer the pairwise interaction weights, and why this criterion is asymmetric. To answer this question, it is again convenient to look at the case $k = 1$ (larger values of k are just meant to generalize this intuition to

higher dimensions) and consider a graph on n vertices, one vertex per feature. Let the edge between two vertices be assigned the weight of the pairwise interaction between those features. Assume for simplicity that all weights are non-zero. Then one can prove that FMs can model precisely those graphs where:

1. For every cycle of even length, the two alternating products of edge weights are equal.

2. For every cycle of odd length, the product of its weights is positive.

From this, it is clear that FMs (with $k = 1$) have only n degrees of freedom: fix the weights of a spanning tree and an edge forming an odd cycle (in a way that does not violate condition 2), and from the above restrictions all other weights are uniquely determined. FM_1^- changes (2) to read 'for every cycle of odd length, the product of its weights is **negative**'. With some poetic license, we can say that FMs work under the assumption that *the friend of my friend is my friend, and the friend of my enemy is my enemy.* FMs 'fail' in our toy dataset because our intuition want us to infer from two negative edges in a triangle a third negative edge, but the inductive bias of FMs will do the exact opposite.

Explaining why this issue does not arise in the toy user-movie dataset is a bit more subtle, and has to do with the fact that in this dataset the pairwise weights that are only ever exercised are induced by a bipartite graph with users on one side and movies on the other, but FM_k^+ and FM_k^- are equally expressive in this context: they are both equivalent to MF. Formally, $FM_k^+ = FM_k^-$ when we restrict the domain of the functions in FM_k^+ and FM_k^- to the set X of vectors $x \in \{0,1\}^{U+I}$ ($U, I \in \mathbb{N}$ fixed) such that exactly one of $x_1, \ldots, x_U$ is 1, and exactly one of $x_{U+1}, \ldots, x_{U+I}$ is 1. The proof for this is easy: if $f \in FM_k^+$ is parametrized by the weights $(w_0, \ldots, w_n, v_1, \ldots, v_{U+I})$, then $-f \in FM_k^+$, parametrized by the weights $(-w_0, \ldots, -w_n, -v_1, \ldots, -v_U, v_{U+1}, \ldots, v_{U+I})$, hence $FM_k^+ = FM_k^-$.

4.4 Exploiting or Addressing Asymmetry

The previous discussion suggests a trivial trick to try to improve the performance of FMs: since $FM_k^+ \neq FM_k^-$ (except specific cases as the one discussed), *just by reversing the value of the response variable in the dataset* (which is equivalent to training a model from FM_k^- instead of FM_k^+) the performance of FMs might improve. Ensembling models from FM_k^+ and FM_k^- might be another good idea, which we will explore extensively later.

If we talk about 'fixing' the asymmetry of FMs, some good ideas can be taken from the work of [14]. Indeed, we can consider:

$$\hat{y}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2} \sum_{j=1}^{k} \lambda_i \big(\big(\sum_{i=1}^{n} v_{i,j} x_i \big)^2 - \sum_{i=1}^{n} v_{i,j}^2 x_i^2 \big) \tag{4.8}$$

where $\lambda_i \in \mathbb{R}$. Since the absolute value of λ_i can be absorbed into the right hand factor, this hypothesis space is as expressive as limiting each λ_i to take the value -1 or 1. From our elementary point of view, the squared hyperplanes can now be either added or subtracted. A middle ground approach which does not require fitting λ is to just force half the λ_i in equation 4.8 to be 1 and the other half to be -1. We will refer to these

symmetric FMs as *neutral FMs*[1], and denote their family by

$$FM_k^{\pm} = \{\hat{y} : \mathbb{R}^n \to \mathbb{R} \mid \exists w \in \mathbb{R}^{n+1}, V \in \mathbb{R}^{n \times k} : \\ \hat{y}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2} \sum_{j=1}^{k} (-1)^j \Big(\Big(\sum_{i=1}^{n} v_{i,j} x_i \Big)^2 - \sum_{i=1}^{n} v_{i,j}^2 x_i^2 \Big) \}$$

Interestingly, equation 4.8 is closely related to the following model, also derived from [14]:

$$\hat{y}(x) := w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle u_i, v_j \rangle \, x_i x_j \tag{4.9}$$

where $w_0 \in \mathbb{R}, w \in \mathbb{R}^n$ and $U, V \in \mathbb{R}^{n \times k}$. In essence, the asymmetry we have discussed is directly related to the positive semi-definiteness of the pairwise interaction weight matrix W. By relaxing the representation of W from $W = VV^T$ to $W = UV^T$, the positive semi-definiteness of W is bypassed. The results in [14] suggest that the models defined by equations 4.8 and 4.9 should be of comparable performance when the embedding size of the former is twice the one of the latter.

Finally, unlike FMs, FFMs do not typically exhibit the asymmetry that FMs do. Indeed, if only one feature is active per field each time (that is, if $x_i x_j \neq 0 \Rightarrow f(i) \neq f(j)$) which is typically the case by the choice of fields, then the FFM model equation can be easily reversed by multiplying each w_i by -1 and multiplying each $v_{i,g}$ with $f(i) < g$ by -1. The intuition behind this is that in each of the $\binom{f}{2}$ embedding spaces corresponding to two distinct fields we choose the embeddings corresponding to one of the two fields and multiply them by -1. When might a field have more than one feature active at a time, and so this argument fail? For example, this could be the case if for a movie we have a list of one *or more* genres associated to that movie, and we choose to group all genre indicator variables under the same field. Even then, this asymmetry is quite mild and 'local' in nature: notice that in this example simply removing the genre field would restore the symmetry of the FFM model equation. On the other hand, the asymmetry of the FM model equation appears to be more 'global' in nature.

Before moving on, let us remark a trivial fact which we will use later on:

Observation. If we ensemble a model from FM_k^+ with a model from FM_k^-, then the resulting model belongs to the class of models $FM_{2k}^{\pm}$. In fact:

$$\{f + g : f \in FM_k^+, g \in FM_k^-\} = FM_{2k}^{\pm}$$

This is interesting because it tells us that training a model from FM_k^+ and a model from FM_k^- and ensembling them can be seen as a (very unconventional) way to train a model from $FM_{2k}^{\pm}$. In our experiments we will compare this with training a model from $FM_{2k}^{\pm}$ directly. Since FMs are known to overfit, ensembling can be seen as a way to regularize the training of a model from $FM_{2k}^{\pm}$. We will show in our experiments that this ensembling trick is quite effective on the recommendation datasets.

[1] We prefer to call them *neutral FMs* rather than *symmetric FMs* not to overload the already widely used mathematical term *symmetric*.

5. INTERMEDIATE-ORDER FACTORIZATION MACHINES

5.1 Motivation

One line of research on FMs involves exploring *higher-order interactions*. FMs model interactions of order 2, assigning a weight of $w_{ij} = \langle v_i, v_j \rangle$ to the interaction term $x_i x_j$. The idea behind higher-order FMs is to model the 3-way interaction terms $x_i x_j x_l$, the 4-way interaction terms $x_i x_j x_l x_t$, and so on, in a similar way. The original FM paper by Rendle [9] proposes generalizing FMs from 2-way to d-way via:

$$\hat{y}_{\text{HOFM}}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{l=2}^{d} \sum_{i_1=1}^{n} \cdots \sum_{i_l=i_{l-1}+1}^{n} \Big(\prod_{j=1}^{l} x_{i_j} \Big) \Big(\sum_{q=1}^{k_l} \prod_{j=1}^{l} v_{i_j,q}^{(l)} \Big) \tag{5.1}$$

The intuition is simpler than the equation suggests: just as in standard FMs the weight w_{ij} of the pairwise interaction $x_i x_j$ is the inner product $\langle v_i, v_j \rangle$, for 3-way interactions we just let the weight w_{ijl} corresponding to $x_i x_j x_l$ be the 'triple vector product' $\sum_{q=1}^{k} v_{i,q} v_{j,q} v_{l,q}$. We do this for all orders from 2 to d, with different embeddings v each time, and finally add up all terms.

Later work [15] shows how to efficiently implement these Higher-Order Factorization Machines (HOFMs) with dynamic programming. HOFMs have not received much adoption though, possibly because they are a lot more complex than standard 2-way FMs to implement and do not show much performance improvement.

In this work we take a different approach, motivated by equation 4.4. We attempt to simplify the HOFM proposal and at the same time to generalize FMs in a new direction. The idea is that the essence of FMs lies in the distributive law: when squaring a hyperplane pairwise interaction terms arise naturally, and similarly when cubing a hyperplane 3-way interaction terms arise. Of course, this also generates some unwanted terms such as $v_i^2 v_j x_i^2 x_j$ which we must deal with. But clearly, *there is nothing stopping us from exponentiating a hyperplane to an arbitrary d-th power*. By doing so, we can interpret it as giving rise to d-way interactions, but more importantly, provides us a way to continuously parameterize the interaction order in the FM model equation. Concretely, we explore generalizing standard FMs of order 2 to order d in the following way:

$$\hat{y}_{FM_d}(x) := w_0 + \sum_{i=1}^{n} w_i x_i + \frac{1}{2} \sum_{j=1}^{k} \left(\left| \sum_{i=1}^{n} v_{i,j} x_i \right|^d - \Big(\sum_{i=1}^{n} v_{i,j}^2 x_i^2 \Big)^{\frac{d}{2}} \right) \tag{5.2}$$

When $d = 2$ this is the same as the original FM model equation 4.4. Of course, as d varies further away from 2 the model diverges more and more from HOFMs. Think about $d = 4$ for example, where expanding we get a variety of unwanted terms besides $x_i x_j x_l x_t$ $(i < j < l < t)$, such as $x_i x_j^2 x_l$. Even for $d = 3$ the use of the absolute value in equation 5.2 makes it unclear what the interpretation of the weight for term $x_i x_j x_l$ $(i < j < l)$ is. But we do not care about any of this: we abandon our original interpretation in favor of increased flexibility and mathematical convenience. All models are wrong anyway!

5.2 Implementation

Adapting 2-way FMs to support Intermediate-Order FMs as in equation 5.2 is easy in the case of SGD learning. We believe it might not be possible to do so efficiently in the case of ALS and MCMC learning, because the key property of multi-linearity, which leads to multi-convexity, is lost. See [16] for a full exposition on these learning procedures applied to FMs.

When $d \geq 1$, equation 5.2 has continuous partial derivatives with respect to the $v_{i,j}$, given by:

$$\frac{\partial}{\partial v_{i,j}} \hat{y}_{FM_d}(x) = \frac{d}{2} x_i \text{sign}(D_1) |D_1|^{d-1} - \frac{d}{2} v_{i,j} x_i^2 D_2^{\frac{d-2}{2}} \tag{5.3}$$

where $D_1 = \sum_{i=1}^{n} v_{i,j} x_i$ and $D_2 = \sum_{i=1}^{n} v_{i,j}^2 x_i^2$. This is similar to the derivatives of 2-way FMs and can also be computed efficiently with minor modifications to the code. The only caveat is that exponentiating a floating point number to an arbitrary exponent is *a priori* a slow operation, but this can be overcome by restricting ourselves to suitable values of d, such as $d \in \{1.25, 1.50, 1.75, \dots\}$ for which exponentiation can be efficiently computed by taking only square roots. Note also that equation 5.2 can be easily differentiated with respect to d, allowing us to learn the interaction order from the data if we wish (at the expense of computational cost).

6. EXPERIMENTAL RESULTS

We compare our FM variants against state-of-the-art FFMs following the FFM paper [13]. The paper's source code is publicly available[1], allowing us to reproduce their results and extend their code with our own FM variants. Most of our experiments are based off this paper, with the exception of experiments on the MovieLens dataset, which we included because it constitutes the prototypical recommendation scenario of rating prediction and is also a regression problem, unlike all of the other experiments.

The libFFM library[2] benefits from the use of the SSE instruction set and parallelism through the OpenMP library. This speeds up training considerably (by an order of magnitude in our experiments). The libFFM library has also been widely used at Kaggle competitions and was the key to a first place win at both the Criteo Display Advertising Challenge[3] and the Avazu CTR Prediction competition[4], won by the authors of the FFM paper [13].

The datasets consist of binary classification tasks, with the exception of the MovieLens dataset, and can be divided into two categories:

- **Small Datasets**: Four of these are generic datasets unrelated to recommender systems, named **phishing, adult, rna** and **ijcnn**. The other two small datasets are **movielens-100K** and **movielens-100K-all**, from the recommendation domain. The difference between these two variants of MovieLens is that the former contains only user and item information, whereas the latter is enriched with user and item metadata, allowing us to appreciate how the inclusion of metadata increases the performance of FMs.

  The small size of these datasets allows us to tune the hyperparameters of all models with grid search in an automated fashion. The non-recommendation datasets allow us to judge the impact of asymmetry and interaction order on the performance of FMs as a general predictor.

- **Large Datasets**: These are the **criteo** and **avazu** CTR-prediction datasets from the aforementioned Kaggle competitions. Like the MovieLens dataset, this is the kind of setting where FM-based models are meant to excel at, particularly FFMs.

  Unlike the small datasets, exhaustive hyperparameter search is not possible and so we stick to the original hyperparameters in the FFM paper [13]; we just explore varying FM order and symmetry type while keeping the other hyperparameters fixed. The goal in this case is not to find the best possible model, but rather to determine how readily exploitable asymmetry and interaction order are.

The models considered are:

- **LM**: The standard linear model, $\hat{y}_{LM}(x) = w_0 + \sum_{i=1}^{n} w_i x_i$.

[1] https://www.csie.ntu.edu.tw/ cjlin/ffm/exps/
[2] https://www.csie.ntu.edu.tw/ cjlin/libffm/
[3] https://www.kaggle.com/c/criteo-display-ad-challenge
[4] https://www.kaggle.com/c/avazu-ctr-prediction

- **Poly2**: The polynomial model of degree 2 with hashed cross features as in equation 3.2. The number of buckets is always 10^7 (to match the experiments in [13]).

- **FFM**: FFMs as in 3.6. When we explore the effect of reversing the value of the response variable on FFMs we will call these **FFM-Reversed**. Of course, since in most cases explored the class of FFM models is by construction symmetric with respect to the response variable, any difference in performance between FFM and FFM-Reversed is attributed to noise in the optimization procedure (specifically, model initialization).

- **FM**: We explore standard FMs as in 4.1, which we call simply **FM-Standard**. When reversing the value of the response variable as in 4.2 we call these **FM-Reversed**. Finally, we consider the symmetric variant of FMs as in 4.8 and call these **FM-Neutral**.

  We also explore Intermediate-Order FMs as in 5.2. When the interaction order is different from 2, we indicate this by appending the order to the name, i.e. **FM-1.25-Standard** indicates a 1.25-way standard Factorization Machine as in 5.2. The interaction orders explored are $1.25, 1.5, 1.75, 2.0, 2.25, 2.5, 2.75, 3.0$, since these can be efficiently implemented with square root operations only.

- **Gradient-Boosted Decision Trees (GBDT)**: GBDTs are state-of-the-art general predictors. We use the XGBoost implementation on the four small, non-recommendation datasets, and the LightGBM implementation on the MovieLens dataset (because LightGBM is more efficient at handling high-dimensional categorical variables such as user and item).

- When we ensemble two models A and B, we term this model $A + B$. Of particular interest is the ensemble **FM-Standard + FM-Reversed**, which as discussed in section 4.4, belongs to the class of **FM-Neutral** models.

With regard to the inclusion or exclusion of linear terms in the Poly2, FM, and FFM models, these are **included for the small datasets** and **excluded for the large datasets** so as to follow the FFM paper [13]. In our experiments, inclusion of linear terms for the large datasets does not change results much. These results are included in the appendix for completeness. All models are trained with the Adagrad optimizer [17].

One goal of this thesis has been to make our work fully reproducible. As a result, we have provided code for reproducing every result reported. The code can be found here:

`https://bitbucket.org/sebaprillo/thesis-fms/src/master/`

Submissions to the Kaggle platform have also been automated. Running our main script will re-run all experiments and re-compile the thesis with the new results.

6.1 Small Datasets

The following table summarizes the characteristics of the small datasets:

Data set	Type	# Training Instances	# Test Instances	# Features	# Fields
phishing	Classification	8860	2195	100	30
adult	Classification	32561	16281	308	14
cod-rna	Classification	59535	271617	8	8
ijcnn	Classification	49990	91701	22	22
movielens-100K	Regression	80000	20000	2625	2
movielens-100K-all	Regression	80000	20000	2689	6

Fig. 6.1: Characteristics of the small datasets

The training protocol for a model (which is taken from the FFM paper for the sake of reproducibility) is the following:

1. The training set is randomly split with a 80-20 ratio into a sub-training and validation set.

2. For each setting of hyperparameters, the model is trained on the **sub-training** set and evaluated on the **validation** set. We record the best loss achieved on the validation set and the number of iterations required to achieve that loss (this is a form of early-stopping).

3. After all hyperparameter settings have been tried, the best performing one is re-trained on the full training set for the number of iterations recorded, and finally evaluated on the test set. This is what is reported in the tables.

All grids for all models have been chosen of the same size to allow for a fairer comparison. As in the FFM paper, we grid search for the regularization hyperparameter λ and the embedding size k. However, we also grid search for the learning rate, as it has a big impact on performance. For the details, refer to the code of the thesis.

6.1.1 phishing dataset

The **phishing** dataset from the UCI Machine Learning Repository[5] contains 11055 instances each corresponding to a website, with 8860 training instances and 2195 test instances. From each website 30 categorical fields, each of which takes on 2 or 3 different values, are extracted, and the task is to predict whether the website is a phishing website or not. After one-hot encoding the number of features is 100.

We extend the table in the FFM paper ([13, Table 4]) with our best FM variant, and also report the performance of FFM-Reversed to get a grasp of the noise underlying our experimental procedure; recall that FFM and FFM-Reversed should have similar performance because the FFM model equation is symmetric in this case, so any discrepancy can be attributed to optimization noise due to model initialization.

	LM	Poly2	FM	FFM	FFM-Reversed	FM-1.25-Neutral	GBDT
FFM paper	0.14211	0.11512	<u>0.09229</u>	0.10650	-	-	-
Ours	0.13810	0.09123	0.09559	0.08344	0.09020	<u>0.06992</u>	0.06225

Fig. 6.2: Grid search results for the phishing dataset. The best logloss (excluding GBDT) is underlined.

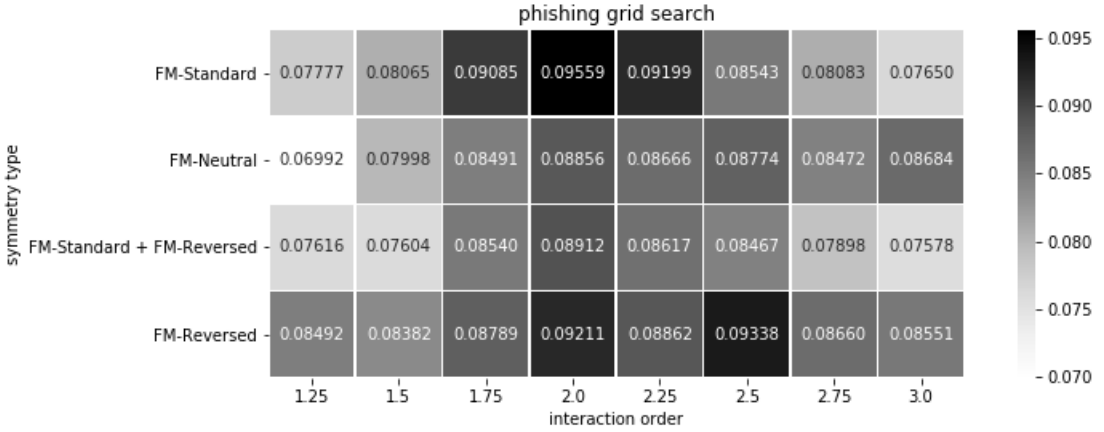


Fig. 6.3: Grid search results (logloss) for varying symmetry type and order of FMs

- The **phishing** dataset is the smallest one considered, and so we should be very careful in our analysis. Indeed, observe the performance gap between FFM and FFM-Reversed: it is quite large and simply due to optimization noise. In particular, the great performance of FM-1.25-Neutral in the table above likely has a noise component to it. This does not mean we cannot spot some consistent patterns from the results, we just have to be careful.

- Unlike the FFM paper, we found that Poly2 and FFMs match or outperform (standard) FMs on this dataset.

[5] http://archive.ics.uci.edu/ml/datasets/Phishing+Websites

- Surprisingly, an order of 2 for FMs appears to be a bad choice on this dataset. Performance seems to improve as we decrease or increase the interaction order, outperforming Poly2 and FFMs.
- Asymmetry has no clear impact on performance given the level of noise we are dealing with.
- There is not much benefit from ensembling FM-Standard and FM-Reversed on this dataset.
- GBDT is the best model, but FM-1.25-Neutral does a reasonable job as a general predictor.

6.1.2 adult dataset

The **adult** dataset from the UCI Machine Learning Repository[6], also known as the **census income** dataset, contains a mix of 14 categorical and continuous variables describing socio-economical characteristics of 48842 people. The training set size is 32561 and the test set size 16281. Attributes include age, occupation, and gender. The goal is to predict whether a person's yearly income exceeds \$50K. The numerical features are discretized into 94 bins (following the FFM paper [13]), leading to a total of 308 features. GBDT is trained on the raw data without any discretization.

	LM	Poly2	FM	FFM	FFM-Reversed	FM-3.0-Neutral	GBDT
FFM paper	0.30970	0.30655	0.30763	0.30565	-	-	-
Ours	0.30899	0.30620	0.30781	0.30568	0.30580	0.30588	0.27514

Fig. 6.4: Grid search results for the adult dataset. The best logloss (excluding GBDT) is underlined.

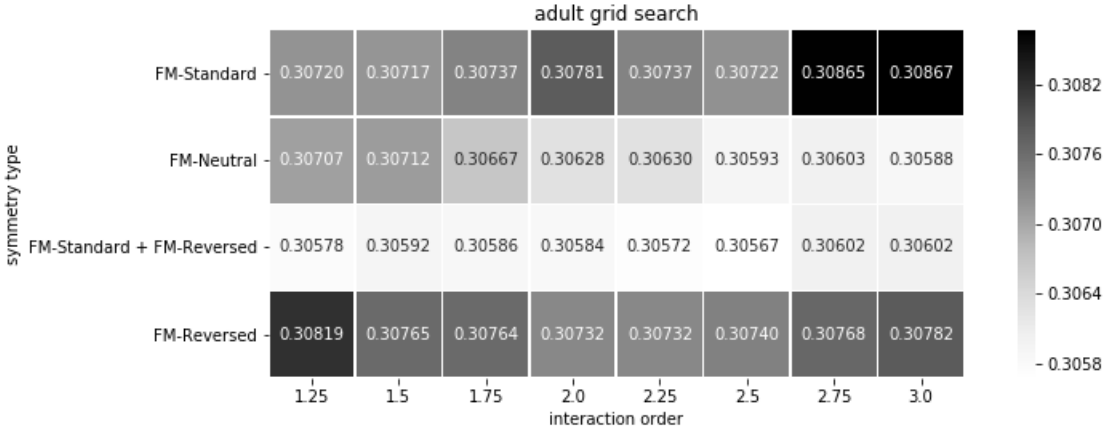


Fig. 6.5: Grid search results (logloss) for varying symmetry type and order of FMs

- We see that the experimental protocol is more robust for this dataset, as FFM and FFM-Reversed both achieved close performance.
- Asymmetry has a big impact on the performance of FMs when contextualized with the performance of FFMs: FM-Neutral outperforms its asymmetric counterparts and significantly closes the performance gap with FFMs.
- Despite the merits of FM-Neutral, the ensemble of FM-Standard and FM-Reversed outperforms FM-Neutral most times. The ensemble is quite robust across all orders, even as the performance of its parts vary. Ensembling seems effective here.
- Order has a marked impact on the performance of FM-Neutral, with higher order performing better, but does not impact much on the performance of FM-Standard

[6] http://archive.ics.uci.edu/ml/machine-learning-databases/adult/

and FM-Reversed.

- GBDT outperforms all other models by a huge margin. In fact, the impact of symmetry type and interaction order on the performance of FMs is meaningless when contextualized with the performance of GBDTs.

6.1.3 rna dataset

The **rna** dataset[7] consists of 331152 instances with 8 numerical fields each. Of these, 59535 belong to the training set and 271617 to the test set (an unusual split). The task is to predict whether a given sequence of RNA is coding or not. Features correspond to statistics on this sequence. No bucketization is performed, so there is the same number of features as there are fields.

	LM	Poly2	FM	FFM	FFM-Reversed	FM-3.0-Reversed	GBDT
FFM paper	0.13829	0.12874	0.12580	0.12914	-	-	-
Ours	0.13310	0.12874	0.12334	0.12545	0.12540	0.12270	0.08825

Fig. 6.6: Grid search results for the rna dataset. The best logloss (excluding GBDT) is underlined.

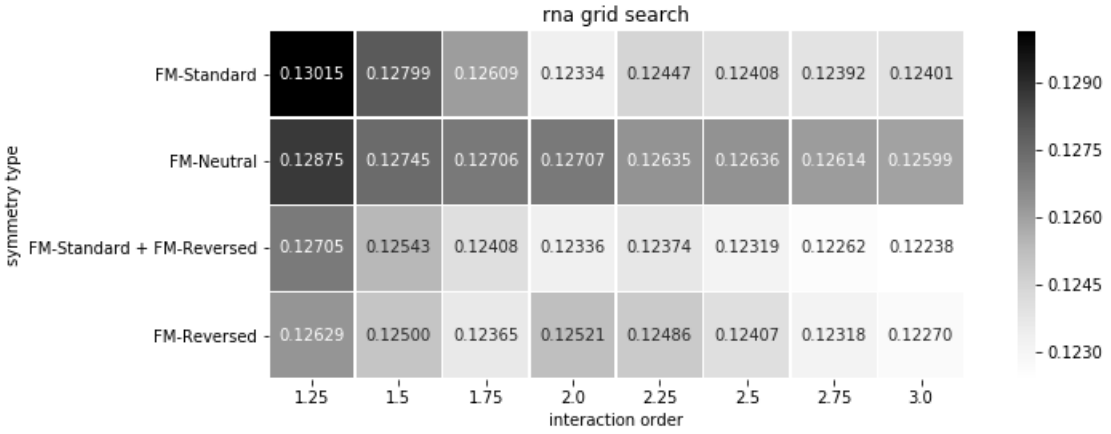


Fig. 6.7: Grid search results (logloss) for varying symmetry type and order of FMs

- FFM and FFM-Reversed have very similar performance, as desired.
- Asymmetry has a visible impact on the performance of FMs when contextualized with the performance of FFMs. FM-Neutral is outperformed by both FM-Standard and FM-Reversed, as well as by their ensemble by a large margin. This is interesting as FM-Neutral should be able to model the best of FM-Standard and FM-Reversed, and so suggests issues with the optimization of FM-Neutral.
- Higher orders generally lead to better performance, but the effect is not as marked as for symmetry type.
- GBDT once again outperforms all other models by a huge margin. And just as in the **adult** dataset, the impact of asymmetry and interaction order on the performance of FMs is meaningless when viewed in the context of the performance of GBDTs.

[7] https://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/binary.html#cod-rna

6.1.4 ijcnn dataset

The **ijcnn** dataset[8] belongs to an old neural network competition. It consists of 141691 instances each 22 fields with of which 49990 belong to the training set and 91701 to the test set. All features are numerical and no bucketization is performed, so as in the **rna** dataset we are left with the same number of features as there are fields.

	LM	Poly2	FM	FFM	FFM-Reversed	FM-1.25-Neutral	GBDT
FFM paper	0.20093	0.08981	0.07087	0.06920	-	-	-
Ours	0.20991	0.07006	0.06984	0.06714	0.06560	0.04346	0.04282

Fig. 6.8: Grid search results for the ijcnn dataset. The best logloss (excluding GBDT) is underlined.

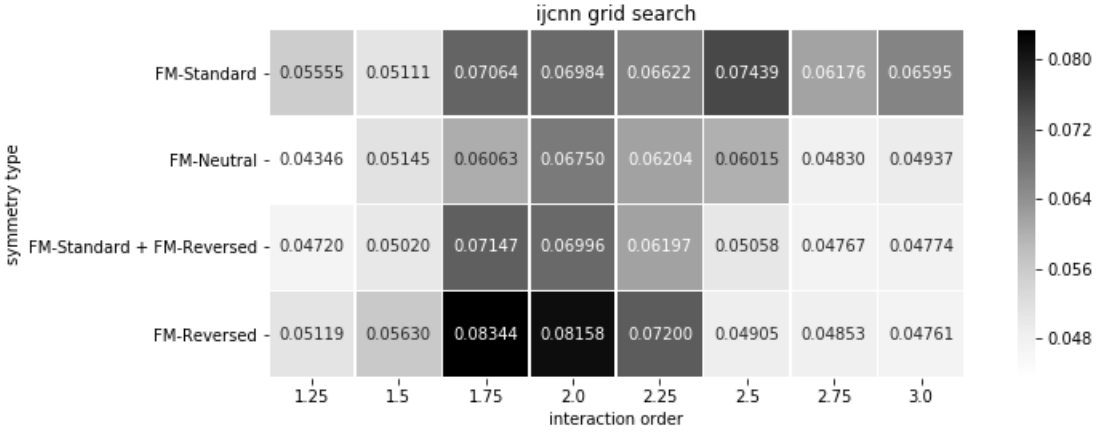


Fig. 6.9: Grid search results (logloss) for varying symmetry type and order of FMs

- FFM and FFM-Reversed have close performance, as desired.
- Results are quite similar to those of the **phishing** dataset (but with significantly less noise thanks to the larger dataset size).
- Order has an outsized impact on performance - even in the context of the performance of GBDTs - with small and large orders performing best.
- Asymmetry has little to no impact on performance, with just FM-Standard performing worse than its counterparts for the higher orders.
- There is no benefit from ensembling FM-Standard and FM-Reversed on this dataset.
- GBDT is the best performing model, but FMs achieve similar performance with FM-1.25-Neutral.

[8] https://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/binary.html

6.1.5 movielens-100K dataset

The **movielens-100K** dataset[9] consists of 100K ratings of 943 users on 1682 movies, on a scale from 1 to 5. This is a prototypical recommendation problem, and the MovieLens dataset in particular has been extensively studied [18].

Note that since the dataset consists of only user and item, then the model families FM-2.0-Standard, FM-2.0-Neutral, FM-2.0-Reversed, FFM, FFM-Reversed *all* coincide. Any difference in performance between these classes is therefore randomness in our experimental procedure.

LM	Poly2	FM	FFM	FFM-Reversed	FM-2.0-Neutral	GBDT
0.93277	0.93275	0.89867	0.89677	0.89917	<u>0.89629</u>	1.00839

Fig. 6.10: Grid search results for the movielens-100K dataset. The best RMSE is underlined.

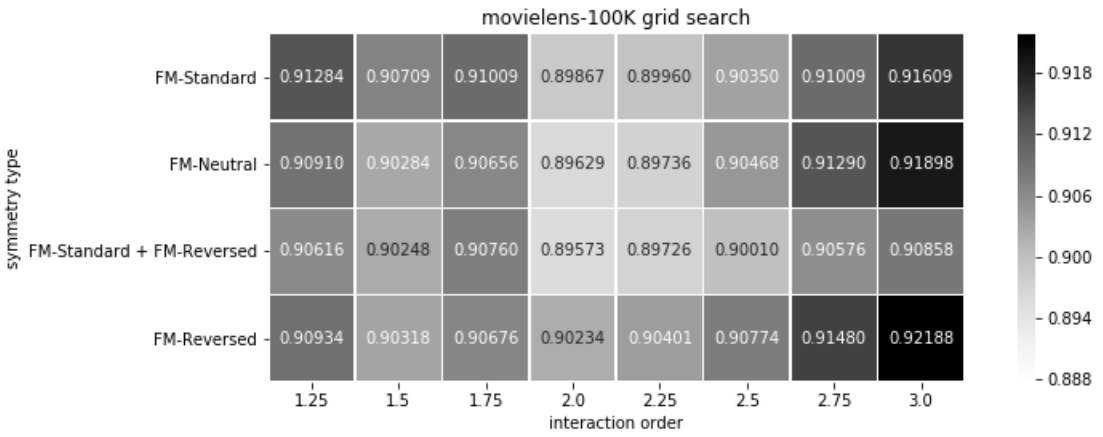


Fig. 6.11: Grid search results (RMSE) for varying symmetry type and order of FMs

- FM-2.0-Standard, FM-2.0-Neutral, FM-2.0-Reversed, FFM and FFM-Reversed all have close performance within a range of 0.006, as desired.
- Order has a mild impact on performance, but unlike the **ijcnn** dataset, standard interaction terms of order 2 work best.
- Asymmetry, on the other hand, has little impact on performance. This we know is true in theory for order 2, but it translates to other orders as well.
- The ensemble of FM-Standard and FM-Reversed (which is the same as ensembling two models from FM-Standard in this case) offers little improvement.
- GBDT performs very poorly, which is no surprise as the dataset consists of two high-cardinality categorical variables (user and item). One of the strengths of GBDTs is that they are robust to the scale of numerical variables (because they work with quantiles), which is also of no use in this setting.

[9] https://grouplens.org/datasets/movielens/

6.1.6 movielens-100K-all dataset

The **movielens-100K-all** dataset extends the **movielens-100K** dataset with user and item metadata. These are:

- the user's age, gender, and occupation
- the movie's genre(s).

User age has been bucketized into 5 same-sized bins. This way, all features are categorical.

Note that due to the inclusion of metadata, the classes FM-Standard, FM-Neutral, FM-Reversed, and FFM are all different.[10]

LM	Poly2	FM	FFM	FFM-Reversed	FM-2.0-Neutral	GBDT
0.93251	0.91715	0.89294	0.89527	0.89557	<u>0.88956</u>	0.93904

Fig. 6.12: Grid search results for the movielens-100K-all dataset. The best RMSE is underlined.

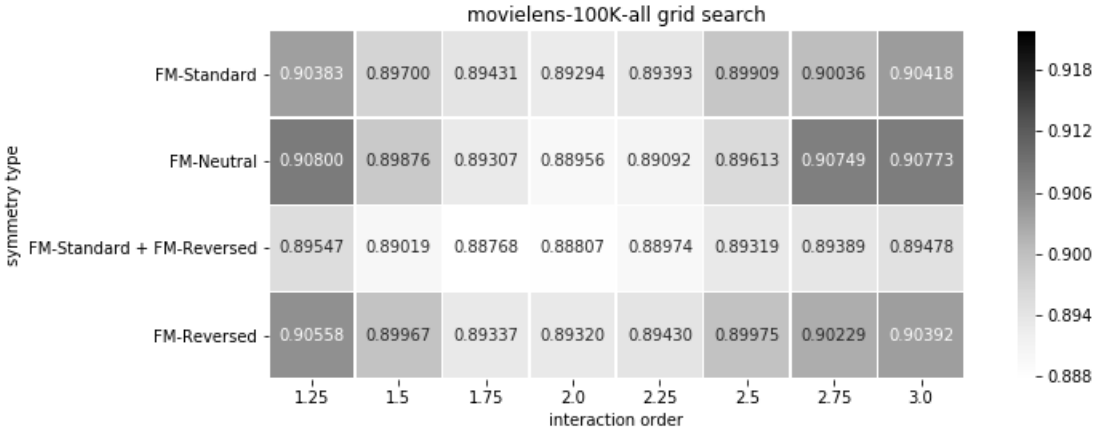


Fig. 6.13: Grid search results (RMSE) for varying symmetry type and order of FMs

It is interesting to note how the performance of the linear model has not improved with the inclusion of metadata (the performance of LM for **movielens-100K** and **movielens-100K-all** is almost the same), whereas for all other models, which are able to capture pairwise feature interactions, the performance has improved. This is most dramatical for the Poly2 and GBDT models, which seem to benefit more from metadata than the latent-factor models (FM and FFM), presumably because the latent-factor models are based on matrix factorization, which is already tailored to the user-item case.

In terms of the impact of order and asymmetry, results are pretty similar to the **movielens-100K** dataset: order has little impact and FM-Standard, FM-Neutral and FM-Reversed all perform similarly. However, there is one clear difference with respect to the **movielens-100K** dataset: the ensemble of FM-Standard with FM-Reversed is

[10] In fact, since a movie can have more than one genre associated to it, and we have chosen all genres to fall under the same field, we have FFM $\neq$ FFM-Reversed for the first time.

stronger this time, across all orders. For example, the drop in RMSE obtained by ensembling FM-Standard with FM-Reversed is 0.0049 for **movielens-100K-all**, which is almost twice the improvement for **movielens-100K**, which is 0.0029. In contrast, we found that the ensemble of FFM with FFM-Reversed dropped its improvement from 0.0029 on **movielens-100K** to 0.0023 on **movielens-100K-all**. The asymmetry of FMs would explain this phenomenon: as features are added beyond user and item, the predictions of FM-Standard and FM-Reversed start to differ, and their ensemble gets stronger.

6.2 Criteo Dataset

The **criteo** dataset[11] from the Criteo Display Advertising Challenge consists of 7 days of display ads served by Criteo, for a total of 45M training instances. The task is to predict whether the ad was clicked or not, and the metric to optimize is logloss. The test set contains 6M instances from the day following the training data. Each display is described by 13 numerical features and 26 categorical features, all undisclosed (no description of the features is given). After preprocessing and bucketizing numerical features, the features are hashed down to 1M different values with the hashing trick[12].

We would like to point out that the hashing trick might actually asymmetrize the FFM model equation, but the picture is somewhat contrived (features suddenly get shared *across* fields), so we leave it out of the discussion. We found this to have no practical impact, for the correlation between the predictions of FFM and FFM-Reversed in our experiments is almost perfect.

6.2.1 Table of Results

We reproduce [13, Table 3 (a)] and extend it with FM-Neutral and FM-Reversed (note that FM-Standard in our table stands for FM in the original table [13, Table 3 (a)]). For readability, we exclude results for intermediate order FMs from this table.

Model	parameters	training time (seconds)	public set logloss	public set rank
LM	$\eta = 0.2, \lambda = 0, t = 13$	289	0.46261	96
Poly2	$\eta = 0.2, \lambda = 0, B = 10^7, t = 10$	4424	0.44969	14
FM-Standard	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 100, t = 9$	2042	0.44862	11
FM-Neutral	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 100, t = 9$	1952	0.44860	11
FM-Standard + FM-Reversed	-	-	0.44671	6
FM-Reversed	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 100, t = 9$	1920	0.44860	11
FFM	$\eta = 0.2, \lambda = 2 \times 10^{-5}, k = 4, t = 9$	3671	0.44616	3
FFM-Reversed	$\eta = 0.2, \lambda = 2 \times 10^{-5}, k = 4, t = 9$	3698	0.44600	3
FFM + FFM-Reversed	-	-	0.44581	3
FM-Standard + Poly2	-	-	0.44726	6
FM-Neutral + Poly2	-	-	0.44718	6
FM-Reversed + Poly2	-	-	0.44723	6
FM-Standard + LM	-	-	0.44999	15
FM-Neutral + LM	-	-	0.44976	14
FM-Reversed + LM	-	-	0.44986	15
LM + Poly2	-	-	0.45188	17

Fig. 6.14: Criteo results. Training time is only for reference as different models are trained for a different number of iterations.

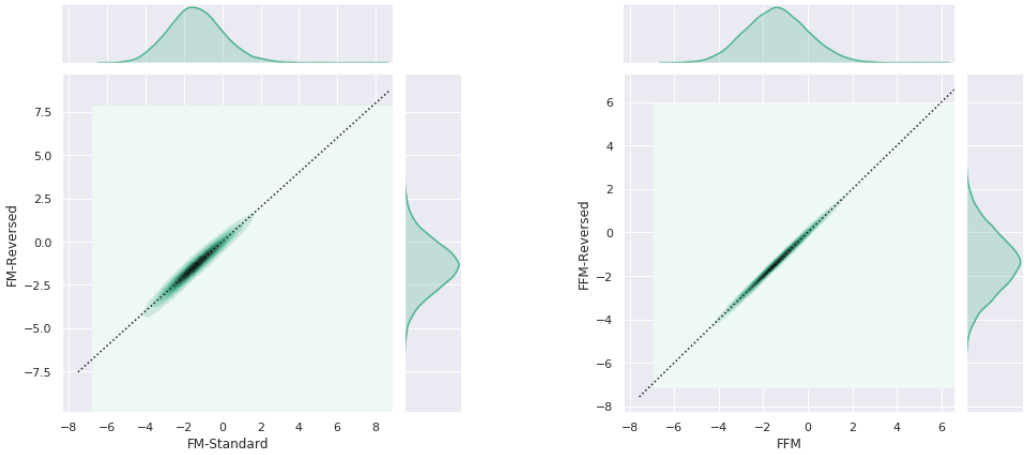
We were able to closely reproduce the results reported in the FFM paper. However, we used 8 threads to train the models, which causes non-determinism, and leads to slightly different results but lower training times compared to [13, Table 3 (a)]. All

[11] http://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/
[12] https://www.csie.ntu.edu.tw/ r01922136/kaggle-2014-criteo.pdf

of FM-Standard, FM-Neutral and FM-Reversed have similar performance. However, the ensemble of FM-Standard and FM-Reversed is a lot stronger, decreasing loss by around 0.4% from 0.44860 down to 0.44671 and surpassing all other ensembles considered (other than the ensemble of FFM with FFM-Reversed, of course). Contrast this to the ensemble of FFM and FFM-Reversed, which decreases loss by less than 0.1%.

6.2.2 Correlation Between Predictions

To understand why the ensemble of FM-Standard with FM-Reversed is so strong, we analyze how the predictions between these two models correlate with each other compared to the predictions between other models.



(a) FM-Standard vs FM-Reversed *(b)* FFM vs FFM-Reversed

Fig. 6.15: Effect of swapping the class label on (logit) predictions of FMs and FFMs, Criteo dataset.

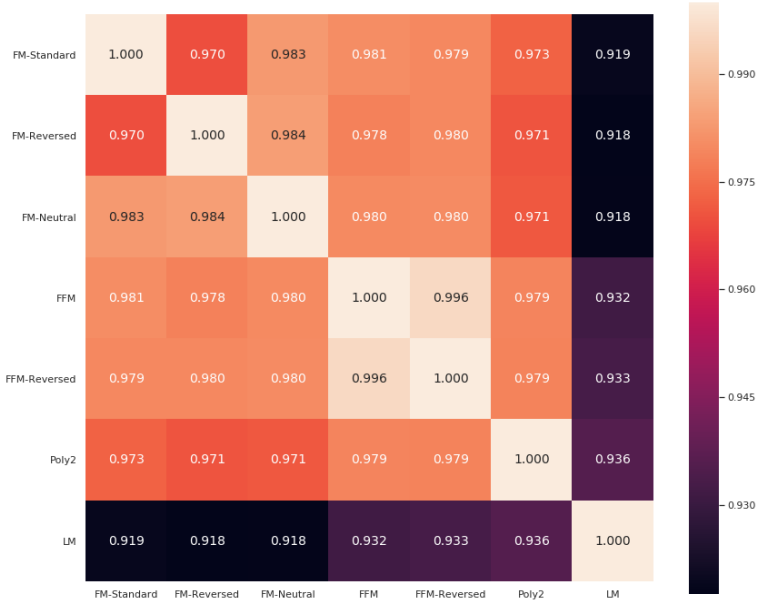


Fig. 6.16: Correlation between test set predictions, Criteo dataset.

We observe that, the linear model left aside, *FM-Standard and FM-Reversed are the less correlated pair of models.* Contrast this to the very high correlation between the predictions of FFM and FFM-Reversed, which is expected since this model class is essentially symmetric. The pairwise plot between the (logit) predictions of FM-Standard and FM-Reversed shows more spread compared to those of FFM and FFM-Reversed.

The performance of the ensemble of FM-Standard and FM-Reversed was not matched by FM-Neutral. To account for the possibility that we need a larger embedding size in FM-Neutral to match the performance of the ensemble, we doubled the embedding size of FM-Neutral from $k = 100$ to $k = 200$, re-computed the number of training epochs by cross-validation (which turned out to be 5) and obtained a score of 0.44760, which is indeed closer to the performance of the ensemble but still far from it. Not content with this, we tried to grid search for better hyperparameters for FM-Neutral too: while keeping $k = 200$ fixed, we grid searched for $\lambda \in \{2 \times 10^{-4}, 2 \times 10^{-5}, 2 \times 10^{-6}\}$ and $\eta \in \{0.02, 0.05, 0.1\}$, but the original setting of $\lambda = 2 \times 10^{-5}$ and $\eta = 0.05$ was already the best. This hints at the value of ensembling.

6.2.3 Varying Symmetry Type and Order

We now explore varying the symmetry type and interaction order of FMs. We use the same hyperparameters as in the FFM paper ($\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 100$). Since we found that using a lower interaction order makes FMs overfit faster, we used cross-validation to find the number of training iterations for each model. When we did this, we found that FMs of orders $1.25, 1.5, 1.75, 2.0, 2.25, 2.5, 2.75, 3.0$ required $3, 5, 6, 7, 7, 7, 7, 7$ epochs respectively before starting to overfit. Intermediate order FMs were at most 15% slower to train per epoch than standard FMs. The test set performances are shown in the next matrix.

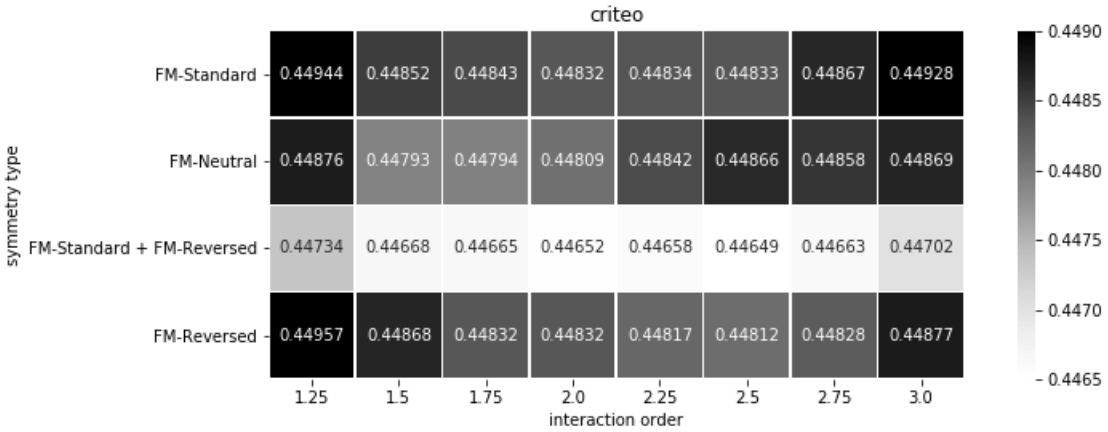


Fig. 6.17: Grid search results (logloss) for varying symmetry type and order of FMs

We found the results on the **criteo** dataset to be largely insensitive to order, and an order of 2.0 already works best, just like on the MovieLens dataset. The ensemble of FM-Standard and FM-Reversed consistently outperforms all of the variants, and is never matched by FM-Neutral.

6.3 Avazu Dataset

The **avazu** dataset[13] from the Avazu Click-Through Rate Prediction Kaggle competition consists of 11 days of click-through data. The data from the first 10 days makes up the training set and the data from the last day makes up the test set. The task is to predict whether an impression was clicked or not, and the metric to optimize is logloss. All available fields are categorical, such as the ad identifier, hour, banner position and site id. As in the **criteo** dataset, after feature engineering the features are hashed down to 1M different values[14].

For this dataset, rather than training one model on all the data, the dataset is split into two parts and a different model is trained on each. This is the strategy used by the winning team of the competition (the authors of the FFM paper [13]).

6.3.1 Table of Results

Model	parameters	training time (seconds)	public set logloss	public set rank
LM	$\eta = 0.2, \lambda = 0, t = 10$	55	0.39016	55
Poly2	$\eta = 0.2, \lambda = 0, B = 10^7, t = 10$	170	0.38555	10
FM-Standard	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 40, t = 8$	221	0.38611	11
FM-Neutral	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 40, t = 8$	183	0.38485	7
FM + FM-Reversed	-	-	0.38449	6
FM-Reversed	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 40, t = 8$	219	0.39217	159
FFM	$\eta = 0.2, \lambda = 2 \times 10^{-5}, k = 4, t = 4$	137	0.38389	6
FFM-Reversed	$\eta = 0.2, \lambda = 2 \times 10^{-5}, k = 4, t = 4$	138	0.38394	6
FFM + FFM-Reversed	-	-	0.38348	6
FM-Standard + Poly2	-	-	0.38463	6
FM-Neutral + Poly2	-	-	0.38412	6
FM-Reversed + Poly2	-	-	0.38558	10
FM-Standard + LM	-	-	0.38541	10
FM-Neutral + LM	-	-	0.38518	10
FM-Reversed + LM	-	-	0.38675	11
LM + Poly2	-	-	0.38529	11

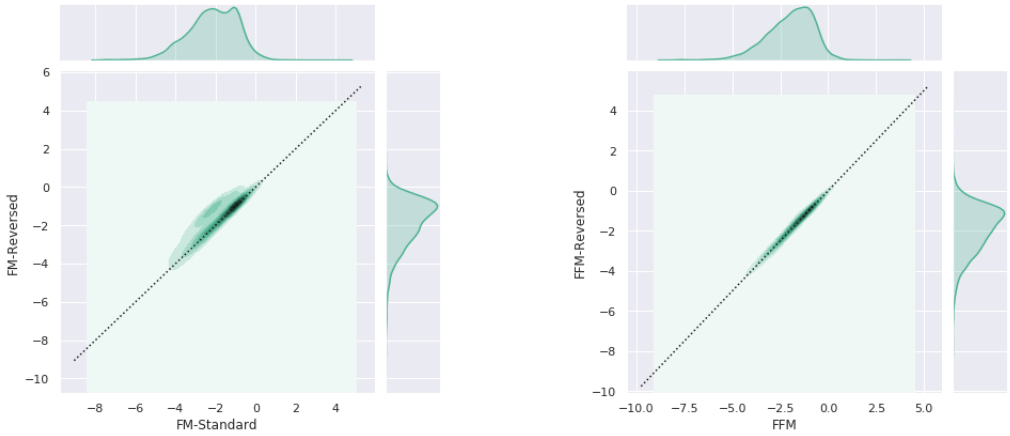
Fig. 6.18: Avazu results. Training time is only for reference as different models are trained for a different number of iterations.

Again, we were able to closely reproduce the results reported in the FFM paper. This time, surprisingly, reversing the value of the response variable significantly degrades the performance of FMs, to the point that *they are outperformed even by the linear model.* However, against all odds, the ensemble of FM-Standard with FM-Reversed outperforms all other ensembles considered except for FM-Neutral + Poly2. The performance boost over FM-Standard by ensembling FM-Standard with FM-Reversed is again around 0.4%, compared to barely 0.1% for FFMs. We found it surprising that a model ranked 11th in the competition would climb up to the 6th position when ensembled with a model ranked

[13] https://www.kaggle.com/c/avazu-ctr-prediction
[14] https://www.csie.ntu.edu.tw/ r01922136/slides/kaggle-avazu.pdf

as low as 159th. We attribute this to the complementing inductive biases of FM-Standard and FM-Reversed.

6.3.2 Correlation Between Predictions



(a) FM-Standard vs FM-Reversed *(b)* FFM vs FFM-Reversed

Fig. 6.19: Effect of swapping class labels on (logit) predictions of FMs and FFMs, Avazu dataset.

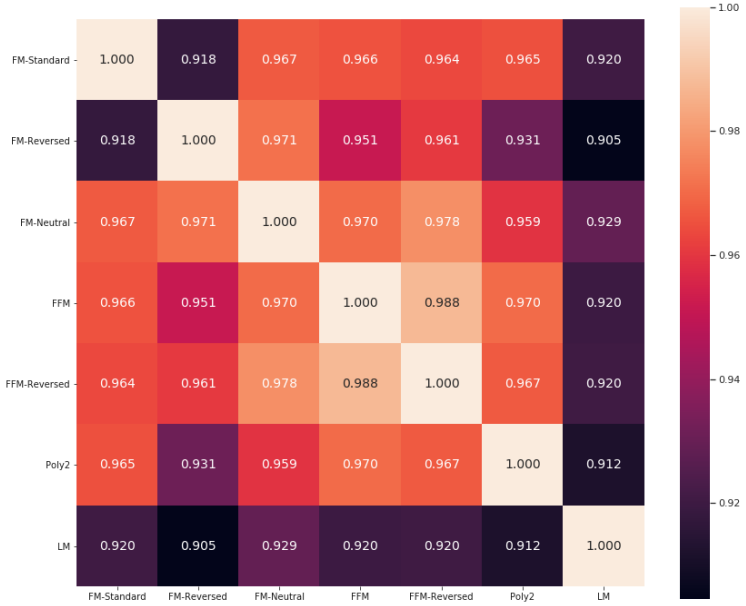


Fig. 6.20: Correlation between test set logits, Avazu dataset.

The pairwise plot between the predictions of FM-Standard and FM-Reversed confirms the conflicting predictions made by the two models, revealing an asymmetric and bimodal shape. This bimodality is explained by the fact that the predictions are the result of merging the predictions from two different models. Again, we see that the correlation between the (logit) predictions of FM-Standard and FM-Reversed is the lowest among any pair of models, even comparable to those involving the linear model. In general, we note that the predictions across all model pairs are less correlated than for the Criteo dataset.

Finally, FM-Neutral shows much better performance compared to FM-Standard and FM-Reversed but is worse than their ensemble. As in the **criteo** dataset, we try to compensate for this by doubling the embedding size of FM-Neutral from $k = 40$ to $k = 80$ and recomputing the number of training epochs with cross validation, but when we do this, the score obtained is 0.38527, which is actually worse than before. Once again we grid search with $\lambda \in \{2 \times 10^{-4}, 2 \times 10^{-5}, 2 \times 10^{-6}\}$ and $\eta \in \{0.02, 0.05, 0.1\}$ keeping $k = 80$ fixed, but the best score obtained was 0.38491 for $\lambda = 2 \times 10^{-4}$ and $\eta = 0.02$, again worse than originally. This shows the difficulties of training FMs that can generalize well.

6.3.3 Varying Symmetry Type and Order

We follow the same protocol as in the **criteo** dataset, whereby we determine the number of iterations to train for with cross-validation. Intermediate order FMs were at most 40% slower to train per iteration than standard FMs.

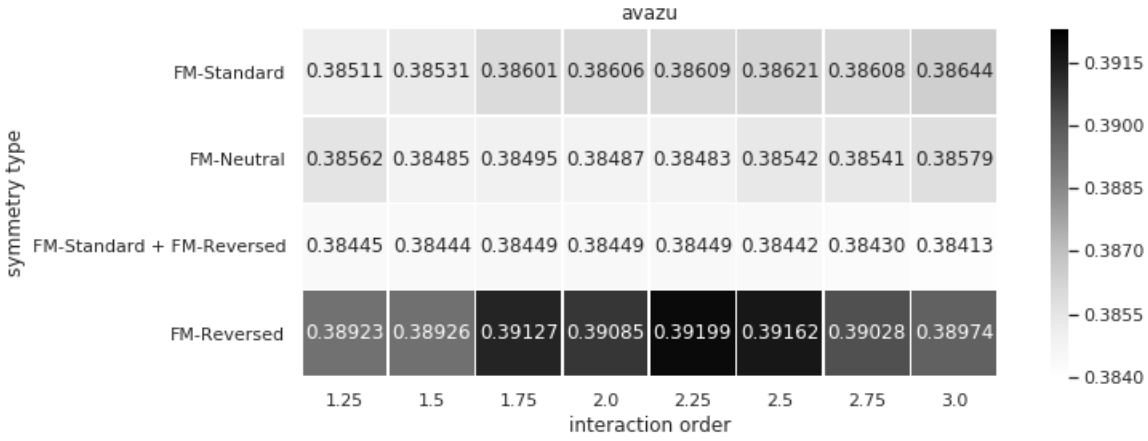


Fig. 6.21: Grid search results (logloss) for varying symmetry type and order of FMs

We see that on the **avazu** dataset asymmetry has a major impact on performance, while interaction order has a mild impact on performance. Lower orders tend to perform better for FM-Standard and FM-Neutral. It is interesting to see that the performance of the ensemble of FM-Standard with FM-Reversed is not necessarily correlated with the performance of its parts: for example, each of FM-3.0-Standard and FM-3.0-Reversed is worse than FM-1.25-Standard and FM-1.25-Reversed respectively, yet the ensemble is stronger at 0.38413. We find this a bit surprising but possible given the low level of correlation that the predictions of FM-Standard and FM-Reversed exhibit on **avazu**.

6.4 Summary

We have seen that symmetry type and interaction order affect the performance of FMs very differently depending on the dataset. In general terms, interaction order proved effective on the non-recommendation datasets, whereas asymmetry was not very relevant. On the other hand, on the recommendation datasets, we could not exploit interaction order but were able to harness the asymmetry of FMs through ensembles. This allowed us to significantly close the performance gap with FFMs. We encountered difficulties training FMs reliably and found ensembling to be an effective strategy to consistently improve the performance of FMs.

7. CONCLUSION AND FUTURE WORK

In this work we have attempted to answer two questions regarding Factorization Machines (FMs): How does asymmetry impact the performance of FMs? Can interaction order as continuously parameterized by Intermediate-Order FMs be exploited to improve performance? We have analyzed these questions in the context of both non-recommendation scenarios (the **phishing**, **adult**, **rna**, **ijcnn** datasets) where FMs are used as general predictors, and in the context of recommendation scenarios (the **movielens-100K**, **movielens-100K-all**, **criteo**, **avazu** datasets).

We have seen that interaction order can significantly improve the performance of FMs as a general predictor, in some cases (the **phishing** and **ijcnn** datasets) nearing the performance of GBDTs. Symmetry can affect the predictions of FMs too, as we saw in the **adult** and **rna** datasets. However, these differences were meaningless when contextualized within the performance of GBDTs.

For the recommendation scenarios explored, the picture was reversed: intermediate orders were of little use, while we were able to successfully exploit the asymmetry of FMs to get unusually large performance improvements by ensembling FM-Standard with FM-Reversed (when compared to other ensembles). We saw how adding metadata to the MovieLens dataset led to stronger ensembles too, in line with our observation that as features are added beyond user and item the FM model equation asymmetrizes and the predictions of FM-Standard and FM-Reversed start to diverge.

Despite our efforts to get the same performance as the ensemble of FM-Standard and FM-Reversed out of FM-Neutral on the **criteo** and **avazu** datasets - which should be theoretically possible - we were not able to achieve this. We attribute this to how easily FMs tend to overfit, and suggest that ensembling might be an effective way to regularize the training of neutral FMs. This is of course not the best scenario as we would like our optimizers to do the hard work, and challenges the effectiveness of our optimization and regularization strategies. Some preliminary analysis (included in the appendix) shows that L2-regularization for FM-based models might be an aspect to investigate in the future.

Also, although we have exploited the additive structure of the FM model class (specifically, that $FM_k^+ + FM_k^- = FM_{2k}^{\pm}$), more work could follow in this direction. For example, boosted versions of FMs could be considered, with ANOVA kernels as weak learners. This sequential way to train FMs might help deal with overfitting in a different way.

One aspect we have not explored in this work and could be interesting is how model asymmetry plays with class disbalance. Specifically, the asymmetry of FMs might be exacerbated by class disbalance. For example, as the ratio of classes changes from 1 (perfectly balanced) to 0.1, 0.01 and so on, the loss gap between FM^+ and FM^- might get worse, and ensembles might get stronger.

Finally, experiments on synthetic data could give us further insights. For example, we could generate data from a model in FM_k^+ and try to fit models from $FM_{k'}^-$ on it with varying values of k'. We know that when $k' = n$ (where n is the number of features), $FM_{k'}^-$ will fit the data perfectly. But what happens for intermediate values of k'? What does loss as a function of k' look like? And hence, for what values of k' - as a function of n - can we consider asymmetry to be noticeable?

8. APPENDIX

8.1 Results with Linear Terms

Recall that FM-based models were trained without linear terms $w_0 + \sum_{i=1}^{n} w_i x_i$ for the **criteo** and **avazu** datasets. In this appendix we include the results for varying symmetry type and order when linear terms are included. Results are mostly similar, perhaps with the exception of FM-Reversed on the **criteo** dataset, which seems to perform slightly worse when linear terms are included, and as a consequence the ensemble also appears to perform slightly worse.

8.1.1 Results on Criteo Dataset with Linear Terms

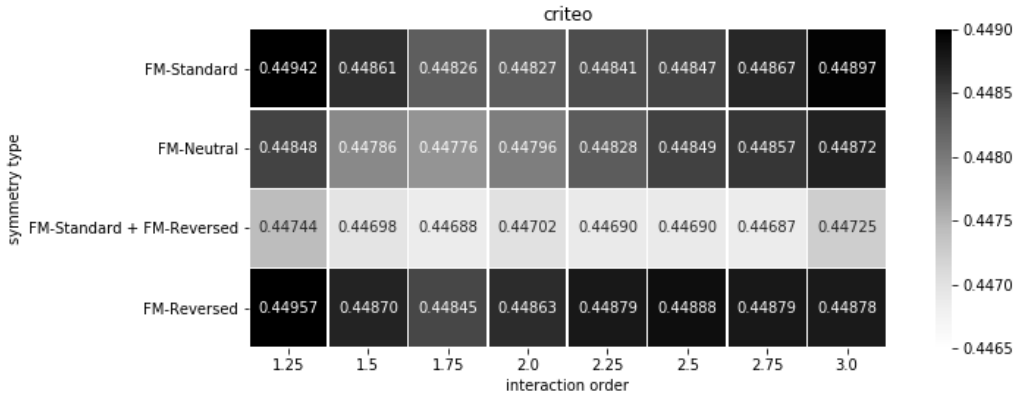


Fig. 8.1: Grid search results (logloss) for varying symmetry type and order of FMs. Linear terms included.

8.1.2 Results on Avazu Dataset with Linear Terms

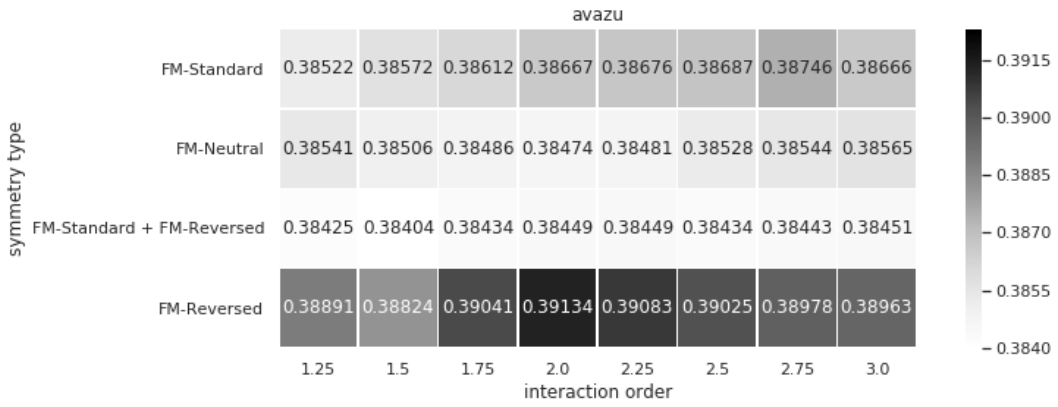


Fig. 8.2: Grid search results (logloss) for varying symmetry type and order of FMs. Linear terms included.

8.2 A Remark on L2-Regularization used for FMs

As studied in [13], FFMs (and FM-based models in general) are quite prone to overfitting. They achieve the best generalization performance with a low level of regularization paired with early stopping. The following figure from [13] illustrates this:

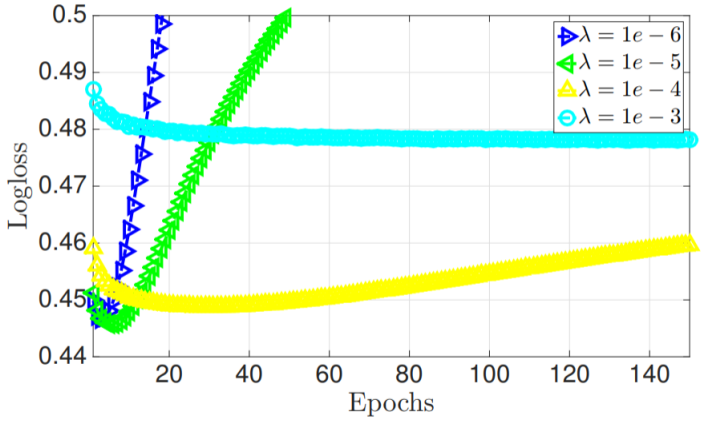


Fig. 8.3: Impact of regularization parameter λ on the generalization performance of FFMs during training

We tried to investigate this and found one interesting fact: both the libFFM [13] and libFM [16] libraries implement a weighted form of L2-regularization for SGD learning. Rather than applying the standard L2-penalty:

$$\Omega = \frac{\lambda}{2}\Big(\sum_{i=1}^{n} w_i^2 + \sum_{j=1}^{k}\sum_{i=1}^{n} v_{i,j}^2\Big) \tag{8.1}$$

what is applied is the following *weighted* L2-penalty:

$$\Omega' = \frac{\lambda}{2}\Big(\sum_{i=1}^{n} p_i w_i^2 + \sum_{j=1}^{k}\sum_{i=1}^{n} p_i v_{i,j}^2\Big) \tag{8.2}$$

where p_i is the probability that $x_i \neq 0$ in the dataset. This happens because regularization updates are only performed for weights whose corresponding feature is non-zero in the current training example. Indeed, the pseudocode for training FMs (and similarly FFMs) with L2-regularization is the following:

Algorithm 2 SGD training of FMs under logarithmic loss with L2-regularization

1: Initialize $w_i \in \mathbb{R}, v_i \in \mathbb{R}^k$ randomly, and let η be the learning rate (which in general can be adaptive and per-coordinate), and λ the regularization parameter.
2: Repeat until convergence:
3: **for** $(x, y) \in$ Training Set **do**
4: $\quad \text{loss} = \log(1 + \exp(-y\hat{y}(x)))$
5: $\quad \frac{d\text{loss}}{d\hat{y}} = \frac{-y}{1+\exp(y\hat{y}(x))}$
6: $\quad w_0 \leftarrow w_0 - \eta \, \frac{d\text{loss}}{d\hat{y}}$
7: $\quad$ **for** $i \in \{1, 2, \ldots, n\}$ such that $x_i \neq 0$ **do**
8: $\quad\quad w_i \leftarrow w_i - \eta \, (\frac{d\text{loss}}{d\hat{y}} \, x_i + \lambda w_i)$
9: $\quad$ **end for**
10: $\quad$ **for** $j \in \{1, 2, \ldots, k\}$ **do**
11: $\quad\quad s = \sum_{i=1}^{n} v_{i,j} x_i$
12: $\quad\quad$ **for** $i \in \{1, 2, \ldots, n\}$ such that $x_i \neq 0$ **do**
13: $\quad\quad\quad v_{i,j} \leftarrow v_{i,j} - \eta \, (\frac{d\text{loss}}{d\hat{y}} \, x_i(s - v_{i,j}x_i) + \lambda v_{i,j})$
14: $\quad\quad$ **end for**
15: $\quad$ **end for**
16: **end for**

As can be seen, the discrepancy arises as a result of only performing weight updates for the non-zero entries of the current training example (lines 7 and 12 in algorithm 2). This is of course convenient for computational reasons. However, the stochastic gradient update due to regularization is now $\nabla\Omega'$ in expectation rather than $\nabla\Omega$.

On one hand, Ω' is not the regularization penalty that is applied when using other solvers such as alternating least squares in libFM (see [19, section 4.3.1]), which uses Ω. On the other hand, the weighted regularization term Ω' is analogous to the weighted-λ-regularization term used in [20] which apparently gave good empirical results. Therefore, the effects of using Ω' over Ω are unclear to us. To the best of our knownledge, we were not able to find comments about this difference in the works on FMs and FFMs. We did find research on efficient regularization for sparse models such as [21]. Applying these strategies to FMs could be a line of future work.

BIBLIOGRAPHY

[1] Sebastian Prillo. An elementary view on factorization machines. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 179–183, New York, NY, USA, 2017. ACM.

[2] Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003.

[3] Brent Smith and Greg Linden. Two decades of recommender systems at amazon.com. *IEEE Internet Computing*, 21(3):12–18, May 2017.

[4] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. The youtube video recommendation system. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 293–296, New York, NY, USA, 2010. ACM.

[5] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 191–198, New York, NY, USA, 2016. ACM.

[6] James Bennett, Stan Lanning Netflix, and Netflix. The netflix prize. 2007.

[7] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, pages 1257–1264, USA, 2007. Curran Associates Inc.

[8] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press.

[9] Steffen Rendle. Factorization machines. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, 2010.

[10] Yehuda Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.

[11] Steffen Rendle and Lars Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 81–90, New York, NY, USA, 2010. ACM.

[12] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 811–820, New York, NY, USA, 2010. ACM.

[13] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, 2016.

[14] Mathieu Blondel, Masakazu Ishihata, Akinori Fujino, and Naonori Ueda. Polynomial networks and factorization machines: New insights and efficient training algorithms. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 850–858. JMLR.org, 2016.

[15] Mathieu Blondel, Akinori Fujino, Naonori Ueda, and Masakazu Ishihata. Higher-order factorization machines. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3351–3359. Curran Associates, Inc., 2016.

[16] Steffen Rendle. Factorization machines with libfm. *ACM Trans. Intell. Syst. Technol.*, 3, 2012.

[17] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[18] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015.

[19] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 635–644, New York, NY, USA, 2011. ACM.

[20] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, AAIM '08, pages 337–348, Berlin, Heidelberg, 2008. Springer-Verlag.

[21] Zachary Lipton and Charles Elkan. Efficient elastic net regularization for sparse linear models. 05 2015.