Field-aware Factorization Machines for CTR Prediction

Yuchin Juan
Criteo Research*
Palo Alto, CA
yc.juan@criteo.com

Yong Zhuang
Dept. of ECE*
Carnegie Mellon Univ., USA
yong.zhuang22@gmail.com

Wei-Sheng Chin
Dept. of Computer Science
National Taiwan Univ., Taiwan
d01944006@csie.ntu.edu.tw

Chih-Jen Lin
Dept. of Computer Science
National Taiwan Univ., Taiwan
cjlin@csie.ntu.edu.tw

ABSTRACT

Click-through rate (CTR) prediction plays an important role in computational advertising. Models based on degree-2 polynomial mappings and factorization machines (FMs) are widely used for this task. Recently, a variant of FMs, field-aware factorization machines (FFMs), outperforms existing models in some world-wide CTR-prediction competitions. Based on our experiences in winning two of them, in this paper we establish FFMs as an effective method for classifying large sparse data including those from CTR prediction. First, we propose efficient implementations for training FFMs. Then we comprehensively analyze FFMs and compare this approach with competing models. Experiments show that FFMs are very useful for certain classification problems. Finally, we have released a package of FFMs for public use.

Keywords

Machine learning; Click-through rate prediction; Computational advertising; Factorization machines

1. INTRODUCTION

Click-through rate (CTR) prediction plays an important role in advertising industry [1, 2, 3]. Logistic regression is probably the most widely used model for this task [3]. Given a data set with m instances $(y_i, \boldsymbol{x}_i), i = 1, \ldots, m$, where y_i is the label and $\boldsymbol{x}_i$ is an n-dimensional feature vector, the model $\boldsymbol{w}$ is obtained by solving the following optimization problem.

$$\min_{\boldsymbol{w}} \quad \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{m} \log(1 + \exp(-y_i \phi_{\mathrm{LM}}(\boldsymbol{w}, \boldsymbol{x}_i))). \qquad (1)$$

*Part of the work was done when these authors were in National Taiwan University.

RecSys '16, September 15-19, 2016, Boston , MA, USA

 ISBN 978-1-4503-4035-9/16/09…$15.00

DOI: http://dx.doi.org/10.1145/2959100.2959134

		Publisher	Advertiser
+80	−20	ESPN	Nike
+10	−90	ESPN	Gucci
+0	−1	ESPN	Adidas
+15	−85	Vogue	Nike
+90	−10	Vogue	Gucci
+10	−90	Vogue	Adidas
+85	−15	NBC	Nike
+0	−0	NBC	Gucci
+90	−10	NBC	Adidas

Table 1: An artificial CTR data set, where + (−) represents the number of clicked (unclicked) impressions.

In problem (1), λ is the regularization parameter, and in the loss function we consider the linear model:

$$\phi_{\mathrm{LM}}(\boldsymbol{w}, \boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}.$$

Learning the effect of feature conjunctions seems to be crucial for CTR prediction; see, for example, [1]. Here, we consider an artificial data set in Table 1 to have a better understanding of feature conjunctions. An ad from Gucci has a particularly high CTR on Vogue. This information is however difficult for linear models to learn because they learn the two weights Gucci and Vogue separately. To address this problem, two models have been used to learn the effect of feature conjunction. The first model, degree-2 polynomial mappings (Poly2) [4, 5], learns a dedicate weight for each feature conjunction. The second model, factorization machines (FMs) [6], learns the effect of feature conjunction by factorizing it into a product of two latent vectors. We will discuss details about Poly2 and FMs in Section 2.

A variant of FM called pairwise interaction tensor factorization (PITF) [7] was proposed for personalized tag recommendation. In KDD Cup 2012, a generalization of PITF called "factor model" was proposed by "Team Opera Solutions" [8]. Because this term is too general and may easily be confused with factorization machines, we refer to it as "field-aware factorization machines" (FFMs) in this paper. The difference between PITF and FFM is that PITF considers three special fields including "user," "item," and "tag," while FFM is more general. Because [8] is about the overall solution for the competition, its discussion of FFM is limited. We can conclude the following results in [8]:

1. They use stochastic gradient method (SG) to solve the optimization problem. To avoid over-fitting, they only train with one epoch.
2. FFM performs the best among six models they tried.

In this paper, we aim to concretely establish FFM as an effective approach for CTR prediction. Our major results are as follows.

- Though FFM is shown to be effective in [8], this work may be the only published study of applying FFMs on CTR prediction problems. To further demonstrate the effectiveness of FFMs on CTR prediction, we present the use of FFM as our major model to win two world-wide CTR competitions hosted by Criteo and Avazu.
- We compare FFMs with two related models, Poly2 and FMs. We first discuss conceptually why FFMs might be better than them, and conduct experiments to see the difference in terms of accuracy and training time.
- We present techniques for training FFMs. They include an effective parallel optimization algorithm for FFMs and the use of early-stopping to avoid over-fitting.
- To make FFMs available for public use, we release an open source software.

This paper is organized as follows. Before we present FFMs and its implementation in Section 3, we discuss the two existing models Poly2 and FMs in Section 2. Experiments comparing FFMs with other models are in Section 4. Finally, conclusions and future directions are in Section 5.

Code used for experiments in this paper and the package LIBFFM are respectively available at:

http://www.csie.ntu.edu.tw/~cjlin/ffm/exps
http://www.csie.ntu.edu.tw/~cjlin/libffm

2. POLY2 AND FM

Chang et. al [4] have shown that a degree-2 polynomial mapping can often effectively capture the information of feature conjunctions. Further, they show that by applying a linear model on the explicit form of degree-2 mappings, the training and test time can be much faster than using kernel methods. This approach, referred to as Poly2, learns a weight for each feature pair:

$$\phi_{\text{Poly2}}(\boldsymbol{w}, \boldsymbol{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} w_{h(j_1,j_2)} x_{j_1} x_{j_2}, \tag{2}$$

where $h(j_1, j_2)$ is a function encoding j_1 and j_2 into a natural number. The complexity of computing (2) is $O(\bar{n}^2)$, where $\bar{n}$ is the average number of non-zero elements per instance.

FMs proposed in [6] implicitly learn a latent vector for each feature. Each latent vector contains k latent factors, where k is a user-specified parameter. Then, the effect of feature conjunction is modelled by the inner product of two latent vectors:

$$\phi_{\text{FM}}(\boldsymbol{w}, \boldsymbol{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} (\boldsymbol{w}_{j_1} \cdot \boldsymbol{w}_{j_2}) x_{j_1} x_{j_2}. \tag{3}$$

The number of variables is $n \times k$, so directly computing (3) costs $O(\bar{n}^2 k)$ time. Following [6], by re-writing (3) to

$$\phi_{\text{FM}}(\boldsymbol{w}, \boldsymbol{x}) = \frac{1}{2} \sum_{j=1}^{n} (\boldsymbol{s} - \boldsymbol{w}_j x_j) \cdot \boldsymbol{w}_j x_j,$$

where

$$\boldsymbol{s} = \sum_{j'=1}^{n} \boldsymbol{w}_{j'} x_{j'},$$

the complexity is reduced to $O(\bar{n}k)$.

Rendle [6] explains why FMs can be better than Poly2 when the data set is sparse. Here we give a similar illustration using the data set in Table 1. For example, there is only one negative training data for the pair (ESPN, Adidas). For Poly2, a very negative weight $w_{\text{ESPN,Adidas}}$ might be learned for this pair. For FMs, because the prediction of (ESPN, Adidas) is determined by $\boldsymbol{w}_{\text{ESPN}} \cdot \boldsymbol{w}_{\text{Adidas}}$, and because $\boldsymbol{w}_{\text{ESPN}}$ and $\boldsymbol{w}_{\text{Adidas}}$ are also learned from other pairs (e.g., (ESPN, Nike), (NBC, Adidas)), the prediction may be more accurate. Another example is that there is no training data for the pair (NBC, Gucci). For Poly2, the prediction on this pair is trivial, but for FMs, because $\boldsymbol{w}_{\text{NBC}}$ and $\boldsymbol{w}_{\text{Gucci}}$ can be learned from other pairs, it is still possible to do meaningful prediction.

Note that in Poly2, the naive way to implement $h(j_1, j_2)$ is to consider every pair of features as a new feature [4].[1] This approach requires the model as large as $O(n^2)$, which is usually impractical for CTR prediction because of very large n. Vowpal Wabbit (VW) [9], a widely used machine learning package, solves this problem by hashing j_1 and j_2.[2] Our implementation is similar to VW's approach. Specifically,

$$h(j_1, j_2) = (\frac{1}{2}(j_1 + j_2)(j_1 + j_2 + 1) + j_2) \bmod B,$$

where the model size B is a user-specified parameter.

In this paper, for the simplicity of formulations, we do not include linear terms and bias term. However, in Section 4, we include them for some experiments.

3. FFM

The idea of FFM originates from PITF [7] proposed for recommender systems with personalized tags. In PITF, they assume three available fields including User, Item, and Tag, and factorize (User, Item), (User, Tag), and (Item, Tag) in separate latent spaces. In [8], they generalize PITF for more fields (e.g., AdID, AdvertiserID, UserID, QueryID) and effectively apply it on CTR prediction. Because [7] aims at recommender systems and is limited to three specific fields (User, Item, and Tag), and [8] lacks detailed discussion on FFM, in this section we provide a more comprehensive study of FFMs on CTR prediction. For most CTR data sets like that in Table 1, "features" can be grouped into "fields." In our example, three features ESPN, Vogue, and NBC, belong to the field Publisher, and the other three features Nike, Gucci, and Adidas, belong to the field Advertiser. FFM is a variant of FM that utilizes this information. To explain how FFM works, we consider the following new example:

Clicked	Publisher (P)	Advertiser (A)	Gender (G)
Yes	ESPN	Nike	Male

Recall that for FMs, $\phi_{\text{FM}}(\boldsymbol{w}, \boldsymbol{x})$ is

$$\boldsymbol{w}_{\text{ESPN}} \cdot \boldsymbol{w}_{\text{Nike}} + \boldsymbol{w}_{\text{ESPN}} \cdot \boldsymbol{w}_{\text{Male}} + \boldsymbol{w}_{\text{Nike}} \cdot \boldsymbol{w}_{\text{Male}}.$$

[1] More precisely, [4] includes the original features as well, though we do not consider such a setting until the experiments.

[2] See http://github.com/JohnLangford/vowpal_wabbit/wiki/Feature-interactions for details.

	#variables	complexity
LM	n	$O(\bar{n})$
Poly2	B	$O(\bar{n}^2)$
FM	nk	$O(\bar{n}k)$
FFM	nfk	$O(\bar{n}^2k)$

Table 2: Comparison of the number of variables and the complexity for prediction among LM, Poly2, FM, and FFM.

In FMs, every feature has only one latent vector to learn the latent effect with any other features. Take ESPN as an example, $\boldsymbol{w}_{\text{ESPN}}$ is used to learn the latent effect with Nike ($\boldsymbol{w}_{\text{ESPN}} \cdot \boldsymbol{w}_{\text{Nike}}$) and Male ($\boldsymbol{w}_{\text{ESPN}} \cdot \boldsymbol{w}_{\text{Male}}$). However, because Nike and Male belong to different fields, the latent effects of (EPSN, Nike) and (EPSN, Male) may be different.

In FFMs, each feature has several latent vectors. Depending on the field of other features, one of them is used to do the inner product. In our example, $\phi_{\text{FFM}}(\boldsymbol{w}, \boldsymbol{x})$ is

$$\boldsymbol{w}_{\text{ESPN,A}} \cdot \boldsymbol{w}_{\text{Nike,P}} + \boldsymbol{w}_{\text{ESPN,G}} \cdot \boldsymbol{w}_{\text{Male,P}} + \boldsymbol{w}_{\text{Nike,G}} \cdot \boldsymbol{w}_{\text{Male,A}}.$$

We see that to learn the latent effect of (ESPN, NIKE), $\boldsymbol{w}_{\text{ESPN,A}}$ is used because Nike belongs to the field Advertiser, and $\boldsymbol{w}_{\text{Nike,P}}$ is used because ESPN belongs to the field Publisher. Again, to learn the latent effect of (EPSN, Male), $\boldsymbol{w}_{\text{ESPN,G}}$ is used because Male belongs to the field Gender, and $\boldsymbol{w}_{\text{Male,P}}$ is used because ESPN belongs to the field Publisher. Mathematically,

$$\phi_{\text{FFM}}(\boldsymbol{w}, \boldsymbol{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} (\boldsymbol{w}_{j_1,f_2} \cdot \boldsymbol{w}_{j_2,f_1}) x_{j_1} x_{j_2}, \quad (4)$$

where f_1 and f_2 are respectively the fields of j_1 and j_2. If f is the number of fields, then the number of variables of FFMs is nfk, and the complexity to compute (4) is $O(\bar{n}^2k)$. It is worth noting that in FFMs because each latent vector only needs to learn the effect with a specific field, usually

$$k_{\text{FFM}} \ll k_{\text{FM}}.$$

Table 2 compares the number of variables and the computational complexity of different models.

3.1 Solving the Optimization Problem

The optimization problem is the same as (1) except that $\phi_{\text{LM}}(\boldsymbol{w}, \boldsymbol{x})$ is replaced by $\phi_{\text{FFM}}(\boldsymbol{w}, \boldsymbol{x})$. Following [7, 8], we use stochastic gradient methods (SG). Recently, some adaptive learning-rate schedules such as [10, 11] have been proposed to boost the training process of SG. We use AdaGrad [10] because [12] has shown its effectiveness on matrix factorization, which is a special case of FFMs.

At each step of SG a data point $(y, \boldsymbol{x})$ is sampled for updating $\boldsymbol{w}_{j_1,f_2}$ and $\boldsymbol{w}_{j_2,f_1}$ in (4). Note that because $\boldsymbol{x}$ is highly sparse in our application, we only update dimensions with non-zero values. First, the sub-gradients are

$$\boldsymbol{g}_{j_1,f_2} \equiv \nabla_{\boldsymbol{w}_{j_1,f_2}} f(\boldsymbol{w}) = \lambda \cdot \boldsymbol{w}_{j_1,f_2} + \kappa \cdot \boldsymbol{w}_{j_2,f_1} x_{j_1} x_{j_2}, \quad (5)$$

$$\boldsymbol{g}_{j_2,f_1} \equiv \nabla_{\boldsymbol{w}_{j_2,f_1}} f(\boldsymbol{w}) = \lambda \cdot \boldsymbol{w}_{j_2,f_1} + \kappa \cdot \boldsymbol{w}_{j_1,f_2} x_{j_1} x_{j_2}, \quad (6)$$

where

$$\kappa = \frac{\partial \log(1 + \exp(-y\phi_{\text{FFM}}(\boldsymbol{w}, \boldsymbol{x})))}{\partial \phi_{\text{FFM}}(\boldsymbol{w}, \boldsymbol{x})} = \frac{-y}{1 + \exp(y\phi_{\text{FFM}}(\boldsymbol{w}, \boldsymbol{x}))}.$$

Algorithm 1 Training FFM using SG

1: Let $G \in R^{n \times f \times k}$ be a tensor of all ones
2: Run the following loop for t epochs
3: **for** $i \in \{1, \cdots, m\}$ **do**
4: Sample a data point $(y, \boldsymbol{x})$
5: caclulate κ
6: **for** $j_1 \in$ non-zero terms in $\{1, \cdots, n\}$ **do**
7: **for** $j_2 \in$ non-zero terms in $\{j_1 + 1, \cdots, n\}$ **do**
8: calculate sub-gradient by (5) and (6)
9: **for** $d \in \{1, \cdots, k\}$ **do**
10: Update the gradient sum by (7) and (8)
11: Update model by (9) and (10)

Second, for each coordinate $d = 1, \ldots, k$, the sum of squared gradient is accumulated:

$$(G_{j_1,f_2})_d \leftarrow (G_{j_1,f_2})_d + (g_{j_1,f_2})_d^2 \quad (7)$$

$$(G_{j_2,f_1})_d \leftarrow (G_{j_2,f_1})_d + (g_{j_2,f_1})_d^2 \quad (8)$$

Finally, $(w_{j_1,f_2})_d$ and $(w_{j_2,f_1})_d$ are updated by:

$$(w_{j_1,f_2})_d \leftarrow (w_{j_1,f_2})_d - \frac{\eta}{\sqrt{(G_{j_1,f_2})_d}} (g_{j_1,f_2})_d \quad (9)$$

$$(w_{j_2,f_1})_d \leftarrow (w_{j_2,f_1})_d - \frac{\eta}{\sqrt{(G_{j_2,f_1})_d}} (g_{j_2,f_1})_d, \quad (10)$$

where η is a user-specified learning rate. The initial values of $\boldsymbol{w}$ are randomly sampled from a uniform distribution between $[0, 1/\sqrt{k}]$. The initial values of G are set to one in order to prevent a large value of $(G_{j_1,f_2})_d^{-\frac{1}{2}}$. The overall procedure is presented in Algorithm 1.

Empirically, we find that normalizing each instance to have the unit length makes the test accuracy slightly better and insensitive to parameters.

3.2 Parallelization on Shared-memory Systems

Modern computers are widely equipped with multi-core CPUs. If these cores are fully utilized, the training time can be significantly reduced. Many parallelization approaches for SG have been proposed. In this paper, we apply HOGWILD! [13], which allows each thread to run independently without any locking. Specifically, the **for** loop at line 3 of Algorithm 1 is parallelized.

In Section 4.4 we run extensive experiments to investigate the effectiveness of parallelization.

3.3 Adding Field Information

Consider the widely used LIBSVM data format:

```
label  feat1:val1  feat2:val2  ···,
```

where each (feat, val) pair indicates feature index and value. For FFMs, we extend the above format to

```
label  field1:feat1:val1  field2:feat2:val2  ···
```

That is, we must assign the corresponding field to each feature. The assignment is easy on some kinds of features, but may not be possible for some others. We discuss this issue on three typical classes of features.

Categorical Features

For linear models, a categorical feature is commonly transformed to several binary features. For a data instance

Yes P:ESPN A:Nike G:Male,

we generate the following LIBSVM format.

Yes P-ESPN:1 A-Nike:1 G-Male:1

Note that according to the number of possible values in a categorical feature, the same number of binary features are generated and every time only one of them has the value 1. In the LIBSVM format, features with zero values are not stored. We apply the same setting to all models, so in this paper, every categorical feature is transformed to several binary ones. To add the field information, we can consider each category as a field. Then the above instance becomes

Yes P:P-ESPN:1 A:A-Nike:1 G:G-Male:1.

Numerical Features

Consider the following example to predict if a paper will be accepted by a conference. We use three numerical features "accept rate of the conference (AR)," "h-index of the author (Hidx)," and "number of citations of the author (Cite):"

Accepted	AR	Hidx	Cite
Yes	45.73	2	3
No	1.04	100	50,000

There are two possible ways to assign fields. A naive way is to treat each feature as a dummy field, so the generated data is:

Yes AR:AR:45.73 Hidx:Hidx:2 Cite:Cite:3

However, the dummy fields may not be informative because they are merely duplicates of features.

Another possible way is to discretize each numerical feature to a categorical one. Then, we can use the same setting for categorical features to add field information. The generated data looks like:

Yes AR:45:1 Hidx:2:1 Cite:3:1,

where the AR feature is rounded to an integer. The main drawback is that usually it is not easy to determine the best discretization setting. For example, we may transform 45.73 to "45.7," "45," "40," or even "int(log(45.73))." In addition, we may lose some information after discretization.

Single-field Features

On some data sets, all features belong to a single field and hence it is meaningless to assign fields to features. Typically this situation happens on NLP data sets. Consider the following example of predicting if a sentence expresses a good mood or not:

good mood	sentence
Yes	`Hooray! Our paper is accepted!`
No	`Well, our paper is rejected..`

In this example the only field is "sentence." If we assign this field to all words, then FFMs is reduced to FMs. Readers may ask about assigning dummy fields as we do for numerical features. Recall that the model size of FFMs is $O(nfk)$. The use of dummy fields is impractical because $f = n$ and n is often huge.

4. EXPERIMENTS

In this section, we first provide the details about the experimental setting in Section 4.1. Then, we investigate the impact of parameters. We find that unlike LM or Poly2, FFM is sensitive to the number of epochs. Therefore, in Section 4.3, we discuss this issue in detail before proposing an early stopping trick. The speedup of parallelization is studied in Section 4.4.

After checking various properties of FFMs, in Sections 4.5-4.6, we compare FFMs with other models including Poly2 and FMs. They are all implemented by the same SG method, so besides accuracy we can fairly compare their training time. Further in the comparison we include state-of-the-art packages LIBLINEAR [14] and LIBFM [15] for training LM/Poly2 and FMs, respectively.

4.1 Experiment Settings

Data Sets

We mainly consider two CTR sets `Criteo` and `Avazu` from Kaggle competitions,[3] though in Section 4.6 more sets are considered. For feature engineering, we mainly apply our winning solution but remove complicated components.[4] For example, our winning solution for `Avazu` includes the ensemble of 20 models, but here we only use the simplest one. For other details please check our experimental code. A hashing trick is applied to generate 10^6 features. The statistics of the two data sets are:

Data Set	# instances	# features	# fields
`Criteo`	45,840,617	10^7	39
`Avazu`	40,428,967	10^7	33

For both data sets, the labels in the test sets are not publicly available, so we split the available data to two sets for training and validation. The data split follows from how test sets are obtained: For `Criteo`, the last 6,040,618 lines are used as the validation set; for `Avazu`, we select the last 4,218,938 lines. We use the following terms to represent different sets of a problem.

- `Va`: The validation set mentioned above.
- `Tr`: The new training set after excluding the validation set from the original training data.
- `TrVa`: The original training set.
- `Te`: The original test set. The labels are not released, so we must submit our prediction to the original evaluation systems to get the score. To avoid over-fitting the test set, the competition organizers divide this data set to two subsets "public set" on which the score is visible during the competition and "private set" on which the score is available after the end of competition. The final rank is determined by the private set.

For example, `CriteoVa` means the validation set from `Criteo`.

[3]Criteo Display Advertising Challenge: http://www.kaggle.com/c/criteo-display-ad-challenge. Avazu Click-Through Rate Prediction: http://www.kaggle.com/c/avazu-ctr-prediction.

[4]The code and documents of our winning solution to the two competitions can be found in http://github.com/guestwalk/kaggle-2014-criteo and http://github.com/guestwalk/kaggle-avazu

Platform

All experiments are conducted on a Linux workstation with 12 physical cores on two Intel Xeon E5-2620 2.0GHz processors and 128 GB memory.

Evaluation

Depending on the model, we change $\phi(\boldsymbol{w}, \boldsymbol{x})$ in (1) to $\phi_{\text{LM}}(\boldsymbol{w}, \boldsymbol{x})$, $\phi_{\text{Poly2}}(\boldsymbol{w}, \boldsymbol{x})$, $\phi_{\text{FM}}(\boldsymbol{w}, \boldsymbol{x})$, or $\phi_{\text{FFM}}(\boldsymbol{w}, \boldsymbol{x})$ introduced in Sections 1-3. For the evaluation criterion, we consider the logistic loss defined as

$$\text{logloss} = \frac{1}{m}\sum_{i=1}^{m}\log(1+\exp(-y_i\phi(\boldsymbol{w}, \boldsymbol{x}_i))),$$

where m is the number of test instances.

Implementation

We implement LMs, Poly2, FMs, and FFMs all in C++. For FMs and FFMs, we use SSE instructions to boost the efficiency of inner products. The parallelization discussed in Section 3.2 is implemented by OpenMP [16]. Our implementations include linear terms and bias term as they improve performance in some data sets. These terms should be used in general as we seldom see them to be harmful.

Note that for code extensibility the field information is stored regardless of the model used. For non-FFM models, the implementation may become slightly faster by a simpler data structure without field information, but our conclusions from experiments should remain the same.

4.2 Impact of Parameters

We conduct experiments to investigate the impact of k, λ, and η. The results can be found in Figure 1. Regarding the parameter k, results in Figure 1a show that it does not affect the logloss much. In Figure 1b, we present the relationship between λ and logloss. If λ is too large, the model is not able to achieve a good performance. On the contrary, with a small λ, the model gets better results, but it easily overfits the data. We observe that the training logloss keeps decreasing. For the parameter η, Figure 1c shows that if we apply a small η, FFMs will obtain its best performance slowly. However, with a large η, FFMs are able to quickly reduce the logloss, but then over-fitting occurs. From the results in Figures 1b and 1c, there is a need of early-stopping that will be discussed in Section 4.3.

4.3 Early Stopping

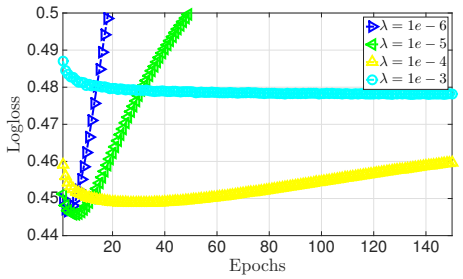
Early stopping, which terminates the training process before reaching the best result on training data, can be used to avoid over-fitting for many machine learning problems [17, 18, 19]. For FFM, the strategy we use is:

1. Split the data set into a training set and a validation set.
2. At the end of each epoch, use the validation set to calculate the loss.
3. If the loss goes up, record the number of epochs. Stop or go to step 4.
4. If needed, use the full data set to re-train a model with the number of epochs obtained in step 3.

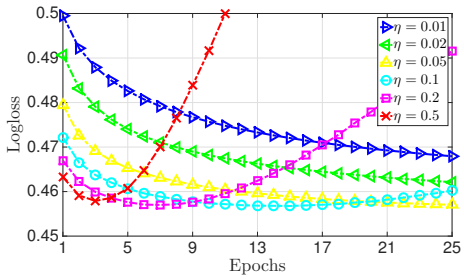
A difficulty in applying early stopping is that the logloss is sensitive to the number of epochs. Then the best epoch on the validation set may not be the best one on the test set. We have tried other approaches to avoid the overfitting such as lazy update[5] and ALS-based optimization methods. However, results are not as successful as that by early stopping of using a validation set.

k	time	logloss
1	27.236	0.45773
2	26.384	0.45715
4	27.875	0.45696
8	40.331	0.45690
16	70.164	0.45725

(a) The average running time (in seconds) per epoch and the best logloss with different values of k. Because we use SSE instructions, the running time of $k = 1, 2, 4$ is roughly the same.



(b) The impact of λ



(c) The impact of η

Figure 1: The impact of λ, η, and k on FFMs. To make experiments faster, we randomly select 10% instances from `CriteoTr` and `CriteoVa` as the training and the test sets, respectively.

4.4 Speedup

Because the parallelization of SG may cause a different convergence behavior, we experiment with different numbers of threads in Figure 2. Results show that our parallelization still leads to similar convergence behavior. With this property we can define the speedup as:

$$\frac{\text{Running time of one epoch with a single thread}}{\text{Running time of one epoch with multiple threads}}.$$

The result in Figure 3 shows a good speedup when the number of threads is small. However, if many threads are used, the speedup does not improve much. An explanation is that if two or more threads attempt to access the same memory address, one must wait for its term. This kind of conflicts can happen more often when more threads are used.

[5] http://blog.smola.org/post/943941371/lazy-updates-for-generic-regularization-in-sgd

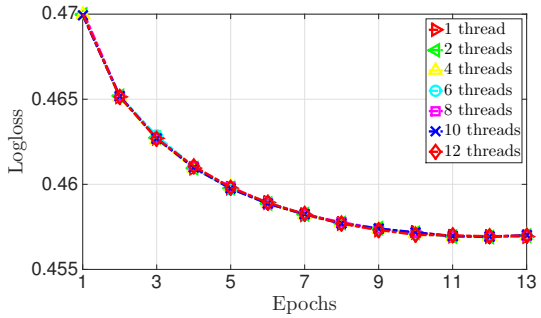


Figure 2: The convergence of using different number of threads.

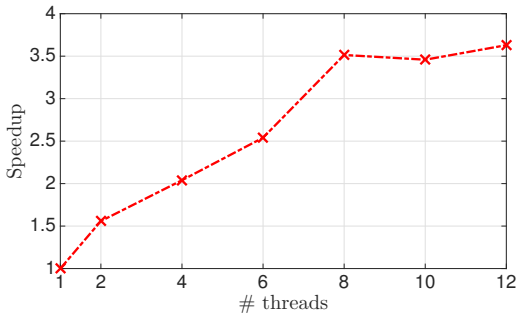


Figure 3: The speedup of using multi-threading. We respectively use `CriteoTr` and `CriteoVa` as the training and the test sets.

4.5 Comparison with LMs, Poly2, and FMs on Two CTR Competition Data Sets

To have a fair comparison, we implement the same SG method for LMs, Poly2, FMs, and FFMs. Further, we compare with two state-of-the-art packages:

- LIBLINEAR: a widely used package for linear models. For L2-regularized logistic regression, it implements two optimization methods: Newton method to solve the primal problem, and coordinate descent (CD) method to solve the dual problem. We used both for checking how optimization methods affect the performance; see the discussion in the end of this sub-section. Further, the existing Poly2 extension of LIBLINEAR does not support the hashing trick,[6] so we conduct suitable modifications and denote it as LIBLINEAR-Hash in this paper.
- LIBFM: As a widely used library for factorization machines, it supports three optimization approaches including stochastic gradient method (SG), alternating least squares (ALS), and Markov Chain Monte Carlo (MCMC). We tried all of them and found that ALS is significantly better than the other two in terms of logloss. Therefore, we consider ALS in our experiments.

For the parameters in all models, from a grid of points we select those that lead to the best performance on the validation sets. Every optimization algorithm needs a stopping condition; we use the default setting for Newton method and coordinate descent (CD) method by LIBLINEAR. For each of the other models, we need a validation set to check which iteration leads to the best validation score. After we obtain the best number of iterations, we re-train the model up to that iteration. Results on `Criteo` and `Avazu` with the list of parameters used can be found in Table 3. Clearly, FFMs outperform other models in terms of logloss, but it also requires longer training time than LMs and FMs. On the other end, though the logloss of LMs is worse than other models, it is significantly faster. These results show a clear trade-off between logloss and speed. Poly2 is the slowest among all models. The reason might be the expensive computation of (2). FM is a good balance between logloss and speed.

For LIBFM, it performs closely to our implementation of FMs in terms of logloss on `Criteo`.[7] However, we see that our implementation is significantly faster. We provide three possible reasons:

- The ALS algorithm used by LIBFM is more complicated than the SG algorithm we use.
- We use an adaptive learning rate strategy in SG.
- We use SSE instructions to boost inner product operations.

Because logistic regression is a convex problem, ideally, for either LM or Poly2, the three optimization methods (SG, Newton, and CD) should generate exactly the same model if they converge to the global optimum. However, practically results are slightly different. In particular, LM by SG is better than the two LIBLINEAR-based models on Avazu. In our implementation, LM via SG only loosely solves the optimization problem. Our experiments therefore indicate that the stopping condition of optimization methods can affect the performance of the resulting model even if the problem is convex.

4.6 Comparison on More Data Sets

In the previous section we focus on two competition data sets, but it is important to see how FFMs perform on other data sets. To answer this question, we consider more data sets for the comparison, where most of them are not CTR data. Note that following the discussion in Section 3.3, we do not consider data sets with single-field features. The reason is that depending on how we assign fields, FFMs either become equivalent to FMs, or generate a huge model.

Here we briefly introduce the data sets used.

- `KDD2010-bridge`:[8] This data set includes both numerical and categorical features.
- `KDD2012`:[9] This set contains both numerical and categorical features. Because our evaluation is logloss, we transform the original target value "number of clicks" to a binary value "clicked or not."
- `cod-rna`:[10] This set contains only numerical features.
- `ijcnn`:[10] This set contains only numerical features.

[6] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#fast_training_testing_for_polynomial_mappings_of_data

[7] The performance of LIBFM on `AvazuVa` is as good as the FM we have implemented, but the performance on `AvazuTe` is poor. It is not entirely clear what happened, so further investigation is needed.

[8] http://pslcdatashop.web.cmu.edu/KDDCup/downloads.jsp

[9] http://www.kddcup2012.org/c/kddcup2012-track2/data

[10] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html

Model and implementation	parameters	training time (seconds)	public set logloss	public set rank	private set logloss	private set rank
LM-SG	$\eta = 0.2, \lambda = 0, t = 13$	527	0.46262	93	0.46224	91
LM-LIBLINEAR-CD	$s = 7, c = 2$	1,417	0.46239	91	0.46201	89
LM-LIBLINEAR-Newton	$s = 0, c = 2$	7,164	0.46602	225	0.46581	222
Poly2-SG	$\eta = 0.2, \lambda = 0, B = 10^7, t = 10$	12,064	0.44973	14	0.44956	14
Poly2-LIBLINEAR-Hash-CD	$s = 7, c = 2$	24,771	0.44893	13	0.44873	13
FM	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 40, t = 8$	2,022	0.44930	14	0.44922	14
FM	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 100, t = 9$	4,020	0.44867	11	0.44847	11
LIBFM	$\lambda = 40, k = 40, t = 20$	23,700	0.45012	14	0.45000	15
LIBFM	$\lambda = 40, k = 40, t = 50$	131,000	0.44904	14	0.44887	14
LIBFM	$\lambda = 40, k = 100, t = 20$	54,320	0.44853	11	0.44834	11
LIBFM	$\lambda = 40, k = 100, t = 50$	398,800	0.44794	9	0.44778	8
FFM	$\eta = 0.2, \lambda = 2 \times 10^{-5}, k = 4, t = 9$	6,587	0.44612	3	0.44603	3

(a) `Criteo`

Model and implementation	parameters	training time (seconds)	public set logloss	public set rank	private set logloss	private set rank
LM-SG	$\eta = 0.2, \lambda = 0, t = 10$	164	0.39018	57	0.38833	64
LM-LIBLINEAR-CD	$s = 7, c = 1$	417	0.39131	115	0.38944	119
LM-LIBLINEAR-Newton	$s = 0, c = 1$	650	0.39269	182	0.39079	183
Poly2-SG	$\eta = 0.2, \lambda = 0, B = 10^7, t = 10$	911	0.38554	10	0.38347	10
Poly2-LIBLINEAR-Hash-CD	$s = 7, c = 1$	1,756	0.38516	10	0.38303	9
Poly2-LIBLINEAR-Hash-Newton	$s = 0, c = 1$	27,292	0.38598	11	0.38393	11
FM	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 40, t = 8$	574	0.38621	11	0.38407	11
FM	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 100, t = 9$	1,277	0.38740	17	0.38531	15
LIBFM	$\lambda = 40, k = 40, t = 20$	18,712	0.39137	122	0.38963	127
LIBFM	$\lambda = 40, k = 40, t = 50$	41,720	0.39786	935	0.39635	943
LIBFM	$\lambda = 40, k = 100, t = 20$	39,719	0.39644	747	0.39470	755
LIBFM	$\lambda = 40, k = 100, t = 50$	91,210	0.40740	1,129	0.40585	1,126
FFM	$\eta = 0.2, \lambda = 2 \times 10^{-5}, k = 4, t = 4$	340	0.38411	6	0.38223	6

(b) `Avazu`

Table 3: Comparison among models and implementations on data sets `Criteo` and `Avazu`. The training sets used here are `CriteoTrVa` and `AvazuTrVa`, and the test sets used here are `CriteoTe` and `AvazuTe`. For all experiments, a single thread is used. The public set is around 20% of the test data, while the private set contains the rest. For Criteo, we do not list the result of Poly2-LIBLINEAR-Hash-Newton, because the experiment does not finish after more than 10 days. Note that the we use different stopping conditions for different algorithms, so the training time is only for reference.

- `phishing`:[11] This set contains only categorical features.
- `adult`:[12] This data set includes both numerical and categorical features.

For `KDD2010-bridge`, `KDD2012`, and `adult`, we simply discretize all numerical features into 29, 13, and 94 bins respectively. For `cod-rna` and `ijcnn`, where features are all numerical, we try both approaches mentioned in Section 3.3 to obtain field information: applying dummy fields and discretization.

For the parameter selection, we follow the same procedure in Section 4.5. We split each set into training, validation, and test sets; then for predicting the test set, we use the model trained with parameters that achieve the best logloss on the validation set.

The statistics and experimental results of each data set are described in Table 4. FFMs significantly outperform other models on `KDD2010-bridge` and `KDD2012`. The common characteristic among these data sets are:

- Most features are categorical.
- The resulting set is highly sparse after transforming categorical features into many binary features.

However, on `phishing` and `adult`, FFM is not significantly better. For `phishing`, the reason might be that the data is not sparse so FFM, FM, and Poly2 have close performance; for `adult`, it seems feature conjunction is not useful because all models perform similarly to the linear model.

When a data set contains only numerical features, FFMs may not have an obvious advantage. If we use dummy fields, then FFMs do not out-perform FMs, a result indicating that the field information is not helpful. On the other hand, if we discretize numerical features, though FFMs is the best among all models, the performance is much worse than that of using dummy fields. We summarize a guideline of applying FFMs on different kinds of data sets:

- FFMs should be effective for data sets that contain categorical features and are transformed to binary features.
- If the transformed set is not sparse enough, FFMs seem to bring less benefit.
- It is more difficult to apply FFMs on numerical data sets.

5. CONCLUSIONS AND FUTURE WORKS

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

[12] http://archive.ics.uci.edu/ml/datasets/Adult

	statistics			logloss			
Data set	# instances	# features	# fields	LM	Poly2	FM	FFM
KDD2010-bridge	20,012,499	651,166	9	0.27947	0.2622	0.26372	0.25639
KDD2012	149,639,105	54,686,452	11	0.15069	0.15099	0.15004	0.14906
phishing	11,055	100	30	0.14211	0.11512	0.09229	0.1065
adult	48,842	308	14	0.3097	0.30655	0.30763	0.30565
cod-rna (dummy fields)	331,152	8	8	0.13829	0.12874	0.12580	0.12914
cod-rna (discretization)	331,152	2,296	8	0.16455	0.17576	0.16570	0.14993
ijcnn (dummy fields)	141,691	22	22	0.20093	0.08981	0.07087	0.0692
ijcnn (discretization)	141,691	69,867	22	0.21588	0.24578	0.20223	0.18608

Table 4: Comparison between LM, Poly2, FM, and FFMs. The best logloss is underlined.

In this paper we discuss efficient implementations of FFMs. We demonstrate that for certain kinds of data sets, FFMs outperform three well-known models, LM, Poly2, and FM, in terms of logloss, with a cost of longer training time.

For the future work, the over-fitting problem discussed in Section 4.3 is an issue that we plan to investigate. Besides, for the ease of implementation we use SG as the optimization method. It is interesting to see how other optimization methods (e.g., Newton method) work on FFMs.

Acknowledgements

This work was supported in part by MOST via grants 104-2221-E-002-047-MY3 and 104-2622-E-002-012-CC2 and MOE of Taiwan via grant 105R7872.

6. REFERENCES

[1] O. Chapelle, E. Manavoglu, and R. Rosales, "Simple and scalable response prediction for display advertising," *ACM Transactions on Intelligent Systems and Technology*, vol. 5, no. 4, pp. 61:1–61:34, 2015.

[2] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica, "Ad click prediction: a view from the trenches," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2013.

[3] M. Richardson, E. Dominowska, and R. Ragno, "Predicting clicks: estimating the click-through rate for new ADs," in *Proceedings of the 16th international conference on World Wide Web*, 2007.

[4] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin, "Training and testing low-degree polynomial data mappings via linear SVM," *Journal of Machine Learning Research*, vol. 11, pp. 1471–1490, 2010.

[5] T. Kudo and Y. Matsumoto, "Fast methods for kernel-based text analysis," in *Proceedings of the 41st Annual Meeting of the Association of Computational Linguistics (ACL)*, 2003.

[6] S. Rendle, "Factorization machines," in *Proceedings of IEEE International Conference on Data Mining (ICDM)*, pp. 995–1000, 2010.

[7] S. Rendle and L. Schmidt-Thieme, "Pairwise interaction tensor factorization for personalized tag recommendation," in *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining (WSDM)*, pp. 81–90, 2010.

[8] M. Jahrer, A. Töscher, J.-Y. Lee, J. Deng, H. Zhang, and J. Spoelstra, "Ensemble of collaborative filtering and feature engineered model for click through rate prediction," in *KDD Cup 2012 Workshop*, ACM, 2012.

[9] J. Langford, L. Li, and A. Strehl, "Vowpal Wabbit," 2007. https://github.com/JohnLangford/vowpal_wabbit/wiki.

[10] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[11] H. B. McMahan, "Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[12] W.-S. Chin, Y. Zhuang, Y.-C. Juan, and C.-J. Lin, "A learning-rate schedule for stochastic gradient methods to matrix factorization," in *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, 2015.

[13] F. Niu, B. Recht, C. Ré, and S. J. Wright, "HOGWILD!: a lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems 24* (J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, eds.), pp. 693–701, 2011.

[14] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: a library for large linear classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[15] S. Rendle, "Factorization machines with libFM," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 3, no. 3, p. 57, 2012.

[16] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, pp. 46–55, 1998.

[17] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006.

[18] G. Raskutti, M. J. Wainwright, and B. Yu, "Early stopping and non-parametric regression: An optimal data-dependent stopping rule," *Journal of Machine Learning Research*, vol. 15, pp. 335–366, 2014.

[19] T. Zhang and B. Yu, "Boosting with early stopping: convergence and consistency," *The Annals of Statistics*, vol. 33, no. 4, pp. 1538–1579, 2005.