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 \((y_i, \mathbf{x}_i)\), \(i = 1, \ldots, m\), where \(y_i\) is the label and \(\mathbf{x}_i\) is an \(n\)-dimensional feature vector, the model \(\mathbf{w}\) is obtained by solving the following optimization problem.

\[
\min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^{m} \log(1 + \exp(-y_i \phi_{LM}(\mathbf{w}, \mathbf{x}_i))).
\]

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

\[
\phi_{LM}(\mathbf{w}, \mathbf{x}) = \mathbf{w} \cdot \mathbf{x}.
\]

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]:

<table>
<thead>
<tr>
<th>Publisher</th>
<th>Advertiser</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPN</td>
<td>Nike</td>
</tr>
<tr>
<td>ESPN</td>
<td>Nike</td>
</tr>
<tr>
<td>Adidas</td>
<td>Nike</td>
</tr>
<tr>
<td>Vogue</td>
<td>Nike</td>
</tr>
<tr>
<td>Vogue</td>
<td>Adidas</td>
</tr>
<tr>
<td>NBC</td>
<td>Nike</td>
</tr>
<tr>
<td>NBC</td>
<td>Gucci</td>
</tr>
<tr>
<td>NBC</td>
<td>Adidas</td>
</tr>
</tbody>
</table>

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

RecSys ’16, September 15-19, 2016, Boston, MA, USA
© 2016 ACM. ISBN 978-1-4503-4035-9/16/09. . . $15.00
DOI: http://dx.doi.org/10.1145/2959100.2959134
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:

\[
\phi_{\text{Poly2}}(\mathbf{w}, \mathbf{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} w_{(j_1, j_2)} x_{j_1} x_{j_2},
\]

(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 \(O(n^2)\), where \(n\) is the average number of non-zero elements per instance.

FFMs 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 modeled by the inner product of two latent vectors:

\[
\phi_{\text{FM}}(\mathbf{w}, \mathbf{x}) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} (w_{j_1} \cdot w_{j_2}) x_{j_1} x_{j_2},
\]

(3)

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

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

where

\[
s = \sum_{j'=1}^{n} w_{jj'} x_{j'},
\]

the complexity is reduced to \(O(nk)\).

Rendle [6] explains why FFMs 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 \(w_{\text{ESPN}, \text{Adidas}}\) might be learned for this pair. For FFMs, because the prediction of (ESPN, Adidas) is determined by \(w_{\text{ESPN}} \cdot w_{\text{Adidas}}\), and because \(w_{\text{ESPN}}\) and \(w_{\text{Adidas}}\) 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 FFMs, because \(w_{\text{NBC}}\) and \(w_{\text{Gucci}}\) 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].

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 \(j_2\).

Our implementation is similar to VW’s approach. Specifically,

\[
h(j_1, j_2) = \frac{1}{2} (j_1 + j_2)(j_1 + j_2 + 1) + j_2 \mod B,
\]

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 FFMs, 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:

<table>
<thead>
<tr>
<th>Clicked</th>
<th>Publisher (P)</th>
<th>Advertiser (A)</th>
<th>Gender (G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>ESPN</td>
<td>Nike</td>
<td>Male</td>
</tr>
</tbody>
</table>

Recall that for FFMs, \(\phi_{\text{FM}}(\mathbf{w}, \mathbf{x})\) is

\[
w_{\text{ESPN}} \cdot w_{\text{Nike}} + w_{\text{ESPN}} \cdot w_{\text{Male}} + w_{\text{Nike}} \cdot w_{\text{Male}}.
\]

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

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

<table>
<thead>
<tr>
<th>Model</th>
<th>#variables</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM</td>
<td>$n$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Poly2</td>
<td>$B$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>FM</td>
<td>$nk$</td>
<td>$O(nk)$</td>
</tr>
<tr>
<td>FFM</td>
<td>$nk$</td>
<td>$O(n^2k)$</td>
</tr>
</tbody>
</table>

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

$$\phi_{FFM}(w, x) = \sum_{j_1=1}^{n} \sum_{j_2=j_1+1}^{n} (w_{j_1, f_1} \cdot w_{j_2, f_2}) x_{j_1} x_{j_2},$$

(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 $O(n^2k)$. It is worth noting that in FFMs because each latent vector only needs to learn the effect with one specific field, usually $k_{FFM} \ll k_{FM}$.

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 $\phi_{LM}(w, x)$ is replaced by $\phi_{FFM}(w, x)$. 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 $(y, x)$ is sampled for updating $w_{j_1, f_2}$ and $w_{j_2, f_1}$ in (4). Note that because $x$ is highly sparse in our application, we only update dimensions with non-zero values. First, the sub-gradients are

$$g_{j_1, f_2} = \nabla w_{j_1, f_2} f(w) = \lambda \cdot w_{j_1, f_2} + \kappa \cdot w_{j_2, f_1} x_{j_1} x_{j_2},$$

(5)

$$g_{j_2, f_1} = \nabla w_{j_2, f_1} f(w) = \lambda \cdot w_{j_2, f_1} + \kappa \cdot w_{j_1, f_2} x_{j_1} x_{j_2},$$

(6)

where

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

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

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

$$\cdot (w_{j_1, f_2})_d \leftarrow (w_{j_1, f_2})_d - \eta \frac{\nabla (G_{j_1, f_2})_d}{\sqrt{(G_{j_1, f_2})_d}} (G_{j_1, f_2})_d,$$

(9)

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

$$(G_{j_1, f_2})_d \leftarrow (G_{j_1, f_2})_d + (g_{j_1, f_2})_d$$

$$(G_{j_2, f_1})_d \leftarrow (G_{j_2, f_1})_d + (g_{j_2, f_1})_d$$

(7)

(8)

where $\eta$ is a user-specified learning rate. The initial values of $w$ 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 $(G_{j_1, f_2})_d^{1/2}$. The overall procedure is presented in Algorithm 1.

Algorithm 1: Training FFN 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 $i \in \{1, \ldots, m\}$ do
4. Sample a data point $(y, x)$
5. $\kappa$ = calculate $\kappa$
6. for $j_1 \in \{1, \ldots, n\}$ do
7. for $j_2 \in \{1, \ldots, n\}$ do
8. Sample a data point $(y, x)$
9. $\kappa$ = calculate sub-gradient by (5) and (6)
10. for $d \in \{1, \ldots, k\}$ do
11. Update the gradient sum by (7) and (8)
12. Update model by (9) and (10)

### 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:

```
1
label fea1:val1 fea2:val2 ... 
```

where each (feat, val) pair indicates feature index and value.

For FFMs, we extend the above format to

```
1
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
well, our paper is rejected.

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-ESPN:1 A-Nike:1 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):”

<table>
<thead>
<tr>
<th>Accepted</th>
<th>AR</th>
<th>Hidx</th>
<th>Cite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>45.73</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>No</td>
<td>1.04</td>
<td>100</td>
<td>50,000</td>
</tr>
</tbody>
</table>

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,” “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:

<table>
<thead>
<tr>
<th>good mood</th>
<th>sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Hooray! Our paper is accepted!</td>
</tr>
<tr>
<td>No</td>
<td>Well, our paper is rejected.</td>
</tr>
</tbody>
</table>

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 Kaggel competitions, though in Section 4.6 more sets are considered. For feature engineering, we mainly apply our winning solution but remove complicated components. 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:

<table>
<thead>
<tr>
<th>Data Set</th>
<th># instances</th>
<th># features</th>
<th># fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>Criteo</td>
<td>45,840,617</td>
<td>10^7</td>
<td>39</td>
</tr>
<tr>
<td>Avazu</td>
<td>40,428,967</td>
<td>10^7</td>
<td>33</td>
</tr>
</tbody>
</table>

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 4,218,939 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.


4The 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 $\phi(w, x)$ in (1) to $\phi_{L}(w, x)$, $\phi_{Poly2}(w, x)$, $\phi_{FM}(w, x)$, or $\phi_{FFM}(w, x)$ introduced in Sections 1-3. For the evaluation criterion, we consider the logistic loss defined as

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

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 $k$, $\lambda$, and $\eta$. 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 $\lambda$ and logloss. If $\lambda$ is too large, the model is not able to achieve a good performance. On the contrary, with a small $\lambda$, the model gets better results, but it easily overfits the data. We observe that the training logloss keeps decreasing. For the parameter $\eta$, Figure 1c shows that if we apply a small $\eta$, FFMs will obtain its best performance slowly. However, with a large $\eta$, 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

![Figure 1: The impact of $\lambda$, $\eta$, 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.](http://blog.smola.org/post/943941371/lazy-updates-for-generic-regularization-in-sgd)
mappings
data polynomial of for testing

Logloss training condition; we use the default setting for Newton method and
dagation sets. Every optimization algorithm needs a stopping
select those that lead to the best performance on the vali-

• 

• 

Figure 2: The convergence of using different number of

Figure 3: The speedup of using multi-threading. We respec-

catively use CriteoTr and CriteoVa as the training and the
test sets.

4.5 Comparison with LMs, Poly2, and FMs

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

• LIBLINEAR: a widely used package for linear models. For
L2-regularized logistic regression, it implements two opti-
mization methods: Newton method to solve the primal
problem, and coordinate descent (CD) method to solve
the dual problem. We used both for checking how opti-
mization methods affect the performance; see the discus-
sion in the end of this sub-section. Further, the exist-
ing Poly2 extension of LIBLINEAR does not support the
hashing trick, so we conduct suitable modifications and
denote it as LIBLINEAR-Hash in this paper.

• LIBFM: As a widely used library for factorization ma-
achines, it supports three optimization approaches includ-
ing 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 bet-
ter 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 vali-
dation 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 re-
quires 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
FFMs 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 opera-
tions.

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 catego-
rical features. Because our evaluation is logloss, we trans-
form the original target value “number of clicks” to a bi-
ary value “clicked or not.” Further, we sub-sample 10%
positive instances because the data set is huge and con-
tains much more negative instances than positive ones.
• cod-rna:10 This set contains only numerical features.

7The 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.
9http://www.kddcup2012.org/c/kddcup2012-track2/data
10http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
 binary.html
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:

- ijcnn:10 This set contains only numerical features.
- 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:

- ijcnn:10 This set contains only numerical features.
- 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:

- ijcnn:10 This set contains only numerical features.
- 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:

- ijcnn:10 This set contains only numerical features.
- phishing:11 This set contains only categorical features.
- adult:12 This data set includes both numerical and categorical features.

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.

<table>
<thead>
<tr>
<th>Model and implementation</th>
<th>parameters</th>
<th>training time (seconds)</th>
<th>public set logloss</th>
<th>private set logloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM-SG</td>
<td>( \eta = 0.2, \lambda = 0, t = 10 )</td>
<td>164</td>
<td>0.39018</td>
<td>0.38833</td>
</tr>
<tr>
<td>LM-LIBLINEAR-CD</td>
<td>( s = 7, c = 1 )</td>
<td>417</td>
<td>0.39131</td>
<td>0.38944</td>
</tr>
<tr>
<td>LM-LIBLINEAR-Newton</td>
<td>( s = 0, c = 1 )</td>
<td>650</td>
<td>0.39269</td>
<td>0.39079</td>
</tr>
<tr>
<td>Poly2-SG</td>
<td>( \eta = 0.2, \lambda = 0, B = 10^t, t = 10 )</td>
<td>911</td>
<td>0.38554</td>
<td>0.38347</td>
</tr>
<tr>
<td>Poly2-LIBLINEAR-Hash-CD</td>
<td>( s = 7, c = 1 )</td>
<td>1,756</td>
<td>0.38516</td>
<td>0.38303</td>
</tr>
<tr>
<td>Poly2-LIBLINEAR-Hash-Newton</td>
<td>( s = 0, c = 1 )</td>
<td>27,292</td>
<td>0.38598</td>
<td>0.38393</td>
</tr>
<tr>
<td>FM</td>
<td>( \eta = 0.05, \lambda = 2 \times 10^{-0.5}, k = 40, t = 8 )</td>
<td>547</td>
<td>0.38621</td>
<td>0.38407</td>
</tr>
<tr>
<td>FM</td>
<td>( \eta = 0.05, \lambda = 2 \times 10^{-5}, k = 100, t = 9 )</td>
<td>1,277</td>
<td>0.38740</td>
<td>0.38531</td>
</tr>
<tr>
<td>LIBFM</td>
<td>( \lambda = 40, k = 40, t = 20 )</td>
<td>18,712</td>
<td>0.39137</td>
<td>0.38963</td>
</tr>
<tr>
<td>LIBFM</td>
<td>( \lambda = 40, k = 40, t = 50 )</td>
<td>41,720</td>
<td>0.39786</td>
<td>0.39635</td>
</tr>
<tr>
<td>LIBFM</td>
<td>( \lambda = 40, k = 100, t = 20 )</td>
<td>39,719</td>
<td>0.39644</td>
<td>0.39470</td>
</tr>
<tr>
<td>LIBFM</td>
<td>( \lambda = 40, k = 100, t = 50 )</td>
<td>91,210</td>
<td>0.40740</td>
<td>0.40585</td>
</tr>
<tr>
<td>FFM</td>
<td>( \eta = 0.2, \lambda = 2 \times 10^{-4}, k = 4, t = 4 )</td>
<td>340</td>
<td>0.38411</td>
<td>0.38223</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Data set</th>
<th># instances</th>
<th># features</th>
<th># fields</th>
<th>logloss</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDD2010-bridge</td>
<td>20,012,499</td>
<td>651,166</td>
<td>9</td>
<td>0.13829</td>
</tr>
<tr>
<td>KDD2012</td>
<td>20,950,284</td>
<td>19,147,857</td>
<td>14</td>
<td>0.09461</td>
</tr>
<tr>
<td>phishing</td>
<td>11,055</td>
<td>100</td>
<td>8</td>
<td>0.20627</td>
</tr>
<tr>
<td>adult</td>
<td>48,842</td>
<td>308</td>
<td>8</td>
<td>0.20627</td>
</tr>
<tr>
<td>cod-rna (dummy fields)</td>
<td>331,152</td>
<td>2,296</td>
<td>8</td>
<td>0.16455</td>
</tr>
<tr>
<td>cod-rna (discretization)</td>
<td>331,152</td>
<td>2,296</td>
<td>8</td>
<td>0.16455</td>
</tr>
<tr>
<td>icas (dummy fields)</td>
<td>141,691</td>
<td>22</td>
<td>100</td>
<td>0.12874</td>
</tr>
<tr>
<td>iwannet (discretization)</td>
<td>141,691</td>
<td>69,867</td>
<td>100</td>
<td>0.12580</td>
</tr>
</tbody>
</table>

5. CONCLUSIONS AND FUTURE WORKS

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


