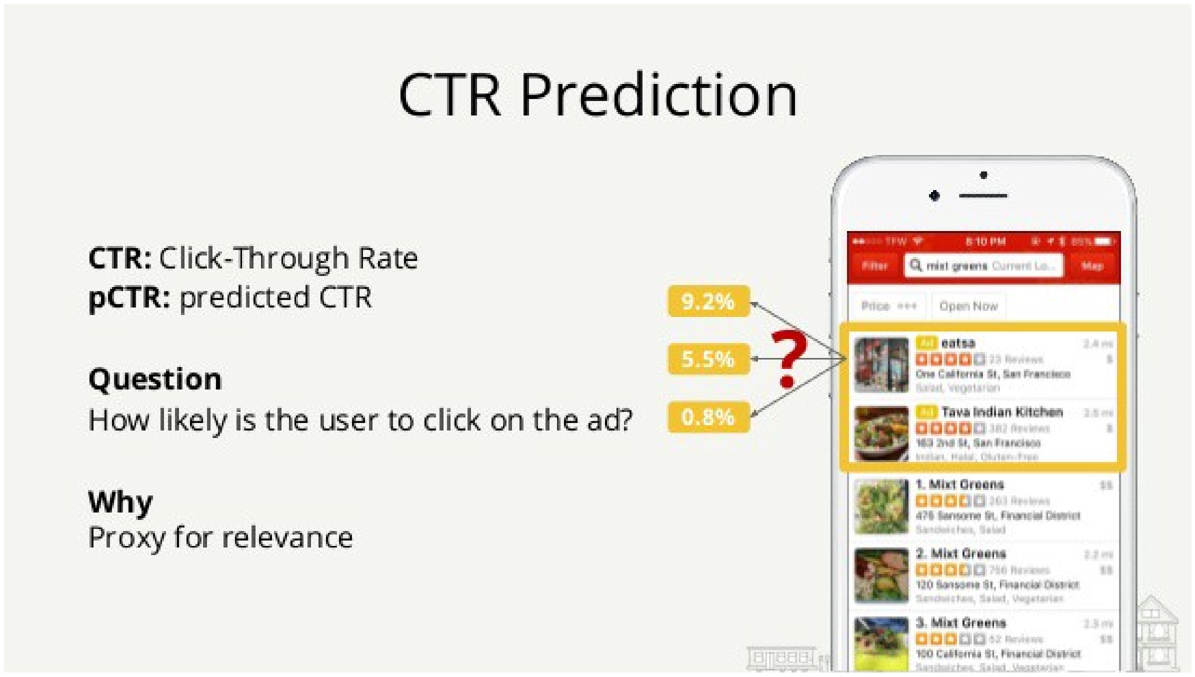
Explaining the Field-Aware Factorization Machines (FFMs) for CTR-Prediction

vineet singh Follow
Dec 26, 2018 · 7 min read

Original Paper:- Field-aware Factorization Machines for CTR Prediction



CTR Prediction

CTR: Click-Through Rate
pCTR: predicted CTR

Question
How likely is the user to click on the ad?

Why
Proxy for relevance

· · ·

“I vividly remember my first encounter with a CTR prediction problem. Before this, I had been learning data science and I was feeling good about my progress. I was starting to feel confident in ML, but the first look at the data-set gave me jitters. The data after unzipping was well over 50 GB! — I had no clue how to predict a click on such a huge data-set. A blessing in disguise, Factorization machines came to my rescue.”

Anyone who has worked on a CTR Prediction problem or Recommendation Systems would have faced a similar crisis. As the data-sets are huge, predicting these data-sets becomes very challenging with limited computation resources at disposal.

However, in most of the cases these data-sets are sparse hence, there are several features which are not important for prediction, this is where factorization comes to rescue — it helps to extract the most important features (hidden or latent) from the existing raw ones.

> *In this article, I discuss Factorization Machines(FMs) and Field-Aware Factorization Machines(FFMs) which allows us to take advantage of factorization in a regression/classification problem.*

. . .

ABSTRACT

Click-through rate (CTR) prediction plays an important role in computational advertising. The most commonly used models for CTR estimates are Poly2 mappings and factorization machines (FMs). In recent times, a variant of factorization machines, the FFMs performed well in the worldwide CTR estimation competition, which is mainly effective in large-scale sparse data classification tasks.

. . .

INTRO

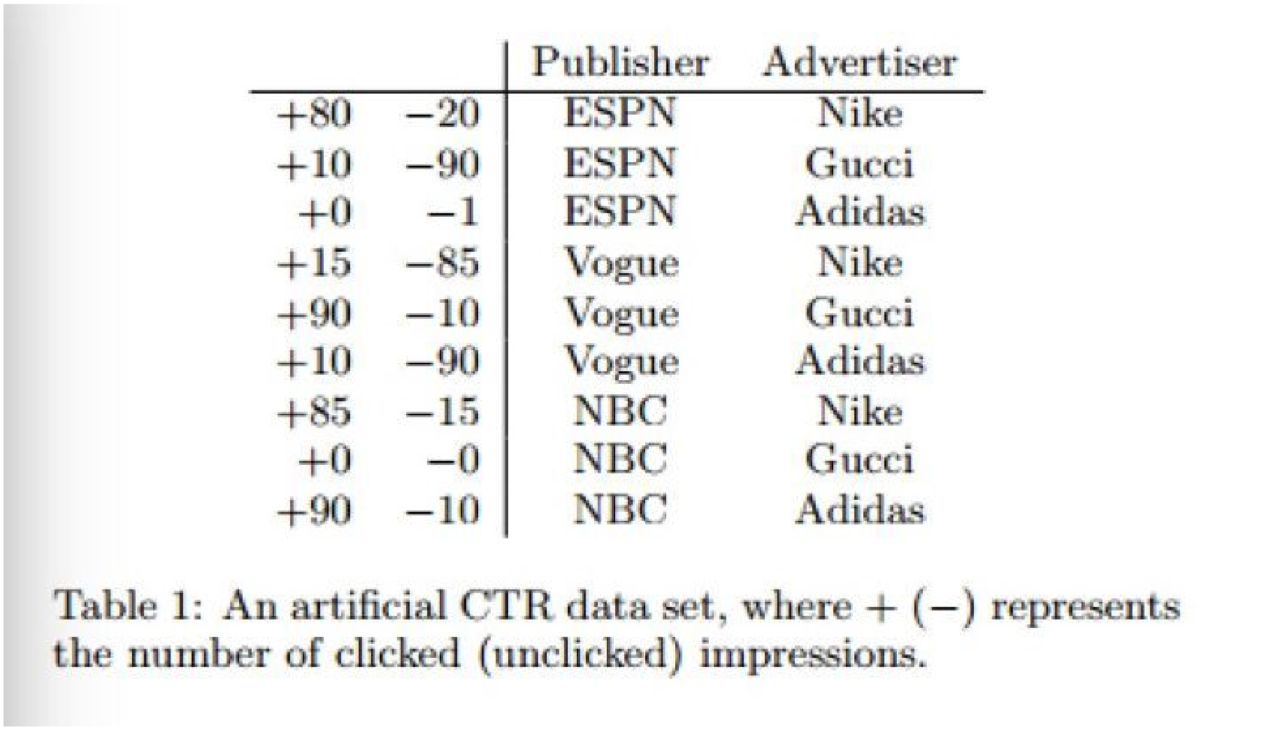
Logistic regression is the most widely used model in CTR estimation. For a data-set with *m* samples *(yi, xi), i = 1,…….,m, yi* stands for label, *xi* is a feature vector, the model parameter *w* is obtained by solving the following optimization problem:

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

among them, This is a linear model:

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

In the CTR predictions, it is crucial to learn the effects of feature conjunctions, such as in the following examples:



		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.

Ads from Gucci have a higher click-through rate at Vogue. However, linear models face difficulties in learning this information. Because, linear models learn the two weights Gucci and Vogue separately, to solve this problem, the degree-2 polynomial mapping (Poly2) model and the FMs model were proposed. The Poly2 model is for each feature conjunctions. The FMs model, learns weight by expressing the effects of feature conjunctions by breaking into the inner product of two hidden vectors.

. . .

Poly2 MODEL & FM MODEL

Studies show that the Poly2 model can capture the effects of feature conjunctions well, and when the linear model is applied to the degree-2 mappings, the training time and test time are significantly faster than the method of applying the kernel. The Poly2 model learns a weight for each feature pair:

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

Where *h(j1,j2)* is a function that encodes *j1* and *j2* into a natural number.

> *The computational complexity of the above formula is $O(n^2)$, n, representing the average of the non-zero numbers of each sample.*

FM learns an implicit vector representation for each feature, and each hidden vector contains *k* dimensions, so that the effects of feature conjunctions are modeled as the inner products between the two hidden vectors:

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

> *The computational complexity of the above formula is $O(n^2k)$.*

The $O(n^2k)$ computational complexity can be reduced to *O(nk)* by some computational techniques.

On a sparse data-set,

> FM model > Poly2 model

How Factorization Machines trump Polynomial and linear models?

For example, consider the pair (ESPN, Adidas), there is only one unique negative sample. The Poly2 model will learn a large negative weight for this pair. However, for FMs, because it is a hidden vector representation of ESPN and Adidas, all samples containing ESPN and all samples containing Adidas will be used to learn these two hidden vectors, so its prediction will be more accurate.

. . .

FIELD-AWARE FACTORIZATION MACHINES

The idea of the FFM model is derived from the PITF model. The PITF model is used in the recommendation system with personalized tags. In the PITF model, there are 3 fields,

user | item | tag

these 3 fields are decomposed in the independent hidden spaces as,

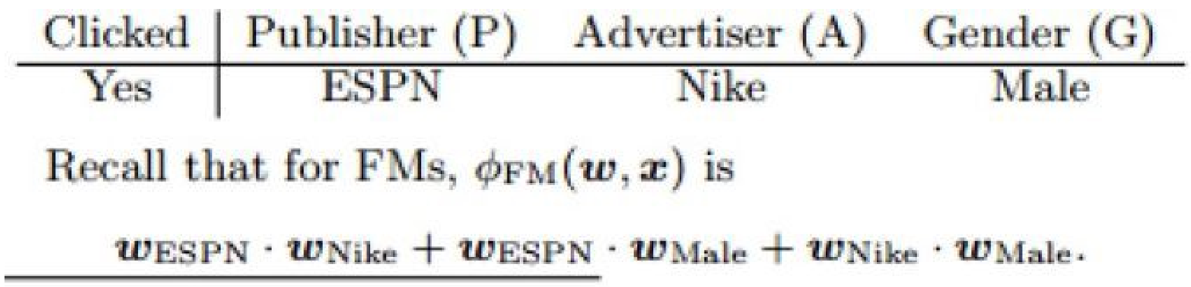
(user, item), (user, tag), (item, tag).

Mostly for CTR predictions, features can generally be grouped into fields. In the example above,

ESPN, Vogue, and NBC can be grouped into Publisher,

Nike, Gucci, and Adidas belong to the Advertiser.

FFMs will make full use of group information. Consider the following example:



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

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

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

In FMs,

each feature has a unique implicit vector representation. This hidden feature is used to learn the effect of any other feature. Considering **ESPN**, *w(ESPN)* is used to learn the implicit impact of,

1. **Nike** → *w(ESPN)*w(Nike)*
2. **Male** → *w(ESPN)*w(Male)*

but due to Nike and Male belongs to different fields, their latent effects are different.

In FFMs,

each feature has several different hidden vectors. The FFMs of the above examples are as follows:

$$w_{\text{ESPN,A}} \cdot w_{\text{Nike,P}} + w_{\text{ESPN,G}} \cdot w_{\text{Male,P}} + w_{\text{Nike,G}} \cdot w_{\text{Male,A}} \cdot$$

Thus, to learn the latent effect of,

1. **(ESPN, NIKE)** → *w(ESPN, A) & w(Nike,P)* is used, because *Nike* belongs to the field *Advertiser* and *ESPN* belongs to the field *Publisher*.

Similarly,

2. **(ESPN, Male)** → *w(ESPN,G) & w(Male,P) is used.*

Mathematically, the overall calculation formula is as follows:

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

here, *f2* represents the field of *j2*, and *f1* represents the field of *j1*.

. . .

EXPERIMENTS

Data Sets:

Data Set	# instances	# features	# fields
Criteo	45,840,617	10^6	39
Avazu	40,428,968	10^6	33

Evaluation method:

logistic loss defined as,

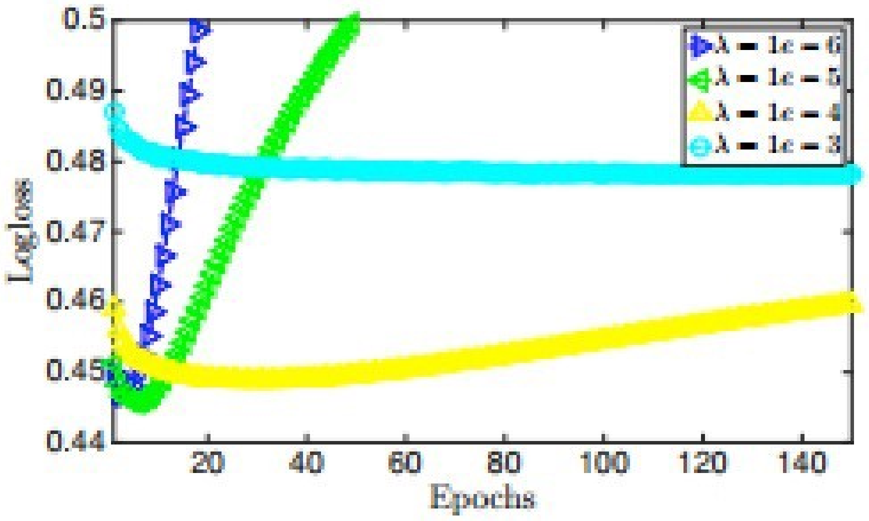
$$\text{logloss} = \frac{1}{m} \sum_{i=1}^{m} \log(1 + \exp(-y_i \phi(\boldsymbol{w}, \boldsymbol{x}_i))),$$

where m is the number of test instances.

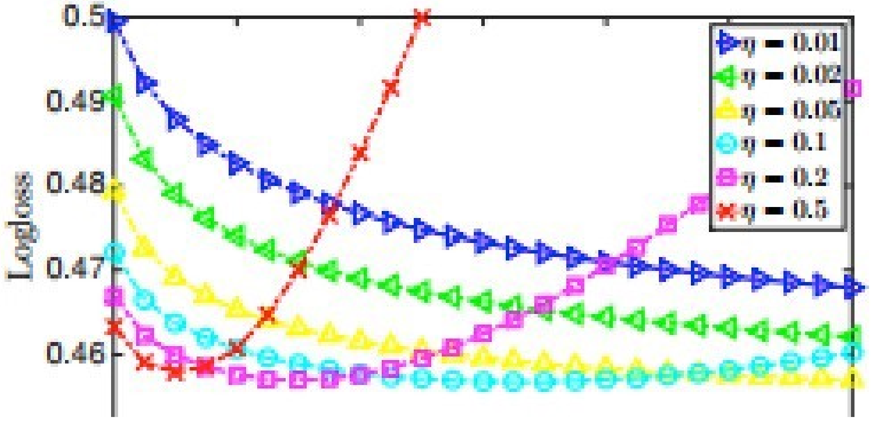
Impact of the Parameters:

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.

It can be evidently seen that the effect of *k* on *logloss* is not significantly large; if the **regular term (λ)** is relatively large, the model is prone to high deviation, the effect is not good. On the contrary, if the **regular term** is relatively small, the model gets better results, but it easily overfits the data.

For the **learning rate (η)**, if it is too small, the model will be slower. If it is too large, the model loss will decrease very quickly, but then it will be fitted, so it needs **early stopping**,

Early stopping is a strategy to reduce overfitting for many Machine Learning problems, for FFMs the strategy used is:

1. Divide the training data-set into training data and verification data.

2. At the end of each epoch(iteration), use the validation set to evaluate the loss.

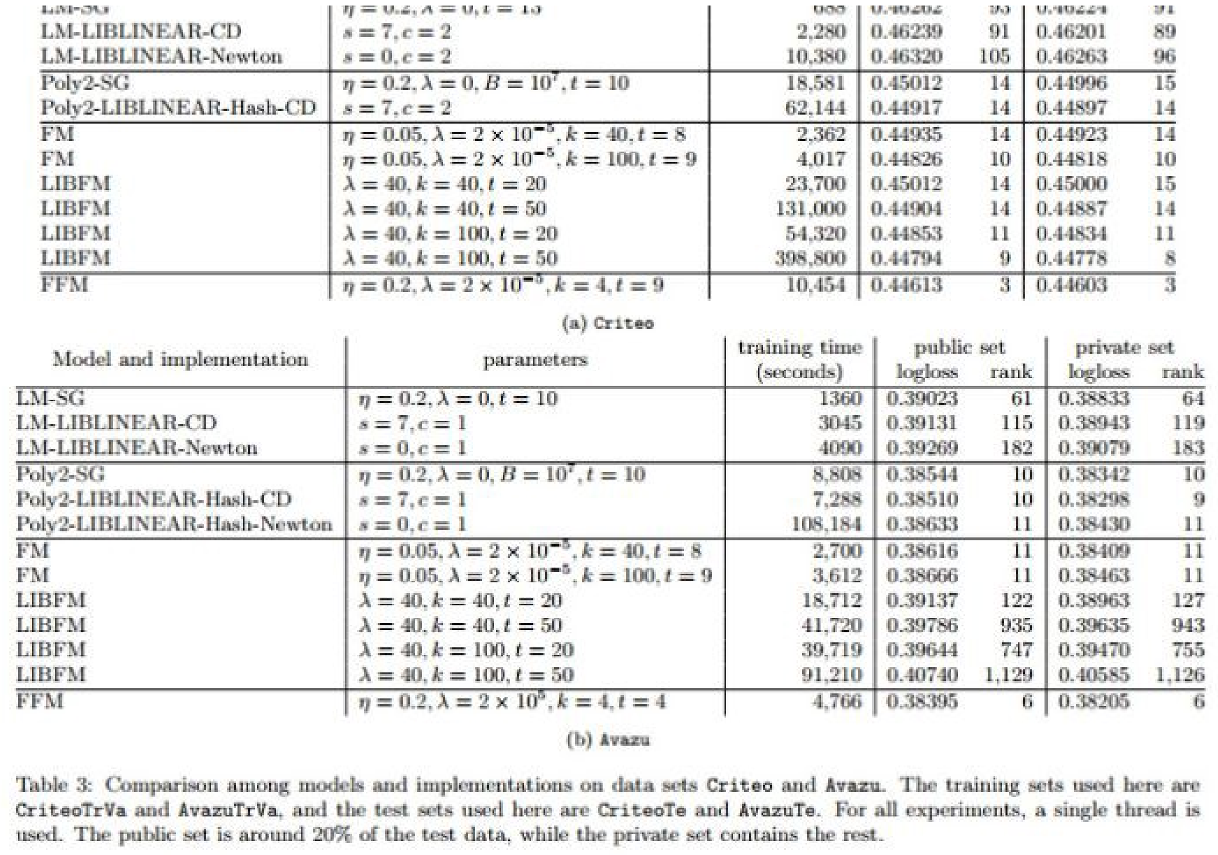
3. If the loss rises, record the number of epochs. Stop or implement the *Step 4*.

4. If requires, retrain the model with the complete data-set, using the number of epochs recorded in *Step 3*.

The more difficult thing is that *logloss* is very sensitive to the number of iterations. The number of iterations that perform well on the validation set is not necessarily good on the test set.

Comparison with other models:

Model and implementation	parameters	training time (seconds)	public set logloss	public set rank	private set logloss	private set rank



LM-SG	$\eta = 0.2, \lambda = 0, t = 13$	[illegible]	0.46262	93	0.46224	91
LM-LIBLINEAR-CD	$s = 7, c = 2$	2,280	0.46239	91	0.46201	89
LM-LIBLINEAR-Newton	$s = 0, c = 2$	10,380	0.46320	105	0.46263	96
Poly2-SG	$\eta = 0.2, \lambda = 0, B = 10^7, t = 10$	18,581	0.45012	14	0.44996	15
Poly2-LIBLINEAR-Hash-CD	$s = 7, c = 2$	62,144	0.44917	14	0.44897	14
FM	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 40, t = 8$	2,362	0.44935	14	0.44923	14
FM	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 100, t = 9$	4,017	0.44826	10	0.44818	10
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	$\eta = 0.2, \lambda = 2 \times 10^{-5}, k = 4, t = 9$	10,454	0.44613	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	$\eta = 0.2, \lambda = 0, t = 10$	1360	0.39023	61	0.38833	64
LM-LIBLINEAR-CD	$s = 7, c = 1$	3045	0.39131	115	0.38943	119
LM-LIBLINEAR-Newton	$s = 0, c = 1$	4090	0.39269	182	0.39079	183
Poly2-SG	$\eta = 0.2, \lambda = 0, B = 10^7, t = 10$	8,808	0.38544	10	0.38342	10
Poly2-LIBLINEAR-Hash-CD	$s = 7, c = 1$	7,288	0.38510	10	0.38298	9
Poly2-LIBLINEAR-Hash-Newton	$s = 0, c = 1$	108,184	0.38633	11	0.38430	11
FM	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 40, t = 8$	2,700	0.38616	11	0.38409	11
FM	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 100, t = 9$	3,612	0.38666	11	0.38463	11
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	$\eta = 0.2, \lambda = 2 \times 10^5, k = 4, t = 4$	4,766	0.38395	6	0.38205	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.

It is evident that FFM works well on CTR estimates!

CONCLUSIONS:

1. *FFMs also supports parallelization, so they are faster.*

2. *FFM is much better than LM, poly2, and FM in the processing of sparse data.*

3. *FFMs incorporate the concept of field into FMs and are worth learning.*

4. *In actual use, it is necessary to adjust the parameters according to the requirements of specific problems for achieving better results.*

. . .

If this blog helped you in any possible way then please don't forget to give it a Clap!! Thank you for the read……

If you wish to discuss on relative topics or wanna recommend anything then feel free to reach me out on my e-mail : vinitvssingh5@gmail.com . Adios!!

Machine Learning Deep Learning Ctr Ffm Factorization Machines