Ad Click Prediction

Ruichen Wang

December 25, 2018

Abstract

There are a lot of algorithms designed for ad ctr prediction. Those algorithms take contextual and historical features as inputs, often combines massive feature engineering work, then predict whether or not the user will click some certain ads. Here I'd like to summarize some popular traditional solutions and recent deep learing technics.

Contents

1 Online Advertising and Linear Model Sparsity

For learning in massive scale, generalized linear models have many advantages. Easily implement, memory saving, efficient training on streaming data.

Supposing we choose logistic regression as our model. Given feature $x_t \in R^d$, model parameter w_t. we can predict $p_t = \sigma(w_t \cdot x_t)$, where $\sigma(x) = \frac{1}{1+exp(-x)}$

is the sigmoid function. And our observation $y_t \in \{0, 1\}$. The log loss can be easily denoted as :

$$\mathcal{L}(w_t) = -y_t log p_t - (1 - y_t) log(1 - p_t)$$

It is obvious that:

$$\frac{\partial \mathcal{L}}{\partial w} = (\sigma(w \cdot x_t) - y_t)x_t = (p_t - y_t)x_t$$

And this gradient is all we need for optimization purpose.

1.1 Online Gradient Descent (OGD)

OGD is essentially the same with SGD. OGD can provide excellent prediction accuracy with minimum computing resources. However, as x might have billions of dimensions, the size of the final model is another key consideration. Here we want the w be sparse, while keep the accuracy or little accuracy loss.

However, OGD is not good at producing sparse result. In fact, simply adding subgradient penalty will never induce sparsity.

1.2 Truncated Gradient (TG)

There is a very simple solution to produce sparse solution. Just rounding small weights to 0. but doing so may cause some problems, because a weight may be useless or small because it has been updated only once, or at the begining of the training. Truncated gradient[3, 7] introduce another natural method to round small coefficients.

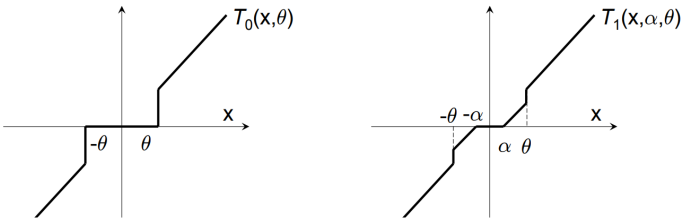

Figure 1: Plots for the truncation functions below.

For each K steps, perform simple coefficient rounding.

$$T_0(v_j, \theta) = \begin{cases} 0 & if \ |v_j| \leq \theta \\ v_j & otherwise \end{cases}$$

For each K steps, truncate gradient.

$$T_1(v_j, \alpha, \theta) = \begin{cases} max(0, v_j - \alpha) & if \ v_j \in [0, \theta] \\ min(0, v_j + \alpha) & if \ v_j \in [-\theta, 0] \\ v_j & otherwise \end{cases}$$

1.3 Forward-Backward Splitting (FOBOS)

In FOBOS[4], the problem is considered as two parts,

$$f(w) + r(w)$$

where f is the empirical loss and r is a regularization term that penalized for excessively complex vectors. FOBOS updates the w in 2 steps, forward-looking subgradients and forward-backward spliting. May sound complex, Actually it's very simple.

$$w_{t+\frac{1}{2}} = w_t - \eta_t g_t$$

$$w_{t+1} = \arg\min_{w_t} \left\{ \frac{1}{2} ||w_t - w_{t+\frac{1}{2}}||^2 + \eta_{t+\frac{1}{2}} r(w_t) \right\}$$

The first step is apply the gradient to the weight. The second step mainly contributes to the sparsity. It can be viewed as (i)let the updated weight be close to the previous one (ii) attain a low complexity.

1.4 Regularized Dual Averaging Algorithm (RDA)

RDA[12] condsider the learning variables are adjusted by solving a simple minimization problem that involves the running average of all past subgradients of the loss function and the whole regularization term, not just its subgradient.

$$w_{t+1} = \arg\min_{w} \{ \frac{1}{t} \sum_{i=1}^{t} \langle g_i, w \rangle + \Psi(w) + \frac{\beta_t}{t} h(w) \}$$

At each iteration, this method minimizes the sum of three items: (i) all previous subgradients (dual average). (ii)the original regularization funciton ($\lambda ||w||_1$). (iii) an additional strongly convex regularization term($\frac{1}{2}||w||_2^2$). It can be denoted as :

$$w_{t+1} = \arg\min_{w} \{ \frac{1}{t} \sum_{i=1}^{t} \langle g_i, w \rangle + \lambda ||w||_1 + \frac{1}{2} ||w - 0||_2^2 \}$$

In result, RDA producs even better accuracy vs sparsity tradeoffs than FOBOS.

1.5 Follow the Proximally Regularized Leader (FTRL-Proximal)

FTRL [8] is a hybrid of FOBOS and RDA, and significantly outperforms both on large real-world dataset.

$$w_{t+1} = \arg\min_{w} \{ g_{1:t} \cdot w + t\lambda ||w||_1 + \frac{1}{2} \sum_{s=1}^{t} \sigma_s ||w - w_s||^2 \}$$

where σ is the learning rate schedule, such that $\sigma_{1:t} = \frac{1}{\eta_t}$. The update of this fomula is actually very simple. As the fomula above can also be written as :

$$(g_{1:t} - \sum_{s=1}^{t} \sigma_s w_s) \cdot w + \frac{1}{\eta_t}||w||_2^2 + \lambda_1||w||_1 + const$$

If we store $z_{t-1} = g_{1:t-1} - \sum_{s=1}^{t-1} \sigma_s w_s$ in memory. At each round t, we update z_t:

$$z_t = z_{t-1} + g_t + (\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}})w_t$$

and the w_{t+1} can be solved in closed form on:

$$w_{t+1} = \begin{cases} 0 & if\ |z_t| \leq \lambda_1 \\ -\eta_t(z_t - sgn(z_t)\lambda_1) & otherwise \end{cases}$$

Thus FTRL-Proximal stores z in memory, whereas OGD stores w.

Per-coordinate leaning Standard online learning gradient descent suggest using a global learning rate schedule $\eta = \frac{1}{\sqrt{t}}$, which is common for all coordinates. As some coordinate updates frequently and some not. It makes more sense to represent the gradient per coordinate:

$$\eta_{t,i} = \frac{\alpha}{\beta + \sqrt{\sum_{s=1}^{t} g_{s,i}^2}}$$

β is simply for numeric stable sake, ensures that early learning rate not too high. $\beta = 1$ is usually good enough.

2 Collaborative Filtering

2.1 Factorization Machines (FM)

In recommender system, we may want the model to be able to estimate interactions between large feature dimensions, while keeping the sparsity. FM [10] is designed to deal with highly sparse features.It has linear complexity, and can work with any real valued feature vector. It can be used in many prediction task like regression, binary classification and ranking.

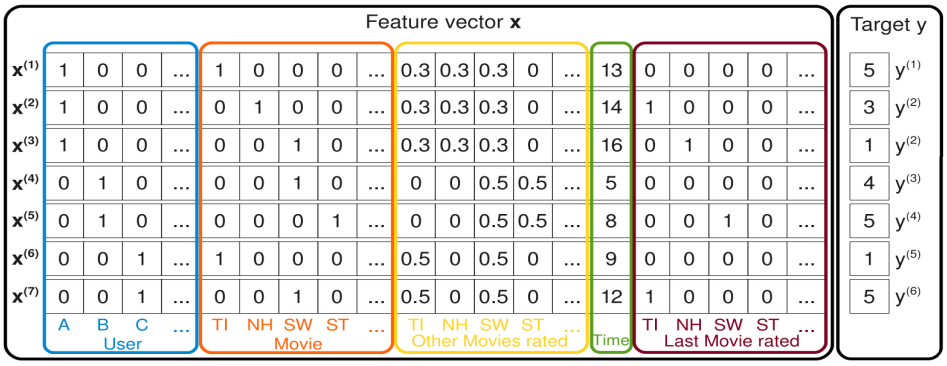

Figure 2: Sparse user features and target y.

Model equation Given $x \in R^n$, the model parameters that have to be estimated are $w_0 \in R, \mathbf{w} \in R^n, \mathbf{V} \in R^{n \times k}$.

$$\widehat{y} = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j$$

where

$$\langle \mathbf{v}_i, \mathbf{v}_j \rangle = \sum_{m=1}^{k} v_{i,m} \cdot v_{j,m}$$

Based on the theory that for any positive definite matrix (semi-positive definite) $\mathbf{w}$, there exists a matrix $\mathbf{v}$ such that $\mathbf{w} = \mathbf{v} \cdot \mathbf{v}^{\mathrm{T}}$ if k is not limited. In practice, we often pick a small k for better generalization and thus improved interaction matrices under sparsity.

Computation If we follow the straight forward computation, the cost will be $O(kn^2)$. We can apply some mathematics transformations to reduce the computation to linear time $O(kn)$.

$$\begin{aligned}
\sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j &= \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j - \frac{1}{2} \sum_{i=1}^{n} \langle \mathbf{v}_i, \mathbf{v}_i \rangle x_i x_i \\
&= \frac{1}{2} \sum_{m=1}^{k} \left(\left(\sum_{i=1}^{n} v_{i,m} x_i \right)^2 - \sum_{i=1}^{n} v_{i,m}^2 x_i^2 \right)
\end{aligned}$$

Learning FM model Introduced by libFM[11], there are three ways to optimize this task, SGD, ALS and MCMC inference. Details and comparison is presented in the paper, I'm not going too deep here.

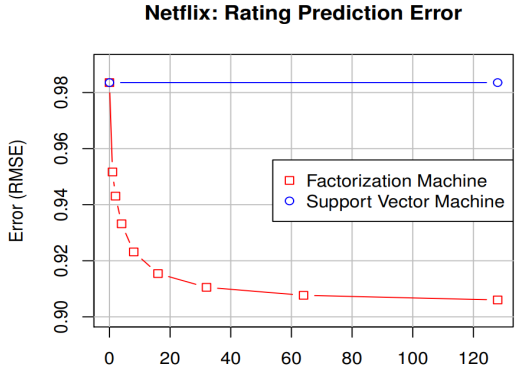

Figure 3: Original paper shows that FM succeed in 2-way variable interactions while SVM doesn't converge at all.

2.2 Field-aware Factorization Machines (FFM)

In FM, every feature has only one latent vector to learn the latent effect with any other features. For example, conside two pair of interaction, $(w_{male}, w_{games}), (w_{male}, w_{dance})$. As games and dance clearly belong to different field. FFM [6] claims that the latent effects at different field should be differ too. Mathematically,

$$\phi_{FFM}(\mathbf{w}, \mathbf{x}) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} (\mathbf{w}_{i,f_j}, \mathbf{w}_{j,f_i}) \mathbf{x}_i \mathbf{x}_j$$

If f is the number of fields, then the number of variables of FFMs is nfk, and the complexity to compute is $O(n^2k)$. It is worth noting that in FFMs because each latent vector only needs to learn the effect with a specific field (represent feature using a few of short vectors instead of a long vector), usually

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

In kaggle Criteo and Avazu competition, FFM outperform FM. The practical stochastic gradient leaning rate schedule is proposed in [2].

3 Incorporate with Deep Learing

3.1 FNN and PNN

FM performs a very good job at collaborative filtering on recommendation. It is regarded as one of the most successful embedding models.

Proposed by *Weinan Zhang*, published at IJCAI, factorization machine supported neuaral network (FNN) **pretrains** FM and then applys the DNN part

on it. FM mainly used to convert the high dimensional sparse binary features into a dense feature.

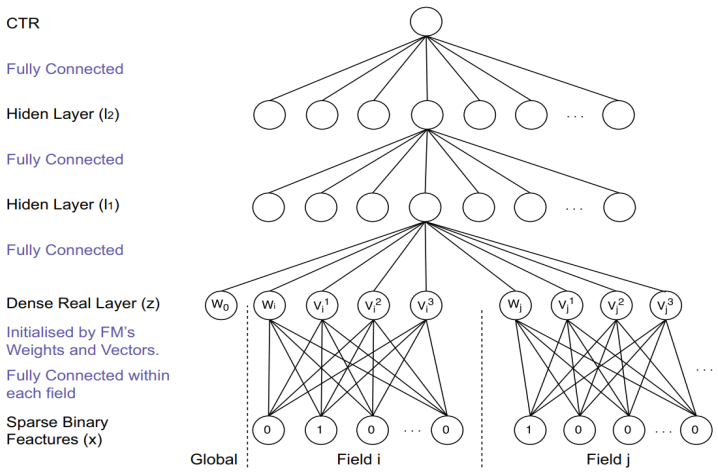

Figure 4: A 4-layer FNN structure.

It key idea of this paper is that it trys a combination of FM and DNN. In the same year, PNN[9], published at ICDM, was proposed by the same group of people. The only different is that it adds a product layer between FM and DNN.

3.2 Deep & Wide Learning

Both FNN and PNN fails to capture the low order feature interactions. And it is often the case that there should be no interactions between most query-item pairs. But the dense embeddings will lead to nonzero predictions for all pairs.

While in Google's deep & wide network[1], wide linear models and deep neural networks are jointly trained. Combines the **memorization** and **generalization** for recommender system.

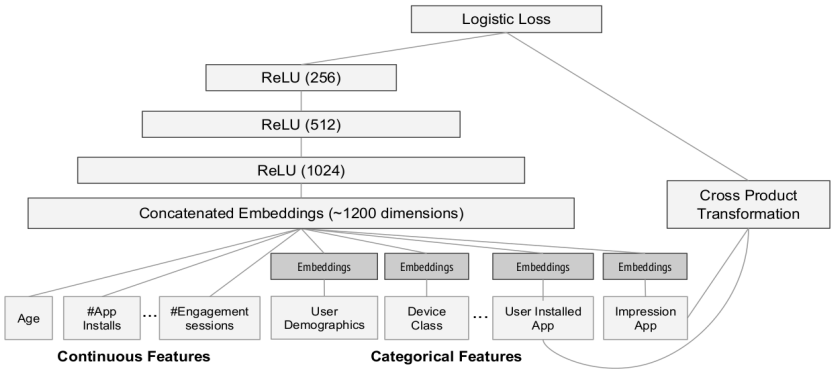

Figure 5: Deep & Wide model.

Google claims that jointly training the deep and wide model needs less data to achieve reasonable accuracy, as the wide part only needs to complement the weeknesses of deep part, rather than a full-size wide model. In experiments, they use FTRL with L_1 regularization on the wide part, and use AdaGrad for the deep part. The model can be denoted as :

$$P(Y = 1|\mathbf{x}) = \sigma(\mathbf{w}_{wide}^T[\mathbf{x}, \phi(\mathbf{x})] + \mathbf{w}_{deep}^T a^{(l_f)} + b)$$

where the $\sigma(\cdot)$ is the sigmoid function, $\phi(\mathbf{x})$ is the cross product transformations, and $a^{(l_f)}$ is the activations. Note that the cross product transformation(wide part) only takes user installed app and impression app as input.

3.3 DeepFM

DeepFM[5]

3.4 DNN

References

[1] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In Alexandros Karatzoglou, Balázs Hidasi, Domonkos Tikk, Oren Sar Shalom, Haggai Roitman, Bracha Shapira, and Lior Rokach, editors, *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRS@RecSys 2016, Boston, MA, USA, September 15, 2016*, pages 7–10. ACM, 2016.

[2] Wei-Sheng Chin, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. A learning-rate schedule for stochastic gradient methods to matrix factorization. In Tru Cao, Ee-Peng Lim, Zhi-Hua Zhou, Tu Bao Ho, David Wai-Lok Cheung, and Hiroshi Motoda, editors, *Advances in Knowledge Discovery and Data Mining - 19th Pacific-Asia Conference, PAKDD 2015, Ho Chi Minh City, Vietnam, May 19-22, 2015, Proceedings, Part I*, volume 9077 of *Lecture Notes in Computer Science*, pages 442–455. Springer, 2015.

[3] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[4] John C. Duchi and Yoram Singer. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2899–2934, 2009.

[5] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for CTR prediction. *CoRR*, abs/1703.04247, 2017.

[6] Yu-Chin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for CTR prediction. In Shilad Sen, Werner Geyer, Jill Freyne, and Pablo Castells, editors, *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*, pages 43–50. ACM, 2016.

[7] John Langford, Lihong Li, and Tong Zhang. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10:777–801, 2009.

[8] H. Brendan McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and implicit updates. *CoRR*, abs/1009.3240, 2010.

[9] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. Product-based neural networks for user response prediction. In Francesco Bonchi, Josep Domingo-Ferrer, Ricardo A. Baeza-Yates, Zhi-Hua Zhou, and Xindong Wu, editors, *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, pages 1149–1154. IEEE, 2016.

[10] Steffen Rendle. Factorization machines. In Geoffrey I. Webb, Bing Liu, Chengqi Zhang, Dimitrios Gunopulos, and Xindong Wu, editors, *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010*, pages 995–1000. IEEE Computer Society, 2010.

[11] Steffen Rendle. Factorization machines with libfm. *ACM TIST*, 3(3):57:1–57:22, 2012.

[12] Lin Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11:2543–2596, 2010.