A Fast Linear Computational Framework for User Action Prediction in Tencent MyApp

Yaochen Hu
University of Alberta
Edmonton, AB
yaochen@ualberta.ca

Di Niu
University of Alberta
Edmonton, AB
dniu@ualberta.ca

Jianming Yang
Mobile Internet Group, Tencent
Shenzhen, China
kimmyyang@tencent.com

ABSTRACT

User action modeling and prediction has long been a topic of importance to recommender systems and user profiling. The quality of the model or accuracy of prediction plays a vital role in related applications like recommendation, advertisement displaying, searching, etc. For large scale systems with a massive number of users, beside the pure prediction performance, there are other practical factors like training and prediction latency, memory overhead, that must be optimized to ensure smooth operation of the system. We propose a fast linear computational framework to handle a vast number of second order crossed features with dimensionality reduction. By leveraging the training and serving system architecture, we shift heavy calculation burden from online serving to offline preprocessing, at the cost of a reasonable amount of memory overhead. The experiments on a 15-day data trace from Tencent MyApp shows that our proposed framework can achieve comparable prediction performance to much complex models like the field-aware factorization machine (FFM) while being served in 2 ms with a reasonable amount of memory overhead.

CCS CONCEPTS

• **Information systems → Collaborative filtering**; *Computational advertising*;

KEYWORDS

Machine learning; speedup prediction; large scale system; user behavior modeling

ACM Reference Format:
Yaochen Hu, Di Niu, and Jianming Yang. 2018. A Fast Linear Computational Framework, for User Action Prediction in Tencent MyApp. In *The 27th ACM International Conference on Information and Knowledge Management (CIKM '18), October 22–26, 2018, Torino, Italy.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3269206.3272015

1 INTRODUCTION

The accurate online prediction of user behavior or user-item interaction in large-scale web and mobile applications plays a vital role in many revenue-generating applications, such as content/product recommendation [4, 18], advertisement displaying [2, 9], search result ranking [6, 13], etc. Tencent MyApp is the largest Android app store in China, with a market share of 24.7% in China in 2017, followed by 360 Mobile Assistant (15.5%) and Xiaomi App Store (13.0%)[1].

Like other e-commerce sites or content aggregators, Tencent MyApp critically relies on the accurate and *fast* prediction of the activation rates of a user on different apps to perform online personalized app recommendation, ranking, and advertisement placement, when a user opens MyApp. Moreover, this system is extremely large, with 2×10^8 users, 5×10^4 apps, and more than 10^4 features spanning users, apps and the context. Given the sheer scale of the system, it is a great challenge to make and *serve* such predictions both instantaneously and accurately whenever a user opens MyApp, within only several milliseconds, the tolerable delay of a typical user. Note that pre-computing all the user-item activation scores and using the pre-computed scores for serving is not an option in MyApp, since the large number of features considered are constantly changing; the context features depend on the location and time at which the user opens MyApp. Moreover, some user interest features are derived from extensive feature engineering based on the past digested content and are evolving over time.

Up to date, a variety of models with different levels of accuracy and complexity have been adopted in industry for such prediction and recommendation tasks. Simple statistical models and logistic regression with extensive feature engineering are the first successful models [11] that are still widely used in some cases today. They can achieve a satisfactory prediction accuracy, with small cost on computational resources during both the training phase and serving phase. To overcome the limitation of a linear model, carefully selected second-order crossed features have been added into logistic regression to improve the performance [2, 13]. To further effectively and efficiently model all second-order feature crossing, the factorization machine (FM) [15] and, more recently, the field-aware factorization machine (FFM) [9] have been proposed, where the coefficient of each second-order crossed feature is modeled as the inner product of two projected latent variables. Recently, even more complex models based on the deep neural network (DNN) have been proposed to further capture the higher-order feature interactions [12, 14, 20, 21]. In addition, hybrid methods that combine the powers of different models are proposed to further characterize different levels of feature interactions and combine information from human feature engineering with automatically learned features [3, 7]. Finally, even more complex approaches are adopted as winning solutions in several data mining challenges, such as Kaggle

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.
CIKM '18, October 22–26, 2018, Torino, Italy
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6014-2/18/10…$15.00
https://doi.org/10.1145/3269206.3272015

[1]Google Play Store ranked the 8th among Android app stores in China in 2017, with a market share of 3.7%

and the Netflix Challenge, which are ensemble solutions combing a large number of weaker estimators.

However, in real industrial environment, although carefully designed complex models may bring about better prediction performance, there are other practical constraints and goals in addition to the pure prediction performance. One of the biggest concern is the computational complexity [9]. In fact, as has been reported in [9, 19], even for models with second-order feature crossing, the system still suffers from long training and serving times when the feature dimension and sample size scales up to size of Tencent MyApp. A more critical constraint is the prediction serving time or prediction complexity. A longer prediction latency due to the use of more complex models could directly lead to observable fewer user accesses and potential revenue loss [17]. Moreover, more complex models could consume much more resources during serving, which also harms the system scalability. Therefore, simple lower-order polynomial feature interactions are still preferable in a production environment in an extremely large-scale system like MyApp, which still could achieve satisfactory prediction performance comparing to much complex model [1].

Motivated by these observations, we present a practical algorithm and system framework that strikes a balance between the accuracy and model complexity (focusing on the serving latency), as well as other system resource consumptions such as storage overhead. We take the insights from FFM [20] and the hash trick [2] that dimension reduction techniques can be used for *sparse* features, which are largely present in our problem, especially for the even more sparse second-order crossed features. We propose a linear framework for prediction serving, which can leverage the sparsity of second-order crossed features for dimensionality reduction via a technique inspired by FFM. From a practical point of view, we reformulate the prediction serving procedure and propose a pre-calculation scheme to speedup the prediction for any second-order models including logistic regression, FM and FFM.

Note that the original prediction serving complexity of any second-order models is $O(N^2)$, N being the dimension of the features. Our *linear framework* only takes $O(N)$ time to serve each prediction, at the additional cost for storing some pre-calculated *condensed* features for each item, greatly reducing the serving time as the feature space scales up. Furthermore, the memory cache cost only scales linearly as the feature dimension, the user number and the app number.

We evaluate our proposed scheme on a large dataset collected from Tencent MyApp over a 15-day period. Since in MyApp, only a part of all the apps downloaded by a user will be actively used by the user, the problem is to predict the probability that a user-app pair will become active in a certain context, for all the candidate apps for this user. Experimental results suggest that our linear framework can finish prediction procedure in 2 ms while achieving an accuracy comparable to the much more complex FFM model.

In the following sections, We first introduce the background and the problem setting in Sec. 2. In Sec. 3, we present our linear framework. The speedup technique for online prediction calculation is proposed in Sec. 4. We present performance evaluation in Sec. 5 based on a large dataset collected from Tencent MyApp. Related work is discussed in Sec. 6. The paper is concluded in Sec. 7.

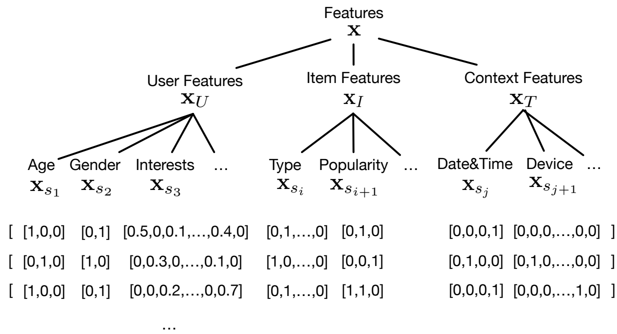

Figure 1: An illustration of the hierarchical feature space in MyApp. Fine-grained features are "one-hot" encoded or produced by some embedding methods so that the feature vector x is sparse and has a large dimension.

2 PROBLEM BACKGROUND

Tencent MyApp is a large mobile app distribution platform, which recommends and distributes apps for Android users. Whenever a user accesses the MyApp on his or her specific device, a *request* is sent to the MyApp service platform, where different apps are evaluated immediately and ranked according to relevant features. A personalized ranked list of apps and scores are generated for this request to help determine what apps the user will view in what order as well as advertisement placement. In this app evaluation procedure, one of the most important tasks is to predict the so called *download activation rate* of each user-app pair. "Activation" represents the behavior that the user will keep using the downloaded app in the following series of days, while "non-activation" represents the downloaded apps that are not actively used by the user. All app downloads are logged by the platform, with their activation status tracked by the corresponding apps for a certain period of time. Any predicted download activation rate is made based on the past logs collected from both the MyApp platform and individual apps.

It is challenging to improve prediction accuracy due to the large feature space (usually with a dimension of 10^4) as well as an enormous number of users. Although more complex models usually yield better performance, it also takes longer time to train them and make predictions. In our problem, the prediction latency, which is the key portion of the serving time, is especially critical, since scores must be evaluated instantaneously upon receiving a request, i.e., when a user opens MyApp. As download activation rate prediction is only a part of the entire serving phase, with other parts including display arrangement, advertisement insertion, and logging, etc., there is a very strict latency requirement for the prediction. Bearing these practical constraints in mind, we must optimize the prediction of the download activation rate from a comprehensive perspective, simultaneously considering prediction accuracy, online prediction latency as well as the computational and storage resource overhead.

2.1 High Dimensional Feature Space in MyApp

As Fig. 1 shows, there are mainly three groups of features recorded by MyApp platform. The first group is user features, including *fine-grained features* which are either natural features like gender, age,

occupation, etc., or *engineered features* extracted based on past user behavior, such as a user's short-term and long-term interests into one of the predefined app categories. The second group contains the item features, which are also either natural features like app type or *engineered features* extracted from past logs, such as the app popularity (among different groups of users). The third group contains context features of the request, such as the time, date and location for the request, device type and model, the recent searching queries of the user, etc.

Let $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$ denote a vector concatenating all the fine-grained feature vectors. In our problem, the dimension d of the entire feature space is over 10^4 (partly due to the use of "one-hot" encoding to be described). Let $\mathbf{x}_s$ be the feature vector for a fine-grained feature s, where $s \subset \{1, \ldots, d\}$ is the subset of indices corresponding to the specific feature s. If $\mathbf{x}_s$ is categorical, e.g., gender, location, it is represented as a vector of length $|s|$ with "one-hot" encoding, e.g., for the gender feature, $[1, 0]$ and $[0, 1]$ represent female and male, respectively. Other features like user interests into different classes of apps are embedded measures and are thus represented as vectors with multiple non-zero values.

Furthermore, let $U, I, T \subset \{1, \ldots, d\}$ represent the subsets of indices corresponding to all user features, all item features and all context features, respectively. As is shown in Fig. 1, $\mathbf{x}$ is a concatenation of $\mathbf{x}_U$, $\mathbf{x}_I$ and $\mathbf{x}_T$, which can be further divided into a number of fine-grained features. For example, $\mathbf{x}_U = \{\mathbf{x}_s\}$ such that $s \subset U$.

2.2 Problem Definition

In the collected training data, each sample with features $\mathbf{x}$ will have a $0-1$ valued response y, where $y = 0$ represents non-activation and $y = 1$ indicates a download activation. Let $D = \{(\mathbf{x}^{(0)}, y^{(0)}), (\mathbf{x}^{(1)}, y^{(1)}), \ldots\}$ denote the set of all samples. Let $f(\mathbf{x})$ represent the predicted probability of download activation. In MyApp, $f(\mathbf{x})$ will be characterized by a certain learned model to minimize

$$\sum_{(\mathbf{x}, y) \in D} -y \cdot \log(f(\mathbf{x})), \tag{1}$$

where the *log loss* is a natural measure of how close the estimated activation probability f is to the true $0-1$ label in a statistical view [5]. We will also use (1) as one of the metrics in Sec. 3 to evaluate the prediction accuracy.

3 PREDICTION MODELS

Different statistical models are all trying to infer a function $f(\mathbf{x})$ based on all possible features in $\mathbf{x}$ to predict the probability that a download action in MyApp will turn into an activation. This is similar to the typical click through rate (CTR) prediction task in many other applications in industry, where feature engineering plays a central role. In this paper, we suppose that the features in the vector $\mathbf{x}$ have already been properly pre-preprocessed via typical binning or embedding methods [8]. Thus, we only consider the modeling and system issues.

3.1 Logistic Regression

In such a prediction task, we do not have any concrete prior knowledge about how the available features are mapped to the final predicted probability. Logistic regression is one of the most popular models for a scalable and stable solution. The specific form is

$$f(\mathbf{x}) = \frac{\exp(-h(\mathbf{x}))}{1 + \exp(-h(\mathbf{x}))}, \tag{2}$$

where

$$h(\mathbf{x}) = \sum_{i=1}^{d} \beta_i x_i + \beta_0, \tag{3}$$

is the linear *logit* function, and β_i are the coefficients to be estimated in the training procedure. The optimal coefficients can be obtained by solving an optimization problem over all the training samples:

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \sum_{(\mathbf{x}, y) \in D} -y \cdot \log(f(\mathbf{x})) + \frac{1}{2}\lambda\|\boldsymbol{\beta}\|, \tag{4}$$

where $\|\cdot\|$ represents L_1 or L_2 norm that acts as a regularization term and λ is a tuning parameter. In our work, we adopt the L_1 norm.

3.2 Second Order Feature Crossing

The direct Logistic Regression model cannot express the nonlinear relationship between the logit and the input features. There are a variety of methods to improve the expressive power of the model. Feature engineering is one of them. Typical feature engineering methods are feature binning or embedding to map feature combinations into a space that has larger linear correlation to the predicted probability [8]. Feature engineering involves human efforts by experienced engineers who bears deep understanding of the application, with many trials and errors. In this paper, we do not consider the details of feature engineering and assume that the feature vectors $\mathbf{x}$ are already a result of extensive feature engineering from raw features.

We explore the interaction between features at the model level. To capture the nonlinearity in h, a variety of basis expansion methods [5] can be introduced into the model $h(\mathbf{x})$ in order to enlarge the model expressiveness. Through basis expansion, instead of evaluating the logit function, $h(\mathbf{x})$ is modeled in the following general form:

$$h(\mathbf{x}) = \sum_{b} g_b(\mathbf{x}) + \sum_{i=1}^{d} \beta_i x_i + \beta_0, \tag{5}$$

where each g_b is a transformation function over all input features $\mathbf{x}$.

Second-order feature crossing has been reported to be effective in CTR prediction problems [1, 10]. In our problem, we also adopt second order feature crossing as an effective basis expansion method, that strikes a balance between accuracy and complexity. Note that the features in our problem have over 10^4 dimensions. In this case, full feature crossing would bring about millions of weights, which will impose heavy burdens not only on the training process, but also on the online serving process. Besides, not all the crossed features have a significant impact on the prediction. Hence, typically only a subset of all second order crossed features is added into a model.

Now we introduce second order feature crossing between different fine-grained feature vectors, e.g., between gender and the category of the app. Let $\mathbb{C}$ denote the set of all pairs of fine-grained

features (s, t) to be considered in logistic regression. Then, we have

$$h(\mathbf{x}) = \sum_{(s,t)\in\mathbb{C}} \phi(\mathbf{x}_s, \mathbf{x}_t) + \sum_{i=1}^{d} \beta_i x_i + \beta_0, \quad (6)$$

where

$$\phi(\mathbf{x}_s, \mathbf{x}_t) = \sum_{i\in s, j\in t} w_{ij} x_i x_j, \quad (7)$$

and w_{ij} is the weight to be determined. In fact, which pairs of features to be included in $\mathbb{C}$ depends on both human expertise over the problem and offline empirical tests.

3.3 Handling Feature Sparsity with *Partial FFM*

In the training process, e.g., using stochastic gradient descent (SGD), β_i (or w_{ij}) is updated only if the corresponding sample contains a non-zero x_i (or non-zero $x_i x_j$). However, most of the feature vectors $\mathbf{x}_s$ are very sparse. And for some feature vector $\mathbf{x}_s$, some of the entries in that feature vector is even rarely non-zero, which results into a even more sparse and unbalanced distribution of non-zeros in the terms $x_i x_j$. Under the SGD training, the weights of the rare terms (rarely non-zero terms) can be unstable and oscillating, degrading the effectiveness of the model.

Low-rank or dimension reduction based methods, such as factorization machine (FM)[15] and field aware factorization machine (FFM) [10], are reported to be effective to address this issue. Specifically, FM trains a hidden vector for each entry of the feature vector $\mathbf{x}$ and models the weight w_{ij} of a second order crossed term $x_i x_j$ as the inner product of the corresponding hidden vectors of x_i and x_j. With this idea, the cross term is modeled as

$$\phi(\mathbf{x}_s, \mathbf{x}_t) = \sum_{i\in s, j\in t} < \mathbf{v}_i, \mathbf{v}_j > x_i x_j, \quad (8)$$

where $< \cdot, \cdot >$ denotes the vector inner product, and $\mathbf{v}_i \in \mathbb{R}^k$ represents the k-dimensional hidden vector of the entry x_i. Furthermore, FFM expands the capacity and expressiveness of the FM model, and keeps a separate hidden vector for each pair of feature crossing products. In this way, the cross term is modeled as

$$\phi(\mathbf{x}_s, \mathbf{x}_t) = \sum_{i\in s, j\in t} < \mathbf{v}_{it}, \mathbf{v}_{js} > x_i x_j, \quad (9)$$

where $\mathbf{v}_{it} \in \mathbb{R}^k$ represents the k-dimensional hidden vector of the entry x_i *specifically for* its crossing product with the feature t. In our model, to address the sparsity issue for the feature crossing terms and since different fine-grained features have different distributions, we integrate the idea of FFM into the logistic regression model in (6). Instead of expressing the second-order feature crossing via (7), we model it as (9).

Finally, the prediction function is defined by combining (2), (6) and (9). Unlike the original FFM, we only include the selected crossed features to regularize the model and reduce the overall model complexity during both the training and prediction phases. We call the model resulted from the combination of (2), (6) and (9) a *partial FFM*.

We use stochastic gradient descent (SGD) to train the model. Typical momentum methods are integrated to help speedup the convergence speed [16].

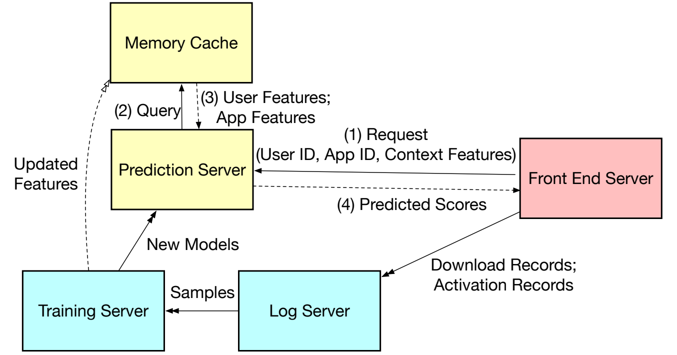

Figure 2: A typical user action prediction system, cyanogen blocks are the offline training module and yellow blocs are the online serving module.

Remarks: Linear models are efficient and robust in prediction tasks. Essentially, the linear model acts as a premium fusion technique to effectively combine a variety of features and weak predictors. The direct linear part combines different features while the (second order) basis expansion part increases the capacity of the model. By leveraging FFM to model the coefficients of the second order crossed features, the model simultaneously yields the hidden feature vectors and adaptively merges the prediction powers of different features and basis.

4 SPEEDUP THE ONLINE PREDICTION

Since improvement of the accuracy brings about high profit in such problems, we always take all means to make the prediction as accurate as possible. One approach in the linear framework is to include more crossed features and test the results. Although linear models with second order feature crossing are very effective models and efficient to train and serve, they still suffers high latency when problem scales up to the size in *Tencent* if more and more crossed features are included. To address this issue, we propose a technique to speedup the online prediction for the linear based models with vast second order crossed features. We take advantage of the typical system architecture and effectively separate the evaluation function into two stages. In the offline stage we pre-calculate some middle results and store them as features. In the online stage, the final prediction can be efficiently calculated with the help of those middle results. We show the whole offline training and online serving system fist, and present our speedup techniques based on the typical system structure.

4.1 Training and Serving System Overview

Fig. 2 shows the whole procedure for the prediction task. There are mainly two parts in the system: the offline training module and the online serving module. The offline module train the model from the collected samples and do necessary preprocessing for the features. The online serving modules handles the request, retrieving necessary feature data and predict the results. The memory cache system stores the user features and item features, which are updated on a daily basis along with the model.

For online serving, each request comes to one of the prediction servers with a single user ID, a lists of candidate app IDs and context features, i.e., a tuple of (user ID, app IDs, context features) . Before calling a prediction procedure, user features and item features are retrieved from the memory cache system by their IDs. These user features, item features and the context features are joined together to construct complete prediction samples. The prediction server calculates the predicted scores over the complete samples and return the score list back.

The offline training module keeps updating both the prediction model and features of users and apps in a daily basis. Models are trained based on samples from past period of days and validated on the samples of the last day. In our problem, all samples are tracked at the front end servers and logged. After a certain time window, those download records that are proved to be activated will be marked as positive samples and other download records are marked as negative samples. When models are trained and validated, they are pushed on to the prediction servers. As for the user features and item features, they are evaluated by different algorithms or rules and updated within the memory cache system in a daily basis.

Note that in the whole system, user features and item features are updated regularly in a daily basis and stored in the memory cache system. Usually only the raw or engineered user or item features are stored there. However, there is no strict constraints on what we store and we can leverage the memory cache system to store some pre-computed middle results to speed up the online procedure.

4.2 Speedup the Serving Time via Integrating the crossed terms and Features

The request response time is critical to quality of user's experience. In the online prediction procedure, response time comes from two parts. The first part is the feature preprocessing time, which include the inquiry time for retrieving features from the memory cache system and the time for joining those features to the samples and dealing with some feature preprocessing. The second part is the time consumed to infer the score from the prediction model. The first part usually related to the performance of the memory cache system and the data structure. We don't consider the optimization for that in this work. In the second part, the time can vary a lot from models to models with different complexity. Specifically, for the linear model with second order feature crossing, evaluating the linear aggregation of the raw features takes constant small time while evaluating the crossed terms contribute most proportion of time consumed in the prediction phase.

To speedup the calculation of (2), we need to carefully handle the calculation for the second order feature crossing terms in (6). A simple observation of evaluating (7) and (9) inspired our approach. Taking evaluating (9) as an example,

$$\begin{aligned}\phi(s,t) &= \sum_{i\in s, j\in t} < \mathbf{v}_{it}, \mathbf{v}_{js} > x_i x_j \\ &= \sum_{i\in s, j\in t} w_{ij} x_i x_j \qquad (10)\\ &= \sum_{i\in s} x_i \sum_{j\in t} w_{ij} x_j. \qquad (11)\end{aligned}$$

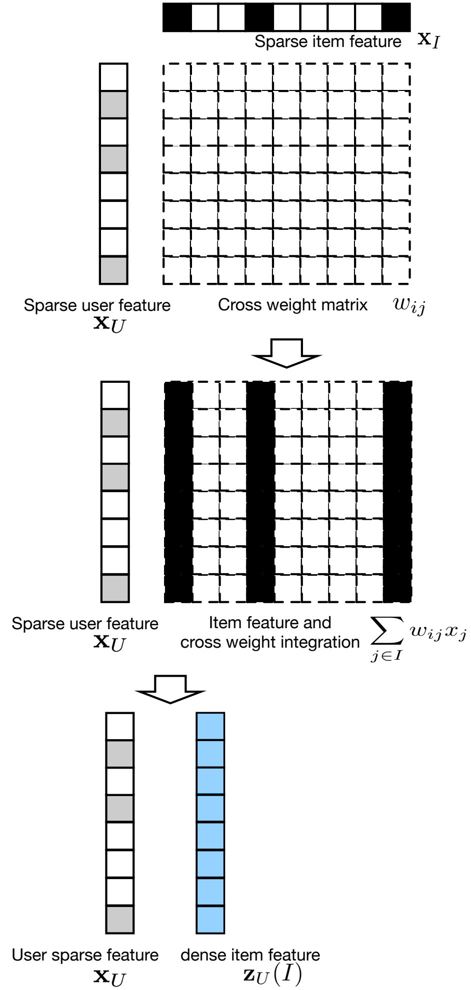

Figure 3: Demonstration of how the cross weights are integrated into dense feature vectors. Original calculation is a Cartesian product and the integrated one is only a dot product.

If we denote the aggregation term in (11) as

$$z_t(i) = \sum_{j\in t} w_{ij} x_j, \qquad (12)$$

and $\mathbf{z}_t(s)$ as the vector that combines all $z_t(i), \forall i \in s$. we can get

$$\begin{aligned}\phi(s,t) &= \sum_{i\in s} x_i z_t(i) \\ &=< \mathbf{x}_s, \mathbf{z}_t(s) >, \qquad (13)\end{aligned}$$

Note that for evaluating the second order feature crossing terms, the trick in (11) can be applied no matter what kind of methods we use to model the second order feature crossing terms, since we can always evaluate the weights for the crossing terms first before the transformation. In the above example, (10) produces the weights of the crossing terms for the partial FFM algorithm.

In a general, the transformation idea in (13) can be applied to evaluate any feature crossing terms from two sub-vectors of $\mathbf{x}$ with

no common entries. As we have discussed in Sec. 4.1, to speedup the prediction procedure, we can leverage the memory cache system to store some of the middle results. Specifically, the feature crossing terms between user features and item features can be evaluated as

$$\begin{aligned}\phi(U, I) &= \sum_{i \in U} x_i \sum_{j \in I} w_{ij} x_j \\ &= \sum_{i \in U} x_i z_I(i) \\ &= < \mathbf{x}_U, \mathbf{z}_I(U) >, \end{aligned} \quad (14)$$

where $z_I(i)$ and $z_I(U)$ is defined similarly to $z_t(i)$ and $z_t(s)$ with index sets notations changed. By (14), the evaluation for crossed terms between two sparse feature vectors are deduced to evaluate a dot product between a sparse feature vector and a dense preprocessed feature vector. The model weights and the item features are simultaneously integrated to a new item features vector that calculated and cached into the memory cache system before serving the model. Note that we can also integrate the model weights and user features into new user feature vectors as well.

From this observation, in an online prediction system, we can integrate the complex feature crossed terms into some new feature vectors. This helps to reduce the complexity of evaluating the Cartesian product of two sparse feature vectors to evaluating a inner vector product between a sparse vector and a dense vector. Fig 3 shows a toy example of this procedure.

4.3 Online Prediction Decomposition

We propose a general online prediction calculation routine for a certain class of system that adopts linear based models with selected or full second order crossed features. In these system, features for each sample are associate to several groups of objects. When requests comes, samples are first joined with the features in some memory cache. Those memory cache allows us the flexibility of pre-calculating some middle result to reduce the online prediction complexity.

In these systems, the most time consuming part is the second order crossed feature terms. Those terms can be divided into two groups: the ones of crossed features from the same feature group and the ones from different feature groups. For the first type of crossed features, the aggregation can be evaluated offline and saved directly as a feature under the individual owners of that group. For the second type, aggregation can be calculated with the trick in (14) while a middle result will be calculated offline and stored as a new historical feature vector for the features from one of the two groups. In this way, the original complexity of $O(N^2)$ to evaluate the crossed features is reduced to $O(N)$. Note that the linear terms can also be aggregated offline and stored in the same feature of that for the crossed features for each group.

There are additional storage overhead in the online memory cache system. For each pair of feature groups, the storage cost is $O(nN)$, where n denotes the total number of items in all groups and N is the size of the features. The storage overhead is $O(G^2nN)$ for crossed features evaluation. For the aggregated result within the same group, the storage will be $O(n)$. The overall storage overhead is $O(G^2nN + n)$. Note that in real system, the number of feature groups will not be large. In our case, it is 2. The storage overhead scales linearly to both the total number of items and feature size.

4.4 Saving the Memory Cache

Theoretically the proposed calculation framework can be applied to any prediction procedure if some of the features are stored in some memory cache. Therefore we can store the integrated new feature vector in that memory cache system. However, in real application, the storage overhead of the new scheme is not negligible. We demonstrate the practical concerns for the proposed calculation scheme. In our problem, we have three classes of features from user features, app features and context features in which user features and item features are stored in memory cache.

As we pointed out in Sec. 4.2, the offline transformation can be aggregated on one of the two groups of features if we cache the new integrated features for that group. We choose the one that require less offline computation and/or storage overhead. In our problem, we have a user scale of 10^8 with user feature size 10^3 while we have app scale of 10^5 with app feature size 10^3. We would have roughly 10^5 calculation and 10^8 storage overhead if we aggregate the crossed feature weights and the app features into new app features. The offline calculation complexity would be 10^8 and the storage overhead would be 10^{11} if aggregating the user features. Therefore we choose to aggregate the app features in this case. In our framework, since we can integrate the crossed weight terms into either side of the related two feature groups, we can always select the side that takes less memory cost.

4.5 Adding Selected Crossed Terms

Our proposed speedup technique can be applied to evaluate any crossed features as long as we can retrieve the feature vectors for one of the two groups. And both the storage cache overhead and the offline computation complexity only scales linearly to the number of feature owners and size of features. However, it might be still too large. In our scenario, for the crossed terms between users and context features, we can only integrate the crossed terms onto the user features since context features are directly collected beside the request. But we have user size of 10^8, which is an intolerable storage overhead. What's more, some of the cross features between these two group might be important to the model accuracy. In this case, we can simply add selected important crossed terms and evaluate them term by term as a normal logistic regression model with crossed features. When doing the prediction, we directly calculate those crossed terms instead of applying our speedup trick.

5 EXPERIMENTS

We conduct experiments based on a 15-day trace dataset of the download-activation logs from *Tencent MyApp* from Sept 1, 2017 to Sept 15, 2017 to evaluate our proposed schemes. In this dataset, we have about 2×10^8 users and 5×10^4 apps. The raw features have a dimension over 1×10^4. We mainly focus on the balance between the prediction performance, prediction latency and resource consumptions.

We test the prediction performance and speed for several schemes to demonstrate the effectiveness of our proposed algorithm. Specifically, we evaluate the following different scenarios.

DLR. Direct logistic regression models with no cross features.

LR. Logistic regression model with fine selected set of cross features.

FFM. Field aware factorization machine algorithm that taking all cross features.

pFFM. Partial FFM model that takes all the item-user, item-context cross features into consideration.

pFFM+. Partial FFM model that takes all the item-user, item-context cross features, along with additional selected other cross features added.

When we test the prediction speed, for the models with crossed features, we test both the ordinary approach that compute the crossed feature directly and our proposed speedup approach. Those scenarios will have the same prediction performance but different time delay performance. For the convenience of presentation, we denote the scenario with our speedup technique as **FFM'**, **pFFM'**, **pFFM+'** respectively.

5.1 Evaluation Metrics

We adopt several different performance metrics to comprehensively demonstrate the prediction performance power of different models. In our task, the predicted scores are some middle results for other task, so we select several effective metric from both classification problem and regression problem. For those regression metric, we regard the 0-1 label as the true value and compare the predicted probability against those numerical value. For those classification metric, we adopt standard methods.

Logloss. As we mentioned in Sec. 2, logloss is a natural metric to evaluate how well the predicted probability where a smaller log loss implicit a larger likelihood that the overall predicted scores are close to the true label distribution. Specifically, for sample $D = \{(\mathbf{x}^{(0)}, y^{(0)}), (\mathbf{x}^{(1)}, y^{(1)}), \ldots\}$, the logloss is

$$logloss = \sum_{(\mathbf{x},y)\in D} -y \cdot \log(f(\mathbf{x})). \tag{15}$$

AUC. Area under the receiver operating curve, which measures the probability that a randomly drawn positive sample is scored higher than a negative sample.

$$AUC = \int_0^1 \mathrm{TPR}(t) d\mathrm{FPR}(t), \tag{16}$$

where $\mathrm{TPR}(t)$ is the true positive rate and $\mathrm{FPR}(t)$ is the false positive rate with a threshold t. AUC indicate the global ranking performance over the test samples.

F1 scores. The common metric for classification that combines both precision and recall.

$$F1 = \frac{2PR}{P+R}, \tag{17}$$

where P is the precision and R is the recall. F1 score usually varies with different threshold value. In our experiment, we simply select the threshold that yields the best F1 score.

MAE. Mean average error.

$$MAE = \frac{1}{\|D\|} \sum_{(\mathbf{x},y)\in D} \|f(\mathbf{x}) - y\|, \tag{18}$$

where $\|D\|$ denotes the number of samples in the set D.

MSE. Mean squared error.

$$MSE = \frac{1}{\|D\|} \sum_{(\mathbf{x},y)\in D} (f(\mathbf{x}) - y)^2. \tag{19}$$

5.2 Model Training and Accuracy Performance

We use the data of the first 14 days as the training set and use the data from the last day as testing set. We test all the algorithms of DLR, LR, FFM, pFFM and pFFM+. Specifically, DLR only does a plain logistic regression on the raw features. FFM takes all the second order crossed features. pFFM includes all the item-user crossed features and item-context crossed features. LR contains manually selected sets of crossed features with continuously effort of fine tuning, which is a stable version of model for the current Tencent MyApp. The pFFM+ expands the pFFM by adding all the selected user-context cross terms in the LR model.

Table 2 presents the prediction performance for those schemes. We can see that models with second order crossed features significant out performs the model with raw features, which confirms the observations in [1, 10]. Besides, in our problem, the performance generally gets better if more second order crossed features are included in the model, where FFM achieves the best model accuracy. Note that the pFFM, although not contains all the crossed features, also get a great boost compared to the manually tuned LR. Surprisingly, with addition crossed features in pFFM+, it has a very close prediction performance to the full FFM. However, the serving complexity can be much less than the FFM.

5.3 Prediction Speedup and Storage Overhead

To evaluate the effectiveness of speedup for our proposed technique, we do a prediction experiment on the data trace on a platform with Intel(R) Xeon(R) CPU E5-2670 v3 and 128 GB memory. As we discussed in Sec. 4.2, the actual serving time includes two parts: feature retrieving and preprocessing part and prediction part. In our system, the first part takes a constant time of several milliseconds. We do simulation to test the second part and simply cache all the necessary features locally. As we will discuss later, the FFM' will actually consumes an extremely large memory and we are not able to really cache the features for that. Since we only need to test the prediction speed for each scenario, we simply use the same integrated dense user vector for all the users. Although the prediction score makes no sense in this manner, the prediction speed should be close to real performance. To simulate the real calculation load, we replay the trace to generate requests. We aggregate the time for evaluating 2000 samples to simulate a request with one user and a list items.

Fig. 4 and Fig. 5 presents the distribution of the total prediction time for different scenarios. Specifically, Fig. 6 and Fig. 7 presents the distribution of the prediction time for evaluating the crossed features. Note that the results are based on simulation without consideration of concurrency issue and in real case the performance should follow the similar trend where the gap between scenarios will be much larger. We can see that our proposed speedup technique can greatly reduce the prediction time. In our problem, the FFM', pFFM' and pFFM' can finish the prediction within nearly

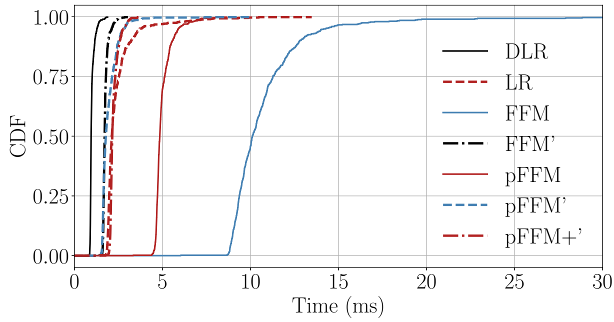

Figure 4: CDF of total calculation time for each request.

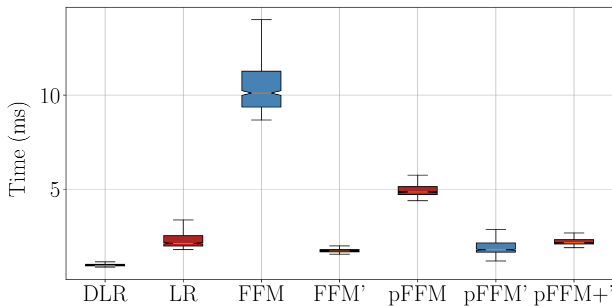

Figure 5: Box plot of total calculation time for each request.

the same time to LR. Bearing mind that the LR only includes several manually selected crossed features while the FFM', pFFM' and pFFM+' contains at least vast number of item-user crossed features and item-context crossed features. Moreover, comparing to pFFM', pFFM+' contains several important crossed features between user features and context features, but the prediction time is still close to pFFM'. Note that in terms of prediction performance, the pFFM+' has an obvious improvement over pFFM', which is a practical trick to further improve the model performance with tiny prediction delay increase as we discussed in Sec. 4.5

Table 1 shows the estimated additional storage overhead in the memory cache system due to applying our speedup technique. To deploy the speedup in FFM', we have to generate new user features for such a huge user pool of size 2×10^8. As we discussed in Sec. 4, it will be unwise to integrate the crossed terms onto the group that contains too much owners like user features. In the contrary, since the number of the items are not large, integrating the crossed terms onto the item features only incurs a reasonable extra memory cache resource. To conclude, the pFFM' achieve a close prediction performance to a FFM that contains all the second order crossed features, while yields a very fast prediction speed close to LR at a reasonable extra storage overhead.

6 RELATED WORKS

User action prediction problem has been a hot topic. Previous works are mainly focused on improving the prediction performance with varieties of models and practical tricks.

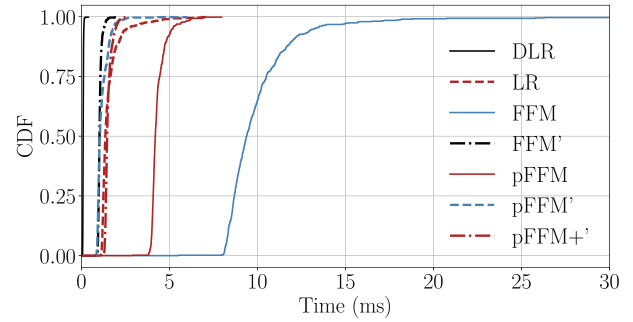

Figure 6: CDF of calculation time to calculate $\sum_{(s,t)\in\mathbb{C}} \phi(\mathbf{x}_s, \mathbf{x}_t)$, the aggregation from all crossed features.

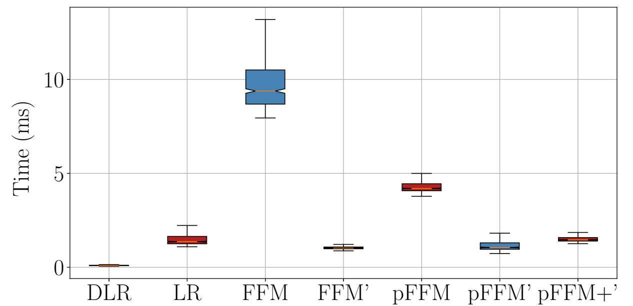

Figure 7: Box plot of calculation time to calculate $\sum_{(s,t)\in\mathbb{C}} \phi(\mathbf{x}_s, \mathbf{x}_t)$, the aggregation from all crossed features.

Table 1: The Estimated extra memory cache overhead for different algorithms.

Algorithm Name	FFM'	pFFM'	pFFM+'
Extra Storage	44.5 TB	11.8 GB	11.8GB

Table 2: The Performance of Different Algorithms

algorithm name	**MAE**	**MSE**	**AUC**	**Log Loss**	**F1 score**
DLR	0.3159	0.1583	0.8154	0.4767	0.6446
LR	0.2883	0.1463	0.8458	0.4409	0.6713
pFFM	0.2762	0.1408	0.8575	0.4273	0.6859
pFFM+	0.2734	0.1397	0.8603	0.4231	0.6879
FFM	0.2742	0.1380	0.8637	0.4179	0.6915

[11] estimates the advertisement displaying conversion rate via inferencing over binomial distribution along specific data hierarchies. Logistical regression is applied to combine different estimators. [2, 13] uses linear logistic regression models with second order feature cross. In [2], a novel hash trick is also proposed to deal with the sparse features of vast dimensions. It is reported that low polynomial degree feature interaction models can effectively and efficiently capture most of the feature interaction informations in [1]. Specifically, a full second order polynomial cross is so call a

polynomial kernel of 2. [15] combines both power of a full second order polynomial feature cross and dimension reduction. They propose factorization machine (FM) model that learn a latent feature vector for every feature and estimate the feature cross via the inner product of those features. [9, 10] propose field-aware factorization machine (FFM) model that further improves the model capacity of FM by keeping different latent feature vectors for each pair of fields of feature interactions. As pointed in [9, 19], in a linear based framework, second order terms are still time consuming in both training and testing phase. We seek the insight of those linear based approach and present a practical linear based model frame work.

Recently, deep neural network based models with much higher complexity and model capacity are proposed. [12, 21] proposes CNN and RNN based models in which the CNN based model explores the relationship between neighboring features and RNN model addresses the sequential information. [20] proposes factorization machine supported neural network (FNN) model which pre-trains the network by FM. [14] proposes Product-based Neural Network (PNN) that introduces a product layer for the embedded features to model the feature interactions. [3] seeks the insight of DNN and proposes a hybrid network that combine both good estimators from feature engineering in the shallow network and feature interactions in the deep network. [7] combines FNN and DNN part to simultaneously capture both low level and high level feature interactions. Those models improves the prediction accuracy at some certain scenarios at the price of much higher computation complexity at both the training and prediction phase.

For application in real systems, [13] presents an online learning scheme for a linear based model. They also addressed several practical challenges on on memory saving, performance monitoring etc. for real system. [8] also presents practical issues and tricks on them. A hybrid model combining gradient boosting tree and logistic regression is proposed to help learn the feature binning and interactions. [9] focus on optimizing the training speed by deploying a distributed training procedure and applying a warm start. A much more CPU resource consumption is also reported and they handle it by reducing the latent factor size of the model regardless of the degrade of performance. In our work, we leverage the online prediction system architecture to speedup the online prediction.

7 CONCLUSIONS

In this paper, we propose a fast linear computational framework to handle the user action prediction task in a large-scale system, namely Tencent MyApp. In our framework, we carefully decompose the time consuming online serving procedure into several phases and used preprocessing in exchange for dimensionality reduction, such that upon the serving of a request, the corresponding prediction result can be rapidly evaluated with the preprocessed intermediate results. With this technique, we reduce the online prediction time from $O(N^2)$ to $O(N)$, N being the dimension of the large feature space under consideration. Meanwhile, the prediction performance is comparable with such models as complex as FFM that takes all second order crossed features into account. Experiments on a 15-day data trace from Tencent MyApp shows the effectiveness of our proposed framework.

REFERENCES

[1] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. 2010. Training and testing low-degree polynomial data mappings via linear SVM. *Journal of Machine Learning Research* 11, Apr (2010), 1471–1490.

[2] Olivier Chapelle, Eren Manavoglu, and Romer Rosales. 2015. Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5, 4 (2015), 61.

[3] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*. ACM, 7–10.

[4] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 191–198.

[5] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning*. Vol. 1. Springer series in statistics New York.

[6] Thore Graepel, Joaquin Q Candela, Thomas Borchert, and Ralf Herbrich. 2010. Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine. In *Proceedings of the 27th international conference on machine learning (ICML-10)*. 13–20.

[7] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. *arXiv preprint arXiv:1703.04247* (2017).

[8] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*. ACM, 1–9.

[9] Yuchin Juan, Damien Lefortier, and Olivier Chapelle. 2017. Field-aware factorization machines in a real-world online advertising system. In *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee, 680–688.

[10] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware factorization machines for CTR prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 43–50.

[11] Kuang-chih Lee, Burkay Orten, Ali Dasdan, and Wentong Li. 2012. Estimating conversion rate in display advertising from past erformance data. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 768–776.

[12] Qiang Liu, Feng Yu, Shu Wu, and Liang Wang. 2015. A convolutional click prediction model. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. ACM, 1743–1746.

[13] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1222–1230.

[14] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 1149–1154.

[15] Steffen Rendle. 2010. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 995–1000.

[16] Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016).

[17] Eric Schurman and Jake Brutlag. 2009. The user and business impact of server delays, additional bytes, and HTTP chunking in web search. In *Velocity Web Performance and Operations Conference*.

[18] Guy Shani and Asela Gunawardana. 2011. Evaluating recommendation systems. *Recommender systems handbook* (2011), 257–297.

[19] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. 2012. Recent advances of large-scale linear classification. *Proc. IEEE* 100, 9 (2012), 2584–2603.

[20] Weinan Zhang, Tianming Du, and Jun Wang. 2016. Deep learning over multi-field categorical data. In *European conference on information retrieval*. Springer, 45–57.

[21] Yuyu Zhang, Hanjun Dai, Chang Xu, Jun Feng, Taifeng Wang, Jiang Bian, Bin Wang, and Tie-Yan Liu. 2014. Sequential Click Prediction for Sponsored Search with Recurrent Neural Networks.. In *AAAI*. 1369–1375.