Contents

1 Introduction

1.1 Recommender System

Recommender system is one of the most popular research problems in the field of data mining. It becomes an urgent matter that how to mine the personalized behavior from the large-scale user behavior data, which is high dimensional and sparse, and employ it to generate personalized recommendation with high accuracy for users. With the data from KDD cup 2012 about predicting the click-through rate of ads given the query and user information, we are expected to accurately predict the CTR of ads in the testing instances.

$$\mathrm{CTR} = \frac{\mathrm{Click}}{\mathrm{Impression}} \times 100\% \quad (1)$$

1.2 KDD Cup 2012

Track 2 of KDD Cup 2012 is a competition for search advertising. The task of the competition is to predict the click-through rate (CTR) of ads in a web search engine given its logs in the past. The dataset, which is provided by Tencent, includes a training set, a test set and files for additional information. The training set contains 155,750,158 instances that are derived from log messages of search sessions, where a search session refers to an interaction between a user and the search engine. During each session, the user can be impressed with multiple ads; then, the same ads under the same setting (such as position) from multiple sessions are aggregated to make an instance in the dataset. Each instance can be viewed as a vector (#click, #impression, DisplayURL, AdID, AdvertiserID, Depth, Position, QueryID, KeywordID, TitleID, DescriptionID, UserID), which means that under a specific setting, the user (UserID) had been impressed with the ad (AdID) for #impression times, and had clicked #click times of those. In addition to the instances, the dataset also contains token lists of query, keyword, title and description, where a token is a word represented by its hash value. The gender

and segmented age information of each user is also provided in the dataset.

The test set contains 20,297,594 instances and shares the same format as the training set, except for the lack of #click and #impression. The test set is generated with log messages that come from sessions latter than those of the training set.

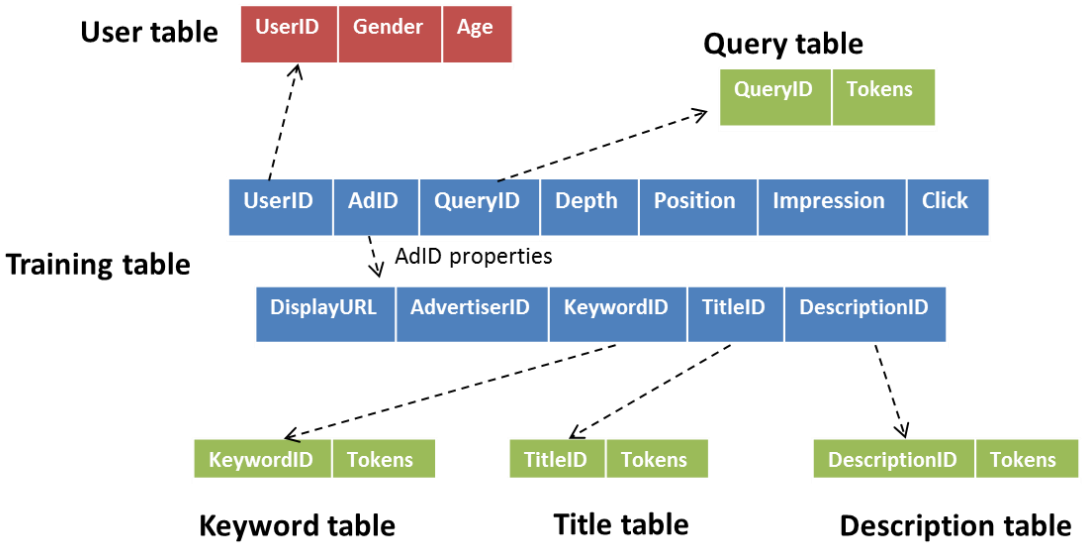

Fig. 1 Data Sets Components

1.3 Purpose of the Competition

The goal of the competition is to predict the click-through rate for each instance in the test set. The goodness of the predictions is evaluated by the area under the ROC curve (AUC), which is equivalent to the probability that a random pair of a positive sample (clicked ad) and a negative one (unclicked ad) is ranked correctly using the predicted click-through rate. That is, an equivalent way of maximizing the AUC is to divide each instance into (#click) of positive samples and (#impression-#click) negative samples, and then minimize the pairwise ranking loss of those samples using the predicted click-through rate. During the competition, teams are allowed to upload the predictions on the test set. Teams are allowed to select up to 3 submissions to be evaluated on the private test set before the end of the competition.

The paper is organized as follows. Section 2 describes the framework of our system. Section 3 discusses the Deep FM model. Section 4 then introduce how we use the individual Deep FM model using the validation set and the test set, respectively, to form

the final system. Finally, we conclude in Section 5.

2 Framework

We first provide an overview of our proposed system. Then, we discuss a key step in building the system: generating the internal validation set. The set not only is used for parameter and model selection, but also plays an important role in the blending stage. Finally, we show our efforts in another key step: generating meaningful features for training the individual models.

2.1 System Overview

The proposed system can be divided into three stages: generating sparse features, blending with the validation set, and ensemble learning with the test set. In the first stage, we apply several different approaches to capture different concepts and many sparse features. In the second stage, we apply non-linear blending method which called DeepFM to predict the results from the features in the first stage. Then, in the final stage, we pick predictions from the second stage, along with every model trained by DeepFM with different hyper-parameters. The final solution thus aggregates all our models and gets a final result using AUC in test set, 78.7%.

2.2 Training Set, Validation Set and Test Set

The validation set is important in data mining systems to select and combine models. In this competition, the training set comes from earlier time period than that of the test set. Thus, we naturally want to use the later data in the training set for validation. However, the training and test sets do not contain time information, and it is non-trivial to directly obtain such a validation set.

We take a simple alternative for generating the validation set. We randomly sampled 1/11 of the training instances as the validation instances. Table 1 shows the data statistics of the validation and the sub-training sets.

Table 1: Statistics of the Sub-training Set and Validation Set

	#instance	#click	#impression
sub-training	137169180	7550609	216038149
validation	12469925	667024	19544730

We have looked at other different methods to generate the validation set, but none of them appears more robust then the simple instance-based sampling. For example, we have tried impression-based sampling; that is, we randomly extract 1/11 of the impressions to make the validation set.

The generated validation set contains 6422303 users, and 6422303 of them are also in the sub-training set. However, in the test set, there are only 3263681 users, and only 1364217 of them are in the training set. The statistics show a drastic difference between such a validation set (with respect to the sub-training set) and the test set (with respect to the whole training set). In particular, such a validation set does not contain enough cold-start users. Although the simple instance-based sampling also suffers from the same problem (possibly because of the lack of time information), but the ratio of cold-start users appears closer to the statistics of the training and test sets.

The validation set also plays an important role in our 3-stage framework. In the first stage, we generate all sparse features.

Table 2 Stat. of Attr. on Training Set, Validation Set and Test Set

Attributes	Training Set	Validation Set	Test Set	Validation Set & Training Set	Test Set & Training Set
AdID	631849	355071	300012	355071	269995
AdvertiserID	14842	14486	11157	14486	10753
QueryID	23295643	4950152	3801978	4950152	1660789
KeywordID	1161369	521682	495421	521682	431313
TitleId	3626120	1315048	1262498	1315048	937550
Description	2848249	1046266	981985	1046266	737008

UserID	21473870	6422303	3263681	6422303	1364217

2.3 General Features

Before introducing our model, we first describe features here.

2.3.1 Categorical Features

The dataset contains information on UserID, AdID, user's gender, ad's position, etc. We can take those fields as categorical features directly. We use AdID, QueryID, KeywordID, UserID, and ad's position as categorical features in our models.

2.3.2 Basic Sparse Features

As mentioned above, we are provided with a set of categorical features such as UserID, AdID, user's gender, ad's position. We have also tried expanding those categorical features into binary features. For example, there are 21473870 different UserIDs in the training data, and we expand UserID as a 21473870-dimensional binary feature vector. We expand the following categorical features into binary features: AdID, AdvertiserID, QueryID, KeywordID, TitleID, DescriptionID, UserID, DisplayURL, user's gender, user's age, depth of session, position of ad and combination of position and depth. We also expand query's tokens, title's tokens, description's tokens and keyword's tokens into binary features. That is, if a token occurs in title, query, description or keyword, the corresponding value in the feature vector will be 1, or 0 otherwise. For some of our models, we only construct the expanded features for those IDs with clicks. For those IDs without any clicks, we do not generate any binary indicator for them. Eventually we have features with more than ten million dimensions, but only a few of them are non-zero. That is, the feature vectors are sparse.

2.3.3 Click-Through Rate Features

Click-through rate features are simple but very useful in our models. For each

categorical feature, we compute the average click-through rate as an additional one-dimensional feature. Take AdID as an example. For each AdID, we compute the average click-through rate for all instances with the same AdID, and use this value as a single feature. This feature represents the estimated click-through rate given its category. We compute this kind of feature for AdID, AdvertiserID, depth, position and $\frac{\text{depth}-\text{position}}{\text{depth}}$.

However, we observe that some categories come with only a few or even no instances. Computing the click-through rate directly for those categories would result in inaccurate estimations because of the insufficient statistics. Thus, we apply smoothing methods during click-through rate estimation. We mainly use a simple additive smoothing and we name it pseudo click-through rate (pseudo-CTR). In our experiments, we set α as 0.05 and β as 75. We generate pseudo-CTR features for AdID, AdvertiserID, QueryID, KeywordID, TitleID, DescriptionID, UserID, DisplayURL, user's age, user's gender and $\frac{\text{depth}-\text{position}}{\text{depth}}$.

2.3.4 ID Raw Value Features

We observe that the numerical value of IDs contains some information. For instance, we observe that #impression decreases when the value of KeywordID increases. We suspect that IDs values may contain time information if they were sequentially assigned. To exploit this information, we use the raw value of IDs as features directly. We have this kind of feature for TitleID, QueryID, KeywordID, DescriptionID and UserID.

To capture the trends in the raw ID values more generally, we quantize IDs by their value into 10000 categories. We generate this kind of grouped-ID value feature for AdID, AdvertiserID, QueryID, KeywordID, TitleID, DescriptionID, and UserID.

2.3.5 Other Numerical Features

Other numerical features that we use include numerical value of depth, numerical value of position and the relative position, which is $\frac{\text{depth}-\text{position}}{\text{depth}}$. Another kind of numerical feature we take is called the length feature. In particular, we calculate the number of

tokens for each QueryID, KeywordID, TitleID and DescriptionID, and use a numerical value to represent it. A weighted version is also applied, where each token is weighted by their IDF value instead of 1 above. We also use #impression of AdID, AdvertiserID, ad's position, session's depth, relative position as features in some of our models.

3 DeepFM

In this problem, we use DeepFM algorithm. DeepFM is aimed to learn both low- and high-order feature interactions, which is a Factorization-Machine based neural network (Deep FM). As depicted in Figure 3.1, Deep FM consists of two components, FM component and deep component, which share the same input. For feature i, a scalar w_i is used to weigh its order-1 importance, a latent vector V_i is used to measure its impact of interactions with other features. V_i is fed in FM component to model order-2 feature interactions, and fed in deep component to model high-order feature interactions. All parameters, including w_i, V_i, and the network parameters are trained jointly for the combined prediction model:

$$\hat{y} = sigmoid(y_{FM} + y_{DNN}), \tag{2}$$

where $\hat{y} \in (0, 1)$ is the predicted CTR, y_{FM} is the output of FM component, and y_{DNN} is the output of deep component.

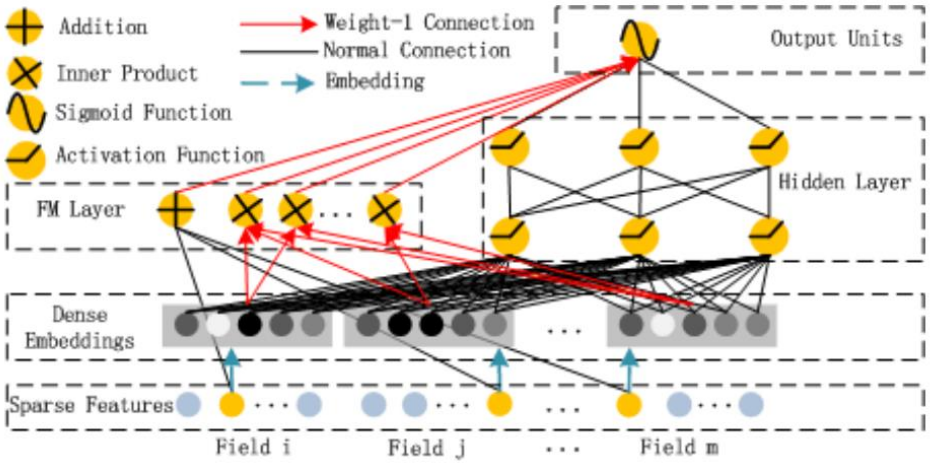

Fig. 3.1 The Architecture of DeepFM

It is worth pointing out that FM component and deep component share the same feature embedding, which brings two important benefits.

1) It learns both low- and high-order feature interactions from raw features;

2) There is no need for expertise feature engineering of the input.

3.1 FM Components

The FM component is a factorization machine, which is aimed to learn feature interactions for recommendation. Besides a linear (order-1) interactions among features, FM models pairwise (order-2) feature interactions as inner product of respective feature latent vectors.

In FM, the parameter of an interaction of features i and j is measured via the inner product of their latent vectors V_i and V_j. Thanks to this flexible design, FM can train latent vector V_i (V_j) whenever feature i (or j) appears in a data record. Therefore, feature interactions, which are never or rarely appeared in the training data, are better learnt by FM. As Figure 3.2 shows, the output of FM is the summation of an Addition unit and a number of Inner Product units:

$$y_{FM} = \langle w, x \rangle + \sum_{j_1=1}^{d} \sum_{j_2=j_1+1}^{d} \langle V_i, V_j \rangle x_{j_1} \cdot x_{j_2}, \tag{3}$$

where the Addition unit $\langle w, x \rangle$ reflects the importance of order-1 features, and the Inner Product units represent the impact of order-2 feature interactions.

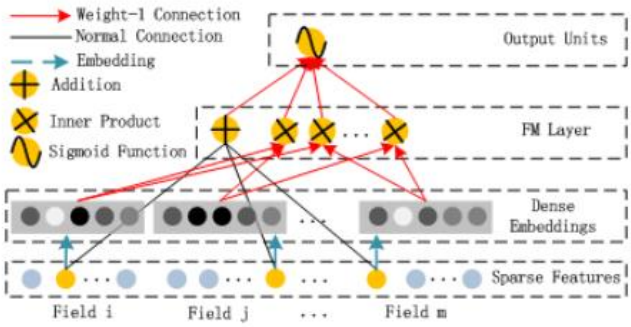

Fig. 3.2 The Architecture of FM

3.2 Deep Components

The deep component is a feed-forward neural network, which is used to learn high-order feature interactions. As shown in Figure 3.3, a data record (a vector) is fed into

the neural network, which is purely continuous and dense, and the input of CTR prediction is quite different, which requires a new network architecture design. Specifically, the raw feature input vector for CTR prediction is usually highly sparse, super high-dimensional, categorical-continuous-mixed, and grouped in fields (e.g., gender, location, age). This suggests an embedding layer to compress the input vector to a low dimensional, dense real-value vector before further feeding into the first hidden layer, otherwise the network can be overwhelming to train. In the proposed Deep FM framework used in solving CTR, there is no implicit requirement on the network structure of the deep component.

Fig. 3.4 highlights the sub-network structure from the input layer to the embedding layer. We would like to point out the two interesting features of this network structure:

1) While the lengths of different input field vectors can be different, their embeddings are of the same size (k);

2) The latent feature vectors (V) in FM now server as network weights which are learned and used to compress the input field vectors to the embedding vectors. In this problem, rather than using the latent feature vectors of FM to initialize the networks, we include the FM model as part of our overall learning architecture, in addition to the other DNN model.

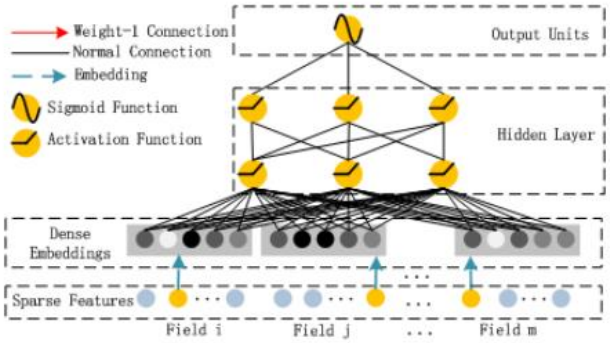

Fig. 3.3 The Architecture of DNN

As such, we eliminate the need of pre-training by FM and instead jointly train the overall network in an end-to-end manner. Denote the output of the embedding layer as:

$$a^{(0)} = [e_1, e_2, \dots, e_m], \tag{4}$$

where e_i is the embedding of i-th field and m is the number of fields.

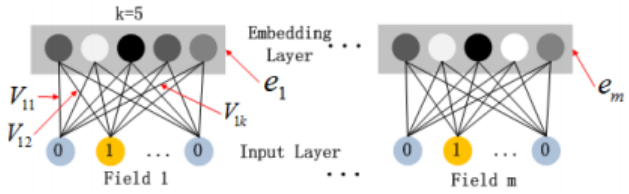

Figure 3.4 The Structure of the Embedding Layer

Then, $a^{(0)}$ is fed into the deep neural network, and the forward process is:

$$a^{(l+1)} = \sigma\left(W^{(l)}\right)a^{(l)} + b^{(l)}, \tag{5}$$

where l is the layer depth and σ is an activation function. $a^{(l)}$, $W^{(l)}$, $b^{(l)}$ are the output, model weight, and bias of the l-th layer. After that, a dense real-value feature vector is generated, which is finally fed into the sigmoid function for CTR prediction:

$$y_{DNN} = \sigma\left(W^{|H|+1} \cdot a^{H} + b^{|H|+1}\right), \tag{6}$$

where $|H|$ is the number of hidden layers.

4 Process of the Project

4.1 Features Construction

In order to construct the features introduced before, we need to explore in the original data. For each value of each feature in the training data, we compute the number of corresponding #click of the value, the number of corresponding impressions of the value and the number of times that the value occurred. For ID's features with tokens, we compute the number of words that the corresponding tokens contain of each ID. To construct sparse feature, we construct the mapping from value to one-hot encoding. To construct dense feature, we compute the maximum and minimum value of each dense feature to normalize them. To construct multi-values features, we compute the IDF of each token and construct the vector to represent each token.

The steps are shown as follows:

1. # get the statistics from original data

   python feature_statistic.py

2. # get the min and max value for normalizing dense features

python features_min_max.py

3. # construct the mapping for one-hot encoding

   python construct_mapping_fn.py

4. # compute the dimension of sparse features

   python count_features_fn.py

5. # compute the IDF of each token

   python tokens_vector.py

6. # get the sum of IDF for each token

   python sum_idf.py

7. # construct the vector to represent each token

   python construct_tokens_vectors.py

4.2 Shuffle the Data

Due to the large size of training data, it is better to shuffle the data first, because we cannot load all data into memory, especially after combining all constructed features into a file, the size increases to 50.5GB.

The step is shown as follows:

8. python shuffle_big_file.py or python shuffle_file_enough_memory.py

4.3 Combine the Data

We combine the CTR, dense features and sparse features into a file, almost all the features need to be mapped from the statistics generated before. The result is shown as follow.

CTR	DisplayURL	AdID	AdvertiserID	Depth	Position	QueryID	KeywordID	TitleID	DescriptionID	UserID	Gender	Age	RelativePosition
0.000	13756257544627676222	9024825	24354	2	1	13161399	53578	2621	1658	0	0	0	0.500
0.000	9285878113705690801	10322557	8640	3	2	13325	6168	327862	279526	0	0	0	0.333
0.000	14340390157469404125	20643971	23808	1	1	10	242	9	9	4186681	1	4	0.000
0.000	6481596396038807703	20187829	6240	3	1	5188	1931	145797	776	446843	2	3	0.667
0.000	12057878999086460853	20192654	27961	2	1	677890	7720	24801	49297	7997286	2	6	0.500
0.000	7207348955256813361	20180120	8747	3	1	32635	407	23907	22458	0	0	0	0.667
0.000	12837504198869453988	21947194	30627	2	1	8	15	847	1053	14016381	2	1	0.500
0.000	7269043601771363193	10365638	4862	3	3	8398981	30685	981	1188	404933	2	3	0.000
0.000	3768407867059173065	10398833	28459	2	1	205	324	21303	19366	221370	1	3	0.500
0.000	1729963849377387605	9583568	23637	3	2	1474034	5997	283	185	0	0	0	0.333
0.000	14347859242938208407	21022176	33931	1	1	307	1059	4647	5092	146290	1	4	0.000
0.000	1776246728347557473	10485506	7121	2	2	6517105	67447	611132	37853	1279378	1	3	0.000
0.000	10645261502079610968	21344606	36632	3	1	22442	11992	360795	159089	133893	1	3	0.667
0.000	12057878999086460853	20147383	27961	3	3	7115586	55196	234204	205515	0	0	0	0.000
0.000	1298151165289752009	20718532	24283	2	2	5752007	20752	263050	49740	884723	1	3	0.000
0.000	2412771796110463309	20067154	23781	3	3	592	1507	1308	1	2702595	2	2	0.000
0.000	17184088135401750265	21401040	28954	3	1	27	2771	11195	6691	6061022	1	5	0.667
0.000	14340390157469404125	3109451	23777	2	2	14617792	1115	1550	1816	0	0	0	0.000

group_Query	group_Keyword	group_Title	group_Description	group_User
5015	428	6	5	0
5	49	809	881	0
0	1	0	0	1751
1	15	359	2	186
258	61	61	155	3345
12	3	59	70	0
0	0	2	3	5862
3200	245	2	3	169
0	2	52	61	92
561	47	0	0	0
0	8	11	16	61
2483	539	1508	119	535
8	95	890	501	56
2711	441	578	647	0
2191	166	649	156	370
0	12	3	0	1130
0	22	27	21	2535
5570	8	3	5	0

aCTR_Ad	aCTR_Advertiser	aCTR_Depth	aCTR_Position	aCTR_RPosition
0.000	0.034	0.040	0.044	0.053
0.010	0.016	0.020	0.024	0.017
0.031	0.043	0.040	0.044	0.031
0.009	0.012	0.020	0.044	0.034
0.039	0.027	0.040	0.044	0.053
0.008	0.009	0.020	0.044	0.034
0.034	0.017	0.040	0.044	0.053
0.023	0.024	0.020	0.011	0.031
0.000	0.009	0.040	0.044	0.053
0.012	0.023	0.020	0.024	0.017
0.064	0.040	0.040	0.044	0.031
0.038	0.028	0.040	0.024	0.031
0.035	0.053	0.020	0.044	0.034
0.005	0.027	0.020	0.011	0.031
0.000	0.024	0.040	0.024	0.031
0.028	0.043	0.020	0.011	0.031
0.006	0.011	0.020	0.044	0.034
0.017	0.028	0.040	0.024	0.031

pCTR_Url	pCTR_Ad	pCTR_Advertiser	pCTR_Query	pCTR_Keyword	pCTR_Title	pCTR_Description	pCTR_User	pCTR_Gender	pCTR_Age	pCTR_RPosition
0.050	0.007	0.034	0.049	0.007	0.007	0.011	0.025	0.042	0.050	0.053
0.004	0.021	0.016	0.006	0.016	0.025	0.025	0.025	0.042	0.050	0.017
0.037	0.031	0.043	0.046	0.034	0.040	0.042	0.070	0.041	0.040	0.031
0.013	0.010	0.013	0.025	0.013	0.021	0.013	0.071	0.044	0.041	0.034
0.027	0.039	0.027	0.049	0.057	0.085	0.097	0.047	0.044	0.050	0.053
0.009	0.010	0.009	0.012	0.007	0.008	0.008	0.025	0.042	0.050	0.034
0.017	0.034	0.017	0.028	0.027	0.038	0.038	0.049	0.044	0.042	0.053
0.029	0.023	0.024	0.048	0.072	0.035	0.035	0.040	0.044	0.041	0.031
0.009	0.001	0.009	0.000	0.004	0.001	0.001	0.028	0.041	0.041	0.053
0.023	0.013	0.023	0.043	0.013	0.027	0.025	0.025	0.042	0.050	0.017
0.046	0.064	0.040	0.023	0.068	0.075	0.075	0.027	0.041	0.040	0.031
0.028	0.047	0.028	0.047	0.072	0.047	0.037	0.039	0.041	0.041	0.031
0.054	0.038	0.053	0.066	0.038	0.039	0.036	0.013	0.041	0.041	0.034
0.027	0.006	0.027	0.048	0.019	0.035	0.036	0.025	0.042	0.050	0.031
0.016	0.015	0.024	0.047	0.012	0.026	0.016	0.059	0.041	0.041	0.031
0.043	0.028	0.043	0.074	0.029	0.030	0.043	0.057	0.044	0.043	0.031
0.011	0.006	0.011	0.003	0.006	0.007	0.009	0.047	0.041	0.045	0.034
0.037	0.018	0.028	0.049	0.015	0.018	0.018	0.025	0.042	0.050	0.031

num_Depth	num_Position	num_RPosition	num_Query	num_Keyword	num_Title	num_Description	num_Imp__Ad	num_Imp__Advertiser	num_Imp_Depth	num_Imp_Position	num_Imp_RPosition
71604486	88801640	35408469	8	4	11	24	434	2145384	106166677	141392659	53083296
36518055	48438568	12242551	2	2	9	18	203	64880	61660205	73636845	20553464
41516564	88801640	90111478	2	4	7	24	123876	15027642	67755997	141392659	141392753
36518055	88801640	11876607	2	2	13	26	4708	78780	61660205	141392659	20553366
71604486	88801640	35408469	4	1	8	24	31647	19730164	106166677	141392659	53083296
36518055	88801640	11876607	2	3	11	23	1638	16024	61660205	141392659	20553366
71604486	88801640	35408469	1	1	8	19	53602	372808	106166677	141392659	53083296
36518055	12398897	90111478	9	2	15	19	11671	242597	61660205	20553375	141392753
71604486	88801640	35408469	5	2	4	23	3228	10565	106166677	141392659	53083296
36518055	48438568	12242551	6	3	12	25	4428	1188904	61660205	73636845	20553464
41516564	88801640	90111478	3	2	4	21	25102	940364	67755997	141392659	141392753
71604486	48438568	90111478	3	1	4	14	26	16098	106166677	73636845	141392753
36518055	88801640	11876607	2	2	11	27	257	3971	61660205	141392659	20553366
36518055	12398897	90111478	4	3	8	16	2563	19730164	61660205	20553375	141392753
71604486	48438568	90111478	8	3	8	22	175	11160	106166677	73636845	141392753
36518055	12398897	90111478	4	3	15	28	751995	1398942	61660205	20553375	141392753
36518055	88801640	11876607	5	6	8	20	5317	12853	61660205	141392659	20553366
71604486	48438568	90111478	5	1	4	13	8888	9880555	106166677	73636845	141392753

sparse_Url	sparse_Ad	sparse_Advertiser	sparse_Depth	sparse_Position	sparse_Query	sparse_Keyword	sparse_Title	sparse_Description	sparse_Gender	sparse_Age
19649	220238	7755	2	1	0	53542	2596	1637	0	0
13200	250736	2752	3	2	13262	6168	323296	276337	0	0
20460	380865	7582	1	1	10	242	9	9	1	4
9238	318878	2077	3	1	5143	1931	143978	768	2	3
17228	319319	8784	2	1	0	7719	24533	48819	2	6
10259	316570	2779	3	1	32541	407	23654	22248	0	0
18336	605496	9974	2	1	8	15	841	1042	2	1
10335	252764	1679	3	3	0	30671	973	1172	2	3
5371	254981	9017	2	1	204	324	21088	19189	1	3
2431	226373	7513	3	2	0	5997	282	184	0	0
20471	439051	11090	1	1	306	1059	4600	5034	1	4
2512	263407	2275	2	2	0	67402	601379	37486	1	3
15216	506100	12930	3	1	22367	11989	355693	157378	1	3
17228	309142	8784	3	3	0	55160	231082	203249	0	0
1829	394648	7740	2	2	0	20746	259514	49260	1	3
3417	298049	7555	3	3	587	1507	1296	1	2	2
24529	521828	9221	3	1	27	2771	11086	6621	1	5
20460	43730	7551	2	2	0	1115	1535	1795	0	0

Fig. 4.3 Features

The step is shown as follow:

9. python combine_data.py train/test

4.4 Split the Data into Training data, Validation and Test Data

Because the data after combining is too large to load it into memory, so we need to train the model by generator. Thus, we need to divide the data into training data and validation data.

The step is shown as follows:

10. python divide_data_to_train_valid.py

4.5 Model Selection

Since this is an older game title, the traditional models, such as logistic regression, naive Bayes and matrix factorization models, are almost being used. Furthermore， we are not familiar with recommended system, do not have much knowledge for features engineering. So, we choose the DeepFM as our model, which can reduce the requirements for feature engineering, the model downloads from GitHub. Because the code did not support multi-value input before, so we modified the Deep FM model to adapt to multi-values input.

Additional, almost all the models are updated by shenweichen in the GitHub to adapt to multi-values input a few days ago.

```
if len(multi_val_input) > 0:
    # multi_val_embedding_list = [[(multi_val_input[i]) for i in range(len(multi_val_input))]]

    # multi_val_embedding_list, multi_val_linear_embedding = get_pooling(multi_val_input, feature_dim_dict, embedding_size,
    #                                     init_std, seed, l2_reg_embedding, l2_reg_deep)

    multi_val_embedding, multi_val_linear_embedding, = get_embeddings_multi_val(
        feature_dim_dict, 'multi_val', embedding_size, init_std, seed, l2_reg_embedding, l2_reg_linear)

    multi_val_embedding_list = [multi_val_embedding[i](multi_val_input[i])
                                for i in range(len(multi_val_input))]
    multi_val_linear_embedding_list = [multi_val_linear_embedding[i](multi_val_input[i])
                                for i in range(len(multi_val_input))]

    multi_val_pooling = [SequencePoolingLayer(seq_len_max=feature_dim_dict['multi_val'][feat][0], mode='mean',
                                              name='multi_val_pool_' + str(i) + '-' + feat)
                         for i, feat in enumerate(feature_dim_dict['multi_val'])]
    multi_val_pooling_list = [multi_val_pooling[i]([multi_val_embedding_list[i], valid_len_input[i]])
                              for i in range(len(valid_len_input))]

    multi_val_linear_pooling_list = [SequencePoolingLayer(seq_len_max=feature_dim_dict['multi_val'][feat][0], mode='mean',
                                                  name='linear_pool_' + str(i) + '-' + feat)
                                     ([multi_val_linear_embedding_list[i], valid_len_input[i]])
                                     for i, feat in enumerate(feature_dim_dict['multi_val'])]

    embed_list += multi_val_pooling_list

    if len(multi_val_linear_pooling_list) > 1:
        multi_val_linear_pooling_list = add(multi_val_linear_pooling_list)
    elif len(multi_val_linear_pooling_list) > 0:
        multi_val_linear_pooling_list = multi_val_linear_pooling_list[0]
    else:
        multi_val_linear_pooling_list = 0

    linear_term = add([multi_val_linear_pooling_list, linear_term])
```

Fig. 4.3 DeepFM Code from GitHub

4.6 Training

4.6.1 Feature Selection

Due to the limitation of time, we cannot add the feature one by one into the model, so we add a group of features into the model, if the AUC is up, then reserve the features.

Table 3 Features Selection

Features	AUC
Sparse	—
Add pCTR	↑
Add aCTR	—
Add Group	↑
Add num_tokens	↑
Add age, depth, position, rposition	↑
Add num_impression	↓
Add num_occurs	↑
Add sum_idf	—

Add tokens_vector	↑
Add ID_val	—
Add mean_idf	—

4.6.2 Model Tuning

The parameters that have been transferred is shown as follow:

- Hidden Size: the layer number and units in each layer of deep net.
- Embedding Size: positive integer, sparse feature and multi-value feature embedding_size.
- Batch Size: the number of samples in a batch.
- Keep Prob: float in (0,1]. keep_prob used in deep net.
- L2 Linear: float. L2 regularizer strength applied to linear part.
- L2 Embedding: float. L2 regularizer strength applied to embedding vector.
- L2 Deep: float. L2 regularizer strength applied to deep net.

The result of model tuning is shown as follow.

Table 4 Model Tuning

Hidden Size	AUC
[128, 128]	—
[512, 256]	↑
[512, 256, 128]	↑↑
[512, 256, 256, 256, 128]	↑↑

Embedding Size	AUC
8	—
20	↑
25	↑↑

Batch Size	AUC

6000	—
10000	↑
15000	↑↑
18000	↑↑↑
30000	↑↑↑

Keep Prob	AUC
1	—
0.9	↓
0.8	↓

L2 Linear	AUC
0.00001	—
0.0001	↓
0	↓
0.000001	↑

L2 Embedding	AUC
0.00001	—
0.000001	↑
0	↓
0.0000001	↓

L2 Deep	AUC
0	—
0.00001	↑
0.0001	↑
0.001	↑
0.01	↑↑

0.1	↑↑

4.7 Ensemble

Due to the fact that we only have time to tuning a model, so we can only generate different result from the same model with random selecting input data, and then simply compute the mean of all results.

5 Summary

Finally, we will conclude our model in a concise way. There are many dense features and sparse features being used in the DeepFM model. We use dense features shown in Fig. 5-1 and sparse features in Fig. 5-2.

```
dense_features = [
    'DisplayURL', 'AdID', 'AdvertiserID', 'QueryID', 'KeywordID', 'TitleID', 'DescriptionID',
    'Depth', 'Position', 'Gender', 'Age', 'RelativePosition',
    'group_Ad', 'group_Advertiser', 'group_Query', 'group_Keyword', 'group_Title', 'group_Description', 'group_User',
    'pCTR_Url', 'pCTR_Ad', 'pCTR_Advertiser', 'pCTR_Query', 'pCTR_Keyword', 'pCTR_Title', 'pCTR_Description',
    'pCTR_User', 'pCTR_Gender', 'pCTR_Age', 'pCTR_RPosition',
    'num_Depth', 'num_Position', 'num_RPosition',
    'num_Query', 'num_Keyword', 'num_Title', 'num_Description',
]
```

Fig. 5-1 Dense Features

```
dense_need_combine_features = [
    'sum_Query', 'sum_Keyword', 'sum_Title', 'sum_Description',  # sum of idf
    'mean_Query', 'mean_Keyword', 'mean_Title', 'mean_Description',  # mean of idf
]

sparse_features = [
    'sparse_Url', 'sparse_Ad', 'sparse_Advertiser', 'sparse_Depth', 'sparse_Position',
    'sparse_Query', 'sparse_Keyword', 'sparse_Title', 'sparse_Description',
    'sparse_Gender', 'sparse_Age']

multi_val_features_dim = {
    'vec_Query': [int(features_min_max['num_Query'][1]), N_WORDS_QUERY],
    'vec_Keyword': [int(features_min_max['num_Keyword'][1]), N_WORDS_KEYWORD],
    'vec_Title': [int(features_min_max['num_Title'][1]), N_WORDS_TITLE],
    'vec_Description': [int(features_min_max['num_Description'][1]), N_WORDS_DESCRIPTION],
}
```

Fig. 5-2 Sparse Features

The AUC result is OK, it's about 78.7% in the test set. We also have a long way to go in recommender system. Recommender systems has grown as an area of both research and practice. As the field moves forward, we believe user experience concerns will continue to drive advances in algorithms, interfaces, and systems.