Deep Convolutional Neural Network and Multi-View Stacking Ensemble in Ali Mobile Recommendation Algorithm Competition

The solution to the winning of Ali Mobile Recommendation Algorithm

Xiang Li, Suchi Qian, Furong Peng, Jian Yang, Rui Xia
School of Computer Science and Engineering
Nanjing University of Science and Technology
Nanjing 210094, China
{implusdream; qiansuchi2012}@gmail.com
{pengfr; csjyang; rxia}@njust.edu.cn

Xiaolin Hu
Tsinghua National Laboratory for Information Science and Technology (TNList)
Department of Computer Science and Technology
Tsinghua University, Beijing 100084, China
xlhu@mail.tsinghua.edu.cn

***Abstract*—We proposed a deep Convolutional Neural Network (CNN) approach and a Multi-View Stacking Ensemble (MVSE) method in Ali Mobile Recommendation Algorithm competition Season 1 and Season 2, respectively. Specifically, we treat the recommendation task as a classical binary classification problem. We thereby designed a large amount of indicative features based on the logic of mobile business, and grouped them into ten clusters according to their properties. In Season 1, a two-dimensional (2D) feature map which covered both time axis and feature cluster axis was created from the original features. This design made it possible for CNN to do predictions based on the information of both short-time actions and long-time behavior habit of mobile users. Combined with some traditional ensemble methods, the CNN achieved good results which ranked No. 2 in Season 1. In Season 2, we proposed a Multi-View Stacking Ensemble (MVSE) method, by using the stacking technique to efficiently combine different views of features. A classifier was trained on each of the ten feature clusters at first. The predictions of the ten classifiers were then used as additional features. Based on the augmented features, an ensemble classifier was trained to generate the final prediction. We continuously updated our model by padding the new stacking features, and finally achieved the performance of F-1 score 8.78% which ranked No. 1 in Season 2, among over 7,000 teams in total.**

Keywords—CNN; Multi-View Stacking; ensemble; Ali Mobile Recommendation Algorithm; feature cluster

I. Introduction

The rapid development of Alibaba (Ali) Group's M-Commerce business in recent years has put forward high requirements in mobile big-data recommendation algorithms. For example, the Gross Merchandise Volume (GMV) on mobile terminals in the Nov. 11 great sale of 2014 accounts for 42.6% of total GMV. Compared with the PC era, access to the network on mobile terminals can take place anytime anyplace. Therefore, a competition was held by Ali for pursuing algorithm solutions to develop an efficient mobile big-data recommendation system in order to recommend appropriate commodities to mobile users at the right time and the right place. This competition is named Ali Mobile Recommendation Algorithm based on the real users-commodities behavior data on Ali's M-Commerce platforms. Over 7,000 teams have participated on it. Precisely, there are two seasons in rules of the competition. Teams are offered with smaller size of data in purpose of designing an efficient recommendation system in Season 1 from March 20 to April 25. In Season 2, Open Data Processing Service (ODPS) is provided for all the teams to deal with the real big data. Season 2 lasts from April 30 to July 1. During the seasons, Ali's judge will conduct the evaluation once per day.

In the past years, the collaborative filtering based approaches [1][2][3] have been proved to be efficient for recommendation systems. It is a common technique widely used in electronic commerce. These systems aggregate data about customers' purchase habits or preferences and make recommendations to other users based on similarity in overall patterns. Unfortunately, drawbacks [4] with traditional collaborative filtering methods become obvious when it comes to big data. Along with the amount of data becoming larger, collaborative filtering algorithm will show a drop in accuracy and data sparsity [5].

We were facing five million users and tens of millions of items in Ali's ODPS platform in this competition. Inspired by [6] we regard recommendation as a binary classification problem. Based on statistical analysis, we extracted a large amount (over 2,000) of indicative features for classification, from the record of history actions. We grouped them into ten clusters according to their properties. On this basis, the statistical machine learning algorithms and ensemble methods were employed for the classification problem.

During recent years, Convolutional Neural Network [7] has achieved extremely excellent performance in the fields of computer vision [7], speech recognition [24], natural language processing [25] and so on [8]. In Season 1 of the competition, we made the first attempt to introduce Deep CNN into the Mobile Recommendation System. It is known in computer vision, Deep CNN can extract local features through well trained convolutional kernels and transfer the local information into high-level features for classification or detection. A natural idea is then to apply convolutional kernels to recommendation system. For this purpose a 2D feature map was constructed along the time dimension and feature cluster dimension. Such a design of input can make full use of local features and high-level features, because the rectangle window of convolutional kernels can cover both the time information and the relation among feature clusters. Moreover, convolutional kernels can easily extract discriminate features from information in short-time actions to long-time behavior habit of mobile users. The proposed Deep CNN approach achieved a result that ranked No. 2 in Season 1 of the competition.

In Season 2, due to the limitation of both time and resource, we failed to apply the Deep CNN approach for dealing with the large amount of data, although we implemented a prototype (a distributed CNN) in Ali's ODPS platform. Instead, we proposed a Multi-View Stacking Ensemble (MVSE) approach based on the Gradient Boosted Decision Trees (GBDT) [9], and conducted an efficient ensemble for recommendation. The MVSE approach was motivated as follows: (1) we have designed ten feature clusters consisting of totally more than 2,000 features. Each individual feature cluster had its own view on the data distribution. In this case, we might need a more appropriate way to efficiently combine these different views. The multi-view ensemble method is a good choice; (2) The Stacking framework [10] has been proved to be an effective ensemble method in preventing overfitting in many applications [17][18][19]. (3) Each stacking feature in Multi-View Stacking can be generalized and used independently of each other. Therefore the system can be constructed in high parallel. In our practice, a GBDT classifier (basic classifier) was trained on each feature cluster. The prediction of each basic classifier was used as a stacking feature in the second-round learning phase. By adding the ten stacking features into original feature space, we once again trained another GBDT classifier and used it to make the final prediction. With the help of the new stacking framework, we finally improved our best result from F1-score 8.71% to 8.78%, and from rank No. 2 to No. 1.

The rest of this paper is organized as follows. In Section II, some related work of both Deep CNN and Stacking Ensemble is introduced. The description of the problem and basic solutions are presented in Section III. In Section IV, we introduce the feature clusters designed in our task in detail. We present our Deep CNN model and MVSE model, as well as the experimental results, in Sections V and VI, respectively. Section VII finally concludes the paper.

II. Related Work

A. Deep Convolutional Neural Network

Convolutional Neural Network was invented by LeCun Yann in nearly 1989 [26]. He designed a standard convolutional neural network called LeNet [11], which made big progress in digital recognition. In year 2012, Hinton's group beat the second best with considerable advantage in ImageNet competition while using Deep Convolutional Neural Network structure [7]. Since then, Deep CNN has become very popular. As shown by many works, CNN has great potential in extracting both local features and high-level features. More and more research institutions and scholars have come to this area. In 2014, google designed an extremely Deep CNN called GoogLeNet [12]. The huge deep structure achieved very high performance in the ImageNet competition. It is interesting and not strange to find that the top 3 winners all used the Deep CNN. The basic algorithm to train Deep CNN is called back propagation (BP) which is a method of gradient descent.

B. Stacking Ensemble Method

The Stacking Method [10] was first proposed by David Wolpert in 1992. He gave an interesting idea in an article appearing in the Neural Network literature: if we have a set of predictors (linear or nonlinear) $v_1(x), \dots, v_K(x)$ then instead of selecting a single one from this set, a more accurate predictor can be gotten by combining the $v_1, \dots, v_K$. The method for combination is based on level one data defined as follows: Operate the procedures for constructing the predictors, getting $v_k(x), k = 1, \dots, K$. Define the K-variable vector z_n by

$$z_{kn} = v_k(x_n). \tag{1}$$

Then the level one data consists of $\{(y_n, \mathrm{z}_n), n = 1, \dots, N\}$, where y_n is the label of nth sample.

Ordinarily the level one data would be used to select one of the v_k, i.e. use v_k where k minimize $\sum_k (y_n - z_{kn})^2$. Wolpert's idea is that the level one data has more information in it, and can be used to construct "good" combinations of the v_{kn}. He illustrated this idea by using level one data to form nonlinear combinations of nearest neighbor predictors.

Following Wolpert's stacked scheme, Li et al. constructed many simplified neural network modules (SNNM) further stacked to build a Deep Stacking Network (DSN) [13], which was previously named the Deep Convex Network. In the past few years, DSN has received increasing attentions due to its successful application in speech classification and information retrieval. In 2015, according to the base DSN structure, Li et al. designed a Sparse Deep Stacking Network (S-DSN) [14] for image classification, finally made breakthrough results on many datasets. Xia et al. also obtained the best results in sentiment classification [15] using stacking ensemble in comparison with other ensemble methods.

III. Problem Formulation

A. Description of the Task

The target of Ali's Mobile Recommendation Algorithm competition is to develop an individualized recommendation system for a subset of all items by utilizing the user behavior data in such subset of items and more comprehensive user behavior data. We use these notations for convenience: U (the set of users), I (the whole set of items), P (the subset of items, P $\subseteq$ I), D (the user behavior dataset in the set of all items).

Our objective is to develop a recommendation model for users in U on the business domain P using the data D. The data contains two parts. The first part is the dataset D, the mobile behavior data of users in the set of all items, which is described in Table I. The second part is Table II about the details of dataset P.

The behavior dataset D contains user behavior records from 2014-11-18 to 2014-12-18, dataset P contains the subset of items we need to concern about and their corresponding information. We should give our predictions about which user would buy which item on 2014-12-19 according to our model. An example of possible answers is shown in Table III.

TABLE I. TIANCHI_MOBILE_RECOMMEND_TRAIN_USER

Column	***Description***	***Comment***
user_id	Identity of users	Sampled & desensitized
item_id	Identity of items	Desensitized
behavior_type	The user behavior type	Including click, collect, add-to-cart and payment, the corresponding values are 1, 2, 3 and 4, respectively
user_geohash	Latitude (user location when the behavior occurs, which may be null)	Subject to fuzzing
item_category	The category id of the item	Desensitized
time	The time of the behavior	To the nearest hours

TABLE II. TIANCHI_MOBILE_RECOMMEND_TRAIN_ITEM

Column	***Description***	***Comment***
item_id	Identity of items	Sampled & desensitized
item_geohash	User location where the behavior occurs (may be null)	Generated by longitude and altitude through a certain privacy-preserving algorithm
item_category	The category id of the item	Desensitized

TABLE III. EXAMPLE OF POSSIBLE ANSWERS

user_id	***item_id***
100000	2345
100000	2478
100001	127900
100002	207245

Season 1 offered 5,000 users' behavior records and millions of items' information. Season 2 provided complete behavior data of five million users and tens of millions of items' information.

The evaluation metric is the F1-score which is defined based on precision and recall as described as follows.

$$precision = \frac{|\cup(predictionSet, referenceSet)|}{|predictionSet|}$$

$$recall = \frac{|\cup(predictionSet, referenceSet)|}{|referenceSet|}$$

$$F_1 = \frac{2 \times precision \times recall}{precision + recall} \quad (2)$$

PredictionSet contains the predicted purchase data and referenceSet contains the real purchase data.

B. *Classification problem formulation*

We transfer this prediction task into a binary classification problem [6]. We construct samples in the form of tuples (user_id, item_id, act_day). For a specific act_day, we consider all the interactive pairs (user_id, item_id) during the last seven days just before the act_day (see Figure 1). For example, if we take act_day equals to 2014-12-10, then we will go through all the interactive pairs (user_id, item_id) during 2014-12-03 to 2014-12-09, and constructing (user_id, item_id, act_day = 2014-12-10) tuples. If user A really bought item a in 2014-12-10, we set the label of the tuple (A, a, 2014-12-10) to be 1, which means a positive sample. If user A didn't buy item a in 2014-12-10, we set the label of the tuple (A, a, 2014-12-10) to be 0, which means a negative sample.

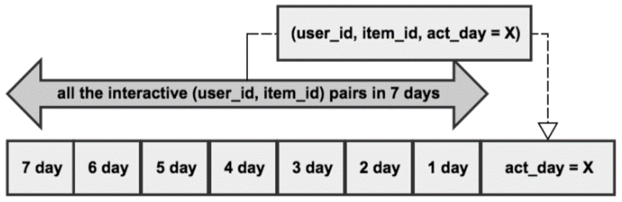
(user_id, item_id, act_day = X)
all the interactive (user_id, item_id) pairs in 7 days
7 day
6 day
5 day
4 day
3 day
2 day
1 day
act_day = X

Fig. 1. The strategy to construct samples for both training and testing. We design this 7 days' sliding window due to human's weekly periodicity and some experimental checks.

Next, we will mention the offline/online training phase, offline/online testing phase. In the offline training period, we take all tuples (user_id, item_id, act_day) whose act_day is between 2014-11-18 and 2014-12-17 to train our model, then use tuples whose act_day is exactly 2014-12-18 as candidate to make predictions. The answers are selected as the ones with higher prediction scores. Because the real purchase data has been given by the records whose time is 2014-12-18 and behavior_type is 4 (see in Table I), we can test our predictions through the ground truth by F1-score. This is called offline testing phase. There is only a little difference between offline and online, while in online phase, training set will include the tuples whose act_day is 2014-12-18, and the final candidate set will be switched into the ones whose act_day is exactly 2014-12-19. Online testing result will be given by Ali's judge one time per day, on account of we do not know the real data in 2014-12-19. What we can do is to improve the model or strategy in order to get higher F1-score in offline testing, which means we can increase our F1-score in online testing simultaneously.

IV. FEATURE CLUSTERS

Here comes the very important part of the tuples: feature clusters. In typical machine learning problems, the features of an objective are the keys to identify themselves. We designed lots of features as statistical values calculated from the history actions, which can be grouped into ten feature clusters according to their properties. The following describes the feature clusters.

- User (U) feature cluster. In this cluster, the statistical values are only related to a specific user_id. For a target tuple (user_id, item_id, act_day), we will consider all the data records corresponding to this user which happened before act_day. From those data, we can extract information like what is the average number of goods had this user bought in last week, in last three days or in last day and so on. The cluster of these features will give us a better understanding towards every user, especially in their different purchase habit.

- Item (I) feature cluster. The same situation with User feature cluster. We only look at all the records related to this specific item_id before the act_day to calculate some values such as how many times had the item been sold in last week or in last three days. These features reflect the popularity of the item, which can affect the probability of purchase from users.
- Category (C) feature cluster. It is similar to Item feature cluster. In this case, everything for item can be replaced by everything for the category of item. If some user has the demand for specific category, he/she will definitely choose the goods in this category.
- User Item (UI) feature cluster. This is the most basic and fundamental feature cluster. It covers exactly the first two elements in the tuple (user_id, item_id, act_day), so it plays the most important role in feature vector. Meanwhile, more information can be extracted from the records referring to both user and item. Something like how many times do this user click this item or add this item to favor yesterday, how long had this user paid his/her attention on this item will be listed in this cluster. These information can definitely show discriminative features to determine the possibility of purchase.
- User Category (UC) feature cluster. Like UI feature cluster, UC also played an essential part in feature vector. The cluster contains the relationship between this user and the category of this item. It aims to find the possible category that a specific user may want in act_day.
- Item Category (IC) feature cluster. It is proved to be very useful in purpose of designing a serial of competitive features. Competitive usually means that when we want to pick out the most important item or several ones in the same category, all the items in the same category are considered competitive with each other. It seems unreasonable that one user bought lots of different items which belongs to one category in one day. These features will help us to prevent this absurd situation. The higher rank or ratio values of an item accounting for its category will let the classifier pick it up firstly.
- Geometry (Geo) feature cluster. Due to a lot of missing values in the original data, we only designed two dimension of this feature cluster: (1) the minimal distance between user and item if it exists, (2) whether or not the item contains a geometry position. The closer distance between user and item is, the more possible the purchase will happen. An item with a specific position will give us sense of security and assurance of credits. These two features are necessary to judge the prediction.
- User & User Item (U&UI) feature cluster. In this cluster, features are mainly ratios generated from combining the values in U cluster and UI cluster. For example, what is the ratio that the statistical value of clicks in UI cluster accounts for the total clicks in U cluster? This will somehow display the importance of an item or the relationship between specific user and item from the ratio aspect of view.
- User & User Category (U&UC) feature cluster. This is totally the same with U&UI feature cluster except for replacing item by its category.
- User Item & User Category (UI&UC) feature cluster. In this cluster, there are extraordinary features which are also competitive features. Different from IC cluster, we take user into account at this time. For a specific user, the items with same category that he/she paid attention to are considered. These features will give more precise competitive information between similar items to prevent the unreasonable prediction that one user may purchase huge amount of similar items in one day.

A brief description of feature clusters is given in Table IV, as well as their dimensions.

TABLE IV. FEATURE CLUSTERS

Feature Cluster Name	*Brief Description*	*Dimension*
U	This user behavior habit	437
I	This item's popularity	388
C	The popularity of the category of this item	191
UI	This user's behavior to this item	404
UC	This user's behavior to this item's category	233
IC	How the item's popularity in its category	65
Geo	Information about geometry position	2
U&UI	Combine U and UI features	122
U&UC	Combine U and UC features	157
UI&UC	Combine UI and UC, particularly competitive features	65

For comparison between these feature clusters, we separately trained and tested each of them to get the summarized performance results in Figure 2. As we can see, the clusters related to UI performed extremely well.

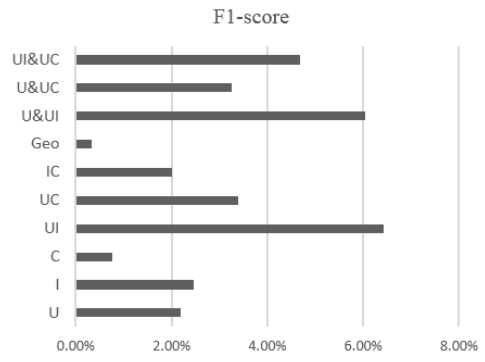

Fig. 2. F1-score for each feature cluster

V. Deep Convolutional Neural Network

We will describe our Convolutional Neural Network used in Season 1 in this section. In order to give convolutional kernels the ability to extract discriminate features from the information in short-time actions to long-time behavior habit of mobile users, a two-dimension (2D) feature map including time and features was designed by us. Meanwhile, we also introduced drop out [16] into our model and found an interesting way to generate a different ensemble method in CNN's prediction results. The basic architecture of Deep CNN is referred to LeNet [11], using two convolutional layers and three full-connected layers with rectified linear units (ReLUs) [7] and drop out regularization.

A. 2D feature map design

For Convolutional Neural Network, it is easy to come up with the following idea in feature design: Making convolutional kernels one-dimensional, fitting the original one-dimension features from all feature clusters. In this case, CNN can be applied like other classical classifiers such as Logistic Regression (LR), Random Forest (RF) or Gradient Boosted Decision Trees (GBDT) which all accepts one-dimension features to train and predict. But this approach ignores the temporal information which may be useful for prediction. Therefore we proposed a two-dimension (2D) feature design according to existing system and our feature clusters:

a) First dimension: The time axis. It illustrates the time before the act_day. In our practice, we expanded time in this way: the last 7 days before, the last 6 days before, …, the last 1 day before. Due to basic experience and several experimental checks, we also separated the period of one day into two parts. One part is before 19:00, another part is after 19:00. By padding two more values about the day for the last operation and the time span between last operation and the act_day, the first dimension was set to be 16 finally.

b) Second dimension: The feature clusters. In order to let CNN extract high-level features from the original ones, we omitted lots of complicated features in feature clusters, only remained some basic statistical values. In practice, only 4 basic statistical values from each UI, UC, U, I feature clusters, also 16 dimension (4 basic dimension chosen from each four cluster which displayed the better performance in Figure 2), same with the first one, were reserved.

Figure 3 shows two examples of 16 ×16 feature maps in our practice. It is clear for humans to see the notable differences between samples from different labels.

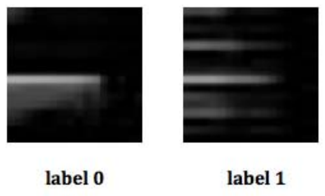

Fig. 3. Examples of feature map with label 0 and lable 1. The differences between positive ones and negative ones are obvious to see.

B. The architecture of Deep CNN

Now we are ready to describe the overall architecture of our CNN. The net contains five layers with weights; the first two layers are convolutional and the remaining three are fully connected. The output of the last fully connected layer is fed to a 2-way softmax which produces a distribution over the 2 class labels. Our network maximizes average across training cases of the log-probability of the correct label under the prediction distribution.

In order to maintain all the information of the original features, we do not use any pooling method. The first convolutional layer filters the 16 ×16 input feature map with 64 kernels of size 3 × 3 with a stride of 2 pixels (this is the distance between the receptive field centers of neighboring neurons in a kernel map). The second convolutional layer takes as input the output of the first convolutional layer and filters it with 64 kernels of size 3 × 3. We flatten the kernels' outputs into one-dimension features with ReLUs and drop out. Two fully connected layers have 1000 neurons each. Figure 4 gives the overall architecture of our Deep CNN model.

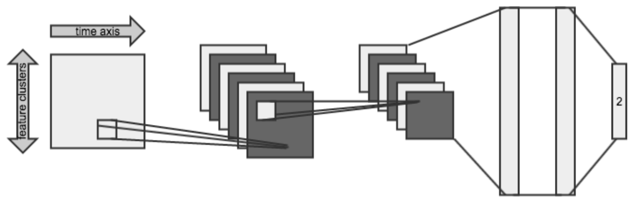

Fig. 4. Overall architecture of Deep CNN model, first two layers are convolutional layers with 64 kernels each, second three layers are full-connected layers with ReLUs and drop out regularization. The input is 2D feature map that designed by us: one dim is time axis, another is feature cluster axis.

C. Drop out regularization and self-ensemble

Combining the predictions of many different models is a very successful way to reduce testing errors, but it appears to be too expensive for big neural networks to train. To deal with this situation, Hinton proposed the drop out technique [16], consists of setting to zero the output of each hidden neuron with probability of 0.5. The neurons which are "dropped out" in this way do not contribute to the forward pass and do not participate in back propagation. So the architecture of network keeps changing all the time. In some extent, after every epoch of training, CNN will become a series of different models, which uses random subset of the total neurons. It helps a lot in learning more robust features.

For our recommendation problem, we can make better use of this property to get ensemble predictions from many epochs' results of a single Deep CNN model. In our practice, we applied drop out before the first two fully connected layers with ratio of 0.5. After a period of training, every epoch of Deep CNN can be regarded as an individual classifier. The results of their corresponding predicted answers will be considerably distinct because of the drop out technique. All the results are combined from these different epochs to get the final prediction. It is interesting to find out that we only trained one model, but we can get ensemble result from different epochs of this single model. We call it self-ensemble for convenience.

D. Results with Deep CNN in Season 1

In Season 1 competition, the amount of users and items is not quite large. The data can be downloaded into our computers, and we can use our resources for training. We use cxxnet, an open source Deep Learning library whose project is on github (https://github.com/dmlc/cxxnet) as a main frame for Deep CNN part, and some basic settings are listed in Table V.

We used one Tesla GPU K20 for CNN to train and test. For comparison, some classical classifiers were implemented for tests including their best ensemble results. Self-ensemble for CNN was also applied to get more stable and robust answers. Our results are shown in Table VI.

TABLE V. DEEP CNN PARAMETERS IN CXXNET

Parameter Name	***Value***	***Meaning***
momentum	0.9	stochastic gradient descent with momentum
wmat:lr	0.01	learning rate of weight matrix
wmat:wd	0.0005	weight decay of weight matrix
bias:lr	0.02	learning rate of bias
bias:wd	0.0005	weight decay of bias

These settings can be found easily in cxxnet configure files in github

TABLE VI. F1-SCORE RESULTS IN SEASON 1

Method	***F1-Score Offline***	***F1-Score Online***
Gradient Boosted Decision Trees (GBDT)	6.85%	-
Random Forest (RF)	6.98%	-
Logistic Regression (LR)	7.36%	10.56%
Deep CNN (self-ensemble)	7.40%	10.84%
LR + RF	7.68%	11.67%
Deep CNN (self-ensemble) + LR	8.02%	12.20%
Deep CNN (self-ensemble)+ LR + RF	**8.12%**	**12.69%**

+ means average ensemble

In Season 1, both LR and CNN can give the best F1-score as a single model. Deep CNN is a new classifier with notable differences from the classical ones, so when we apply average ensemble method in Deep CNN with other classifiers, we get amazing results. There is one important point to illustrate that the gap between offline scores and online scores is not strange, because offline testing happened on 2014-12-18, which is Thursday, meanwhile, online testing happened on 2014-12-19, which is Friday, the purchase situation and data distribution must be different between these two days. For Friday, more purchase predictions will be realized in real life. The higher score in online testing is not odd at all.

The final F1-score result 12.69% of Season 1 gave us rank No. 2 in the end. Therefore, the results tend to prove the effectiveness of Deep CNN in recommendation system. We believe there is large space for Deep CNN to improve in this task in the future.

VI. MULTI-VIEW STACKING ENSEMBLE

As we mentioned before, we already implemented the distributed Deep CNN codes using graph structure (a structure like mapreduce in Ali's ODPS), but due to the limitation of both time and resource, the CNN model in ODPS cannot be well trained. For example, we used more than three days to train but only got 79% accuracy in training set after hundreds of epochs. Finally we applied stacking method in Gradient Boosted Decision Trees (GBDT) which ODPS offered as a tool in algorithm platform. We proposed a new way for stacking according to the abundance of feature clusters. The approach was called Multi-View Stacking Ensemble method which leads to a big improvement in prediction performance in Season 2.

A. Basic stacking ensemble method

Before talking about Multi-View Stacking Ensemble method, we will take a brief view in basic stacking ensemble method. Stacking (sometimes called stacked generalization) involves training a learning algorithm to combine the predictions of several other learning algorithms. First, all of the other algorithms are trained using the available data, then a combiner algorithm is trained to make a final prediction using all the predictions of the other algorithms as additional inputs. If an arbitrary combiner algorithm is used, then stacking can theoretically represent any of the ensemble techniques, although in practice, a single-layer logistic regression model is often used as the combiner.

Stacking typically yields performance better than any single one of the trained models [10]. It has been successfully used on both supervised learning tasks (regression [17], classification and distance learning [18]) and unsupervised learning (density estimation) [19]. It has also been used to estimate bagging's error rate [20][21]. It has been reported to out-perform Bayesian model-averaging [22]. The two top-performers in the Netflix competition utilized blending, which may be considered to be a form of stacking [23].

B. Multi-View Stacking Ensemble based on feature clusters

In our recommendation system, lots of features grouped in ten clusters were designed by us. Instead of using many different predictors or classifiers to produce stacking features in traditional stacking ensemble method, we proposed a new stacking method that regarding one cluster of all feature clusters as a view applied in the same predictor or classifier (GBDT in practice) to produce one stacking feature. That is to say, when we are ready to train sub-predictor for stacking, we only choose a subset of all features for the sub-predictor. For ten clusters in total, we will get ten stacking features. This is so called Multi-View Stacking Ensemble. This approach has two steps.

First, we regard one of the feature clusters as a view to see data relation or pattern based on its feature property. In this aspect of view, only these features were used to train one basic classifier like GBDT. We denote the original feature vector as x, the ten clusters of features are written as $x_{cluster}$. More precisely, the form of the ten feature clusters is shown in (3).

$$x = (x_U, x_I, x_C, x_{UI}, x_{UC}, x_{IC}, x_{Geo}, x_{U\&UI}, x_{U\&UC}, x_{UI\&UC}) \quad (3)$$

The basic classifier is shown as $v(x)$. Then the prediction score of label 1 can be written as (4).

$$z_{cluster} = v(x_{cluster}) \quad (4)$$

If we apply these into all feature clusters, we will get exactly ten prediction scores between 0 and 1 for each sample both in training set and candidate set. The more closely the prediction score comes up to 1, the more positive the sample is. In this way, we obtain ten scores as new features.

Second, the original features x and new features z are concatenated to form a new feature vector x^* as in (5).

$$x^* = (x_U, x_I, x_C, \dots, x_{UI\&UC}, z_U, z_I, z_C, \dots, z_{UI\&UC}) \quad (5)$$

The newest multi-view features generated from different feature clusters can be seen as a serial of self-adaptive weighted discriminative feature values appended into all samples, leading to produce a more general model which reduces overfitting problem to a large extent. The overall stacking structure is displayed in Figure 5.

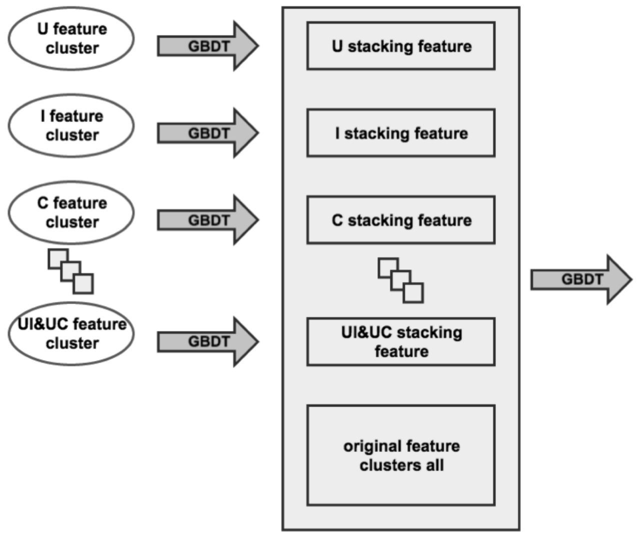

Fig. 5. The details show how to stack ten GBDTs of different feature clusters into the new feature vector.

C. *Results with Multi-View Stacking Ensemble in Season 2*

Season 2 was held on Ali's ODPS called yushanfang (http://yushanfang.com/). The number of original training samples reached 300 million, while the negative ones accounted for the majority. We randomly down sampled the amount of negative ones into nearly ten million, up sampled the amount of positive ones into one million, with ratio of positive: negative being 1:10. Table VII gives some basic settings for GBDT.

DfeatureSplitValueMaxSize means the max size of a feature that can be split, DmaxLeafCount means the max amount of leafs, DminLeafSampleCount shows the least required number of samples in one leaf, the number of trees stands for DtreeCount, The max height of one tree is DmaxDepth. In every epoch of training, we will choose 80% samples and only consider 60% of the feature set to train.

TABLE VII. BASIC GBDT PARAMETERS FOR STACKING

Parameter Name	*Value*	*Meaning*
DfeatureSplitValueMaxSize	100	maximal splits of a feature
DmaxLeafCount	64	maximal number of leaves
DminLeafSampleCount	100	minimal leaf size
DmaxDepth	6	maximal depth of tree
DsampleRatio	0.8	ratio of instances for training
DfeatureRatio	0.6	ratio of features for training
DtreeCount	1200	number of trees

Table VIII shows the experimental results of our team in both offline and online. Our team rank is based on F1-score online. By the way, in big data of Season 2, both Logistic Regression (LR) and Random Forest (RF) showed extremely bad performance, so nearly no team used these two algorithms in practice.

TABLE VIII. F1-SCORE RESULTS IN SEASON 2

Method	*F1-Score Offline*	*F1-Score Online*	*Our Team Rank*
GBDT (single model)	7.90%	8.65%	2
GBDTs (average ensemble models)	7.94%	8.71%	2
MVSE GBDT (single model)	7.95%	8.71%	2
MVSE GBDT + GBDTs (average ensemble models)	**8.01%**	**8.78%**	**1**

The real amount of purchase in 2014-12-19 is 169320, and we submit 170000 (user_id, item_id) pairs which have the highest possibilities to happen for purchase as the answers in both offline testing and online testing. The same with Season 1, there is also a gap between offline score and online score due to the data distribution. This gap becomes smaller when dealing with the big data.

TABLE IX. F1-SCORE LEADERBOARD IN SEASON 2

rank	*F1-Score Online*	*Basic Method*
1	**8.78%**	**MVSE GBDT + GBDTs ensemble**
2	8.75%	GBDTs ensemble
3	8.66%	GBDTs ensemble
4	8.64%	GBDTs ensemble

+ means average ensemble

One trick for ensemble method is to combine the same level classifiers' prediction scores with average strategy. This approach of average ensemble will improve the performance of models quite a lot. In our practice, some basic GBDTs had the same ability, probably achieved the same F1-score. We combined them together by average ensemble, and finally got 0.06% improvement. The Multi-View Stacking Ensemble

(MVSE) already had the same ability with the basic ensemble one, so we put them together again using average strategy, then got another 0.07% improvement in the end, leading us to the top 1 rank of Season 2. For better comparison, the leaderboard for Season 2 is given in Table IX, including the results from other teams.

VII. Conclusion

This paper describes our systems participated in Ali Mobile Recommendation Algorithm Competition in 2015. Two approaches, namely Deep Convolutional Neural Network (CNN) and Multi-View Stacking Ensemble (MVSE), were used in Season 1 and Season 2 of the competition, respectively. In Season 1, we made an attempt to employ Deep CNN for recommendation, and designed a 2D feature map as input for CNN. In Season 2, Multi-View Stacking Ensemble method was applied to efficiently combine different views of features clusters for recommendation. The excellent performance of these approaches demonstrated their great potential for accurate prediction in recommendation system based on big data.

Acknowledgment

This work was also partially supported by the National Science Fund for Distinguished Young Scholars under Grant Nos. 61125305, 61472187, 61233011 and 61373063, the Key Project of Chinese Ministry of Education under Grant No. 313030, the 973 Program No. 2014CB349303, Fundamental Research Funds for the Central Universities No. 30920140121005, Program for Changjiang Scholars and Innovative Research Team in University No. IRT13072, and Natural Science Foundation of China No. 61305090.

References

[1] Breese, J. S., Heckerman, D., & Kadie, C. (1998, July). Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence (pp. 43-52). Morgan Kaufmann Publishers Inc..

[2] Linden, G., Smith, B., & York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. Internet Computing, IEEE, 7(1), 76-80.

[3] Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001, April). Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th international conference on World Wide Web (pp. 285-295). ACM.

[4] Burke, R. (1999, July). Integrating knowledge-based and collaborative-filtering recommender systems. In Proceedings of the Workshop on AI and Electronic Commerce (pp. 69-72).

[5] Ma, H., King, I., & Lyu, M. R. (2007, July). Effective missing data prediction for collaborative filtering. In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval (pp. 39-46). ACM.

[6] Basu, C., Hirsh, H., & Cohen, W. (1998, July). Recommendation as classification: Using social and content-based information in recommendation. In AAAI/IAAI (pp. 714-720).

[7] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[8] LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks, 3361(10).

[9] Ye, J., Chow, J. H., Chen, J., & Zheng, Z. (2009, November). Stochastic gradient boosted distributed decision trees. In Proceedings of the 18th ACM conference on Information and knowledge management (pp. 2061-2064). ACM.

[10] Wolpert, D. H. (1992). Stacked generalization. Neural networks, 5(2), 241-259.

[11] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[12] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2014). Going deeper with convolutions. arXiv preprint arXiv:1409.4842.

[13] Deng, L., Yu, D., & Platt, J. (2012, March). Scalable stacking and learning for building deep architectures. In Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on (pp. 2133-2136). IEEE.

[14] Li, J., Chang, H., & Yang, J. (2015). Sparse Deep Stacking Network for Image Classification. arXiv preprint arXiv:1501.00777.

[15] Xia, R., Zong, C., & Li, S. (2011). Ensemble of feature sets and classification algorithms for sentiment classification. Information Sciences, 181(6), 1138-1152.

[16] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

[17] Breiman, L. (1996). Stacked regressions. Machine learning, 24(1), 49-64.

[18] Ozay, M., & Vural, F. T. Y. (2012). A new fuzzy stacked generalization technique and analysis of its performance. arXiv preprint arXiv:1204.0171.

[19] Smyth, P., & Wolpert, D. (1999). Linearly combining density estimators via stacking. Machine Learning, 36(1-2), 59-83.

[20] Rokach, L. (2010). Ensemble-based classifiers. Artificial Intelligence Review,33(1-2), 1-39.

[21] Wolpert, D. H., & Macready, W. G. (1999). An efficient method to estimate bagging's generalization error. Machine Learning, 35(1), 41-55.

[22] Clarke, B. (2003). Comparing Bayes model averaging and stacking when model approximation error cannot be ignored. The Journal of Machine Learning Research, 4, 683-712.

[23] Sill, J., Takács, G., Mackey, L., & Lin, D. (2009). Feature-weighted linear stacking. arXiv preprint arXiv:0911.0460.

[24] Abdel-Hamid, O., Deng, L., & Yu, D. (2013). Exploring convolutional neural network structures and optimization techniques for speech recognition. InINTERSPEECH (pp. 3366-3370).

[25] Kim, Y. (2014). Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882.

[26] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4), 541-551.