

PREDICTING IF AN ADVERTISEMENT WILL RECEIVE USERS' CLICK

A PROJECT REPORT

Submitted by

**CHINTAN PARIKH
SHWETA SINHA
JOY SEQUERIA
DEEPSHI MAHAJAN
RAGAVI ELAVAZHAGAN**

Under the guidance of
Dr. Ming Hwa Wang

PREFACE

This report has been made in fulfillment of the requirement for the subject: Pattern Recognition and Data Mining under the supervision of Dr. Ming Hwa-Wang. For this project we have studied various concepts related to CTR and ad prediction. We have also studied in-depth about Machine Learning Algorithms which can be applied to solve this problem. In this project, we proposed DeepFM, a Factorization-Machine based Neural Network for CTR prediction, to overcome the shortcomings of the state-of-the-art models and to achieve better performance. The DeepFM model simultaneously models low-order feature combinations and high-order feature combinations, so that a combination of different order features can be learned. The model is an end to end model that does not require any feature engineering.

ACKNOWLEDGEMENT

Apart from our efforts, the success of any project depends largely on the encouragement and guidelines of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project. We would like to show our greatest appreciation to Dr. Ming-Hwa Wang. We thank him for his tremendous support and help. The guidance and support received from all the members who contributed and who are contributing to this project, was vital for the success of the project.

TABLE OF CONTENTS

- 2. INTRODUCTION 1
 - 2.1 OBJECTIVE 1
 - 2.2 WHAT IS THE PROBLEM..... 1
 - 2.3 WHY THIS APPROACH IS RELATED TO THIS CLASS..... 2
 - 2.4 WHY OTHER APPROACH IS NO GOOD 2
 - 2.5 WHY YOU THINK YOUR APPROACH IS BETTER 3
 - 2.6 STATEMENT OF THE PROBLEM 4
 - 2.7 AREA OR SCOPE OF INVESTIGATION 5
- 3. THEORETICAL BASES AND LITERATURE REVIEW 6
 - 3.1 DEFINITION OF THE PROBLEM..... 6
 - 3.2 THEORETICAL BACKGROUND OF THE PROBLEM..... 6
 - 3.3 RELATED RESEARCH TO SOLVE THE PROBLEM 6
 - 3.4 SOLUTION TO SOLVE THIS PROBLEM..... 9
 - 3.5 WHERE YOUR SOLUTION IS DIFFERENT FROM OTHERS 9
 - 3.6 WHY YOUR SOLUTION IS BETTER? 9
- 4. HYPOTHESIS 10
 - 4.1 SINGLE HYPOTHESIS 10
- 5. METHODOLOGY 11
 - 5.1 HOW TO GENERATE/COLLECT INPUT DATA..... 11
 - 5.2 HOW TO SOLVE THE PROBLEM? 12
 - 5.2.1 ALGORITHM DESIGN..... 12
 - 5.2.2 LANGUAGE USED 15
 - 5.1.3 TOOLS USED 15
 - 5.3 HOW TO GENERATE OUTPUT 16
 - 5.4 HOW TO PROVE CORRECTNESS..... 16
- 6. IMPLEMENTATION..... 17
 - 6.1 CODE 17
 - 6.2 DESIGN DOCUMENT AND FLOWCHART 18
- 7. DATA ANALYSIS AND DISCUSSION 19
 - 7.1 OUTPUT GENERATION..... 19
 - 7.2 OUTPUT ANALYSIS 21
 - 7.3 COMPARE OUTPUT AGAINST HYPOTHESIS 21
 - 7.4 ABNORMAL CASE EXPLANATION..... 22
- 8. CONCLUSIONS AND RECOMMENDATIONS 23

8.1.SUMMARY AND CONCLUSIONS	23
8.2 RECOMMENDATION FOR FUTURE STUDIES	23
9. BIBLIOGRPAHY	25
10. APPENDICES	26
10.1 INPUT/OUTPUT LISTING	40

2. INTRODUCTION

2.1 OBJECTIVE

The objective of the proposed system is to predict if an ad will be clicked by the user or not.

2.2 WHAT IS THE PROBLEM

Predicting ad click-through rates (CTR) is a massive-scale learning problem that is central to the multi-billion-dollar online advertising industry. Sponsored search advertising, contextual advertising, display advertising, and real-time bidding auctions have all relied heavily on the ability of learned models to predict ad click-through rates accurately, quickly, and reliably.

CTR is a measure of intent of clicking ads of online internet users who view advertisements on their web pages. It is a measure of ratio of numbers of users clicked the ad to the total number of times the ad is displayed. Always, a higher value of CTR plays a crucial role in increasing the revenue of the business. Showing the user an Ad that is relevant to his/her need greatly improves user's satisfaction. It's important to predict the CTR of ads accurately. Unsuccessful online advertising leads to a variety of problems. First, it has a bad influence on user experience, especially when the user is searching on the website. That's because the users of search engine always have clear searching purpose and needs.

Second, bad recommendation of advertising will reduce the revenue of both the advertisers and the search engine company. That is why advertisers need to know if an ad will be clicked by the users or not.

2.3 WHY THIS APPROACH IS RELATED TO THIS CLASS

The problem at hand requires the use of data mining, machine learning and pattern recognition. The project helps to predict whether an ad will receive a user's click or not whose requirements and functionality are related to the topics covered in this class.

2.4 WHY OTHER APPROACH IS NO GOOD

The commonly used CTR prediction models have these problems.

- In the use of linear models, the general way of extracting higher-order features is based on manual and prior knowledge. On one hand, it is almost infeasible when the feature dimension is relatively high. On the other hand, these models also have difficulty modeling the combined features that rarely occur in training sets.
- Factorization Machines (FM) extract feature combinations through implicit product inners for each dimension feature. The result is also very good. However, in theory, FM can be used to model high-order feature combinations. In fact, because of computational complexity, only second-order feature combinations are generally used

- Deep neural networks have great potential in learning complex feature relationships. There are also many models based on CNN (Convolutional Neural Network) and RNN (Recurrent Neural Networks) for CTR estimation. However, CNN-based models are more biased towards neighboring feature extraction, and RNN-based models are more suitable for sequence-dependent click data.
- The FNN (Factorization-machine supported Neural Network) model proposed by zhang et al. first pre-trains FM and applies the trained FM to the DNN. Then a Product Layer is added between the embedding layer and the fully connected layer of the PNN (Product Neural Network) to complete the feature combination. PNN and FNN are like other deep learning models and it is difficult to effectively extract low-order features.
- Wide & deep models have mixed width and depth models. However, the input of the width model still depends on the feature engineering.

2.5 WHY YOU THINK YOUR APPROACH IS BETTER

Deep learning is based on the neural network structure and nonlinear activation function, and automatically learns the complex combination of features. The most popular depth models in the field of APP recommendation are FNN/PNN/Wide & Deep. The DeepFM model combines the breadth and depth models, a joint training FM model and a DNN model to learn low-order feature combinations and high-order feature combinations.

In addition, the DeepFM and Deep FM components share data input from the Embedding layer.

The advantage of DeepFM are as follows:

- The Embedding layer implicit vector can receive both Deep component and FM component information during training (residual back propagation). So that Embedding layer information more accurate and ultimately enhance the recommendation effect.
- The DeepFM model simultaneously models low-order feature combinations and high-order feature combinations, so that a combination of different order features can be learned.
- The DeepFM model is an end-to-end model that does not require any artificial feature engineering.

2.6 STATEMENT OF THE PROBLEM

The prediction of click-through rate (CTR) is critical in recommender system, where the task is to estimate the probability a user will click on a recommended item. Learning sophisticated feature interactions behind user behaviors is critical in maximizing CTR for recommender systems. It is possible to derive an end-to-end learning model that emphasizes both low- and high order feature interactions. DeepFM, combines the power of factorization machines for recommendation and deep learning for feature learning in a new neural network architecture. It models low-order feature interactions like FM and models high-order feature interactions like DNN.

2.7 AREA OR SCOPE OF INVESTIGATION

There are two interesting directions for future study. One is exploring some strategies (such as introducing pooling layers) to strengthen the ability of learning most useful high order feature interactions. The other is to train DeepFM on a GPU cluster for large-scale problems.

3. THEORETICAL BASES AND LITERATURE REVIEW

3.1 DEFINITION OF THE PROBLEM

Predict the Click Through Rates of advertisements using DeepFM to improve the accuracy and prediction of an advertisement receiving users' click.

3.2 THEORETICAL BACKGROUND OF THE PROBLEM

Many ads are sold on a "pay-per-click" (PPC) basis, meaning the company only pays for ad clicks, not ad views. Thus, your optimal approach (as a search engine) is to choose an ad based on "expected value", meaning the price of a click times the likelihood that the ad will be clicked. In other words, a \$1.00 ad with a 5% probability of being clicked has an expected value of \$0.05, whereas a \$2.00 ad with a 1% probability of being clicked has an expected value of only \$0.02. In this case, you would choose to display the first ad.

For you to maximize expected value, you therefore need to accurately predict the likelihood that a given ad will be clicked, also known as "click-through rate" (CTR)

3.3 RELATED RESEARCH TO SOLVE THE PROBLEM

Since these past few years, there has been a lot of interesting work being carried out to achieve accurate and efficient ad click prediction.

The most widely used model for CTR prediction was the Logistic Regression model, in which a model \mathbf{w} is solved by the following optimization problem:

$$\min_{\mathbf{w}} \sum_{(y_i, \mathbf{x}_i) \in D} \log(1 + \exp(-y_i \phi(\mathbf{w}, \mathbf{x}_i))),$$

Where x_i impression, $y_i \in \{1, -1\}$ is the label, and D is the training set. Here $\Phi(\mathbf{w}, \mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$ is called a linear model.

Before that was Degree-2 Polynomial Mappings, which in fact was the most naive way to learn a dedicated weight for it. For example, let us consider the dataset with two features publisher and advertiser and some ad impressions from Nike are displayed on ESPN:

Publisher	Advertiser
ESPN	NIKE

For Poly-2, $\Phi(\mathbf{w}, \mathbf{x}) = \mathbf{w}_{\text{ESPN, NIKE}}$

The problem with Poly-2 was if the data was sparse, there would be unseen pairs in the test set.

To address this problem, Factorization Machines (FM) was proposed, a method that learns the feature conjunction in latent space. In FMs, each

feature has an associated latent vector and the coexistence of the two is modelled by the inner-product of two latent vectors.

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

Based on our example, we would now have $\Phi(\mathbf{w}, \mathbf{x}) = \mathbf{w}_{\text{ESPN}} \cdot \mathbf{w}_{\text{NIKE}}$. The benefit of FMs is that in the case of predicting on unseen data from the test set, we may still be able to do a reasonable prediction.

FMs have only one latent space, which means each feature has only one latent vector and this latent vector is used to interact with any other latent vector from other features.

From this, the idea of Field-aware Factorization Machines or FFM was conceived. Here the original latent space is split into smaller latent spaces and depending on the fields of features, one of them is used.

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

Here f_1 and f_2 are respectively the fields of j_1 and j_2 and \mathbf{w}_{j_1, f_2} and \mathbf{w}_{j_2, f_1} are two vectors with length k which is a user specified parameter.

3.4 SOLUTION TO SOLVE THIS PROBLEM

Our solution is to use DeepFM, to combine the power of factorization machines and deep learning for feature learning and overcoming the shortcomings of the current systems and using the proposed neural network architecture to generate lesser log loss and hence a greater prediction accuracy.

3.5 WHERE YOUR SOLUTION IS DIFFERENT FROM OTHERS

Our solution is different as the proposed model, DeepFM, combines the power of factorization machines for recommendation and deep learning for feature learning in a new neural network architecture. Compared to the latest Wide & Deep model from Google, DeepFM has a shared input to its “wide” and “deep” parts, with no need of feature engineering besides raw features.

3.6 WHY YOUR SOLUTION IS BETTER?

The proposed solution is better than the linear models as DeepFm can handle high feature dimensions. Comprehensive experiments are conducted to demonstrate the effectiveness and efficiency of DeepFM over the existing models for CTR prediction, on both benchmark data and commercial data.

The DeepFM model simultaneously models low-order feature combinations and high-order feature combinations, so that a combination of different order features can be learned. The DeepFM model is an end-to-end model that does not require any artificial feature engineering.

4. HYPOTHESIS

4.1 SINGLE HYPOTHESIS

We assume that by using a machine learning model we can achieve click predictions better and more efficiently than other pre-existing models.

5. METHODOLOGY

5.1 HOW TO GENERATE/COLLECT INPUT DATA

In online advertising, click-through rate (CTR) is a very important metric for evaluating ad performance. As a result, click prediction systems are essential and widely used for sponsored search and real-time bidding.

For this project, we are taking eleven days' worth of **Avazu** data to build and test prediction models. This data set has been taken from Kaggle. One of the keys to proper machine learning is model evaluation. The goal of model evaluation is to estimate how well your model will "generalize" to future data. In other words, we want to build a model that accurately predicts the future, not the past! One of the most common evaluation procedures is to split your data into a "training set" and a "testing set". 80% of the data is used for training and 20% is used for testing.

Data Fields:

The data fields present in the data set are id, click, time, C1, banner_pos, site_id, site_domain, site_category, app_id, app_domain, app_category, device_id, device_ip, device_model, device_type, device_conn_type and C14-C21.

5.2 HOW TO SOLVE THE PROBLEM?

5.2.1 ALGORITHM DESIGN

DeepFM consists of two components, FM component and deep component, that 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 ($W(l)$, $b(l)$ below) are trained jointly for the combined prediction model:

$\hat{y} = \text{sigmoid}(y_{\text{FM}} + y_{\text{DNN}})$, 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.

FM Component:

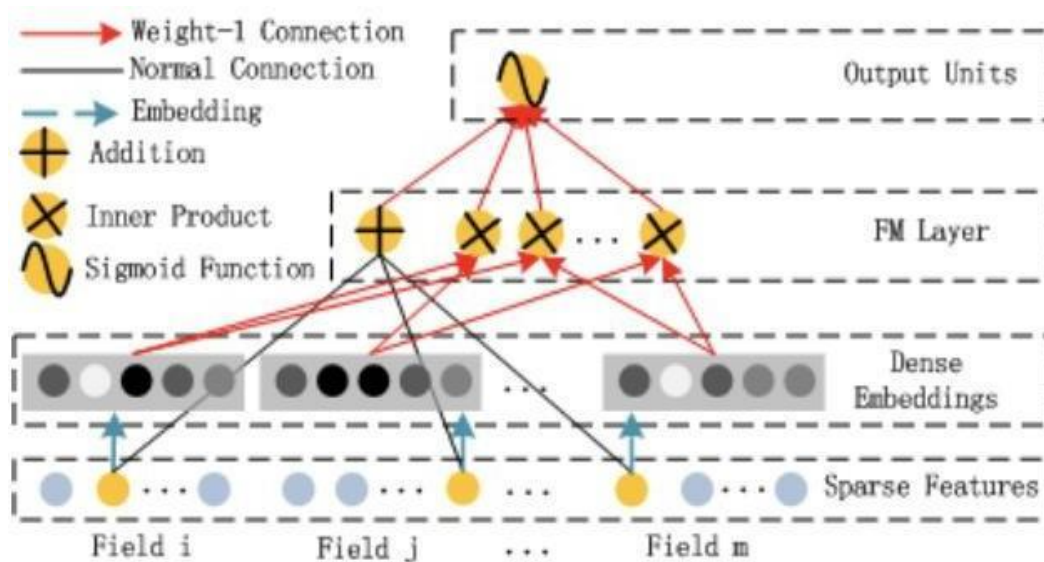


Figure 1: The architecture of FM component

The FM component is a factorization machine, which is proposed in [Rendle, 2010] 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. It can capture order-2 feature interactions much more effectively than previous approaches especially when the dataset is sparse. FM can train latent vector V_i (V_j) whenever 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. The output of FM is the summation of an Addition unit and a number of Inner Product units:

$$y_{FM} = \langle w, x \rangle + \sum_{j_1=j_1+1}^d \sum_{j_2=j_2+1}^d \langle V_{j_1}, V_{j_2} \rangle x_{j_1} \cdot x_{j_2}.$$

where $w \in \mathbb{R}^d$ and $V_i \in \mathbb{R}^k$ (k is given). 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.

Deep Component

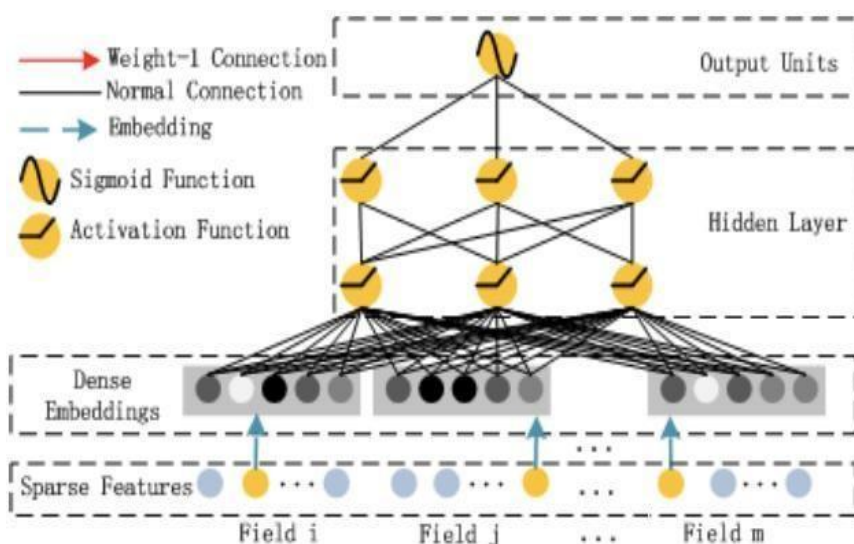


Figure 2: The architecture of DNN layer.

The deep component is a feed-forward neural network, which is used to learn high-order feature interactions. The depth part is a feed-forward neural network that can learn higher-order feature combinations. It should be noted that the original input data is high dimensional sparse data of many fields. Therefore, an embedding layer is introduced to compress the input vector to a low-dimensional dense vector. The structure of embedding layer is shown below:

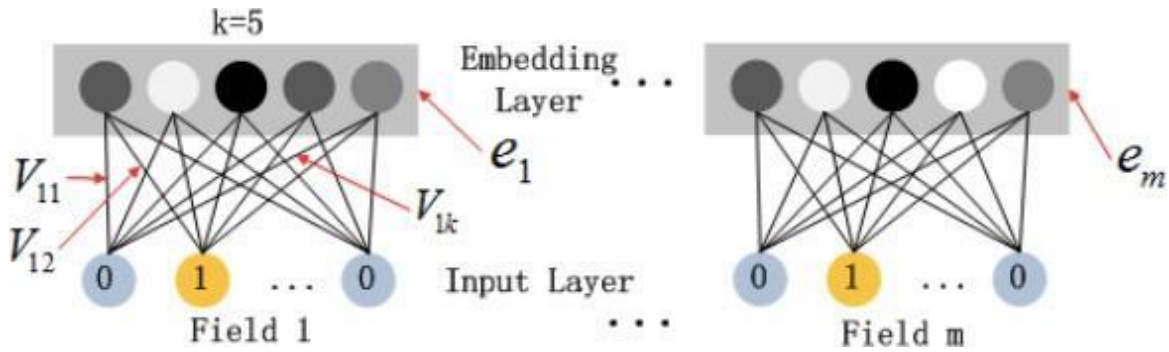


Figure 3: The structure of embedding layer.

The output of the embedding layer as:

$a^{(0)} = [e_1, e_2, \dots, e_m]$, where e_i is the embedding of i -th field and m is the number of fields. Then, $a^{(0)}$ is fed into the deep neural network, and the forward process is: $a^{(l+1)} = \sigma(W^{(l)} a^{(l)} + b^{(l)})$, 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: $y_{DNN} = \sigma(W^{(|H|+1)} \cdot a^{(H)} + b^{(|H|+1)})$, where $|H|$ is the number of hidden layers.

5.2.2 LANGUAGE USED

Python is an interpreted high-level programming language for general-purpose programming.

Packages used are:

- Pandas is a software library written for the Python programming language for data manipulation and analysis.
- SciKit-Learn is a free software machine learning library for the Python programming language.
- Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- Tensorflow is an open-source software library for dataflow programming across a range of tasks.

5.1.3 TOOLS USED

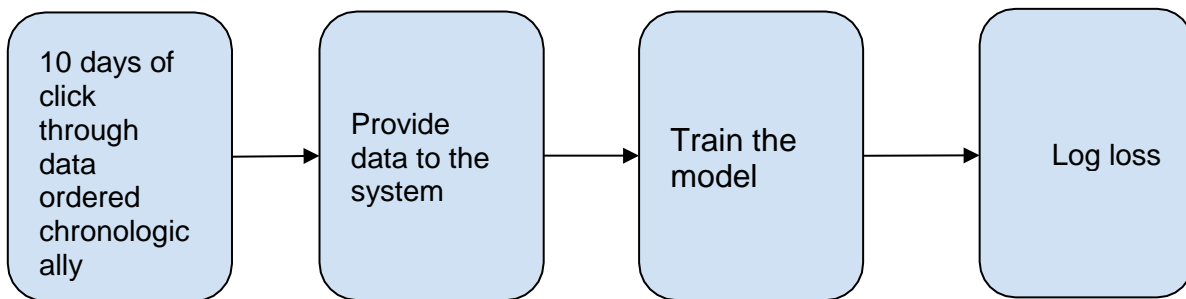
PyCharm 2018.1.3

PyCharm is an Integrated Development Environment used in computer programming, specifically for Python language. It is cross platform with, Windows, macOS and Linux versions.

5.3 HOW TO GENERATE OUTPUT

We will perform the following steps to generate the output :

1. Data is collected from the Avazu dataset.
2. Dataset is then split into two sets: Validation set and Training set
3. Provide the model with the training set first and then validation to prove over-fitting does not exists.
4. Test the model to achieve assumed output.



5.4 HOW TO PROVE CORRECTNESS

We will prove the correctness using Logarithmic Loss. Logarithmic loss measures the performance of a classification model where the prediction input is a probability value between 0 and 1. The goal of the machine learning model is to minimize this value. A perfect model would have a log loss of 0. Lesser the logarithmic loss, better the prediction of the model.

6. IMPLEMENTATION

6.1 CODE

```
# DeepFM
if self.use_fm and self.use_deep:
    concat_input = tf.concat([self.y_first_order, self.y_second_order, self.y_deep], axis=1)
elif self.use_fm:
    concat_input = tf.concat([self.y_first_order, self.y_second_order], axis=1)
elif self.use_deep:
    concat_input = self.y_deep
self.out = tf.add(tf.matmul(concat_input, self.weights["concat_projection"]), self.weights["concat_bias"])

# loss
if self.loss_type == "logloss":
    self.out = tf.nn.sigmoid(self.out)
    self.loss = tf.losses.log_loss(self.label, self.out)
elif self.loss_type == "mse":
    self.loss = tf.nn.l2_loss(tf.subtract(self.label, self.out))
# l2 regularization on weights
if self.l2_reg > 0:
    self.loss += tf.contrib.layers.l2_regularizer(
        self.l2_reg)(self.weights["concat_projection"])
    if self.use_deep:
        for i in range(len(self.deep_layers)):
            self.loss += tf.contrib.layers.l2_regularizer(
                self.l2_reg)(self.weights["layer_%d"%i])

# optimizer
if self.optimizer_type == "adam":
    self.optimizer = tf.train.AdamOptimizer(learning_rate=self.learning_rate, beta1=0.9, beta2=0.999,
                                             epsilon=1e-8).minimize(self.loss)
elif self.optimizer_type == "adagrad":
    self.optimizer = tf.train.AdagradOptimizer(learning_rate=self.learning_rate,
                                               initial_accumulator_value=1e-8).minimize(self.loss)
elif self.optimizer_type == "gd":
    self.optimizer = tf.train.GradientDescentOptimizer(learning_rate=self.learning_rate).minimize(self.loss)
elif self.optimizer_type == "momentum":
    self.optimizer = tf.train.MomentumOptimizer(learning_rate=self.learning_rate, momentum=0.95).minimize(
        self.loss)
elif self.optimizer_type == "yellowfin":
    self.optimizer = YFOptimizer(learning_rate=self.learning_rate, momentum=0.0).minimize(
        self.loss)
```

6.2 DESIGN DOCUMENT AND FLOWCHART

All parameters, including w_i , v_i , and the network parameters ($W(l)$, $b(l)$ below) are trained jointly for the combined prediction model.

$\hat{y} = \text{sigmoid}(y_{FM} + y_{DNN})$, 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.

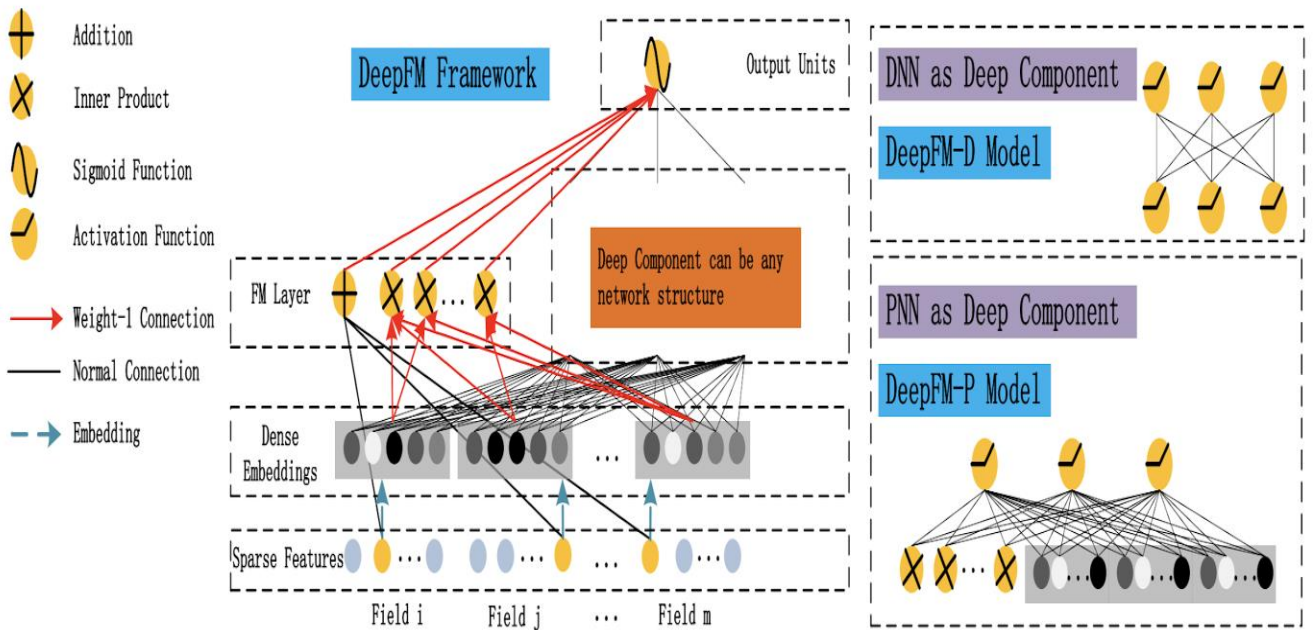


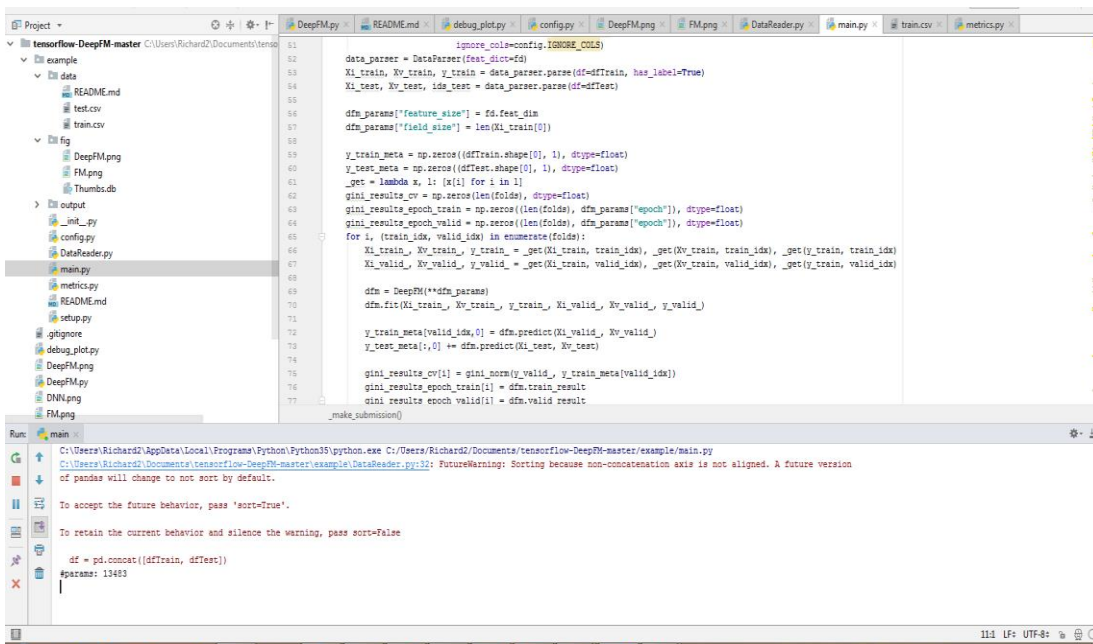
Figure 4: Wide& deep architecture of the DeepFM framework (left).

The wide and deep component share the same input raw feature vector, which enables DeepFM to learn low- and high-order feature interactions simultaneously from the input raw features. The wide component of DeepFM is an FM layer, which we refer to as FM Component. The Deep Component of DeepFM can be any neural network. In this paper, we will study two instances of DeepFM, namely DeepFM-D(top-right) and DeepFM-P (lower-right), the deep components of which are DNN and PNN

respectively.

7. DATA ANALYSIS AND DISCUSSION

7.1 OUTPUT GENERATION



```
51 ignore_cols=config.IGNORE_COLS
52 data_parser = DataParser(feats_dict=fd)
53 Xi_train, Xv_train, y_train = data_parser.parse(df=dfTrain, has_label=True)
54 Xi_test, Xv_test, idv_test = data_parser.parse(df=dfTest)
55
56 dfm_params['feature_size'] = fd.feats_dim
57 dfm_params['field_size'] = len(Xi_train[0])
58
59 y_train_meta = np.zeros((dfTrain.shape[0], 1), dtype=float)
60 y_test_meta = np.zeros((dfTest.shape[0], 1), dtype=float)
61
62 _get = lambda x, i: x[i] for i in 1]
63 gini_results_cv = np.zeros(len(folds), dtype=float)
64 gini_results_epoch_train = np.zeros(len(folds), dfm_params['epoch']), dtype=float)
65 gini_results_epoch_valid = np.zeros(len(folds), dfm_params['epoch']), dtype=float)
66
67 for i, (train_idx, valid_idx) in enumerate(folds):
68     Xi_train_, Xv_train_, y_train_ = _get(Xi_train, train_idx), _get(Xv_train, train_idx), _get(y_train, train_idx)
69     Xi_valid_, Xv_valid_, y_valid_ = _get(Xi_train, valid_idx), _get(Xv_train, valid_idx), _get(y_train, valid_idx)
70
71     dfm = DeepFM(**dfm_params)
72     dfm.fit(Xi_train_, Xv_train_, y_train_, Xi_valid_, Xv_valid_, y_valid_)
73
74     y_train_meta[valid_idx,0] = dfm.predict(Xi_valid_, Xv_valid_)
75     y_test_meta[1,0] += dfm.predict(Xi_test, Xv_test)
76
77     gini_results_cv[i] = gini_norm(y_valid_, y_train_meta[valid_idx])
78     gini_results_epoch_train[i] = dfm.train_result
79     gini_results_epoch_valid[i] = dfm.valid_result
80
81     _make_submission()
```

Run console output:

```
C:\Users\Richard2\AppData\Local\Programs\Python\Python35\python.exe C:/Users/Richard2/Documents/tensorflow-DeepFM-master/example/main.py
C:\Users\Richard2\Documents\tensorflow-DeepFM-master\example\DataReader.py:32: FutureWarning: Sorting because non-concatenation axis is not aligned. A future version
of pandas will change to not sort by default.
To accept the future behavior, pass 'sort=True'.
To retain the current behavior and silence the warning, pass sort=False

df = pd.concat((dfTrain, dfTest))
#params: 13483
```



```

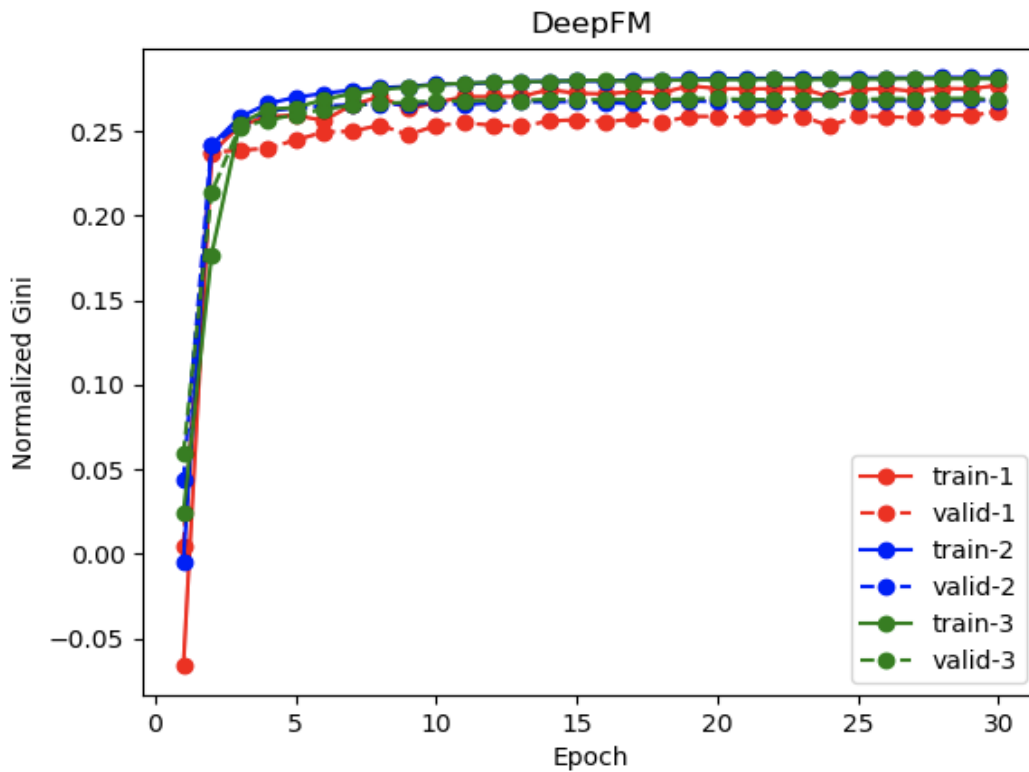
51         ignore_cols=config.IGNORE_COLS
52     data_parser = DataParser(feats_dict=fd)
53     Xi_train, Xv_train, y_train = data_parser.parse(df=dfTrain, has_label=True)
54     Xi_test, Xv_test, y_test = data_parser.parse(df=dfTest)
55
56     dfm_params["feature_size"] = fd.feats_dim
57     dfm_params["field_size"] = len(Xi_train[0])
58
59     y_train_meta = np.zeros((dfTrain.shape[0], 1), dtype=float)
60     y_test_meta = np.zeros((dfTest.shape[0], 1), dtype=float)
61     _get = lambda x, l: [x[i] for i in l]
62     gini_results_cv = np.zeros(len(folds), dtype=float)
63     gini_results_epoch_train = np.zeros((len(folds), dfm_params["epoch"]), dtype=float)
64     gini_results_epoch_valid = np.zeros((len(folds), dfm_params["epoch"]), dtype=float)
65     for i, (train_idx, valid_idx) in enumerate(folds):
66         Xi_train, Xv_train, y_train = _get(Xi_train, train_idx), _get(Xv_train, train_idx), _get(y_train, train_idx)
67         Xi_valid, Xv_valid, y_valid = _get(Xi_train, valid_idx), _get(Xv_train, valid_idx), _get(y_train, valid_idx)
68
69         dfm = DeepFM(**dfm_params)
70         dfm.fit(Xi_train, Xv_train, y_train, Xi_valid, Xv_valid, y_valid)
71
72         y_train_meta[valid_idx,0] = dfm.predict(Xi_valid, Xv_valid)
73         y_test_meta[0,0] = dfm.predict(Xi_test, Xv_test)
74
75         gini_results_cv[i] = gini_norm(y_valid, y_train_meta[valid_idx])
76         gini_results_epoch_train[i] = dfm.train_result
77         gini_results_epoch_valid[i] = dfm.valid_result
78     _make_submission()

```

```

#params: 13483
[1] train-result=0.0908, valid-result=0.1311 [31.9 s]
[2] train-result=0.2131, valid-result=0.2159 [30.5 s]
[3] train-result=0.2444, valid-result=0.2361 [32.5 s]
[4] train-result=0.2638, valid-result=0.2498 [30.7 s]

```



7.2 OUTPUT ANALYSIS

The output is calculated with the help of logloss. At the beginning after initially executing the code, for the first part of the training data, the logloss is calculated. At first the value is high but as it processes through the data the value eventually decreases gradually. At some point there will be minimalistic changes in the value which means the model has achieved optimum performance or in the worst case scenario the model is too attached with the data causing 'over-hitting'.

After 30 epochs, using yellowfin python library, a graph is plotted logloss vs epoch for DeepFM using train and validation sets.

7.3 COMPARE OUTPUT AGAINST HYPOTHESIS

As we have mentioned in the hypothesis that, by using a machine learning model DeepFM we can achieve click predictions better and more efficiently than other pre-existing models. We have taken advantage of DeepFM. The DeepFM model simultaneously models low-order feature combinations and high-order feature combinations, so that a combination of different order features can be learned. Also, it does not require any artificial feature engineering.

Our implementation of DeepFM produces a comparatively lesser log loss than the other existent models while predicting whether an ad will be clicked by the user or not.

7.4 ABNORMAL CASE EXPLANATION

To avoid model being attached to training or in other words to avoid over-fitting, we split the dataset into training set and validation set.

Another abnormality that had occurred was some missing data which discrepancy in the output. This could have happened when the ads clicked couldn't access the device details or some other vital information.

8. CONCLUSIONS AND RECOMMENDATIONS

8.1.SUMMARY AND CONCLUSIONS

DeepFM, a Factorization-Machine based Neural Network for CTR prediction, overcomes the shortcomings of the state-of-the-art models and achieves better performance. DeepFM trains a deep component and an FM component jointly. It gains performance improvement from these advantages:

- It does not need any pre-training.
- It learns both high-and low-order feature interactions.
- It introduces a sharing strategy of feature embedding to avoid feature engineering. We conducted extensive experiments on two real-world datasets to compare the effectiveness and efficiency of DeepFM and the state-of-the-art models.

8.2 RECOMMENDATION FOR FUTURE STUDIES

Our experiment results demonstrate that

- DeepFM outperforms the state-of the-art models in terms of AUC and Log Loss on both datasets.
- The efficiency of DeepFM is comparable to the most efficient deep model in the state-of-the-art.

There are two interesting directions for future study. One is exploring some strategies (such as introducing pooling layers) to strengthen the ability of learning most useful high-order feature interactions. The other is to train DeepFM on a GPU cluster for large-scale problems.

9. BIBLIOGRPAHY

1. H.Guo, R.Tang, Y.Ye, Z.Li, X.He, and Z.Dong, "DeepFM: An End-to-End Wide & Deep Learning Framework for CTR Prediction" in Journal of Latex Class Files, Vol. 14, No. 8, August 2015 arXiv:1804.04950v2
2. 2.H.Guo, R.Tang, Y.Ye, Z.Li, X.He, "DeepFM: A Factorization-Machine based Neural Network for CTR Prediction" in arXiv:1703.04247
3. Y.Juan,Y.Zhuang,W.Chin and C.Lin "Field Aware Factorization Machine for CTR Prediction" in RecSys '16 Proceedings of the 10th ACM Conference on Recommender Systems, Pages 43-50.
4. M.Fire and J.Schler, "Exploring Online Ad Images Using a Deep Convolutional Neural Network Approach" in 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)
5. R. Kumar, S. M. Naik, V. D. Naik, S. Shiralli, Sunil V. G and M. Husain, "Predicting clicks: CTR estimation of advertisements using Logistic Regression classifier," *2015 IEEE International Advance Computing Conference (IACC)*, Bangalore, 2015, pp. 1134-1138.
6. C. Jie-Hao, L. Xue-Yi, Z. Zi-Qian, S. Ji-Yun and Z. Qiu-Hong, "A CTR prediction method based on feature engineering and online learning," *2017 17th International Symposium on Communications and Information Technologies (ISCIT)*, Cairns, QLD, 2017, pp. 1-6.
7. Dataset: <https://www.kaggle.com/atirpetkar/avazu-ctr/data>

10. APPENDICES

```
class DeepFM(BaseEstimator, TransformerMixin):
    def __init__(self, feature_size, field_size,
                 embedding_size=8, dropout_fm=[1.0, 1.0],
                 deep_layers=[32, 32], dropout_deep=[0.5, 0.5, 0.5],
                 deep_layers_activation=tf.nn.relu,
                 epoch=10, batch_size=256,
                 learning_rate=0.001, optimizer_type="yellowfin",
                 batch_norm=0, batch_norm_decay=0.995,
                 verbose=False, random_seed=2016,
                 use_fm=False, use_deep=True,
                 loss_type="logloss", eval_metric=roc_auc_score,
                 l2_reg=0.0, greater_is_better=True):
        assert (use_fm or use_deep)
        assert loss_type in ["logloss", "mse"],
            "loss_type can be either 'logloss' for classification task or 'mse' for regression task"

        self.feature_size = feature_size           # denote as M, size of the feature dictionary
        self.field_size = field_size              # denote as F, size of the feature fields
        self.embedding_size = embedding_size      # denote as K, size of the feature embedding

        self.dropout_fm = dropout_fm
        self.deep_layers = deep_layers
        self.dropout_deep = dropout_deep
        self.deep_layers_activation = deep_layers_activation
        self.use_fm = use_fm
        self.use_deep = use_deep
        self.l2_reg = l2_reg

        self.epoch = epoch
        self.batch_size = batch_size
        self.learning_rate = learning_rate
        self.optimizer_type = optimizer_type
```

```

self.batch_norm = batch_norm
self.batch_norm_decay = batch_norm_decay

self.verbose = verbose
self.random_seed = random_seed
self.loss_type = loss_type
self.eval_metric = eval_metric
self.greater_is_better = greater_is_better
self.train_result, self.valid_result = [], []

self._init_graph()

def _init_graph(self):
    self.graph = tf.Graph()
    with self.graph.as_default():

        tf.set_random_seed(self.random_seed)

        self.feats_index = tf.placeholder(tf.int32, shape=[None, None],
                                         name="feats_index") # None * F
        self.feats_value = tf.placeholder(tf.float32, shape=[None, None],
                                         name="feats_value") # None * F
        self.label = tf.placeholder(tf.float32, shape=[None, 1], name="label") # None * 1
        self.dropout_keep_fm = tf.placeholder(tf.float32, shape=[None], name="dropout_keep_fm")
        self.dropout_keep_deep = tf.placeholder(tf.float32, shape=[None], name="dropout_keep_deep")
        self.train_phase = tf.placeholder(tf.bool, name="train_phase")

        self.weights = self._initialize_weights()

```



```

# model
self.embeddings = tf.nn.embedding_lookup(self.weights["feature_embeddings"],
                                         self.feats_index) # None * F * K
feat_value = tf.reshape(self.feats_value, shape=[-1, self.field_size, 1])
self.embeddings = tf.multiply(self.embeddings, feat_value)

# first order term
self.y_first_order = tf.nn.embedding_lookup(self.weights["feature_bias"], self.feats_index) # None * F * 1
self.y_first_order = tf.reduce_sum(tf.multiply(self.y_first_order, feat_value), 2) # None * F
self.y_first_order = tf.nn.dropout(self.y_first_order, self.dropout_keep_fm[0]) # None * F

# second order term
# sum_square part
self.summed_features_emb = tf.reduce_sum(self.embeddings, 1) # None * K
self.summed_features_emb_square = tf.square(self.summed_features_emb) # None * K

# square_sum part
self.squared_features_emb = tf.square(self.embeddings)
self.squared_sum_features_emb = tf.reduce_sum(self.squared_features_emb, 1) # None * K

# second order
self.y_second_order = 0.5 * tf.subtract(self.summed_features_emb_square, self.squared_sum_features_emb) # None * K
self.y_second_order = tf.nn.dropout(self.y_second_order, self.dropout_keep_fm[1]) # None * K

# Deep component
self.y_deep = tf.reshape(self.embeddings, shape=[-1, self.field_size * self.embedding_size]) # None * (F*K)
self.y_deep = tf.nn.dropout(self.y_deep, self.dropout_keep_deep[0])
for i in range(0, len(self.deep_layers)):
    self.y_deep = tf.add(tf.matmul(self.y_deep, self.weights["layer_%d" % i]), self.weights["bias_%d" % i]) # None * layer[i] * 1
    if self.batch_norm:
        self.y_deep = self.batch_norm_layer(self.y_deep, train_phase=self.train_phase, scope_bn="bn_%d" % i) # None * layer[i] * 1
    self.y_deep = self.deep_layers_activation(self.y_deep)
    self.y_deep = tf.nn.dropout(self.y_deep, self.dropout_keep_deep[1+i]) # dropout at each Deep layer

# DeepFM
if self.use_fm and self.use_deep:
    concat_input = tf.concat([self.y_first_order, self.y_second_order, self.y_deep], axis=1)
elif self.use_fm:
    concat_input = tf.concat([self.y_first_order, self.y_second_order], axis=1)
elif self.use_deep:
    concat_input = self.y_deep
self.out = tf.add(tf.matmul(concat_input, self.weights["concat_projection"]), self.weights["concat_bias"])

```

```

# loss
if self.loss_type == "logloss":
    self.out = tf.nn.sigmoid(self.out)
    self.loss = tf.losses.log_loss(self.label, self.out)
elif self.loss_type == "mse":
    self.loss = tf.nn.l2_loss(tf.subtract(self.label, self.out))
# l2 regularization on weights
if self.l2_reg > 0:
    self.loss += tf.contrib.layers.l2_regularizer(
        self.l2_reg)(self.weights["concat_projection"])
    if self.use_deep:
        for i in range(len(self.deep_layers)):
            self.loss += tf.contrib.layers.l2_regularizer(
                self.l2_reg)(self.weights["layer_%d"%i])

# optimizer
if self.optimizer_type == "adam":
    self.optimizer = tf.train.AdamOptimizer(learning_rate=self.learning_rate, beta1=0.9, beta2=0.999,
                                             epsilon=1e-8).minimize(self.loss)
elif self.optimizer_type == "adagrad":
    self.optimizer = tf.train.AdagradOptimizer(learning_rate=self.learning_rate,
                                                initial_accumulator_value=1e-8).minimize(self.loss)
elif self.optimizer_type == "gd":
    self.optimizer = tf.train.GradientDescentOptimizer(learning_rate=self.learning_rate).minimize(self.loss)
elif self.optimizer_type == "momentum":
    self.optimizer = tf.train.MomentumOptimizer(learning_rate=self.learning_rate, momentum=0.95).minimize(
        self.loss)
elif self.optimizer_type == "yellowfin":
    self.optimizer = YFOptimizer(learning_rate=self.learning_rate, momentum=0.0).minimize(
        self.loss)

# init
self.saver = tf.train.Saver()
init = tf.global_variables_initializer()
self.sess = self._init_session()
self.sess.run(init)

```

```

# number of params
total_parameters = 0
for variable in self.weights.values():
    shape = variable.get_shape()
    variable_parameters = 1
    for dim in shape:
        variable_parameters *= dim.value
    total_parameters += variable_parameters
if self.verbose > 0:
    print("#params: %d" % total_parameters)

def _init_session(self):
    config = tf.ConfigProto(device_count={"gpu": 0})
    config.gpu_options.allow_growth = True
    return tf.Session(config=config)

def _initialize_weights(self):
    weights = dict()

    # embeddings
    weights["feature_embeddings"] = tf.Variable(
        tf.random_normal([self.feature_size, self.embedding_size], 0.0, 0.01),
        name="feature_embeddings") # feature_size * K
    weights["feature_bias"] = tf.Variable(
        tf.random_uniform([self.feature_size, 1], 0.0, 1.0), name="feature_bias") # feature_size * 1

    # deep layers
    num_layer = len(self.deep_layers)
    input_size = self.field_size * self.embedding_size
    glorot = np.sqrt(2.0 / (input_size + self.deep_layers[0]))
    weights["layer_0"] = tf.Variable(
        np.random.normal(loc=0, scale=glorot, size=(input_size, self.deep_layers[0])), dtype=np.float32)
    weights["bias_0"] = tf.Variable(np.random.normal(loc=0, scale=glorot, size=(1, self.deep_layers[0])),
        dtype=np.float32) # 1 * layers[0]

    for i in range(1, num_layer):
        glorot = np.sqrt(2.0 / (self.deep_layers[i-1] + self.deep_layers[i]))
        weights["layer_%d" % i] = tf.Variable(
            np.random.normal(loc=0, scale=glorot, size=(self.deep_layers[i-1], self.deep_layers[i])),
            dtype=np.float32) # layers[i-1] * layers[i]
        weights["bias_%d" % i] = tf.Variable(
            np.random.normal(loc=0, scale=glorot, size=(1, self.deep_layers[i])),
            dtype=np.float32) # 1 * layer[i]

```

```

# final concat projection layer
if self.use_fm and self.use_deep:
    input_size = self.field_size + self.embedding_size + self.deep_layers[-1]
elif self.use_fm:
    input_size = self.field_size + self.embedding_size
elif self.use_deep:
    input_size = self.deep_layers[-1]
glorot = np.sqrt(2.0 / (input_size + 1))
weights["concat_projection"] = tf.Variable(
    np.random.normal(loc=0, scale=glorot, size=(input_size, 1)),
    dtype=np.float32) # layers[i-1]*layers[i]
weights["concat_bias"] = tf.Variable(tf.constant(0.01), dtype=np.float32)

return weights

def batch_norm_layer(self, x, train_phase, scope_bn):
    bn_train = batch_norm(x, decay=self.batch_norm_decay, center=True, scale=True, updates_collections=None,
        is_training=True, reuse=None, trainable=True, scope=scope_bn)
    bn_inference = batch_norm(x, decay=self.batch_norm_decay, center=True, scale=True, updates_collections=None,
        is_training=False, reuse=True, trainable=True, scope=scope_bn)
    z = tf.cond(train_phase, lambda: bn_train, lambda: bn_inference)
    return z

def get_batch(self, Xi, Xv, y, batch_size, index):
    start = index * batch_size
    end = (index+1) * batch_size
    end = end if end < len(y) else len(y)
    return Xi[start:end], Xv[start:end], [[y_] for y_ in y[start:end]]

# shuffle three lists simultaneously
def shuffle_in_unison_scary(self, a, b, c):
    rng_state = np.random.get_state()
    np.random.shuffle(a)
    np.random.set_state(rng_state)
    np.random.shuffle(b)
    np.random.set_state(rng_state)
    np.random.shuffle(c)

```

```

def fit_on_batch(self, Xi, Xv, y):
    feed_dict = {self.feats_index: Xi,
                 self.feats_value: Xv,
                 self.label: y,
                 self.dropout_keep_fm: self.dropout_fm,
                 self.dropout_keep_deep: self.dropout_deep,
                 self.train_phase: True}
    loss, opt = self.sess.run((self.loss, self.optimizer), feed_dict=feed_dict)
    return loss

def fit(self, Xi_train, Xv_train, y_train,
        Xi_valid=None, Xv_valid=None, y_valid=None,
        early_stopping=False, refit=False):
    """
    :param Xi_train: [[ind1_1, ind1_2, ...], [ind2_1, ind2_2, ...], ..., [indi_1, indi_2, ..., indi_j, ...], ...]
                     ind_i_j is the feature index of feature field j of sample i in the training set
    :param Xv_train: [[val1_1, val1_2, ...], [val2_1, val2_2, ...], ..., [vali_1, vali_2, ..., vali_j, ...], ...]
                     vali_j is the feature value of feature field j of sample i in the training set
                     vali_j can be either binary (1/0, for binary/categorical features) or float (e.g., 10.24, for numerical features)
    :param y_train: label of each sample in the training set
    :param Xi_valid: list of list of feature indices of each sample in the validation set
    :param Xv_valid: list of list of feature values of each sample in the validation set
    :param y_valid: label of each sample in the validation set
    :param early_stopping: perform early stopping or not
    :param refit: refit the model on the train+valid dataset or not
    :return: None
    """
    has_valid = Xv_valid is not None
    for epoch in range(self.epoch):
        t1 = time()
        self.shuffle_in_unison_scary(Xi_train, Xv_train, y_train)
        total_batch = int(len(y_train) / self.batch_size)
        for i in range(total_batch):
            Xi_batch, Xv_batch, y_batch = self.get_batch(Xi_train, Xv_train, y_train, self.batch_size, i)
            self.fit_on_batch(Xi_batch, Xv_batch, y_batch)

```

```

# evaluate training and validation datasets
train_result = self.evaluate(Xi_train, Xv_train, y_train)
self.train_result.append(train_result)
if has_valid:
    valid_result = self.evaluate(Xi_valid, Xv_valid, y_valid)
    self.valid_result.append(valid_result)
if self.verbose > 0 and epoch % self.verbose == 0:
    if has_valid:
        print("[%d] train-result=%.4f, valid-result=%.4f [%0.1f s]"
              % (epoch + 1, train_result, valid_result, time() - t1))
    else:
        print("[%d] train-result=%.4f [%0.1f s]"
              % (epoch + 1, train_result, time() - t1))
if has_valid and early_stopping and self.training_termination(self.valid_result):
    break

# fit a few more epoch on train+valid until result reaches the best_train_score
if has_valid and refit:
    if self.greater_is_better:
        best_valid_score = max(self.valid_result)
    else:
        best_valid_score = min(self.valid_result)
    best_epoch = self.valid_result.index(best_valid_score)
    best_train_score = self.train_result[best_epoch]
    Xi_train = Xi_train + Xi_valid
    Xv_train = Xv_train + Xv_valid
    y_train = y_train + y_valid
    for epoch in range(100):
        self.shuffle_in_unison_scary(Xi_train, Xv_train, y_train)
        total_batch = int(len(y_train) / self.batch_size)
        for i in range(total_batch):
            Xi_batch, Xv_batch, y_batch = self.get_batch(Xi_train, Xv_train, y_train,
                                                         self.batch_size, i)
            self.fit_on_batch(Xi_batch, Xv_batch, y_batch)
    # check
    train_result = self.evaluate(Xi_train, Xv_train, y_train)
    if abs(train_result - best_train_score) < 0.001 or \
        (self.greater_is_better and train_result > best_train_score) or \
        ((not self.greater_is_better) and train_result < best_train_score):
        break

```

```

def predict(self, Xi, Xv):
    """
    :param Xi: list of list of feature indices of each sample in the dataset
    :param Xv: list of list of feature values of each sample in the dataset
    :return: predicted probability of each sample
    """
    # dummy y
    dummy_y = [1] * len(Xi)
    batch_index = 0
    Xi_batch, Xv_batch, y_batch = self.get_batch(Xi, Xv, dummy_y, self.batch_size, batch_index)
    y_pred = None
    while len(Xi_batch) > 0:
        num_batch = len(y_batch)
        feed_dict = {self.feat_index: Xi_batch,
                    self.feat_value: Xv_batch,
                    self.label: y_batch,
                    self.dropout_keep_fm: [1.0] * len(self.dropout_fm),
                    self.dropout_keep_deep: [1.0] * len(self.dropout_deep),
                    self.train_phase: False}
        batch_out = self.sess.run(self.out, feed_dict=feed_dict)

        if batch_index == 0:
            y_pred = np.reshape(batch_out, (num_batch,))
        else:
            y_pred = np.concatenate((y_pred, np.reshape(batch_out, (num_batch,))))

        batch_index += 1
        Xi_batch, Xv_batch, y_batch = self.get_batch(Xi, Xv, dummy_y, self.batch_size, batch_index)

    return y_pred

def evaluate(self, Xi, Xv, y):
    """
    :param Xi: list of list of feature indices of each sample in the dataset
    :param Xv: list of list of feature values of each sample in the dataset
    :param y: label of each sample in the dataset
    :return: metric of the evaluation
    """
    y_pred = self.predict(Xi, Xv)
    return self.eval_metric(y, y_pred)

```

```

sys.path.append("../")
from DeepFM import DeepFM

gini_scorer = make_scorer(gini_norm, greater_is_better=True, needs_proba=True)

def _load_data():

    dfTrain = pd.read_csv(config.TRAIN_FILE)
    dfTest = pd.read_csv(config.TEST_FILE)

    def preprocess(df):
        cols = [c for c in df.columns if c not in ["id", "target"]]
        df["missing_feat"] = np.sum((df[cols] == -1).values, axis=1)
        df["ps_car_13_x_ps_reg_03"] = df["ps_car_13"] * df["ps_reg_03"]
        return df

    dfTrain = preprocess(dfTrain)
    dfTest = preprocess(dfTest)

    cols = [c for c in dfTrain.columns if c not in ["id", "target"]]
    cols = [c for c in cols if (not c in config.IGNORE_COLS)]

    X_train = dfTrain[cols].values
    y_train = dfTrain["target"].values
    X_test = dfTest[cols].values
    ids_test = dfTest["id"].values
    cat_features_indices = [i for i,c in enumerate(cols) if c in config.CATEGORICAL_COLS]

    return dfTrain, dfTest, X_train, y_train, X_test, ids_test, cat_features_indices

def _run_base_model_dfm(dfTrain, dfTest, folds, dfm_params):
    fd = FeatureDictionary(dfTrain=dfTrain, dfTest=dfTest,
                          numeric_cols=config.NUMERIC_COLS,
                          ignore_cols=config.IGNORE_COLS)
    data_parser = DataParser(feat_dict=fd)
    Xi_train, Xv_train, y_train = data_parser.parse(df=dfTrain, has_label=True)
    Xi_test, Xv_test, ids_test = data_parser.parse(df=dfTest)

    dfm_params["feature_size"] = fd.feats_dim
    dfm_params["field_size"] = len(Xi_train[0])

```



```

y_train_meta = np.zeros((dfTrain.shape[0], 1), dtype=float)
y_test_meta = np.zeros((dfTest.shape[0], 1), dtype=float)
_get = lambda x, l: [x[i] for i in l]
gini_results_cv = np.zeros(len(folds), dtype=float)
gini_results_epoch_train = np.zeros((len(folds), dfm_params["epoch"]), dtype=float)
gini_results_epoch_valid = np.zeros((len(folds), dfm_params["epoch"]), dtype=float)
for i, (train_idx, valid_idx) in enumerate(folds):
    Xi_train_, Xv_train_, y_train_ = _get(Xi_train, train_idx), _get(Xv_train, train_idx), _get(y_train, train_idx)
    Xi_valid_, Xv_valid_, y_valid_ = _get(Xi_train, valid_idx), _get(Xv_train, valid_idx), _get(y_train, valid_idx)

    dfm = DeepFM(**dfm_params)
    dfm.fit(Xi_train_, Xv_train_, y_train_, Xi_valid_, Xv_valid_, y_valid_)

    y_train_meta[valid_idx,0] = dfm.predict(Xi_valid_, Xv_valid_)
    y_test_meta[:,0] += dfm.predict(Xi_test, Xv_test)

    gini_results_cv[i] = gini_norm(y_valid_, y_train_meta[valid_idx])
    gini_results_epoch_train[i] = dfm.train_result
    gini_results_epoch_valid[i] = dfm.valid_result

y_test_meta /= float(len(folds))

# save result
if dfm_params["use_fm"] and dfm_params["use_deep"]:
    clf_str = "DeepFM"
elif dfm_params["use_fm"]:
    clf_str = "FM"
elif dfm_params["use_deep"]:
    clf_str = "DNN"
print("%s: %.5f (%.5f)%(clf_str, gini_results_cv.mean(), gini_results_cv.std())
filename = "%s_Mean%.5f_Std%.5f.csv"%(clf_str, gini_results_cv.mean(), gini_results_cv.std())
_make_submission(ids_test, y_test_meta, filename)

_plot_fig(gini_results_epoch_train, gini_results_epoch_valid, clf_str)

return y_train_meta, y_test_meta

def _make_submission(ids, y_pred, filename="submission.csv"):
    pd.DataFrame({"id": ids, "target": y_pred.flatten()}).to_csv(
        os.path.join(config.SUB_DIR, filename), index=False, float_format="%.5f")

```

```

y_train_meta = np.zeros((dfTrain.shape[0], 1), dtype=float)
y_test_meta = np.zeros((dfTest.shape[0], 1), dtype=float)
_get = lambda x, l: [x[i] for i in l]
gini_results_cv = np.zeros(len(folds), dtype=float)
gini_results_epoch_train = np.zeros((len(folds), dfm_params["epoch"]), dtype=float)
gini_results_epoch_valid = np.zeros((len(folds), dfm_params["epoch"]), dtype=float)
for i, (train_idx, valid_idx) in enumerate(folds):
    Xi_train_, Xv_train_, y_train_ = _get(Xi_train, train_idx), _get(Xv_train, train_idx), _get(y_train, train_idx)
    Xi_valid_, Xv_valid_, y_valid_ = _get(Xi_train, valid_idx), _get(Xv_train, valid_idx), _get(y_train, valid_idx)

    dfm = DeepFM(**dfm_params)
    dfm.fit(Xi_train_, Xv_train_, y_train_, Xi_valid_, Xv_valid_, y_valid_)

    y_train_meta[valid_idx,0] = dfm.predict(Xi_valid_, Xv_valid_)
    y_test_meta[:,0] += dfm.predict(Xi_test, Xv_test)

    gini_results_cv[i] = gini_norm(y_valid_, y_train_meta[valid_idx])
    gini_results_epoch_train[i] = dfm.train_result
    gini_results_epoch_valid[i] = dfm.valid_result

y_test_meta /= float(len(folds))

# save result
if dfm_params["use_fm"] and dfm_params["use_deep"]:
    clf_str = "DeepFM"
elif dfm_params["use_fm"]:
    clf_str = "FM"
elif dfm_params["use_deep"]:
    clf_str = "DNN"
print("%s: %.5f (%.5f)%(clf_str, gini_results_cv.mean(), gini_results_cv.std())
filename = "%s_Mean%.5f_Std%.5f.csv"%(clf_str, gini_results_cv.mean(), gini_results_cv.std())
_make_submission(ids_test, y_test_meta, filename)

_plot_fig(gini_results_epoch_train, gini_results_epoch_valid, clf_str)

return y_train_meta, y_test_meta

def _make_submission(ids, y_pred, filename="submission.csv"):
pd.DataFrame({"id": ids, "target": y_pred.flatten()}).to_csv(
    os.path.join(config.SUB_DIR, filename), index=False, float_format="%.5f")

```

```

y_train_meta = np.zeros((dfTrain.shape[0], 1), dtype=float)
y_test_meta = np.zeros((dfTest.shape[0], 1), dtype=float)
_get = lambda x, l: [x[i] for i in l]
gini_results_cv = np.zeros(len(folds), dtype=float)
gini_results_epoch_train = np.zeros((len(folds), dfm_params["epoch"]), dtype=float)
gini_results_epoch_valid = np.zeros((len(folds), dfm_params["epoch"]), dtype=float)
for i, (train_idx, valid_idx) in enumerate(folds):
    Xi_train_, Xv_train_, y_train_ = _get(Xi_train, train_idx), _get(Xv_train, train_idx), _get(y_train, train_idx)
    Xi_valid_, Xv_valid_, y_valid_ = _get(Xi_train, valid_idx), _get(Xv_train, valid_idx), _get(y_train, valid_idx)

    dfm = DeepFM(**dfm_params)
    dfm.fit(Xi_train_, Xv_train_, y_train_, Xi_valid_, Xv_valid_, y_valid_)

    y_train_meta[valid_idx,0] = dfm.predict(Xi_valid_, Xv_valid_)
    y_test_meta[:,0] += dfm.predict(Xi_test, Xv_test)

    gini_results_cv[i] = gini_norm(y_valid_, y_train_meta[valid_idx])
    gini_results_epoch_train[i] = dfm.train_result
    gini_results_epoch_valid[i] = dfm.valid_result

y_test_meta /= float(len(folds))

# save result
if dfm_params["use_fm"] and dfm_params["use_deep"]:
    clf_str = "DeepFM"
elif dfm_params["use_fm"]:
    clf_str = "FM"
elif dfm_params["use_deep"]:
    clf_str = "DNN"
print("%s: %.5f (%.5f)"%(clf_str, gini_results_cv.mean(), gini_results_cv.std()))
filename = "%s_Mean%.5f_Std%.5f.csv"%(clf_str, gini_results_cv.mean(), gini_results_cv.std())
_make_submission(ids_test, y_test_meta, filename)

_plot_fig(gini_results_epoch_train, gini_results_epoch_valid, clf_str)

return y_train_meta, y_test_meta

def _make_submission(ids, y_pred, filename="submission.csv"):
    pd.DataFrame({"id": ids, "target": y_pred.flatten()}).to_csv(
        os.path.join(config.SUB_DIR, filename), index=False, float_format="%.5f")

```

```

def _plot_fig(train_results, valid_results, model_name):
    colors = ["red", "blue", "green"]
    xs = np.arange(1, train_results.shape[1]+1)
    plt.figure()
    legends = []
    for i in range(train_results.shape[0]):
        plt.plot(xs, train_results[i], color=colors[i], linestyle="solid", marker="o")
        plt.plot(xs, valid_results[i], color=colors[i], linestyle="dashed", marker="o")
        legends.append("train-%d"%(i+1))
        legends.append("valid-%d"%(i+1))
    plt.xlabel("Epoch")
    plt.ylabel("Normalized Gini")
    plt.title("%s"%model_name)
    plt.legend(legends)
    plt.savefig("./fig/%s.png"%model_name)
    plt.close()

# load data
dfTrain, dfTest, X_train, y_train, X_test, ids_test, cat_features_indices = _load_data()

# folds
folds = list(StratifiedKFold(n_splits=config.NUM_SPLITS, shuffle=True,
                             random_state=config.RANDOM_SEED).split(X_train, y_train))

dfm_params = {
    "use_fm": True,
    "use_deep": True,
    "embedding_size": 8,
    "dropout_fm": [1.0, 1.0],
    "deep_layers": [32, 32],
    "dropout_deep": [0.5, 0.5, 0.5],
    "deep_layers_activation": tf.nn.relu,
    "epoch": 30,
    "batch_size": 1024,
    "learning_rate": 0.001,
    "optimizer_type": "adam",
    "batch_norm": 1,
    "batch_norm_decay": 0.995,
    "l2_reg": 0.01,
    "verbose": True,
    "eval_metric": gini_norm,
    "random_seed": config.RANDOM_SEED
}

```

```
y_train_dfm, y_test_dfm = _run_base_model_dfm(dfTrain, dfTest, folds, dfm_params)

fm_params = dfm_params.copy()
fm_params["use_deep"] = False
y_train_fm, y_test_fm = _run_base_model_dfm(dfTrain, dfTest, folds, fm_params)

dnn_params = dfm_params.copy()
dnn_params["use_fm"] = False
y_train_dnn, y_test_dnn = _run_base_model_dfm(dfTrain, dfTest, folds, dnn_params)
```

10.1 INPUT/OUTPUT LISTING

Input : As we previously mentioned, the input files are in the form of test.csv and train.csv

Output : The final output confirms that the logloss value obtained from our model is less than that of FM.