Recommender Models Comparison on Douban Movie Dataset

Rui Fan 11268011

Yani Mo 11263831

Ning Ye 11267197

1 Introduction

One of the consistently popular topics on the movie and E-commerce websites is to accurately use various recommendation models to provide different suggestions for different users. Douban website is one of the largest Chinese social media websites, focus on providing reviews and recommendations of movies, books, and music for users. In this research project, we will use raw data of the Douban Movie dataset[1] to build a recommender system that could suggest movies to users who are most likely prefer these movies. This dataset was crawled in September 2019, and we used the subset of this dataset which contains $99,598$ users, $21,596$ movies, and $220,411$ ratings, including UserID, MovieID, movie names, genres and ratings, etc.

1.1 Problem Definition

In this project, we aim to build a recommender system on the Douban movie dataset, which recommends movies to users who are most likely to enjoy these movies. From previous papers, we find that the effectiveness of systems varies incredibly among different models. Thus, our project contains two tasks: first, compare different recommendation models with chosen evaluation criteria; Then, we pick the most performant model on the Douban Movie dataset[1] to develop a recommendation system for specific users.

1.2 Literature Review

Recommender systems use multiple data related to users' interests to predict and recommend items to customers. The more efficient recommendation algorithm can leverage the previous interaction between users and items to make more accurate recommendations. Thus, various recommender systems have been developed extensively due to their usefulness in commercial applications, such as ensemble models and deep learning models. However, the implementation of deep learning algorithms in Chinese movie recommender systems remains elusive. In our research, we selected three models to compare on the dataset written in Chinese(the Douban Movie dataset[1]): Wide and Deep, DeepFM, and LightGBM models.

The resources for recommender systems are ample, ranging from books to online open packages. Specifically, Weichen Shen(2017)[2] proposed *DeepCTR*, which provides a set of easy-to-use packages and models API for feature engineering and deep learning models. By mainly using this *DeepCTR* Models API, we built Wide-and-Deep and DeepFM models.

The Wide and Deep model [3] is widely used for large-scale regression and classification problems with sparse inputs. This model has a wide part and a deep part. The wide part is to memorize the feature interactions through a set of cross-product feature transformations for its effectiveness and interpretation. While the deep part can generalize better to unseen feature combinations. Yan Liu et al.(2019) [4] have compared the traditional models such as LR and GBDT, with deep models such as Wide and Deep and DeepFM to detect their performance differences in store site recommendation.

Preprint. Under review.

The second model is DeepFM [5]. The key challenge in the task of building a recommender system is to effectively model the feature interactions, both low and high-order interactions. Existing methods have a strong bias towards low and high-order interactions or require expertise in feature engineering. For example, generalized linear models (GLM) perform well in modeling low-order interactions. However, because of GLM's limited learning capacities, it is hard to model the high-order feature interactions. As for Factorization Machines(FM)[6] model the pairwise feature interactions as the inner product of latent vectors between features. Although FM can model the high-order feature interaction, it suffers from expensive computation cost when the order goes above two.

DeepFM [5] is an FM based Neural Network that could emphasize both low and high-order feature interactions behind user behaviors. DeepFM also consists of two components, the FM component and the deep component, which share the same input and are jointly trained. With the advantages of factorization machines for recommendation and deep learning for feature learning, DeepFM has superior performance in three aspects:

- DeepFM does not need any pre-training, while the model FNN need to pre-train the FM-initialized part then feed-forward neural network;
- The wide and deep component of DeepFM share the same input raw feature vectors, which enables DeepFM to learn both high-order and low-order feature interactions simutaneouly from the input raw features;
- DeepFM introduces a sharing strategy of feature embedding to avoid feature engineering, while the input of Wide&Deep model's "wide" part relies on expertise feature engineering

Another model for our experiments is the LightGBM [7] model based on Gradient Boosting Decision Tree (GBDT) algorithm. The LightGBM contains two novel techniques: Gradient-based One-Side Sampling and Exclusive Feature Bundling. These two techniques help the LightGBM algorithm outperform the conventional GBDT model significantly when dealing with a large number of data instances and features in terms of computational speed and memory consumption while achieving almost the same accuracy. The LightGBM model has been used on the Criteo [8] binary classification task with also dense and sparse features of user and ad-spot information, which is similar to the dataset that we dealt with. Since our dataset is high dimensional features with a high level of sparsity and considered the efficiency of experiments, we also investigate this model as one of our targets.

1.3 Methodology

Figure 1: Experiments Process

As shown in the flow chart 1, first, we preprocess the raw dataset, by splitting Chinese sentences into vocabularies and transforming the sparse and sequence features into embeddings.

Second, to evaluate the effectiveness of different models and the impact of the key hyper-parameters on model performances, we perform experiments on three models. After data preprocessing, we split the dataset into two portions: 80%for training, 20% for testing. Then, we use Wide-and-Deep, DeepFM, and LightGBM to predict the rating of each movie for each user in the dataset. Comparing these two deep learning models with LightGBM, we can identify that which recommender model performs best. Besides, from the Douban Movie Dataset[1], user's rating scores for each movie are all integers between 1 and 5, but the predicted score calculated by the mentioned models may not be an integer. Thus, we round up the predictions to integers. For example, if the model prediction is 3.6, it will be 4 as the final rating.

Next, we measure the prediction performance of our models by using the following four metrics in the test set.

- MSE (Mean squared error): The average of the squares of errors that is the average squared difference between the predicted ratings and the actual ratings. Y_i is the actual value and $\hat{Y}_i$

is the predicted value, and n is the size of test set.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

- (After rounding up the prediction) Recall= tp / (tp + fn), where tp is the number of true positives and fn the number of false negatives.
- (After rounding up the prediction) Precision= tp / (tp + fp), where tp is the number of true positives and fp the number of false positives.
- (After rounding up the prediction) Accuracy= (tp + tn) / (tp + tn + fp + fn), where tp is true positive, fp is false positive, tn is true negative, and fn is false negative.

Finally, we predict movies with the most possible highest rating using the most performant model and use a content similarity-based algorithm to define similarity and capture the relationship between predicted high rating movies and the movies that the user has already interacted with. As a result, we recommend the top k movies that have high rating potential and most similar to users' watched and highly rated movies.

2 Experiments

2.1 Data Preprocessing

(Detailed code for this part is in [9])

Libraries in python have been used in data preprocessing:

- *thulac* package(THU Lexical Analyzer for Chinese) has been used to split sentences in Chinese vocabulary-by-vocabulary into sequecnes features.
- *re* package has been applied to specify a set of strings that matches.
- *sklearn.preprocessing* for label encoding.
- *DeepCTR* library: SparseFeat, VarLenSparseFeat have been used for spare features and sequence features embedding.

By using the above python resources, in this section, we have the following steps (figure 2)to preprocess the dataset:

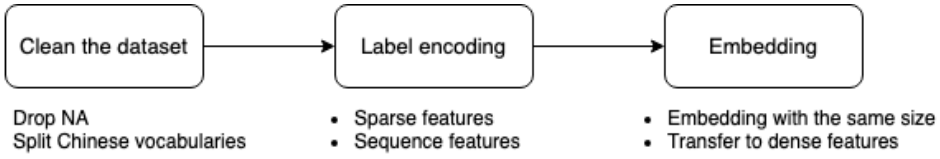

Figure 2: Preprocess steps

1. Considering the low portion of missing data, we simply eliminate rows with missing data.

2. Some variables contain sentences in Chinese, such as "NAME" and "TAGS" without a space between vocabulary, thus, we split them to Chinese vocabularies with " | " to make them easier for encoding, for example:

3. In order to better learn relationships between features, we encoded discrete features ('userID', 'MovieID', 'DOUBANSCORE', 'DOUBANVOTES') and embedded them into vectors of continuous real numbers with the same size for deep learning models.

4. Label encoding the all sparse features with multivalue ('ACTORS', 'DIRECTORS', 'GENRES', 'REGIONS', 'TAGS', 'LANGUAGES', 'MovieNameWord') into sequence features.

5. Then embedding and padding sequence features to make them into the dense features with the same size as the other features.

Movie Name="婚礼之后" in English is “After the wedding”

Before splitting:

Movie Name ="婚礼之后"

After splitting:

"NAME"="婚礼|之后"

2.2 Wide and Deep

(Detailed code for this part is in [10])

Wide and Deep learning combines the wide linear models and deep neural networks (DNN) in one model to memorize and generalize the input information for recommender systems. Based on the research of H. Cheng et al.[3], the architecture of the Wide and Deep model shows as the figure 3:

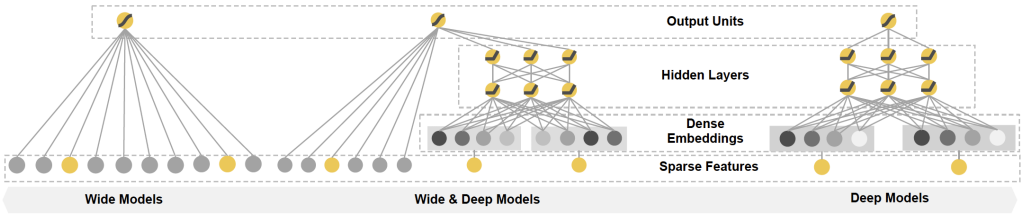

Figure 3: The spectrum of Wide & Deep models

The wide part

The wide part is a linear model with the form $y = w^T x + b$. y is the prediction, $x = \{x_1, x_2, ..., x_n\}$ is a vector of n features, $w = \{w_1, w_2, ..., w_n\}$ is the vector of model parameters and b is the bias[3]. The n features include raw entity features and transformed features. The transformed features is cross-feature transformation, which is define by [3]:

$$\phi_k(x) = \prod_{i=1}^{d} x_i^{c_{ki}}, c_{ik} \in \{0, 1\}$$

where c_{ik} is a boolean variable that is 1 if the i-th feature is part of the k-the transformation ϕ_k , and 0 otherwise[4]. The linear model is trained with a wide set of cross-product feature transformations to identify the interactions between a query-item feature pair correlated and the target(the item ratings from users), as shown in the figure 4,

Figure 4: Wide part

For example, in the Douban Movie Dataset[1], the feature(TAG = "love story", item = "After the wedding") attracts users, but (TAG = "love story", item = "The Company") doesn’t get as much interest though the character is similar. Thus, the wide part is to memorize which movie users like and it can obtain more traction. However, the wide part is not open to more generalization and detects a variety set of related movies such as "family movie".

The deep part

The deep model is trained to build the deep neural network. Each hidden layer performs the following computation[3]:

$$a^{l+1} = f\left(W^l a^l + b^l\right)$$

where l is the layer number and f is the activation function, $a^{(l)}$, $b^{(l)}$, and $W^{(l)}$ are the activation, bias, and model weights at l-th layer.

With the deep learning part, we can learn lower-dimensional embedding vectors for each feature and item. It means that the deep neural network model can generalize by matching movies to queries that are close to each other in the embedding space, as the following figure. For example, in Douban Movie Dataset[1], users who like a movie with TAG = "love story" might not mind whether this love story from Europe or Asia.

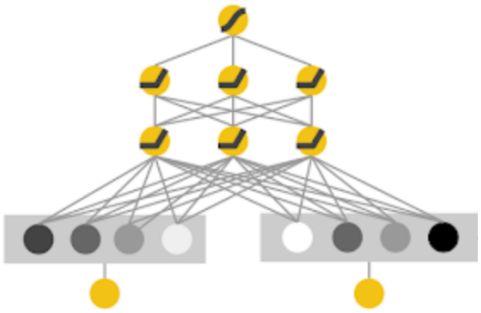

Figure 5: Deep part

Combining Wide and Deep parts

Sometimes the deep neural networks generalize and recommend irrelevant items, but they can not memorize how the co-occurrence of a feature-item pair correlates with the target, like whether a movie has been watched or not. Thus, we need the Wide and Deep model to combine the linear model with the DNN model. As shown in the following graph, the 15 sparse features from the Douban Movie Dataset[1], like TAG = "love story", movieaName = "The Company" are used in both the wide part and DNN part of the model. The cross-feature transformation in the wide part can memorize the relations between sparse features and the target rating. Besides, the DNN part can generalize to similar movies through embeddings.

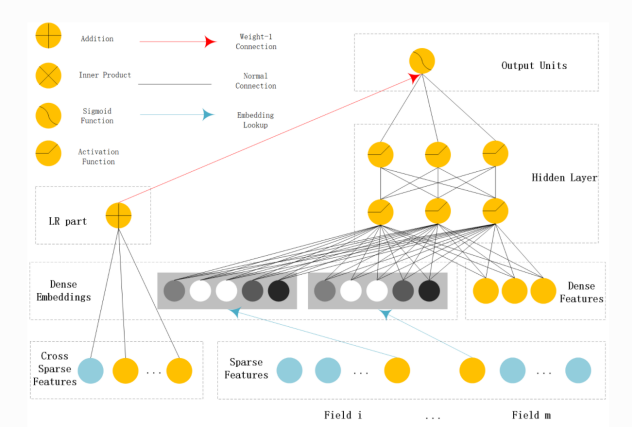

Figure 6: Combining Wide and Deep parts

Wide and Deep model training and hyper-parameters tuning

Library *DeepCTR* has been used for building the Wide-and-Deep model.

The size of the Douban Movie Dataset[1] is 220,411 with 15 features. In this part, by using Deepctr API, we train Wide and Deep model on the split training set (80% of Douban Movie Dataset) and tune hyper-parameters on the test set(20% of Douban Movie Dataset).

We focus on tuning the following hyper-parameters:

- activation function;
- the number of neurons per layer;
- the number of hidden layers

Based on the same hidden-layers, which is hidden layer sizes = (128, 128) and fixed the other hyper-parameters setting, we have the result in figure 7.

Activation Function\Test Set	MSE	Recall	Precision	Accuracy
relu	0.8320	0.584	0.7	0.6387
tanh	0.7088	0.574	0.7	0.6377
linear	0.7081	0.566	0.704	0.6363
softmax	1.4836	0.366	0.59	0.4872
softplus	0.6678	0.576	0.702	0.6367
sigmoid	0.6982	0.556	0.696	0.6322

Figure 7: Activation Function Tuning

From the table 7, we can see that the MSE, recall, precision, accuracy vary slightly when the model applies different activation function except for DNN activation='softmax'. Among the four activation functions, when DNN activation='softmax', the model achieves the highest MSE and the lowest Accuracy. In contrast, when DNN activation='softplus', the model can obtain the lowest MSE and higher accuracy, besides, when DNN activation='relu', the model can get the highest accuracy and recall, and lower MSE. Thus, we select the model whose DNN activation='relu' to make a comparison with the other two models due to its highest accuracy and recall.

Choose hidden layer = (128,128), (128, 128,), (50,), (200,) and fixed the other hyper-parameters setting. We have the result in figure 8.

	MSE	Recall	Precision	Accuracy
hidden_layer_sizes=(128, 128)	0.8320	0.584	0.7	0.6387
hidden_layer_sizes=(128,128,)	0.7338	0.586	0.69	0.6408
hidden_layer_sizes=(50,)	0.8547	0.572	0.704	0.6381
hidden_layer_sizes=(200,)	0.7837	0.574	0.702	0.6434

Figure 8: Hidden Layer Tuning

From the table 8, we can find that test MSE errors, recall, precision, and accuracy change slightly when we have different activation functions, the number of hidden layers, and the number of neurons in DNN. It means that a number of hidden layers and neurons are the hyper-parameters, which make the biggest difference in the model performance. Thus, in the next model comparison, we select hidden layer size=(128,128,) due to its highest accuracy and lower MSE.

2.3 DeepFM

(Detailed code for this part is in [11])

Compared with the Wide and Deep model, DeepFM is similar to Wide and Deep, which is also jointly trained by the deep component and the wide component. There are two distinct differences between the two models.

FM Component

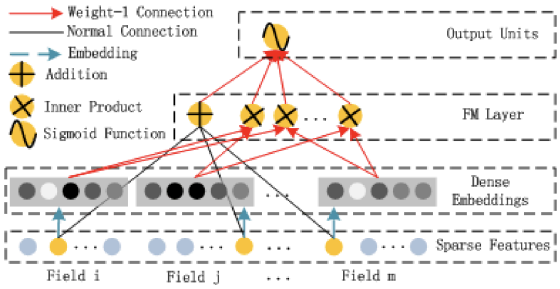

Figure 9: The architecture of FM

First, as shown in figure 9, DeepFM apply FM component instead of Wide component to avoid human feature engineering, enables to learn low and high-order features interactions from input raw features. Since the Wide component uses the linear model to predict, it is more efficient for the FM model to learn feature interactions automatically.

Second, the DeepFM model uses raw features as shared inputs for the FM part and Deep part, which can guarantee the consistency of features.

DeepFM training and hyper-parameter training

Library *DeepCTR* has been used for building the DeepFM model.

The DeepFM model is trained on the train set (80% of the dataset) and tune hyper-parameters on the test set (20% of the dataset).

In this part, by using *Deepctr* API[2], we focus on tuning the following hyper-parameters:

- activation function;
- DNN hidden units;
- dropout layer

Choose different activation function and fixed the other hyper-parameters setting. We have results in figure 10.

dnn_activation (test set)	MSE	Recall	Precision	Accuracy
dnn_activation='relu'	0.64	0.64	0.67	0.6389
dnn_activation='sigmoid'	0.63	0.64	0.66	0.6357
dnn_activation='linear'	0.63	0.64	0.67	0.6371
dnn_activation='tanh'	0.63	0.64	0.67	0.6374

Figure 10: Activation Function Tuning

From the figure 10, we can see that the MSE, recall, precision, accuracy vary slightly when model apply different activation functions. Among the four activation functions, with DNN activation='relu', we achieve highest accuracy.

Choose different hidden units and fixed the other hyper-parameters setting as shown in figure 11. By tuning the hidden units, when hidden units vary from 128 to 512, the MSE, recall, precision, accuracy

dnn_hidden_units (test set)	MSE	Recall	Precision	Accuracy
dnn_hidden_units=(128,128)	0.64	0.65	0.67	0.6456
dnn_hidden_units=(256,256)	0.65	0.65	0.68	0.6525
dnn_hidden_units=(512,512)	0.65	0.65	0.67	0.6517

Figure 11: Hidden units tuning

vary slightly, which could show that hidden units do not have important impacts on DeepFM model performance here. Within comparison, when DNN hidden units=(256,256), the accuracy reaches 0.6525, which is the highest.

Choose different dropout ratios and fixed the other hyper-parameters setting. The dropout ratios tuning process shows in figure 12. The differences between MSE, recall, precision, accuracy are very small. When the dropout is 0, the accuracy is the highest as 0.6394. Thus, in this DeepFM experiment, the dropout layer is useless and can be removed.

dnn_dropout (test set)	MSE	Recall	Precision	Accuracy
dnn_dropout=0	0.64	0.64	0.66	0.6394
dnn_dropout=0.2	0.62	0.63	0.66	0.6261
dnn_dropout=0.3	0.61	0.62	0.66	0.6215
dnn_dropout=0.4	0.63	0.63	0.66	0.6303
dnn_dropout=0.5	0.63	0.63	0.65	0.6293

Figure 12: Dropout tuning

2.4 LightGBM

(Detailed code for this part is in [12])

Libraries *LightGBM*, *StandardScaler*, *Optuna* and *scipy* have been used in this section.

In this experiment, we investigate the LightGBM model performance on the Douban dataset. The LightGBM model is based on Gradient Boosting Decision Tree (GBDT) algorithm with two novel techniques [7]: Gradient-based One-Side Sampling(GOSS) and Exclusive Feature Bundling(EFB). GOSS keeps all the instances with large gradients and performs random sampling and a constant multiplier on the instances with small gradients. By doing so, we put more focus on the under-trained instances without changing the original data distribution by much. EFB merges many sparse features (both the one-hot coding features and implicitly exclusive features) into much fewer features, which effectively reduces the number of features and speed up the training process.

Unlike the other two models, the LightGBM model does not take embedded vectors as input for each variable. Therefore, we applied the one-hot encoding method to preprocess the categorical variables like genres, tags, regions, etc. And since there are multiple values in such variables, we first encoded different keys to indexes as we did in the former models and then encoded indexes into one-hot vectors manually. Besides, we standardize the numerical features by the *StandardScaler* function offered by the *sklearn.preprocessing* library. Then we concatenate all encoded variables into a feature matrix X, which is a scipy sparse matrix with the size of $220411 * 26709$ and sparsity of 99.92% as input to the LightGBM model.

We selected some important hyper-parameters for automatically tuning using the *Optuna* library. Since the *Max_depth, num_leaves*, and *min_data_in_leaf* have more significant impacts on the model performance, we mainly focus on tuning on these hyper-parameters. After one hundred trials to optimize the accuracy, we find the best hyper-parameters showed in figure 13. Other hyper-parameters keep the default values.

Throughout these trials, the best model finally gives us the accuracy of 43.4%, while other metrics shown in figure 15. To further investigate the LightGBM model, we can obtain features' importance based on the split. Only 1,076 out of 26,709 features have impacts on the model prediction. Here,

Hyper-parameter	Value
num_leaves	151
max_depth	21
min_data_in_leaf	350
learning_rate	0.01
feature_fraction	0.73
lambda_l1	0.10

Figure 13: Optimal hyper-parameters of the LightGBM model

we display the ten most important features in figure 14, we could conclude that *userID, MovieID, genre*, and *region* have a significant influence on the tree splitting. Even though this model shows great interpretability, this model gives low accuracy in the task of rating prediction. And we think the reason why this model performs unpleasantly is that this dataset does not have enough interactions between users and movies. When more than two-third of userID only have one instance in the dataset, the model hard to generalize relationships between specific user and movies, which lead to low accuracy in the prediction.

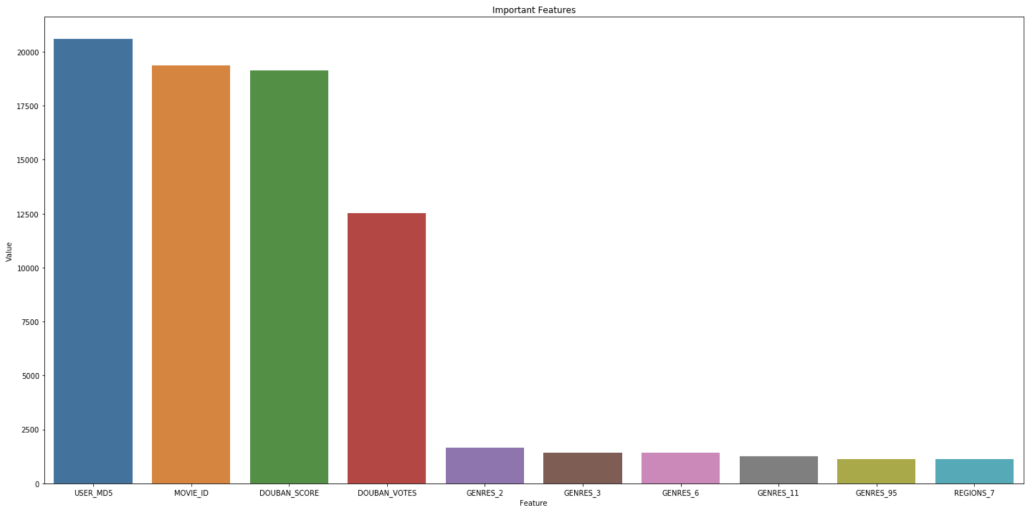

Figure 14: Important features

3 Evaluation

Library in python: *sklearn.metrics* has been used for model evaluation.

	MSE	Recall	Precision	Accuracy
Wide&Deep	0.8320	0.584	0.70	0.6387
DeepFM	0.8377	0.590	0.69	0.6384
Baseline:LightGBM	0.8253	0.323	0.54	0.4338

Figure 15: Models evaluation

From figure 15, we can see that the MSE of DeepFM is 0.8377, which is slightly higher than that of Wide-and-Deep. And the accuracy of Wide-and-Deep is a little higher than that of DeepFM.

Compared to the LightGBM model, the MSE of Wide-and-Deep and DeepFM models are much lower than the MSE of LightGBM. Besides, the precision and accuracy of Wide-and-Deep and DeepFM are much higher than of LightGBM. Thus, in our experiments, the Wide-and-Deep model performs best.

As shown in figure 16 and 17, we have the specific comparison of two neural networks performance and most values are very similar.

Accuracy: 0.6387249275217661

	precision	recall	f1-score	support
1.0	0.87	0.49	0.62	16521
2.0	0.52	0.51	0.51	26893
3.0	0.62	0.73	0.67	74075
4.0	0.62	0.72	0.66	69035
5.0	0.87	0.47	0.61	33887
6.0	0.00	0.00	0.00	0
accuracy			0.64	220411
macro avg	0.58	0.48	0.51	220411
weighted avg	0.66	0.64	0.64	220411

Figure 16: Wide-and-Deep performance

Accuracy: 0.638425486931233

	precision	recall	f1-score	support
1.0	0.85	0.51	0.63	16521
2.0	0.51	0.54	0.52	26893
3.0	0.62	0.72	0.67	74075
4.0	0.62	0.70	0.66	69035
5.0	0.86	0.48	0.61	33887
6.0	0.00	0.00	0.00	0
accuracy			0.64	220411
macro avg	0.58	0.49	0.52	220411
weighted avg	0.66	0.64	0.64	220411

Figure 17: DeepFM performance

The architecture of Wide-and-Deep is similar to DeepFM. Wide-and-Deep has wide and deep components. For the deep component, since the deep neural network has strong generalization ability, it can catch the connections between users and movies in our dataset. That is the reason why both Wide-and-Deep and DeepFM perform much better than LightGBM.

For the wide component, through the linear feature transformation, it can catch specific features and memorize the users' rating and their preferences. For example, when users often watch romantic movies during a period, the deep neural network can recommend movies that can give users similar emotional experiences instead of only focusing on romantic movies, which shows its surprise and specific characteristic. But the limitation is that our dataset has fewer user information than other datasets on the open-source platform, which could lead to the failure of capturing the relationship between users and movies.

When we compare Wide-and-Deep and baseline, the generalization ability can easily find the relationship between users and movies that are fit for their preference. To the baseline, there is no surprise effect and the diversity of recommendation is very limited, that is the reason why Wide-and-Deep performs much better than LightGBM.

Also, after analyzing the result of model performance, we find that the higher rating (around 5) or lower rating (around 1) could be more accurate than the rating within a medium scope (from 3 to 4). The reason why we get this phenomenon is that the limited number of users' features, leading to the loss of connection between users and movies. Thus when the feature engineering map the users' information to movies, the effect of representation could not as good as the same model on other datasets.

4 Further Experiments

(Detailed code for this part is in [13])

Libraries *linear kernel* and *CountVectorizer* have been used in this section.

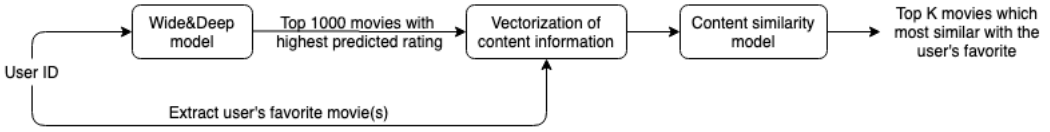

Figure 18: Recommender system

The best model we selected from the comparison is the Wide-and-Deep model. With this model, we learned the latent interactions between user and movie, which could provide a better performance in terms of diversity and serendipity but hard to explain why the model recommends such movies to a specific user. Therefore, we add a content-based recommender model to produce recommendations for the specific user with more consideration of relevance.

For example, as shown in figure 18, for a certain user with a unique ID "*ad74bb4926aa3e77433e01b1d8051a20*", we first sort out a small portion of movies (e.g. a subset of 1000) with the highest predicted ratings using the Wide-and-Deep model.

Second, we go back to the original dataset to find out which movies this user watched before and extract the highest rated one as this user's favorite. Here, this user's favorite movie is *MovieID* "26897547" which has the highest rating of 5.

Finally, we vectorize the content of movies using *CountVectorizer* to encode all text features, like name, actors, directors, etc and investigate the cosine similarity between the user's favorite and the subset of 1,000 movies from step one with *Linear kernel*. Then, we recommend the top k movies with the highest similarity, in other words, the most similar movies with this user's favorite. The figure 19 shows the recommendation list with 10 movies that have the highest cosine similarity with the *movieID* "26897547", which demonstrate relatively obvious connections with this user favorite in terms of either genres or tags.

	NAME_y
359	探长来访
731	布朗宁版本
776	影子大地
252	寂静人生
398	经度
459	抛开自我空间
652	查令十字街84号
828	雨天下的迎神会
885	无人之境
887	永远活下去

Figure 19: Recommendation list

And we also find calculating cosine similarity is a computation expensive task that requires large memory, however, gives very resemble movies compared with the customer's favorite. Therefore, it is not possible to reverse the sequence of the Wide-and-Deep model and the content similarity model with limited computation resources.

To sum up, we build an efficient recommender system using an ensemble method that not only considers the user's implicit interests but also provides meaningful explanations to improve user experience.

5 Conclusion

Given the Douban dataset, models with deep neural networks provide rating predictions more accurately than the tree-based LightGBM model. And the Wide-and-Deep model has the best performance on the accuracy, which is mainly due to the great generalization advantage of the deep neural network. However, the LightGBM model has the advantage in interpretability but not as good as deep neural network models in terms of generalization. To improve the interpretability of results, we suggest combining the Wide-and-Deep model with the content similarity model to generate the final recommendations.

Recommender system design is relatively a subjective task and we believe both accuracy and user experience need to be addressed. In the first part of our model, we focus on the accuracy of rating predictions, which recommend movies that a specific customer will most likely enjoy. In the second part, from these possible enjoyable movie list, we extract the most similar ones as our final top k recommendations, which could provide more explanatory reasons for the user: "This recommendation is given based on you previously watched or highly rated." With two parts of this recommender system, we hope to improve the user experience or intention to click by providing explanations of recommendations and user satisfaction or retention by recommending movies with accurate predictions on user's latent interests.

References

[1] Douban Movie Dataset, available at `https://www.csuldw.com/2019/09/08/2019-09-08-moviedata-10m/`

[2] Weichen Shen. (2017). *DeepCTR: Easy-to-use,Modular and Extendible package of deep-learning based CTR models*. `https://github.com/shenweichen/deepctr`

[3] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, and H. Shah, "*Wide & deep learning for recommender systems*," CoRR, vol. abs/1606.07792, 2016

[4] Yan Liu, Bin Guo, Nuo Li, Jing Zhang, Jingmin Chen, Daqing Zhang, Yinxiao Liu, Zhiwen Yu, Sizhe Zhang, and Lina Yao, *DeepStore: An Interaction-aware Wide-and-Deep Model for Store Site Recommendation with Attentional Spatial Embeddings*, IEEE Internet of things journal 6, no. 4, (2019): 7319-7333.

[5] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, Xiuqiang He, *DeepFM: A Factorization-Machine based Neural Network for CTR Prediction*, In Proceedings of the IJCAI. 2782–2788.

[6] Steffen RENDLE, *Factorization machines*. In Proceedings of the 10th IEEE International Conference on Data Mining. IEEE Computer Society.

[7] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu. "*LightGBM: A Highly Efficient Gradient Boosting Decision Tree". Advances in Neural Information Processing Systems 30* (NIPS 2017), pp. 3149-3157.

[8] Criteo Dataset, available at `http://labs.criteo.com/2013/12/download-terabyte-click-logs/`

[9] Code for data pre-processing at `https://drive.google.com/file/d/1Dzl6LO00tZneUoSehbtRTnnFk4bSr75V/view?usp=sharing`

[10] Code for Wide-and-Deep at `https://drive.google.com/file/d/1S2JLQm_2y7enwNQXdogvlo4HEWbKPHgM/view?usp=sharing`

[11] Code for DeepFM at `https://colab.research.google.com/drive/1ElLNK4Mkc0l016FWZvZuQvzPBJJyrKCz?usp=sharing`

[12] Code for LightGBM at https://drive.google.com/file/d/1Lj7XXSIxKE3DLLP_CiBNiHxWuJAz2E8p/view?usp=sharing

[13] Code for further experiment at https://drive.google.com/file/d/12IqniIGqg-UgjMoAXtyNbaIO6R8RsoNn/view?usp=sharing