Memorization and Generalization for Music Recommendation: From Wide and Deep to Content-based DNN with Psuedo-MF and Transfer Learning

Kaho Chan
kc3137
kc3137@columbia.edu

Yu Wang
yw3025
yw3025@columbia.edu

Jingxi Xu
jx2324
jx2324@columbia.edu

Abstract

In order to handle the challenge of both memorization and generalization, we introduce a new method called contented-based DNN (deep neural network) with transfer learning and pseudo-MF (matrix factorization). We apply our model to a Kaggle challenge on music recommendation and present a better result than using wide and deep model with Tensorflow APIs under these particular problem settings. We also analyze the performance of our model with a variety of other models to show the advantages and limitations of our model and propose potential work for future researchers.

1. Introduction

With the growth of digital music industry and popularity of music software and platform like Spotify, Google Play, etc., automatic music recommendation has become a hot research topic in the field of machine learning/information retrieval.

Two broad directions in music recommendation are *collaborative filtering* vs. *content-based recommendation*. While the collaborative filtering relies on the usage patterns (items that a specific user has consumed or rated which provide information about the user's preference more directly), content-based recommendation uses item content (audio signal) and metadata (artist, year of release, etc.). According to [Sla11], collaborative filtering generally outperforms content-based methods but it cannot handle cold-start problem (no usage data for new songs).

One challenge in developing recommendation systems is to achieve both *memorization* (learning the frequent co-occurrence of features and exploiting the available correlation in the historical data) and *generalization* (based on transitivity of correlation and exploring new feature combinations) [CKH+16]. Memorization can be achieved effectively by cross-product transformations while embedding-based models, such as factorization machines are useful methods to achieve generalization. There are some debates on whether factorization methods should be classified as memorization or generalization. For this project, we regard factorization as memorization methods because even though it can generalize to previously unseen user-song pairs, it cannot handle cold start when new songs or users enter the system.

2. Related Work

Previous pioneering work presented by [HKV08] uses collaborative filtering with implicit datasets (no direct rating/preference information from users but only purchase histories) for TV shows recommendation. They transform the implicit feedback into preference-confidence paradigm and provide a latent vector algorithm to handle it. Inspired by their work, [VdODS13] adopts their *weighted matrix factorization* (WMF) and *alternating least squares* (ALS) optimization methods for content-based music recommendation. However, different from other content-based methods such as Mcfee et al [MBL12], who use metric learning to learn a similarity metric for comparing bag-of-words representations of audio signals, [VdODS13] adopts a deep convolutional neural network for predicting. They achieve a better performance and also handles the cold start problem so that new and unpopular songs can be recommended. Later on, Wang et al [WW14] develops a novel model which can unify the extraction and prediction into an automated simultaneous process, which further improves the performance of hybrid and content-based music recommendation systems.

One of the most commonly used models for collaborative filtering is matrix factorization, which is adopted in [HKV08, VdODS13]. The idea behind matrix factorization is to factorize the rating matrix R comprised of user set U and song set S. Obviously, R is of the size $|U| \times |S|$ and entry r_{ij} is the rating of song s_j given by user u_i. We assume that there are k latent features of each user and song which can truly reflect the preference of a user or the nature of a song. Mathematically, we are trying to find matrix P (a

$|U| \times k$ matrix) and Q (a $|S| \times k$ matrix) where:

$$R = P \times Q^{\intercal}$$

The ith row of P denoted by p_i and the ith row of Q denoted by q_i are the latent vectors of user i and song i respectively. Therefore, we can predict the rating of a song for an unobserved user-song pair (u_i, s_j) by calculating:

$$\hat{r}_{ij} = p_i \cdot q_j$$

We can then introduce the objective function to optimize by general optimization methods like gradient descent:

$$L = \sum_{u_i, s_j, r_{ij}} (p_i \cdot q_j - r_{ij})^2$$

While matrix factorization is a fundamental embedding-based methods for generalization (memorization, in this project), logistic regression is widely used as a baseline model for memorization [LDG+17]. However, Cheng et al [CKH+16] recently has presented a wide & deep learning framework for jointly training neural networks with embeddings and linear model with feature transformations, which effectively handles the challenge of both memorization and generalization. Their methods are the main baseline model we refer to in this project.

3. Problem Formulation

The dataset we use is provided by KKBOX. It is also used by a challenge from Kaggle where we apply our model in practice. For more information about the dataset and Kaggle challenge, please see here.

In this challenge, we train a model to predict the chances of a user listening to a song repetitively after the first observable listening event within a time window was triggered. The training set provides user-song pairs with the `target` variable indicating whether there is a repetitive listening activity of that user to that song within a month. The test set is some new user-song pairs not recorded in the training set where we need to predict their `target` variables. Some extra features like entry point, artist, song length are also provided in training set and extra sets.

Inspired by previous work presented in section 2, in this project, we aim to adopt the hybrid wide & deep learning framework as the baseline model. Then we will try different combinations and preprocessing of the features to implement both collaborative filtering and content-based methods to find an optimal configuration with best performance for music recommendation.

3.1. Criteria

The criteria used in both this paper and the Kaggle contest is area under receiver operating characteristic curve (ROC-AUC or simply AUC).

AUC is the area under the receiver operating characteristic curve (ROC). The latter one works by plotting the true positive rate ($TPR = TP/(TP + FN)$) against the false positive rate ($FPR = FP/(FP + TN)$) at various threshold settings.

4. Feature Engineering

4.1. Raw data

KKBOX supplies three kinds of datasets for this problem: meta dataset (containing both train and test datasets), song related dataset, member related dataset.

- **Meta dataset** The data about the event, including user id, song id, source screen name, source type and source system tab. The last three are information about where the event happens (where the user plays the music for the first time).
- **Song related dataset** Associated with the song id, including the length of the song, the genre categories of the song (may have multiple genres), artist name, composer, lyricist, language of the song, and year of the song.
- **Member related dataset** City of the user, age of the user, gender of the user, where the user registers, the registration time and expiration date.

For feature engineering, we first merge the three datasets into one dataframe according to song id and user id Figure 1. Then for categorical raw features, we create count features, binary features and do label encoding to transform them into real values. For numerical raw features, we standardize them. Finally, we generate historical statistical features based on history events. Note in the following discussion, we only give a few examples for a particular kind of feature engineering. An illustration of the whole features after feature engineering can be seen in the Github repository.

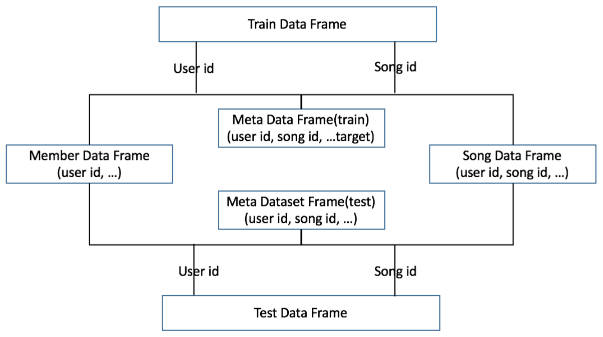

Figure 1. Merge dataset to generate train and test data frame

4.2. Feature Cleaning

The data is not very complete and we have to fill in the empty slots and clear the noises in the feature table. To be specific, we use 'Unknown' tag for each categorical feature. For numerical data, we fill in the holes using the mean value. Also, there are lots of data that do not make sense: for example, some user ages are marked by unreasonable values such as 1000 and 3. We clear these irregular entries and treat them as missing information.

4.3. Categorical feature engineering

- **Count feature** New features based on how many categorical labels appear in one feature column. For instance, 'genre count' indicates the number of genre ids of a song, and 'lyricist count' indicates the number of lyricists of a song.

- **Binary feature** New features based on whether a particular label appears in one feature column. For example, 'song lang' indicates if the song is English/Chinese, 'small song' shows if the song has a duration time less than mean duration time. Note that we generate this feature since some labels in the feature columns are highly relevant with the targets.

- **Feature transformation** Feature transformation maps raw categorical features into real value vectors. We use different strategies for different models.

  For wide and deep model, we transform the categorical features with fewer categories directly using feature hashing to make it a vector with fixed length. For other categorical columns which have many more options than the number of keys we want to put on, we hash them into buckets to reduce the number of categories. Then such features are embedded into one embedding layer.

  For Pseudo-MF based model, we encode the labels into continuous real value numbers and then embed the features into $\log_2 n$ vectors (n is the number of classes in that categorical feature column).

4.4. Numerical feature engineering

- **bucketization** New categorical feature is generated by 'bucketization'; that is, given a set of boundaries $b_0 = -\infty, b_1, \cdots, b_k, b_{k+1} = \infty$, split the data into categories represented by different bins $(b_i, b_{i+1}), i = 0, 1, \cdots, k$. An example will be 'age bucket' since we commonly believe same age groups tend to listen to the same kinds of songs.

- **standardization** All the numerical feature is normalized before being put into the model:

  $$x = \frac{x - mean(x)}{std(x)}$$

  Note numerical feature also includes the count features generated from categorical features.

4.5. Historical feature engineering

The historical statistical features are generated with regards to the targets. It shows how many times a feature value resulting in the song-listening event occurs in the past one month. For example, 'count song played indicates how many times a song has been played (which means the target is 1), and 'count artist played means how many times an artist has appeared in the dataframe where the target is 1. Such feature is highly important since it is used by the model to achieve memorization tasks.

5. Method

5.1. Overview

In this project, we seek to combine memorization and generalization. We will present various methods in this section, which can formed into two parts: in the first part, we present the logistic regression and the deep neural network method, then we introduce the Wide and Deep Network to combine the two models ; in the second part, an alternative Matrix Factorization (MF) model will be discussed, and then we combine it with deep neural network to produce a new method called content-based deep neural network. We will omit some of the implementation details here and focus on the ideas. At last, we make a summary of the methods proposed in this section.

5.2. Wide Model

The Wide Model is a linear model such as a logistic regression. In a binary classification problem, it follows

$$y = f(w^T x + b)$$

where x is the feature vector, w and b is the parameters train, and f is usually a non-linear function. We use sigmoid function, i.e. $f(x) = \frac{1}{1+e^{-x}}$. The generated y indicates the possibility of the label being 1.

The input features in the wide model part contain the one-hot representations of categorical features and crossed features. The idea of crossed feature is to combine two different features into one specific feature which may have more information than a separate feature. We hash the combined features into an integer value and use the hashed values as a new feature. For example, combining feature 'city' and 'language' will produce a crossed feature '(city, language)' and the hashing function may appear as following:

```
                    ...
('New York', 'English') -> 42
 ('Beijing', 'Chinese') -> 97
                    ...
```

Wide models with crossed feature columns can memorize sparse interactions between features effectively. However, one limitation is that it cannot generalize on feature combinations that never appear in the training data before. Thus we introduce the deep model.

5.3. Deep Model

5.3.1 Embedding Layer

To utilize categorical features in a deep neural network, we use embedding techniques to embed them into fixed size vectors. To be more specific, suppose we are embedding a categorical feature 'City' into a 2-dimensional vector: each possible value of the categorical feature is mapped to a 2-dimensional vector. The mapped vectors are first initialized randomly, for example, the i-th category 'Beijing' may be mapped to a random vector $v_i = (x_1^i, x_2^i) = (0.1, -0.4)$. After initialization, the mappings are trained as parameters just as the weights of normal densely connected layers. So x_1^i and x_2^i may be trained to different values, for example, $(0.9, 0.2)$ which minimizes the loss function of the whole model. Embedding layers could greatly reduce the redundancy and noise in the categorical features: compared to simply using one-hot on categorical features, it shrinks the feature dimension to a pre-defined size, which is usually relatively small. By this process of shrinking dimension, redundancies and noises are likely to be filtered away while the most important characteristics are preserved.

5.3.2 Structure

The Deep Model is a fully-connected feed forward network with two layers of hidden units. The input feature of the deep model contains continuous features and densely-embedded vectors from the sparse, high-dimensional categorical features. These features are combined together as input layer to the hidden units in the forward pass and we use rectified linear units (ReLUs) as activation function for each hidden layer and sigmoid as the activation function for the last layer. To avoid overfitting, we also add a dropout layer before the output layer.

$$a^{l+1} = f(w^l a^l + b^l)$$

Through dense embedding, deep models can generalize better and can make predictions on feature pairs that were previously unseen in the training data.

5.4. Wide and Deep Network

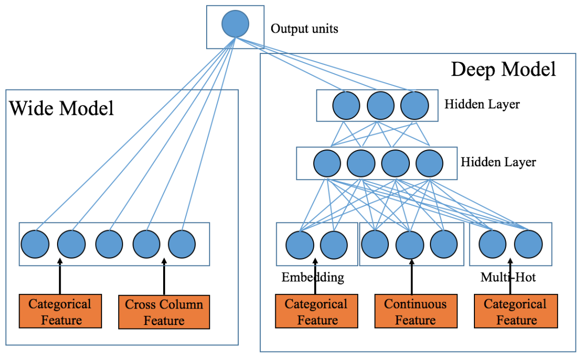

Figure 2. Wide and Deep model structure for music recommendation

In order to combine the previous two methods and take advantage of both memorization and generalization, Wide and Deep Network is proposed, which combines a linear model with a feed forward neural network as in Figure 2. It jointly trains a feed-forward neural network with embedding layers and a linear model.

The wide model and deep model are combined by weighted sum of their final output log odds, then it feeds the prediction into a logistic loss function. Then the model uses gradient descendent methods to update the parameters simultaneously for the wide and deep model.

The relationship can be represented as the following optimization problem, where f_w means the wide model, f_d means the deep model, L means the loss function and h means the sigmoid activation function:

$$\hat{f} = \underset{\alpha,\beta,f_w,f_d,f}{\arg\min}\ L(y, f(x))$$

such that $f(x) = h(\alpha f_w(x) + \beta f_d(x))$ and $\alpha + \beta = 1$

5.5. Pseudo MF

As discussed before, Matrix factorization can achieve memorization and is widely used in recommendation problems. To integrate more neural network components, we now present an alternative of Matrix Factorization, which will be called Pseudo-MF (Matrix Factorization) in the remaining report. In order to get the latent vectors of users and songs using meta data, we use two embedding layers to embed user id and song id into two 64-dimension vectors, latent user vector and latent song vector respectively. We then get the dot product of latent user vector and latent song vector and concatenate the scalar result with the two latent vectors to form a 129-dimensional vector ($129 = 64 + 64 + 1$). In order to take more advantage of the neural networks and make this model more flexible, we add a 128-node hidden layers (fully connected, activated by ReLU) followed by a dropout layer between the concatenated layer and the output layer (fully connected, activated by sigmoid). The process is shown in Figure 3.

This model distinguishes from classical Matrix Factorization in that we still preserve the latent vectors and use neural networks to generate the output from concatenated latent vectors and dot product instead of directly using dot product of latent vectors.

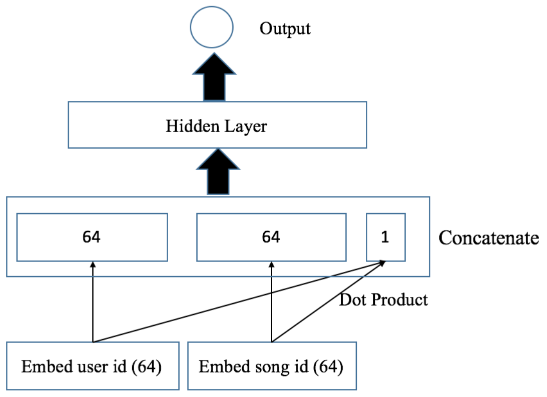

Figure 3. Pseudo-MF

5.6. Transfer Learning: From Pseudo-MF to Content-based DNN

The previous model only takes advantage of the meta data: user id and song id. But in real problems, there will be many completely new users and new songs that have not been recorded by the training data. In order to go from using pure history to a useful recommendation system able to recommend new songs to new users. We combine the previous model and a deep neural network into a model called Content-based Deep Neural Network (Content-based DNN). As shown in the graph Figure 4, it works as follows. First we train the Pseudo-MF model on user id and song id. In order to integrate content-based features (e.g. age of user, language of song, etc.), we train a generalization deep neural network from content-based features to 129-dimension vectors. The content-based features are generated by concatenating numerical features and outputs of embedding layers for categorical features. The embedding layers are trainable layers as mentioned before.

The generalization DNN is then plugged into after the concatenated layer of the Pseudo-MF. To train the final model, we fix the weights of the layers in Pseudo-MF and feed in content-based features and labels.

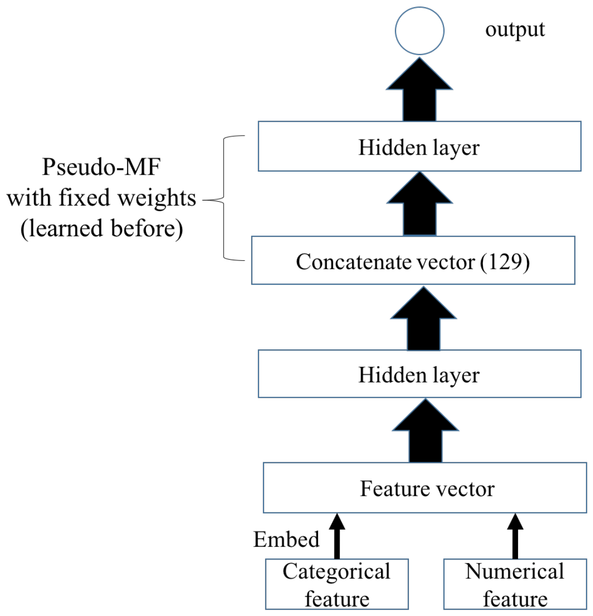

Figure 4. Content-Based DNN

To be more specific, the problem can be expressed in the following optimization problem. Denote f_e the embedding function for user id and song id, f_m the Pseudo MF, f_g the generalization DNN, f the Content-based DNN and L the loss function. We first do the following optimization for the Pseudo-MF:

$$\hat{f_m}, \hat{f_e} = \arg\min_{f_m, f_e} L\Big(f_m(f_e(x)), y\Big)$$

We then train the generalization DNN as following optimization problem:

$$\hat{f_g} = \arg\min_{f_g} L\Big(\hat{f_m}(f_g(x)), y\Big)$$

Together, we get the Content-basd DNN as

$$f(x) \triangleq f_m(f_g(x))$$

5.6.1 Inspiration and Innovation

The idea of the two-step model comes from [VdODS13], where a weighted matrix factorization is used as the first step and a convolutional neural network is trained to learn the embedding layers. We also use a two-step model, but **the model architectures and the connection between the two steps are slightly different**.

We use Pseudo-MF (DNN to simulate MF) for the first step, which allows us to integrate more flexibility. DNN is then used in the second step to take place of CNN as we only have categorical data while in [VdODS13], they directly use audios which CNN can take advantage of. Moreover, instead of training the second part using latent vectors as labels, we directly plug in the neural network after the first model with fixed weights and use binary labels to train.

The idea actually comes from transfer learning. Transfer learning in deep learning means that train a network in one dataset and then transfer it to another dataset to be trained by using the pre-trained model as an initialization or a prefix feature extractor. Instead of pre-training lower layers as in regular uses of transfer learning, we pre-train the higher part as Pseudo-MF and use the generalization DNN to train another 'feature extractor' from features excluding user ids and song ids.

5.7. Summary

The two proposed methods, Wide and Deep Networks and Content-based DNN, have similar goals of combining memorization and generalization using a combination of simple models. **The former one achieves this by jointly training a 'memorization' model (wide part) and a 'generalization' model (deep part). The latter one, instead, trains the 'generalization model' using pre-trained 'memorization model' (Pseudo-MF) as a supervisor: because the weights of upper side in the networks are fixed, the generalization DNN tends to generate similar 129-dimension vectors as Pseudo-MF.** In this way, we achieve the integration of memorization part.

6. Experiments

As in section 5, our experiments can be separated into two parts: Wide and Deep Neural Network and Content-based DNN.

6.1. Code Repository

Our code is shared through private Github repository.

6.2. Configuration

For the Wide and Deep Neural Network, we use the Tensorflow API `DNNLinearCombinedClassifier` in `tensorflow.estimator`. To implement the Pseudo-MF together with Content-based DNN, we take advantage of Keras APIs using Tensorflow as back end.

6.2.1 Wide and Deep Networks

For the wide part, we include all the categorical features mentioned above, together with crossed features, including:

- User id and song id
- Registration year and expiration year
- User id and language of the song
- User age (bucketized) with artist name

Then we concatenate all the features as one vector and pass it into the wide part.

For the deep part, we use all the categorical features by embedding them into dimension of 8, together with numerical features (age, song length, genre count and lyricist count). For user id, song id and artist name, we also include 64-dimension embedding features respectively. Then we concatenate all the features as one vector and pass it into the deep part.

We use two layers of densely connected hidden layers for the deep part. The dimensions of the two layers are 128 and 32 respectively.

To train the model, we use default optimizer as provided by the API, i.e. Adagrad optimizer for the deep part and FTRL optimizer for the wide part.

6.2.2 Content-based DNN

For the Pseudo-MF part, we embed the user id and song id into 64-dimensional vectors. The hidden layer after the concatenate layer has 128 nodes, activated by ReLU. The output layer is activated by sigmoid.

For the generalization DNN, categorical features are embedded into vectors with size $\log(\text{number of categories})$. The input layer concatenates categorical embedding vectors and numerical features and is then passed to a hidden layer with 128 nodes. Then a dropout layer is added before the output layer. The output layer is a fully connected layer without non-linear activation.

Both models are trained using optimizer RMSprop with learning rate set as `RMSprop(lr=1e-3)` and the model is compiled with `loss='binary_crossentropy'`. The setting of all the embedding layers is listed as follows, where we specify the uniform random initializer and L2 regularization:

```
Embedding = (
    ...,
    embeddings_initializer =
        RandomUniform(minval=-0.1,
        maxval=0.1),
    embeddings_regularizer = l2(1e-4),
    ...
)
```

In this phase, we also do baseline experiments using vanilla DNN. In order to compare the results, we train two vanilla DNNs, one with user ids and song ids in the features, one without. They share the same structure: input features are concatenated as before using embedding layers of categorical features and numerical features. The input layers are then followed by three fully connected hidden layers, with $128, 64, 32$ nodes respectively and ReLU as activation function. The layer is then connected to output layer using sigmoid activation after applying a dropout layer with dropout rate 0.5.

6.3. Results

We present both validation AUC (train on 90% of data and validate on the other 10%) and test AUC (validated by the public leader board on Kaggle). We present the performance of three main models on Kaggle as of Dec. 14, 2017. in Table 1, with best performance colored in green.

Model	Rank	Percentage
Wide and deep model	850	77%
Content-based DNN	700	63%
DNN with uid & sid (Best performance on Kaggle)	**390**	**35%**

Table 1. Performance of models on Kaggle

Please note that only the code of these three main models are included in our online Github repository, while code of experiments on different variations of these models are not presented for brevity.

6.3.1 Wide and Deep Neural Network

Table 2 presents the results of three models using `DNNLinearCombinedClassifier`, the wide model just uses the linear part of the API (i.e. logistic regression) while the deep model only uses the deep part (i.e. Deep Neural Networks).

Model	Validation AUC	Test AUC
Wide Model	0.69	0.59
Deep Model	0.55	0.54
Wide and deep Network	0.77	0.62

Table 2. Results of Wide and Deep Networks

We have experimented on numerous modifications on the whole model but most of the experiments do not lead to a satisfying result. Limited by experiences with Tensorflow, we are unable to fully utilize the power of the API `DNNLinearCombinedClassifier`. Also, the problem may come from the limited size of the data to use. However, the comparison between results generated by different usages shows that this approach does make some improvements by combining memorization and generalization.

6.3.2 Content-based DNN

We implement content-based DNN using Keras APIs, the results are shown in Table 3. We add a baseline model using DNN without uid & sid as features. This model is the same as Content-based DNN in that both models predict the target on features excluding user ids and song ids.

Model	Validation AUC	Test AUC
Pseudo MF	0.772	0.626
DNN without uid & sid	0.735	0.652
Content-based DNN	0.745	0.655
DNN with uid & sid (Best performance on Kaggle)	**0.825**	**0.676**

Table 3. Results of Content-based DNN

Simple Pseudo-MF does not work well among these three methods as it only memorize. As we can see, the proposed Content-based DNN does prove performance compared to only using Pseudo-MF and vanilla DNN without user ids and song ids.

Also, we implement several variations of content-based DNN for comparison (results listed in Table 4).

1. **Fix** Just as stated in the 'Method' section, we train the Pseudo-MF and fix the weights for generalization model.

2. **Fix, then unfix** After training the first variation, we un-fix the weights of Pseudo-MF part and continue to train. This variation works slightly worse as it may overfit the data.

3. **Warm start** Similar as the first model, we use the weights of pre-trained Pseudo-MF as initialization (warm start) instead of fixed weights. This variation is more flexible than the first variation as it does not have the constraints of fixing Pseudo-MF layers. It achieves a slightly better result, but takes significantly longer time to train (larger than 100 epochs, compared to around 20 epochs for the first variation).

Model	Validation AUC	Test AUC
Fix	0.745	0.655
Fix, then unfix	0.7488	0.6529
Warm start	0.7447	0.658

Table 4. Variations of Content-based DNN

6.4. Discussion

We can see from these results that the content-based DNN with Psuedo-MF outperforms the wide and deep model implemented by Tensorflow APIs (0.655 v.s. 0.62). However, the wide and deep model has been proved to be working well for online store application recommendation systems as shown in [CKH+16]. A possible reason for this discrepancy might lie in the nature of the dataset. In the work of [CKH+16], the dataset has a much smaller number of features and each feature has a richer pool of categorical values. In the case of music recommendation challenge, we have more features but less choices for each feature. This difference in dataset might have been the key reason for the unexpected performance of wide and deep model. In addition, since we are implementing the wide and deep model with existing Tensorflow APIs, it is hard for us to modify the way the model is implemented to better fit with our problem.

Psuedo-MF on its own turns out to have a similar performance as wide and deep model with much simpler architecture and easier to train but its drawback is also significant – cannot make prediction on new users entering the system or recommend new songs. This problem can be addressed by our content-based DNN with a better prediction accuracy.

The difference between DNN with uid & sid in Table 3 and the deep model in Table 2 lies in the ways of implementation. We only use the deep model part of the Tendorflow API for comparison with wide and deep model in Table 2 but use `Keras` package in Table 3. This also proves that the Tensorflow API for wide and deep model might not be feasible for implementing pure deep learning or logistic regression model.

It turns out that the simple DNN model with all features (including uid and sid) has the best prediction accuracy (0.676). It might because the dataset of our problem is not complicated enough (or not large enough) to fully take advantage of our content-based model with psuedo-MF.

Another important characteristics of this dataset is that the training data and test data seem to have very different distribution, which results in the huge difference between validation AUC and test AUC. We address this problem to some extent by adding dropout and regularization to prevent overfitting.

7. Conclusion

In this project, we present two ways to combine memorization and generalization, Wide and Deep Networks and Content-based DNN. For the former one, we jointly train the 'logistic regression' (memorization) and 'DNN' (generalization). As for the latter one, content-based DNN, we first simulate the matrix factorization by a memorization DNN to learn historical listening events. Then, we fix the model weights in the memorization DNN and by means of transfer learning, train a generalization DNN in the bottom to handle cold start and learn information from pure content-based features. The latter approach proves to be more efficient in both time and accuracy. While the former one can be seen as a 'parallel combination', the latter one is inherently serial. Model combinations in this two structures could also be used in other scenarios (such as recommendation systems) where we need both memorization and generalization for better results.

8. Limitation and Future Work

Even though the content-based model has shown an improvement with regards to the wide and deep model, it still does not surpass the performance of a simple deep learning model using `Keras` API. As discussed in section 6.4, this might be a result of the nature of dataset and that `Keras` package is using better, more mature and well-optimized functions.

More work can be done in modifying the DNN network architectures of our content-based model. For instance, we can change the embedding size in Psuedo-MF or add more hidden layers (currently 129 $\implies$ 128), and we can also try different network structures for the lower DNN in Figure 4.

Another potential improvement can be done in collecting more content data for the model. One potential data category proved helpful by [VdODS13] is the audio track signal of each song, which we did not manage to acquire due to time and resource limitations. The audio signal would be rich and useful data for feature engineering and subsequently, training our Deep Learning models. We may also replace the DNN with CNN in our Content-based DNN model if the audio track signals are involved, as experimented in [VdODS13].

As for wide and deep model, one possible way to improve its performance would be instead of using the API provided by Tensorflow, develop the architectures from scratch so that we can modify and implement different DNN network parameters and structures to make it better fit with our problem settings.

References

[CKH+16] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 7–10. ACM, 2016.

[HKV08] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. Ieee, 2008.

[LDG+17] Xiaoliang Ling, Weiwei Deng, Chen Gu, Hucheng Zhou, Cui Li, and Feng Sun. Model ensemble for click prediction in bing search ads. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 689–698. International World Wide Web Conferences Steering Committee, 2017.

[MBL12] Brian McFee, Luke Barrington, and Gert Lanckriet. Learning content similarity for music recommendation. *IEEE transactions on audio, speech, and language processing*, 20(8):2207–2218, 2012.

[Sla11] Malcolm Slaney. Web-scale multimedia analysis: Does content matter? *IEEE MultiMedia*, 18(2):12–15, 2011.

[VdODS13] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Advances in neural information processing systems*, pages 2643–2651, 2013.

[WW14] Xinxi Wang and Ye Wang. Improving content-based and hybrid music recommendation using deep learning. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 627–636. ACM, 2014.