Deep or Wide? Why Not Both?

Wilson Lui
wl2522@columbia.edu

Xuxan He
xh2343@columbia.edu

Yizhang Tao
yt2591@columbia.edu

Abstract

Our project is concerned with predicting whether or not a KKBox user will play a song again within a month of playing it for the first time. This problem is a binary classification task where we make predictions based on each user's demographic profile as well as each song's metadata. Our goal is to implement a wide and deep recommender system in TensorFlow that pairs a linear classifier with a fully-connected neural network. We then attempt to improve on this model architecture to achieve competitive results in the KKBox Music Recommendation Challenge on Kaggle[1].

1. Introduction

With the rapid growth of various online applications, the issue of recommendation is becoming increasingly critical. Music recommendation systems automatically recommend songs from a large database that match a user's music preferences.

The quality of a match is influenced by many user-related factors, such as personality and social environment. However, existing music recommendation systems usually rely on audio signals from the music and achieve unsatisfactory recommendation performance. A two-stage approach is commonly used: extracting traditional audio content features, such as Mel-frequency cepstral coefficients (MFCC), followed by using these features to predict user preferences. These traditional audio content features were not intended for use in music recommendation, but rather for music classification and speech recognition. Using such features can result in poor recommendation performance because these high-level concepts may not be essential to the user's music preferences. We believe that an effective music recommendation system can be made with a set of good features and a deep and wide neural network model.

2. Related Work

Currently, music recommendation systems can be classified into four categories: collaborative filtering (CF), content-based methods, context-based methods, and hybrid methods. Collaborative filtering recommends songs by considering the preferences of other like-minded users. For instance, if user A and B have similar music preferences, then the songs liked by A but not yet listened to by B will be recommended to B [1]. Content-based methods recommend songs that have similar audio content to the user's preferred songs. Most existing content-based methods first extract traditional audio features such as MFCC and then make recommendations based on the similarities between the feature vectors of the songs [2, 3]. Context-based methods recommend songs to match various aspects of the user context, such as activity, environment, or physiological state [4, 5]. Hybrid methods combine two or more of the above methods. For example, hybrid CF and content-based methods have been explored extensively in recommendation systems [6, 7].

The music field has only recently begun to embrace the power of deep learning. In Hamel et al. [8], a deep belief network was used for music genre classification and auto-tagging, with performance surpassing that of models based on MFCC and MIM feature sets. Schmidt et al. [9] found that a DBN easily outperforms models using traditional features representing rhythm and melody. To the best of our knowledge, the first deep learning based approach for music recommendation was almost concurrently proposed by Oord et al. [10] in 2013. They first performed matrix factorization to obtain latent features for all the songs and then used deep learning to map audio content to those latent features.

The idea of combining wide linear models with cross-product feature transformations and deep neural networks with dense embeddings is inspired by the paper by Cheng et al. [11], which introduces a model that recommends apps to users on Google Play. Specifically, it combines a linear classifier along with a fully-connected neural network with each component of the model being given different inputs to train on. The neural network is trained on categorical features, while the linear classifier is trained on cross-product transformations, which are the interactions between different features. For example, the cross-product interaction "feature1_feature2" will equal to 1 only if "feature1" and "feature2" both equal 1. This idea was previously used in factorization machines [12], which add

[1]https://www.kaggle.com/c/kkbox-music-recommendation-challenge

generalization to linear models by factorizing the interactions between two variables as a dot product between two low-dimensional embedding vectors. Similar ideas are also applied in other research fields. In language models, joint training of recurrent neural networks (RNN) and maximum entropy models with n-gram features has been proposed to significantly reduce RNN complexity [13]. In computer vision, deep residual learning [14] has been used to reduce the difficulty of training deeper models and improve accuracy with shortcut connections which skip one or more layers.

3. Problem formulation

The goal of the KKBox Music Recommendation Challenge is to build a music recommendation system that can predict the probability of a user listening to a song again after the first observable listening event was triggered. If there is another listening event triggered within a month after the user's first observable listening event, the user-song pair is labeled 1. Otherwise, it is labeled 0. Thus, this challenge is a binary classification task. Submissions are evaluated based on the area under the ROC curve (ROC AUC).

4. About the Features

KKBox has provided a training dataset containing information concerning the first observable listening event for each unique user-song pair within a specific time duration. Metadata for each observed user and song are also provided. The features can be divided into three main categories: member features, song features, and app features. The member features include the user's KKBox member ID, location, gender, age (the dataset contains many nonsensical and improbable age values), and subscription registration and expiration dates. The song features include the song ID, song length, genre ID (songs may have multiple genres), artist name, composer, lyricist, song language, and so on. The app features include the name of the tab where the event was triggered, the name of the layout a user saw when the song was played, and an entry point through which the member first plays the song on the mobile app. Based on the different characteristics of each feature, we will use different preprocessing strategies in order to convert them into an appropriate format for use in TensorFlow.

For the wide model, we used the following features from the dataset:

1. User gender
2. User city
3. User age
4. Song language
5. Song genre

The transformation process that we applied to the user age feature is detailed in Section 4.3. We found that cross-product feature transformations using these features gave the best results due to their relatively low cardinalities. Using higher cardinality features with the wide model resulted in worse performance, presumably due to the high sparsity of the resulting transformations.

Feature 1	**Feature 2**	**Cardinality**
User city	Song language	500
User city	User gender	200
User city	Song genre	12,000
User age	Song genre	9,000
User age	Song language	300

Table 1: Cross-product feature interactions used in the wide model and their cardinalities.

For the deep model, we used the following features from the dataset:

1. User gender
2. User city
3. Song language
4. System tab (the section in the app where the listening event occurred)
5. Screen name (the layout that was displayed on the app when the listening event occurred)
6. Source type (entry point in the app where the listening event began, such as the user's library, an online playlist, an album, etc.)
7. User ID
8. Song ID
9. Song genre
10. Song composer
11. Song lyricist

For the wide and deep model, the feature sets used in each of the individual models remain as described above.

4.1. Data Preprocessing

Missing values were imputed with the string "unknown". We consider features 1-6 in the above list to have low cardinality, meaning that each one has less than 50 unique categories. Therefore, they could simply be one hot encoded without encountering any memory constraints. Features 7-10 are considered to have high cardinality since

each feature could possess tens of thousands or millions of unique categories. One hot encoding that many categories cannot be done with the amount of RAM available to us. Therefore, in order to effectively convert these features to an acceptable form for the deep model, we need to embed them into dense, continuous vectors of real numbers, similar to how word embeddings are created in natural language processing. The width of the embedding vector has a significant effect on the embedding process. In this case, we decided to use embedding vectors with approximately $d = \log_2 n$ dimensions (rounded to the nearest integer), where n is the cardinality of the feature. These dense vectors are then concatenated with the one hot encoded vectors from the low cardinality features to form the inputs of the deep model.

Feature	Cardinality	Embedding Dimension
User ID	30,755	15
User gender	3	N/A
User city	21	N/A
Song ID	2,296,320	22
Song language	10	N/A
Song genre	1,046	10
Song composer	329,824	18
Song lyricist	110,926	17
System tab	10	N/A
Screen name	21	N/A
Source type	13	N/A

Table 2: Features used by the deep model and the dimensions of their respective embedding vectors. N/A indicates that a feature was simply one hot encoded.

4.2. Cross-Product Feature Transformations

To create more features for our model, we performed cross-product feature transformations among different features. This operation is defined as

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

In the above equation, c_{ki} equals 1 if the i^{th} feature is part of the k^{th} transformation ϕ_k and 0 otherwise.

As shown in the table below, we can create cross-product feature transformations such as $\boldsymbol{AND}(20 \leq age \leq 30,\ city = Singapore)$, for which the value equals 1 if and only if the user's age is between 20 to 30 and he or she lives in Singapore. This feature can help explain how the co-occurrence of a feature pair correlates with the target label and adds nonlinearity to the linear classifier used in the wide model.

20 ≤ age ≤30	city = Singapore	20 ≤ age <30 & city = Singapore
1	1	1
0	1	0
1	0	0
0	1	0

Table 3: An example of cross-product feature transformations using user demographic information.

4.3. Feature Engineering

We experimented with engineering several new features to use as inputs for the deep model. The most effective one is the registration duration feature. Registration duration is defined as the difference between a user's initial registration date and registration expiration date. This feature represents the number of days a user was registered for KKBox. The addition of this feature resulted in a 4.05% increase in test AUC, making it by far the most effective engineered feature that we tested.

We modified the user age feature that was provided in the original dataset due to the frequent occurrence of improbable and nonsensical values (such as 0 years old or 80 years old). We decided to bucket the age data using the following intervals: [0, 14, 20, 30, 40, 50, 80]. With these intervals, the nonsensical values would be grouped together in the 0-14 bin and the outlier age values would be grouped in the 50-80 column. The addition of this feature resulted in a 0.348% increase in test AUC.

Other engineered features made the model worse. An example of this is song popularity, which is the frequency of a song in the training set divided by the total number of songs in the training set. We created a set of features indicating how many different languages, genres, and songs a user has listened to, as well as a feature for the average length of the songs a user has listened to. All of these features reduced model performance to some degree.

Finally, two engineered features resulted in little to no change in model performance. This first one is the artist repeat rate feature, which is a ratio of how many times each artist's songs were replayed in the training dataset (target = 1) to how many times the artist's songs appear in the dataset. The other feature is the user song history, which is a list of all the songs that each user has listened to in the training set.

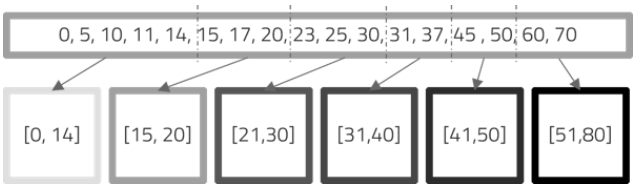

Figure 1: The age bucketing process.

5. Methodology

Prior to implementing the wide and deep model, we implemented and tuned the wide and deep models separately to see how well they could perform.

The wide and deep models use different optimization methods during the training process. In the deep model, the standard Adam optimizer is used. In the wide model, an optimization method called FTRL (Follow the Regularized Leader) is used so that a sparse model can be achieved. FTRL is an online learning optimization method focused on enforcing the sparsity of the weights while allowing the algorithm to converge faster. For regular stochastic gradient descent, the update rule is

$$w_{t+1} = w_t - \eta_t \, g_t$$

where w_t is the updated weight for the t^{th} round, η_t is the learning rate for t^{th} round, and g_t is the gradient of the loss function with respect to w_t in t^{th} round. The FTRL optimizer uses the following update rule instead:

$$w_{t+1} = argmin_w(g_{1:t}w + \frac{1}{2}\sum_{s=1}^{t}\sigma_s\|w - w_s\|_2^2 \; + \; \lambda\|w\|_1)$$

where

$$g_{1:t}w = \sum_{i=1}^{t} g_t$$

This function is an approximation of the function

$$w_{t+1} = argmin_w(g_{1:t}w + t\lambda\|w\|_1 \; + \; r_{1:t}(w))$$

In the FTRL algorithm, there are t copies of the L_1 penalty term $\lambda\|w\|_1$, one for each iteration of the optimization. Due to this, the FTRL algorithm results in a sparser solution.

5.1. Logistic Regression

The wide model is a logistic regression classifier that takes the cross-product feature transformations mentioned in Table 1 as inputs. In addition to this, a separate GBDT model acts as an effective feature selection method for the wide model. This concept will be introduced in more detail in the next section.

5.2. Gradient Boosted Decision Trees

We use a Gradient Boosted Decision Tree (GBDT) model to perform the feature extraction and improve the accuracy of the model. Specifically, in the training stage, we train a GBDT model on the unmodified dataset. This training is performed using Gradient Boosting Machines (GBM) in the LightGBM library.

In each learning iteration, a new tree is created to model the residual of the previous tree. More specifically, GBDT[2] is an additive training (boosting) method. A GBDT model consists of multiple rounds of tree building. Given the input x_i, we define the output of the k^{th} tree as $f^k(x_i)$, and $\hat{y}_i^k$ to be the k^{th} round of additive output. We define $\hat{y}_i^k$ as

$$\hat{y}_i^k = f^k(x_i) + \hat{y}_i^{k-1}$$

We initialize $\hat{y}_i^0 = 0$. Using the mean squared error as the loss function, we then get

$$Loss = \sum_{i=0}^{n}(\hat{y}_i^k - y_i)^2 + R$$

where R is a regularization term.

We then perform a Taylor expansion on the loss function to get

$$Loss = \sum_{i=0}^{n} g_i f^k(x_i) - h_i(f^k(x_i)^2) + R$$

where

$$g_i = \frac{\partial Loss}{\partial \hat{y}_i^{k-1}}, \qquad h_i = \frac{\partial^2 Loss}{\partial (\hat{y}_i^{k-1})^2}$$

[2] https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf

Then, according to this criterion, we can build k trees and use gradients to minimize the loss function.

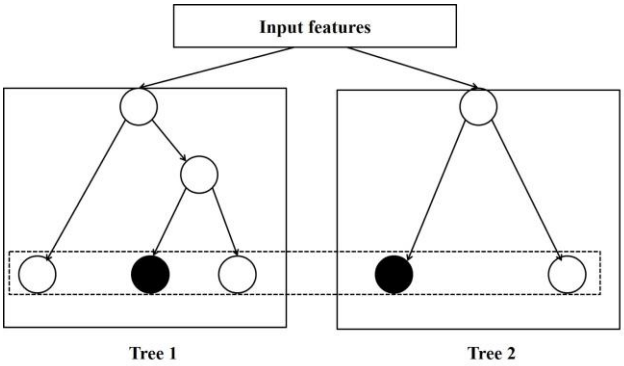

Figure 2: Using predictions from the GBDT model as a feature in the wide model. Here, $k = 2$. Tree 1 contains three leaves and this particular example falls in leaf 2. Tree 2 contains two leaves and this example falls in leaf 1. For each tree, a one hot encoded vector representing which of the leaves the example fell in is produced. Then, these vectors are concatenated and passed along as a feature for the wide model. Here, Tree 1 would result in the vector [0, 1, 0], Tree 2 would result in the vector [1, 0], and the concatenated vector [0, 1, 0, 1, 0] would be passed along to the wide model.

5.3. The Wide Model

In addition to the features mentioned in Table 1, we use the classification results obtained by the GBDT model as a feature in our baseline logistic regression model.

We treat each individual tree as a categorical feature that takes as its value the index of the leaf that an instance falls in. We use 1-of-K encoding for this type of feature. For example, consider the boosted tree model in Figure 2 with two subtrees, where the first subtree has three leaves and the second has two leaves. If an instance falls in leaf 2 in the first subtree and leaf 1 in the second subtree, the overall input to the linear classifier will be the binary vector [0, 1, 0, 1, 0], where the first three entries correspond to the leaves of the first subtree and the last two correspond to those of the second subtree. We can understand GBDT-based transformations as a supervised feature encoding that converts a real-valued vector into a compact binary-valued vector. A traversal from the root node to a leaf node represents a rule on certain features. Fitting a linear classifier on the binary vectors is essentially learning weights for the set of rules.

5.4. The Deep Model

The deep model consists of a fully-connected neural network with three hidden layers containing 1024, 512, and 256 units, respectively. These layers use ReLU activation units. The architecture for the deep model is shown in Figure 3. Categorical features in the dataset are transformed using the feature column API in TensorFlow such that those with low cardinality (containing less than 50 unique categories) are one hot encoded and those with high cardinality are embedded into dense continuous vectors. The model was trained for 150,000 training steps with a minibatch size of 100, a learning rate of 0.001, and Adam as the optimizer.

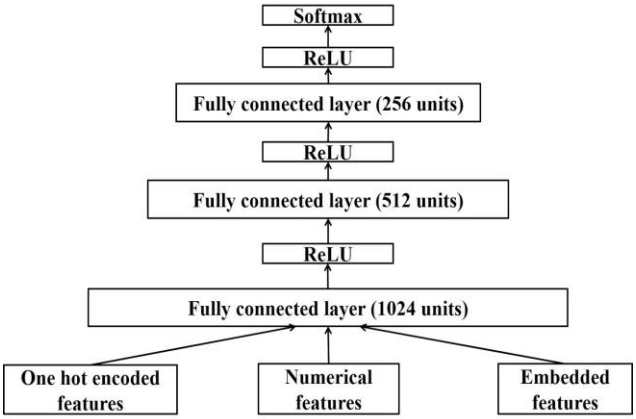

Figure 3: The deep model architecture.

5.5. The Wide & Deep Model

Having implemented the two models separately, we then combine the two to form a wide and deep model. The code for the wide and deep model can be found on Gitlab.[3]

This model is not an ensemble model where they are trained separately and a weighted sum of their predictions are added to form the final prediction. The wide and deep models are jointly trained using the categorical cross entropy loss function

$$L(w) = -\frac{1}{N}\sum_{n=1}^{N}[\, y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)]$$

The outputs of the two models are then concatenated prior to calculating the softmax

$$S(x_n) = \frac{e^{y_i}}{\sum_{i=n}^{N} e^{y_j}}$$

to obtain the predictions for each example. During the

[3] https://gitlab.com/wl2522/KKBox

optimization phase of training, the gradient obtained from the final output is simultaneously propagated through each model.

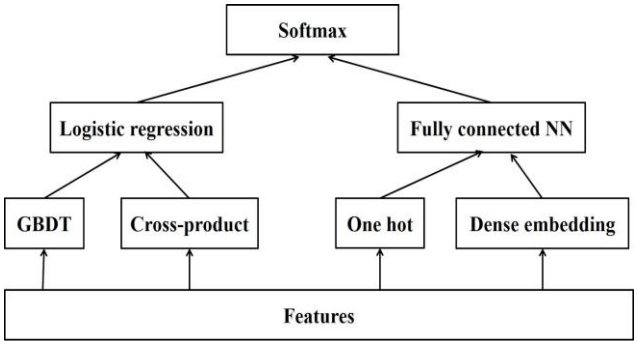

Figure 4: A diagram of the wide and deep model architecture.

6. Results

The results achieved by the wide and deep model as well as each of the separate component models are presented in Table 4. A validation set was created by shuffling the training set and performing an 80/20 split.

Model	Training AUC	Validation AUC	Test AUC
Wide & Deep Model[4]	0.73340	0.7294	0.64235
Deep[5]	0.78946	0.77203	0.66347
Wide[6]	0.76548	0.73201	0.65911
GBDT[7]	0.82107	0.80306	**0.68711**
Logistic Regression[8]	0.70128	0.69238	0.62316
Uniformly Random Prediction	0.49994	0.49982	0.50040
All Majority Class Prediction	0.50352	0.50351	0.50000

Table 4: Performance for each of the models, with the highest achieved test AUC in bold. Test AUC is calculated on the Kaggle leaderboard test set. Two baseline naïve prediction strategies are also shown at the bottom.

ROC AUC obtained using two different naïve prediction strategies are also shown below for comparison. The first naïve strategy is to find the label with the highest frequency in the training set (1 in this case). Then, every example is predicted to be that label. The second naïve strategy is to make a prediction for each example by drawing a random sample from the uniform distribution $U(0, 1)$.

With less than eight million examples to train on, sparsity in the data can negatively influence the model to a large degree. From our observations, adding sparse features to the wide and deep model results in unsatisfactory results. Cross features, especially those with great sparsity, are supposed to be able to remember complicated combinations. However, with so few occurrences of such combinations, the memorization becomes quite difficult. For example, crossing the genre and age bucket features would result in 9,000 different categories, with a great majority of them not appearing in the dataset. Aside from these problems, the dataset also lacks timestamps indicating when users played each song, which could be a critical feature. With this information, we could find which users are active on KKBox based on their last recorded timestamp. Popular songs could also be determined using the temporal information provided by the timestamps.

Furthermore, the data is quite well-balanced in terms of the number of positive and negative samples (3,714,656 positive examples vs. 3,662,762 negative ones). However, in real life, that is rarely the case. The even distribution of the data indicates that this data was filtered to remove the vast majority of the negative examples where the user did not replay the song. This is because one would expect that users would dislike many more songs than they like. With this kind of filtering, a lot of data becomes missing, which harms the wide and deep model's ability to learn from the data. We believe that for these reasons, the models which used the cross-product feature transformations (logistic regression, wide, wide and deep models) were not as successful as the models that did not use them (GBDT and deep models).

We believe that the deep model was more successful in this prediction task due to the fact that it could avoid the sparsity problem by embedding the high cardinality features into dense continuous vectors. In addition, the two engineered features that we developed helped to improve the model's performance.

7. Conclusion

In this paper, we proposed using a deep and wide model in the KKBox Music Recommendation Challenge. During the contest, we were surprised to find that a simple ensemble tree method such as our GBDT model worked better than the complicated wide and deep model. We found that the sparse cross-product feature transformations that we used were not able to learn from the relatively small dataset. Therefore, we feel that the wide and deep model is not appropriate for this particular Kaggle challenge.

[4] Described in Section 5.5
[5] Described in Section 5.4
[6] Described in Section 5.3
[7] Described in Section 5.2
[8] Described in Section 5.1

References

1. Y. Koren, R. Bell, and C. Volinsky, “Matrix Factorization Techniques for Recommender Systems”, Computer, vol. 42, pp. 30–37, Aug. 2009.
2. H. C. Chen and A. L. P. Chen, “A Music Recommendation System Based on Music Data Grouping and User Interests,” in Proceedings of the Tenth International Conference on Information and Knowledge Management, CIKM ’01, (New York, NY, USA), pp. 231–238, ACM, 2001.
3. B. Zhang, J. Shen, Q. Xiang, and Y. Wang, “CompositeMap: a Novel Framework for Music Similarity Measure,” SIGIR, 2009.
4. X. Wang, D. Rosenblum, and Y. Wang, “Context-Aware Mobile Music Recommendation for Daily Activities,”in ACM Multimedia 2012, Oct. 2012.
5. M. Schedl and D. Schnitzer, “Location-Aware Music Artist Recommendation,” in MultiMedia Modeling (C. Gurrin, F. Hopfgartner, W. Hurst, H. Johansen, H. Lee, and N. O’Connor, eds.), vol. 8326 of Lecture Notes in Computer Science, pp. 205–213, Springer International Publishing, 2014.
6. I. Porteous, A. Asuncion, and M. Welling, “Bayesian Matrix Factorization with Side Information and Dirichlet Process Mixtures,” in Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10).
7. L. M. de Campos, J. M. Ferna´ndez-Luna, J. F. Huete, and M. A. Rueda-Morales, “Combining content-based and collaborative recommendations: A hybrid approach based on Bayesian networks,” International Journal of Approximate Reasoning, vol. 51, pp. 785–799, Sept. 2010.
8. P. Hamel and D. Eck, “Learning features from music audio with deep belief networks,” in 11th International Society for Music Information Retrieval Conference, 2010.
9. E. M. Schmidt and Y. E. Kim, “Learning rhythm and melody features with deep belief networks,” in ISMIR, 2013.
10. A. van den Oord, S. Dieleman, and B. Schrauwen, “Deep content-based music recommendation,” in NIPS, 2013.
11. H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al. Wide & deep learning for recommender systems. arXiv preprint arXiv:1606.07792, 2016.
12. S. Rendle. Factorization machines with libFM. ACM Trans. Intell. Syst. Technol., 3(3):57:1–57:22, May 2012.
13. T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. H. Cernocky. Strategies for training large scale neural network language models. In IEEE Automatic Speech Recognition & Understanding Workshop, 2011.
14. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. Proc. IEEE Conference on Computer Vision and Pattern Recognition, 2016.