On the Diversity and Explainability of Recommender Systems: A Practical Framework for Enterprise App Recommendation

Wenzhuo Yang
Salesforce Research
Singapore
wenzhuo.yang@salesforce.com

Jia Li
Salesforce Research
Palo Alto, CA, USA
jia.li@salesforce.com

Chenxi Li
Salesforce Sales Intelligence
San Francisco, CA, USA
chenxi.li@salesforce.com

Latrice Barnett
Salesforce
San Francisco, CA, USA
lbarnett@salesforce.com

Markus Anderle
Salesforce Sales Intelligence
San Francisco, CA, USA
markus.anderle@salesforce.com

Simo Arajarvi
Salesforce Sales Intelligence
San Francisco, CA, USA
sarajarvi@salesforce.com

Harshavardhan Utharavalli
Salesforce Sales Intelligence
Secunderabad, Telangana, India
hutharavalli@salesforce.com

Caiming Xiong
Salesforce Research
Palo Alto, CA, USA
cxiong@salesforce.com

Steven HOI
Salesforce Research
Singapore
shoi@salesforce.com

ABSTRACT

This paper introduces an enterprise app recommendation problem with a new "to-business" use case, which aims to assist a sales team acting as the bridge connecting the applications and developers with the customers who apply these apps to solve their business problems. Our recommender system is an assistant to the sales team, helping recommend relevant apps to the customers for their businesses and increasing the likelihood of improving sales revenue. Besides recommendation accuracy, recommendation diversity and explainability are even more crucial since they provide more exposure opportunities for app developers and improve the transparency and trustworthiness of the recommender system. To allow the sales team to explore unpopular but relevant apps and understand why such apps are recommended, we propose a novel framework for improving aggregate recommendation diversity and generating recommendation explanations, which supports a wide variety of models for improving recommendation accuracy. The model in our framework is simple yet effective, which can be trained in an end-to-end manner and deployed as a recommendation service easily. Furthermore, our framework can also apply to other generic recommender systems for improving diversity and generating explanations. Experiments on public and private datasets demonstrate the effectiveness of our framework and solution.

KEYWORDS

App recommendation; Recommendation diversity; Explainability

CIKM '21, November 1–5, 2021, Virtual Event, Australia.

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3481940

ACM Reference Format:
Wenzhuo Yang, Jia Li, Chenxi Li, Latrice Barnett, Markus Anderle, Simo Arajarvi, Harshavardhan Utharavalli, Caiming Xiong, and Steven HOI. 2021. On the Diversity and Explainability of Recommender Systems: A Practical Framework for Enterprise App Recommendation. In *Proceedings of the 30th ACM Int'l Conf. on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3481940

1 INTRODUCTION

Recommender systems are essential components to various commercial applications, e.g., online advertising [37], online retail [23], video and music services [14], mobile and cloud app stores [8], etc. Given user profiles and contextual information, the goal of many recommender systems especially for consumer markets is to find relevant items and then rank the items for optimizing some metrics such as clicks or purchases. In this paper, we introduce an enterprise app recommendation problem and aim to build a practical recommender system for the sales team. The objective of the task is to help the sales teams recommend meaningful and relevant apps to their customers, which can reduce their manual selection work and improve the app installations from customers.

Our enterprise app recommendation task involves multiple stakeholders including the sales teams, app vendors/developers, and enterprise customers. App developers develop enterprise applications on Salesforce cloud platform for solving some specific business problems and bringing the platform's benefits to real business use cases. The sales teams serve as the bridge connecting the applications and developers with the customers who download and apply these apps to solve their own business problems. The sales teams analyze customers' needs and advise them on how to better meet their goals. To achieve this, they recommend specific apps to the customers based on the customers' goals and behaviors. Figure 1 shows some real-world enterprise apps for recommendation. Our recommender system is designed to assist them in finding such specific apps, allowing them to interact with the system and

obtain more information, e.g., controlling recommendation diversity for exploring unpopular but relevant apps, and understanding why such apps are recommended. Recommendation diversity is important because it provides more exposure opportunities for app vendors/developers and helps them understand customer's requirements better. Explainability is also important since it improves the transparency and trustworthiness of our recommender system and helps them analyze why the recommended apps are relevant and valuable to the customers.

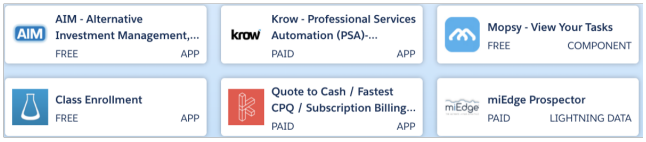


Figure 1: Some enterprise apps and tools from the Salesforce Enterprise cloud marketplace.

Figure 2 presents the overview of our app recommender system and how the sales teams interact with our apps recommendation service. This service is designed to improve app installations by leveraging meaningful information from customer profiles and purchase/installation history. For a certain customer, in order to discover which apps the customer prefers, the sales teams send the user/customer id and the diversity parameter to the apps recommendation engine where the diversity parameter controls the exploration for novel apps. Then the apps recommendation engine submits this request to the model server which generates the top k recommended apps and the corresponding explanations. After the sales teams receive the response from the apps recommendation engine, they can analyze the results with the provided explanations and their experience to determine which apps should be finally recommended to the customer.

As for explanations, one straightforward solution is to take a particular model trained only for recommendation accuracy and apply a certain model-agnostic method such as LIME [27] to generate explanations. The limitations of such solution include: 1) It requires additional training procedures for explanation models which makes the pipeline complicated, and 2) it induces large response latency if LIME [27] or integrated gradients based methods [30] is applied, e.g., LIME requires to train a local linear model for each user/item pair. In this paper, we propose a novel diversity-explanation framework for allowing users to control the aggregate diversity and obtain explainable results efficiently. To develop such framework, we consider three objectives of recommendation: 1) Accuracy: The overall accuracy performance of the enterprise app recommender system should not be sacrificed too much while addressing the diversity and explainability issues, 2) Diversity: The recommended apps should be idiosyncratic or personalized, which can help the sales teams to discover new apps instead of always selling popular apps. Increasing the diversity also provides more exposure opportunities for app providers or developers, and 3) Explainability: Besides presenting the recommended apps to the sales teams, it should also explain why these apps are recommended. The explanation helps to improve the transparency, persuasiveness and trustworthiness of the system, encouraging them to make better sales decisions upon the recommendation results.

Most previous approaches focus on one or two aspects, e.g., models for accuracy only or for both accuracy and diversity. The key question is: for this apps recommendation problem, how to design an algorithm in a unified manner for easy training and deployment that allows to control recommendation diversity, generate explanations and maintain a certain level of accuracy? To answer this question, we need to address the following issues:

(1) How to increase the recommendation diversity? Most existing approaches maximize the objective for accuracy without directly considering diversity.
(2) How to make a trade-off between diversity and accuracy? High accuracy may often be obtained by safely recommending to users the most popular items, which can clearly lead to reduced diversity. And conversely, higher diversity can be achieved by trying to recommend highly personalized items, which often may not have enough data and thus may hurt accuracy. Our method provides a way for the sales team to control this trade-off.
(3) How to generate explanations without knowing the details of the underlying recommendation model? And what types of explanation to provide to the sales team? Since we may frequently update the recommendation model, the explanation module should be suitable for most types of models and generate reasonable results for the sales team to evaluate recommendation results.

In this paper, we propose **a novel framework for improving aggregate recommendation diversity and generating recommendation explanation, which supports a wide variety of models for recommendation accuracy**. The main contributions of the paper include:

(1) The proposed model in our framework is simple yet effective, which can be trained in an end-to-end manner and deployed as a recommendation service easily.
(2) Our approach improves aggregate diversity significantly without reducing recommendation accuracy much and generates reliable explanations that are comparable with the results obtained by LIME [27].
(3) For the enterprise app recommendation, we design a new model for recommendation accuracy by utilizing various kinds of features, e.g., single/multi-group categorical, continuous, text and installation history, which achieves highly competitive results compared with widely-applied models Wide&Deep [8] and DIN [37] in industrial recommender systems.
(4) Our framework can also be applied to generic recommender systems for any recommendation model. We demonstrate it using MovieLens [16] and our apps recommendation dataset.

2 RELATED WORK

In recent years, numerous new algorithms have been proposed to improve recommendation accuracy, e.g., [8, 9, 15, 19, 26, 28, 29, 32, 37], emerging from traditional machine learning to deep learning inspired by the success of deep learning in computer vision [22] and natural language processing [11]. Among them, the wide-and-deep

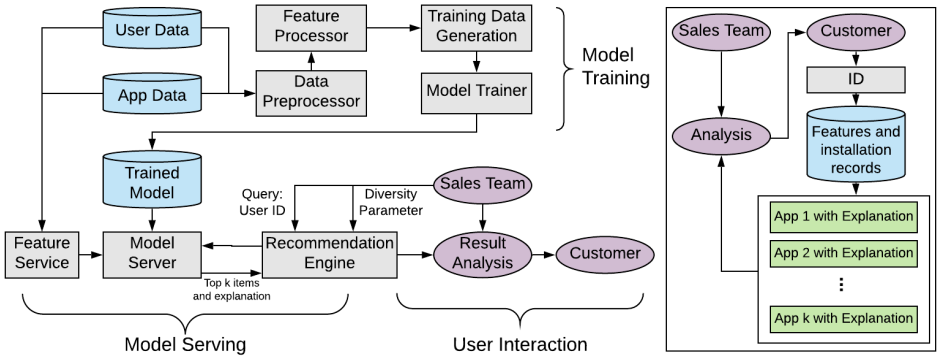


Figure 2: The overview of the proposed Enterprise App Recommender System.

model [8] combines the benefits of memorization and generalization for recommender systems by jointly training the linear model with a DNN model. DeepFM [15] extends factorization machines with a DNN model to represent high-order feature interactions. Deep Interest Network [37] learns the representation of user interests from historical behaviors with a local activation module to improve the expressive ability.

Besides accuracy, recommendation diversity is another dimension of the quality of recommendations. The importance of diverse recommendations has been discussed in several studies, e.g., [1, 6, 18, 20, 33–35, 38]. These studies show that more diverse recommendations create more opportunities for users to get recommended personalized items or relevant but unpopular items. For example, Adomavicius *et al.* [1] explored different item re-ranking techniques that can generate recommendations with substantially higher aggregate diversity across all users while maintaining comparable levels of accuracy. Wilhelm *et al.* [34] proposed a statistical model of diversity based on determinantal point processes and achieved long-term increases in user engagement. Some other studies [2, 3] show that recommending "long-tail" type of items are beneficial for some business models such as online bookstores.

Explainable recommendation is also an important topic in recommender systems beyond accuracy and diversity, which clarifies why such items are recommended. Generating explanations along with recommended items can improve the transparency, trustworthiness, and user satisfaction of the recommender systems. It can also help model developers to diagnose and refine recommendation algorithms. There are two directions for designing explainable recommendation algorithms. One direction focuses on developing intrinsic explainable models such as factorization-based, topic modeling and deep learning methods, e.g., [4, 5, 7, 12, 13, 24, 31, 36]. The other one focuses on the explainability by treating recommendation models as black boxes and developing separate models for explanation, which is also called the model-agnostic or post-hoc methods, e.g., [25, 27, 30]. For example, Ribeiro *et al.* proposed LIME [27] that can explain the predictions of any classifier in an interpretable and faithful manner and be used in click-through rate (CTR) prediction tasks as well.

3 METHOD

To tackle the enterprise app recommendation task, we present a unified recommendation framework, which is capable of dealing with recommendation accuracy, diversity, and explanation simultaneously in an end-to-end trainable solution.

3.1 Feature Representation

The data in our recommendation task contains categorical features, multi-group categorical features, and continuous-valued features. The categorical features form the user profiles, e.g., "country = USA", "market = ESMB" and "industry = {healthcare, life sciences}", while the continuous features encode the user behavior, e.g., cloud usage and application deployment status. For the items (or apps), besides the item IDs, we also utilize the item names, e.g., "Slack" and "LinkedIn Sales Navigator". Each of these categorical features and words is converted into a low-dimensional and dense real-valued embedding vector. The continuous-valued features are concatenated together, forming one dense feature vector. Given user i and item j, let $\mathbf{x}_{ij}$ be the input features (e.g., categorical features and continuous-valued features), and y_{ij} be the user action label ($y = 1$ and $y = 0$ are "installed" and "not installed", respectively).

3.2 Framework Overview

As shown in Figure 3, our framework consists of two parts, i.e., "relevance model" for accuracy and "Diversity and Explanation model" (or "DAE" for short) for diversity and explainability. We designed a specific relevance model for our enterprise app recommendation problem. However, the relevance model in our framework can be replaced with any state-of-the-art approach for a different recommendation task. The objective of the relevance model is to learn the probability $P(y|\mathbf{x})$ of an action label y given the input features $\mathbf{x}$, which can be formulated as minimizing the negative log-likelihood function. For the DAE models, the goal is to provide a convenient way to control aggregate diversity and generate recommendation explanations, which can be achieved by maximizing a diversity objective function discussed below.

Suppose that there are n users and m items. Given user i and item j, suppose that the predicted score of the relevance model is $p(i, j)$ and the output of the DAE model is a distribution $\mathcal{D}(i, j)$ parameterized by its output $g(i, j)$. Let $z(i, j) = p(i, j) + q(i, j)$ be

the combination of their outputs where $q(i,j)$ is drawn from distribution $\mathcal{D}(i,j)$. Let $\mathcal{Q}(i)$ be the distribution of the random vector $\mathbf{z}(i) = (z(i,1), z(i,2), \cdots, z(i,m))^\top$, then the diversity objective function can be defined as the negative KL divergence between $\mathcal{Q}(i)$ and a specific predefined distribution $\mathcal{P}$, e.g., $-D_{KL}(\mathcal{P}||\mathcal{Q}(i))$. For example, if $\mathcal{P}$ is the Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ with $\boldsymbol{\mu} = 0.5, \Sigma = \sigma\mathbf{I}$ where σ is a constant and $\mathbf{I}$ is the identity matrix, then maximizing this diversity objective function makes $\mathcal{Q}(i)$ close to $\mathcal{P}$ so that the distribution of $z(i,j)$ is close to $\mathcal{N}(0.5, \sigma)$, leading to more diverse recommendation results when we recommend items to user i by ranking the scores drawn from distribution $\mathcal{Q}(i)$. To control recommendation diversity in the prediction step, we introduce a weight parameter $w \in [0,1]$ so that the predicted rating of user i for item j is $p(i,j) + w * q(i,j)$ where $q(i,j) \sim \mathcal{D}(i,j)$, providing us ability to explore novel items by tuning w.

The DAE model provides us more flexibility to satisfy other requirements such as generating recommendation explanations. The key idea is that we can decouple different kinds of features from the input features to the relevance model, by designing their specific DAE models, as shown in Figure 3. For example, given user i and item j, one model takes the user categorical features only as its inputs, outputting the distribution $\mathcal{D}(i,j)$ for diversity control and generating the corresponding feature-level explanation at the same time. We will discuss how to design such DAE models in detail in Section 3.4. With K DAE models for different purposes, the total loss function is given by

$$L = \frac{1}{|\mathcal{S}|} \sum_{(i,j)\in\mathcal{S}} \text{Accuracy\_loss}(p(i,j), y_{ij}) + \frac{1}{nK} \sum_{k=1}^{K} \sum_{i\in\mathcal{U}} D_{KL}(\mathcal{P}||\mathcal{Q}_k(i)), \tag{1}$$

where $\mathcal{S}$ is the training dataset, n is the number of users and $\mathcal{P}$ is a predefined distribution. The accuracy loss is the log loss function. The relevance model and DAE models can be trained together in an end-to-end manner.

3.3 Relevance Model

Our framework allows plugging in a wide variety of recommendation models for rating prediction, e.g., deep learning models [8, 37] and traditional machine learning models [19]. For this project, in order to satisfy the requirements and utilizing different features effectively, we design a specific neural network for recommendation accuracy by modeling user and item representations separately as shown in Figure 4.

3.3.1 User vector. We utilize categorical features, multi-group categorical features, continuous-valued features as well as the user installation history to learn user representation. The model is as shown in Figure 4a. The single-group categorical feature embeddings are concatenated together while the average pooling is applied for the multi-group categorical feature embeddings. For user installation history, an attention-based module is applied to learn the similarity between the candidate item and the installed items, which is similar to the DIN model [37]. The input of the attention module is the concatenation of the candidate item, the installed item and their element-wise product, followed by MLP with the sigmoid function as its output. The history representation is the weighted average pooling of these item embeddings based on the attention weights.

3.3.2 Item vector. The model structure for item vectors is shown in Figure 4b whose inputs contain item IDs and item names. For learning a better representation, we apply a linear classifier to highlight the keywords in an item name, e.g., larger weights on "Sales" or "Dashboard" and smaller weights on "and" or "with". Suppose that the item name has n words with embeddings $\{\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_n\}$ and denote $\boldsymbol{\beta}$ the linear classifier, then the importance weight for word i is given by $w_i = \frac{exp(\boldsymbol{\beta}^\top \mathbf{e}_i)}{\sum_{j=1}^{n} exp(\boldsymbol{\beta}^\top \mathbf{e}_j)}$ and the representation for the item name is the weighted sum pooling according to weight $\mathbf{w}$, i.e., $\mathbf{e} = \sum_{i=1}^{n} w_i \mathbf{e}_i$. Our experiments show that this module can indeed improve recommendation accuracy. The predicted score is then modeled by concatenating user and item vectors followed by MLP with the sigmoid output.

For item text features, we didn't apply BERT [10] for modeling their representations due to the concern of the computational cost, i.e., BERT will increase the response time a lot in our deployed app recommender system.

3.4 Diversity and Explanation (DAE) Model

For designing DAE models, we need to select distributions $\mathcal{P}$ and $\mathcal{D}$ so that the loss function (1) can be easily optimized and the corresponding DAE models can generate reasonable explanations. The predefined distribution $\mathcal{P}$ used in this paper is the Gaussian distribution $\mathcal{N}(\boldsymbol{\nu}, \gamma^2\mathbf{I})$ where $\boldsymbol{\nu} = 0.5$ and γ is a constant. For distribution $\mathcal{D}$, we suppose that $\mathcal{D}(i,j)$ given user i and item j is a Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$, where mean μ is a function of both user i and item j while σ depends on item j only. The reason why σ only depends on item features is that variance σ^2 models intrinsic "popular" properties for items. Intuitively, unpopular items have a relatively low number of interactions among most of the users, leading to small variances. On the contrary, popular items have various ratings among users according to different user tastes, which induces large variances. With $\mathcal{P}$ and $\mathcal{D}$ defined above, the loss function (1) can be reformulated as

$$L = \frac{1}{|\mathcal{S}|} \sum_{(i,j)\in\mathcal{S}} \Big[\text{log\_loss}\left(p(i,j), y_{ij}\right) + \frac{1}{K} \sum_{k=1}^{K} D_{KL}\left(\mathcal{N}(\nu, \gamma^2) \Big\| \mathcal{N}_k(p(i,j) + \mu(i,j), \sigma(j)^2)\right) \Big], \tag{2}$$

where $p(i,j)$ is the output of the relevance model.

One of the requirements is that the model should allow the sales team to control recommendation diversity in real-time for exploring new apps to sell. Our framework provides a convenient way to satisfy this requirement. In prediction, the predicted rating of user i for item j is given by

$$r(i,j) = p(i,j) + w * q(i,j),\ q(i,j) \sim \mathcal{N}(\mu(i,j), \sigma(j)^2), \tag{3}$$

where $w \in [0,1]$ controls the trade-off between diversity and offline accuracy, e.g., larger w means doing more exploration for new items. Distribution $\mathcal{N}(\mu(i,j), \sigma(j)^2)$ is a mixture of $\mathcal{N}_k(\mu(i,j), \sigma(j)^2)$, e.g., average or one of $\mathcal{N}_k(\mu(i,j), \sigma(j)^2)$. In practice, we use the average mixture of $\mathcal{N}_k(\mu(i,j), \sigma(j)^2)$.

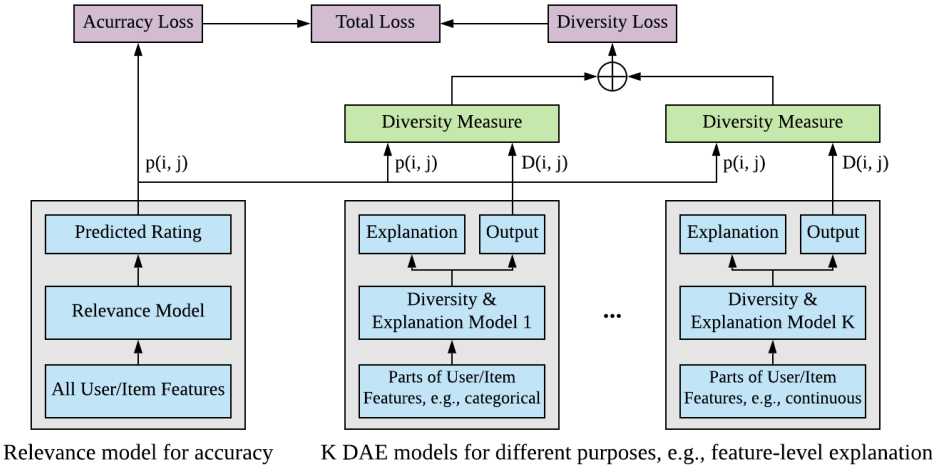
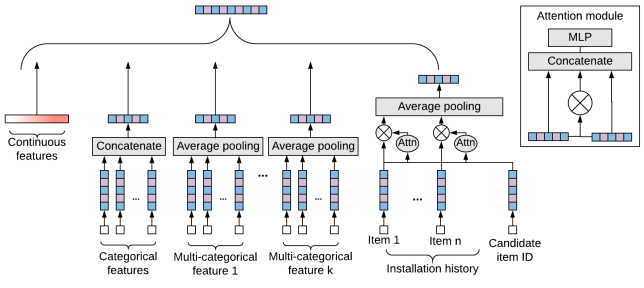
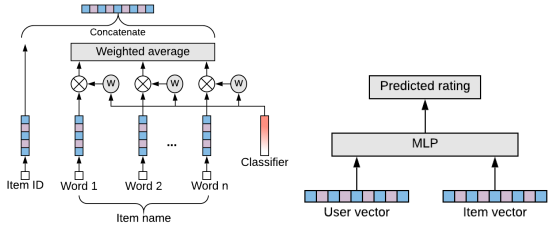


Figure 3: The overall structure of our unified recommendation framework.



(a) The user vector.



(b) The item vector. **(c) The output layer.**

Figure 4: The prediction model for recommending apps.

Another requirement is that our model should explain why recommending those apps instead of only presenting the recommendation results. Now let's discuss why our model can also be used to generate explanations. For demonstration, suppose that $\mathcal{D}(i,j) = \mu(i,j)$ instead of a Gaussian distribution, then the diversity term in Equation (2) reduces to $(p(i,j)+\mu(i,j)-0.5)^2$, meaning that the DAE models try to approximate the predictions of the relevance model, i.e., an estimate of $0.5 - p(i,j)$. Therefore, it can be viewed as a model-agnostic explainable recommendation approach by training a simpler model for explanation. Note that in practice the DAE models need to know what kinds of features the relevance model utilizes for generating proper explanations.

In this paper, we consider the feature-based explanations by highlighting important features or relevant installed apps so that the sales team can judge whether the recommended apps are reasonable or not. We design three types of DAE models, i.e., the hot DAE model for extracting popular items, the categorical DAE model for categorical features, multi-group categorical features and user installation history, and the continuous DAE model for continuous-valued features, as shown in Figure 5.

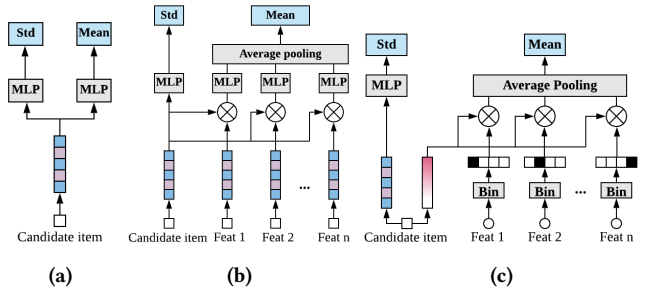


Figure 5: Three types of the DAE models. (a) Hot items. (b) For categorical features. (c) For continuous-valued features. Model (b) can also be used for user installation history, providing item-based explanation.

For the hot DAE model, i.e., model (a) in Figure 5a, the input only contains the candidate item ID. Let $\mathbf{e}_c$ be the candidate item embedding, then the mean μ and the std σ are defined by

$$\mu = 0.5 - \text{sigmoid}(\text{MLP}_m(\mathbf{e}_c)),\ \sigma = \text{sigmoid}\left(\text{MLP}_s(\mathbf{e}_c)\right),$$

where MLP_m and MLP_s are two different MLPs. For convenience, μ has a term "0.5" because we assume the mean of the predefined distribution $\mathcal{P}$ is 0.5. Given an item, we can compute its popularity score $s_{hot} = \text{sigmoid}(\text{MLP}_m(\mathbf{e}_c))$. By sorting these popularity scores, we can obtain a list of hot items. Then the explanation provided by this model is "item X is recommended because it is popular" if item X is in the hot item list.

For the categorical DAE model, i.e., model (b) in Figure 5b, the inputs contain the candidate item ID and the user categorical features (user installation history can also be viewed as categorical

features). Let $\mathbf{e}_c$ be the candidate item's embedding and $\mathbf{e}_i$ be the embedding of the ith feature, then the mean μ and the std σ are given by

$$\mu = 0.5-\text{sigmoid}\left(\frac{1}{n}\sum_{i=1}^{n}\text{MLP}_m(\mathbf{e}_c \odot \mathbf{e}_i)\right),\ \sigma = \text{sigmoid}\left(\text{MLP}_s(\mathbf{e}_c)\right),$$

where $\odot$ is the element-wise product, MLP_m and MLP_s are two different MLPs. Score $s_{cate} = \text{MLP}_m(\mathbf{e}_c \odot \mathbf{e}_i)$ is the ith feature's importance weight which can be used for explanation, namely, the features are sorted by the score s_{cate} and the top k features are selected for explanation. Then the explanation will be "item X is recommended because of features A, B, etc.", for example, "app (RingLead Field Trip – Discover Unused Fields and Analyze Data Quality) is recommended because 1) the customer is in the USA, 2) the market segment is ESMB and 3) it's on the sales and custom cloud". Based on this kind of explanation, the sales team can verify if the recommended apps are reasonable or not.

For the continuous DAE model, i.e., model (c) in Figure 5c, the continuous-valued features are discretized and converted into one-hot vectors. These one-hot vectors are then concatenated and form a $n \times B$ multi-hot vector $\mathbf{v}$ where n is the number of continuous-valued features and B is the number of bins. The candidate item is mapped to two vectors, i.e., one is for modeling σ and the other is the regression coefficient $\boldsymbol{\alpha} \in \mathbf{R}^{nB}$ for modeling μ. μ and σ are given by the following equations

$$\mu = 0.5 - \text{sigmoid}\left(\frac{1}{n}\boldsymbol{\alpha}^\top \mathbf{v}\right),\ \sigma = \text{sigmoid}\left(\text{MLP}(\mathbf{e}_c)\right).$$

The coefficient $\boldsymbol{\alpha}$ can be used for feature-based explanation, i.e., the feature importance weight for the ith continuous-valued feature is $s_{cont} = \boldsymbol{\alpha}(i,k)$ where k is the index of the bin that this feature belongs to. By sorting the scores s_{cont}, we can get similar feature-based explanations for the continuous-valued features as discussed above.

4 EXPERIMENTS

In this section, we evaluate the performance of our proposed framework on both private and public datasets. The main dataset in our experiments is a private enterprise app recommendation dataset, and we use it to compare with other approaches in terms of accuracy, diversity, and explainability. Another public dataset is the popular MovieLens dataset [16], which allows to compare with different existing recommender systems.

4.1 Datasets

MovieLens 1M Dataset. The MovieLens 1M data contains 6040 users, 3883 movies and 1m ratings. We transform it into a binary classification dataset to make it suitable for the recommendation task with implicit feedback. Original user ratings of the movies are ranging from 0 to 5. The samples with a rating of 4 and 5 are labeled as positive examples. We segment the data into training and test dataset based on user rating history in a leave-one-out way. For each user, the movies she has rated are sorted by the timestamp in ascending order, then the last movie is put into the test dataset and the rest are put into the training dataset.

Enterprise App Recommendation Dataset. Our private app recommendation dataset sampled from the whole data on Salesforce cloud platform contains 170k users, 7k apps and 1.4m installation records. The user information includes three types of features: 1) categorical features, e.g., country, city, market segment, 2) multi-group categorical features, e.g., industry, topics, and 3) continuous-valued features, e.g., cloud usage. The app information only includes the app names and app IDs. We take the installation records from 2019-01 to 2019-12 as the training dataset and the records in 2020-01 and 2020-02 as the test dataset. This dataset is used for offline evaluation.

4.2 Experimental Setups

For MovieLens, we consider the setting for demonstration purpose where the relevance model is a feedforward neural network that takes user/movie IDs and user categorical features as its inputs and has four hidden layers with sizes [64, 32, 16, 8], and the DAE models are the ones shown in Figure 5a and Figure 5b which are used to improve diversity and provide explanations like "X is recommended to user A because either A has features P and Q or X is a hot movie".

For the app recommendation dataset, the relevance model we applied here is the one discussed in Section 3.3, where the user vector and item vector are concatenated together and feed to an MLP layer (with sizes [50, 50]) to compute the predicted rating. The attention module has one dense layer to compute the attention weights. The activation functions used in our model are PReLU [17]. The embedding sizes for words, categorical features and item IDs are 16. We consider two simple yet efficient DAE models shown in Figures 5b and 5c for improving diversity and generating feature-level explanations. The MLPs in the models have one dense layer, the number of bins is 5 and the embedding size is 8.

We use Adam [21] as the optimizer with learning rate 1e-3 and batch size 512. The relevance model and the DAE models are trained together by minimizing the loss function shown in Equation (2). For each user, the candidate items in prediction are all the items except those that have been already installed, and the top 10 predicted items are recommended. For diversity, we consider the aggregate diversity of recommendations across all users.

4.3 Experimental Results

4.3.1 Accuracy. We first compare our relevance model with three methods widely applied in industrial recommender systems, i.e., logistic regression (LR), wide and deep model [8] and DIN model [37], on the app recommendation dataset. The metrics for accuracy are hit ratio and NDCG. Table 1 shows the comparison results,

Table 1: Comparison on the app recommendation dataset.

Metric	LR	Wide & Deep	DIN	Ours
Hit Ratio @ 6	0.2434	0.2912	0.2936	**0.2974**
NDCG @ 6	0.2385	0.2563	**0.2671**	0.2669
Hit Ratio @ 10	0.3082	0.3518	0.3530	**0.3567**
NDCG @ 10	0.2598	0.2765	0.2856	**0.2868**

where "@ k" means k items are recommended for each user. Obviously, the deep learning models outperform the LR model. DIN and

our model obtain better performance than the wide & deep model, which demonstrates the importance of utilizing user installation history in our task. Our model performs better than DIN in terms of hit ratio and NDCG @ 10 in this problem. In comparison with DIN, our model has a special module for learning item representation and the experimental results verified its effectiveness.

4.3.2 Diversity. The next experiment evaluates the ability to control diversity with our framework. Recall that the requirement is allowing the sales team to tune the recommendation diversity for exploring new apps to sell after the whole model is trained. Therefore, we compare our approach with different re-ranking methods, e.g., [1, 18]. The aggregate diversity is measured by two metrics. One is the number of unique recommended items among all the users. The other one is the average distance between the recommendation results of two different users. More formally, let $r(i)$ be the set of the recommended items for user i (suppose that $|r(i)| = k$) and $\mathcal{U}$ be the set of user pair (i, j), then the metric is defined by:

$$avg\_dist = \frac{1}{2k|\mathcal{U}|} \sum_{(i,j)\in\mathcal{U}} |r(i) \cup r(j)| - |r(i) \cap r(j)|.$$

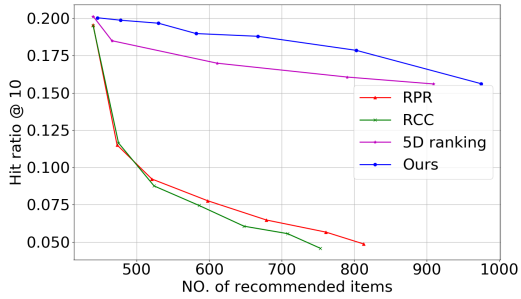
Since $r(i) = k$, avg_dist is in $[0, 1]$, e.g., $avg\_dist = 1$ if $r(i) \cap r(j) = \emptyset$ for all i, j.

The re-ranking methods that we compare with our approach are as follows. Given user i and item j, let $p(i, j)$ be the corresponding predicted rating and $rank(i, j)$ be the rank of item j for user i. For a certain threshold T, the re-ranking function is defined by

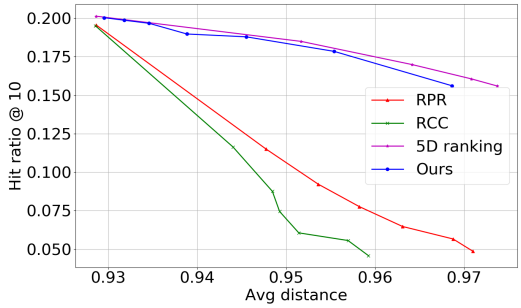
$$rerank(i, j) = \begin{cases} h(i, j) & \text{if } p(i, j) > T; \\ rank(i, j) + z(i) & \text{otherwise,} \end{cases}$$

where $z(i) = \max_{j|p(i,j)>T} h(i, j)$. With different function $h(i, j)$, one can obtain different approaches for improving diversity [1, 18]. We here consider three functions: 1) reverse predicted rating (RPR), i.e., items are sorted based on the predicted ratings from lowest to highest, 2) reverse click/installation counts (RCC), i.e., items are sorted based on the total click/installation times from lowest to highest, and 3) 5D ranking, i.e., Ho *et al.* [18] proposed an approach which improves diversity by considering "recommendations" as resources to be allocated to the items and computing a score named 5D-score for each user/item pair. The diversity is controlled by tuning threshold T. Our method controls diversity by w in Equation (3).

Figure 6 and Figure 7 provides the comparison between our approach and the re-ranking methods mentioned above. The x-axis is the number of the unique recommended items or the average distance between the recommendation results of two users. The y-axis is the hit ratio. These figures show the trade-off between diversity and offline accuracy. For MovieLens, our method performs much better than RPR and RCC, i.e., our method can recommend about 800 items with hit ratio 0.18, while the hit ratios of RPR and RCC drop to 0.06 if 800 items are recommended. Our method beats 5D ranking in terms of the number of the recommended items and has a similar performance with it in terms of the average distance. Note that the 5D ranking method requires to solve a non-convex optimization problem to compute allocation for items and consumes a lot of memory for computing 5D scores especially when the number of users or items are large (dealing with a $n \times m$ matrix



(a) Hit ratio vs NO. of recommended items.



(b) Hit ratio vs average distance.

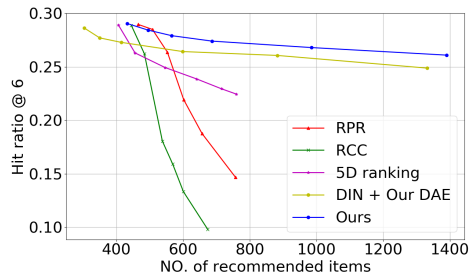
Figure 6: Comparison between our method and the re-ranking methods on MovieLens. "RPR": Reverse predicted rating. "RCC": Reverse click/installation counts. The NO. of the recommended items is the NO. of the unique items recommended among all the users.

where n and m are the numbers of users and items respectively), while our method is much more efficient which only needs to compute the forward propagation of the neural networks.

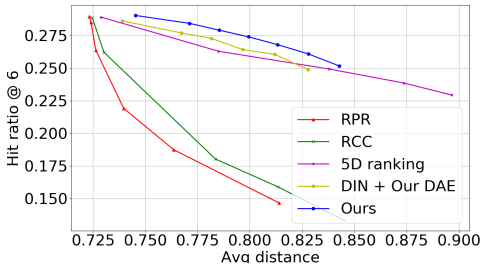
For the app recommendation dataset, our method performs much better, i.e., it can recommend about 1000 items without reducing the hit ratio much, while the re-ranking methods can only recommend about 800 items at most with a certain level of accuracy (e.g., hit ratio @ 10 > 0.2) in this case. We also conducted the experiments ("DIN + DAE") by replacing our relevance model with DIN [37] as shown in Figure 7a and Figure 7b, which demonstrates that our framework can support other recommendation models as well and our relevance model outperforms DIN with the app recommendation dataset in terms of both accuracy and diversity.

Table 2 shows the performance of our method when w used to control diversity varies from 0.1 to 0.4. Clearly, the aggregate diversity can increase a lot without losing offline accuracy much. For this project, we can easily recommend novel items for exploring by adjusting parameter w in real-time without retraining the whole model. Therefore, the sales team can discover new items to sell when they are using our model.

4.3.3 Explainability. We now investigate the explanation generated by the DAE models. Recall that for MovieLens, the DAE models we used for demonstration are the ones shown in Figures 5a and 5b. This "hot" model is applied to construct a list of popular movies and provide explanation "movie X is recommended because it is quite popular" (if X is in this list). Table 3 shows the top 10 movies learned from it. All the movies listed in the table are quite pop-



(a) Hit ratio vs NO. of recommended items.



(b) Hit ratio vs average distance.

Figure 7: Comparison between our method and the re-ranking methods on the app recommendation data. "RPR": Reverse predicted rating. "RCC": Reverse click/installation counts. The number of the recommended items is the number of the unique items recommended among all the users. "DIN + DAE": Replacing our relevance model with DIN and keeping the DAE models.

Table 2: The accuracy-diversity trade-off of our method on the app recommendation dataset.

Parameter w	0.1	0.2	0.3	0.4
Hit ratio @ 6	0.2905	0.2844	0.2740	0.2609
NDCG @ 6	0.2605	0.2521	0.2412	0.2260
Hit ratio @ 10	0.3508	0.3416	0.3306	0.3175
NDCG @ 10	0.2807	0.2712	0.2608	0.2460
NO. of recommended items	433	495	687	1390
Avg distance	0.7452	0.7711	0.7993	0.8280

ular among most age groups. The "categorical" model is applied to compute feature importance and provide explanation "movie X is recommended to user A because A has features P and Q". The user features used here include age, gender, occupation and five preferred genres/topics. Table 4 shows three items recommended by our relevance model and their feature importance scores computed by our DAE model (the account details are listed in the table). From the popular movie list and the feature importance scores, we are able to provide some explanations, e.g., "Planet of the Apes" and "Star Trek VI" are recommended because this user prefers "Action" and "Sci-Fi" movies, while "American Beauty" is recommended because the user is a male at age 45 and interested in "Drama" movies and this movie is popular. We also compare our method with LIME [27].

Table 3: The popular movies extracted by our model.

Rank	Movie
1	American Beauty
2	Star Wars: Episode IV - A New Hope
3	The Silence of the Lambs
4	The Godfather
5	Saving Private Ryan
6	Shawshank Redemption
7	Pulp Fiction
8	Fargo
9	Star Wars: Episode VI - Return of the Jedi
10	Terminator 2: Judgment Day

We did a survey by sampling 50 results and asking "is Explanation A better than Explanation B?" with rating scale 1-5. Explanations A and B are generated by our method and LIME, respectively. We didn't tell in the survey which algorithm generates Explanation A/B. The average rating is 3.29 with variance 1.08, meaning that our method is comparable with LIME.

We now present the explanation results on our private enterprise app recommendation dataset. The explanation has this template "app X is recommended because of features A, B, etc.". In Table 5 and Table 6, we list the top 10 important categorical features learned by the model in Figure 5b and compare them with the results obtained from LIME [27] (Due to the privacy issues, we don't show the explanation related to the continuous-valued features). From these two examples, we can observe that most of the important features extracted by our method and LIME are the same, e.g., CITY, COUNTRY, REGION, ORGTYP. For the first example, our method also highlights the market segment and the account ID which are reasonable for this case. For the second example, our method highlights the cloud service type and the adoption stage, while LIME cannot find them. In these two examples, LIME tends to extract features related to locations, e.g., CITY, COUNTRY and REGION, while the important features extracted by our method are more diverse. We also compare our method with LIME in a quantitative way. For each user/item pair (i, j) in the recommendation results, let $\mathcal{S}_o(i, j)$ be the set of the top 10 important features generated by our method and $\mathcal{S}_l(i, j)$ be the set of the top 10 important features (positive features only) obtained by LIME. Then the metric is defined by explanation_metric $= \frac{1}{10|\mathcal{R}|} \sum_{(i,j)\in\mathcal{R}} |\mathcal{S}_o(i, j) \cap \mathcal{S}_l(i, j)|$, where $\mathcal{R}$ is the set of all the user/item pairs. With the test dataset, the explanation metric is 0.6629, meaning that about 7 out of 10 features extracted by the two methods are the same. This also shows that our framework is able to provide reasonable explanations that are comparable with LIME. Note that LIME is a model-agnostic method for generating explanations, requiring training a local linear model for each user/item pair. It takes much more computational resources and leads to a high response latency, which is not suitable for our case that the system should allow users to tune the diversity and obtain the explanation in real-time. Our method requires no additional training in prediction. We compare the running time when generating explanations (for categorical features and continuous-valued features) for the 170k users with 10 recommended apps for

Table 4: The feature importance scores. The preferred topics are extracted for each user according to the user ratings.

Recommended movie	Age 45	Gender Male	Occupation Executive/Managerial	Topic 0 Action	Topic 1 Sci-Fi	Topic 2 Drama	Topic 3 Adventure	Topic 4 Horror
Planet of the Apes	0.34992	0.70140	0.60264	0.99939	0.87274	0.84117	0.75280	0.27374
American Beauty	0.94194	0.98621	0.87074	0.01474	0.38204	0.98042	0.33694	0.01333
Star Trek VI	0.13612	0.30197	0.54996	0.99992	0.93374	0.53731	0.84779	0.65787

Table 5: Feature-level explanation for "App name: RingLead Field Trip – Discover Unused Fields and Analyze Data Quality". The account ID is hidden due to privacy concerns.

Rank	Our method	LIME
1	**ORGTYP** = UE	**AOVBAND**=200-600k
2	MKT_SEGMENT = ESMB	**ORGTYP** = UE
3	ACCT_OWN_ID = xxxxx	**SUBREGION** = USA
4	**AOVBAND** = 200-600k	ADOPTION_STAGE = assessment
5	**SUBREGION** = USA	**CLOUD** = sales and custom
6	**CLOUD** = sales and custom	**STATUS_DW** = active
7	STATE_NM = NY	COUNTRY_NM = United States
8	**TOP_CITY** = New York	**TOP_CITY** = New York
9	**STATUS_DW** = active	CITY_NM = New York
10	**REGION** = AMER	**REGION** = AMER

Table 6: Feature-level explanation for "App name: 今日から使えるサクセスダッシュボード". This Japanese app is recommended because the customer's country is Japan.

Rank	Our method	LIME
1	**TOP_CITY** = Tokyo	SUBREGION = Japan
2	CLOUD = service	ACCT_OWN_ID = xxxxx
3	**STATE_NM** = 東京都	**USR_ROLE_ID** = yyyyy
4	ADOPTION_STAGE = adoption	REGION = JP
5	STATUS_DW = active	**CITY_NM** = 品川区
6	**CITY_NM** = 品川区	**TOP_CITY** = Tokyo
7	**COUNTRY_NM** = Japan	**COUNTRY_NM** = Japan
8	**INVSTOR_RELT_CD** = small	**STATE_NM** = 東京都
9	**USR_ROLE_ID** = yyyyy	**INVSTOR_RELT_CD** = small
10	**ORGTYP** = EE	**ORGTYP** = EE

each user. The experiment is conducted on a machine with a 3GHz 32-cores CPU and 32GB memory. The running time for LIME is 23 hours (487.1ms per user) while our method requires only 0.6 hours (12.7ms per user) which is much faster.

Besides the feature-level explanation, the categorical DAE model (Figure 5b) can also be used to generate item-based explanation, e.g., "app X is recommended because this user installed apps Y and Z", by replacing the input features with the installed items. Figure 8 shows an example of this kind of explanation, i.e., app "Mass Edit + Mass Update + Mass Delete" is recommended because the user has already installed "Free Blog Core Application" and "Conga Composer". "Mass Edit + Mass Update + Mass Delete" is a developer tool for searching and mass editing multiple accounts, contacts and custom objects, which is useful for cloud-based applications such as "Free blog" and "Conga Composer" which deal with documents

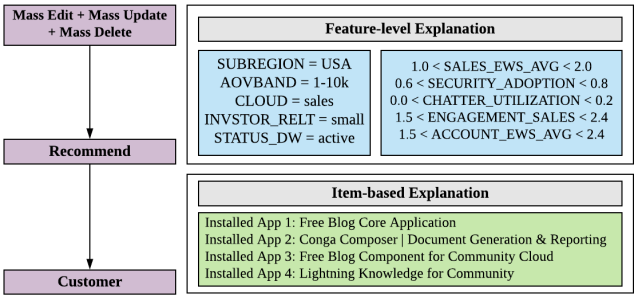


Figure 8: An example of the recommended app.

and contracts. The item-based explanation can be applied to mine some hidden relationship between two different items and provide the sales team more insights about the recommendation results.

Our system has been deployed as a service for our sales teams. We did a user study of our system with our Sales team. Due to the privacy and confidentiality concerns, we can only show parts of our internal users' feedback, i.e., 100% users would use our system and recommend it to others, 87% users trust the information provided by our system, 87% users agree that this tool would replace a longer process for them and 93% users would use the information provided by our system to engage with their customers. Here are some comments we received: "This tool is simple to use and I feel like it's another great one to have in my bag", "The data is a direct reflection of what I know to be true in the account".

5 CONCLUSION

We investigated an enterprise app recommendation problem and proposed a novel framework for improving aggregate diversity and generating explanations. Our framework allows plugging in various diversity-explanation models for different purposes, e.g., extracting popular items, providing feature-level explanations or item-based explanations. For exploring novel items to recommend, our approach provides a convenient way to control the trade-off between recommendation diversity and accuracy. The proposed model can be trained in an end-to-end manner and deployed as a recommendation service easily. We also designed a new model for recommendation accuracy and achieved highly competitive results compared with Wide&Deep [8] and DIN [37] that are widely-applied in industrial recommender systems. The experiments demonstrated that our approach improves recommendation diversity significantly without reducing recommendation accuracy much and generates reliable explanations.

REFERENCES

[1] Gediminas Adomavicius and YoungOk Kwon. 2012. Improving Aggregate Recommendation Diversity Using Ranking-Based Techniques. *IEEE Trans. on Knowl. and Data Eng.* 24, 5 (2012), 896–911.

[2] Erik Brynjolfsson, Yu (Jeffrey) Hu, and Duncan Simester. 2011. Goodbye Pareto Principle, Hello Long Tail: The Effect of Search Costs on the Concentration of Product Sales. *Management Science* 57, 8 (2011), 1373–1386.

[3] Erik Brynjolfsson, Yu (Jeffrey) Hu, and Michael D. Smith. 2003. Consumer Surplus in the Digital Economy: Estimating the Value of Increased Product Variety at Online Booksellers. *Management Science* 49, 11 (2003), 1580–1596.

[4] Shuo Chang, F. Maxwell Harper, and Loren Gilbert Terveen. 2016. Crowd-Based Personalized Natural Language Explanations for Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. Association for Computing Machinery, 175–182.

[5] Jingwu Chen, Fuzhen Zhuang, Xin Hong, Xiang Ao, Xing Xie, and Qing He. 2018. Attention-Driven Factor Model for Explainable Personalized Recommendation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '18)*. 909–912.

[6] Laming Chen, Guoxin Zhang, and Hanning Zhou. 2018. Fast Greedy MAP Inference for Determinantal Point Process to Improve Recommendation Diversity. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NIPS'18)*. 5627–5638.

[7] Xu Chen, Hanxiong Chen, Hongteng Xu, Yongfeng Zhang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2019. Personalized Fashion Recommendation with Visual Explanations Based on Multimodal Attention Network: Towards Visually Explainable Recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'19)*. 765–774.

[8] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS 2016)*. 7–10.

[9] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. 191–198.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 4171–4186.

[12] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. 2017. Sequential User-Based Recurrent Neural Network Recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems (RecSys '17)*. 152–160.

[13] Azin Ghazimatin, Oana Balalau, Rishiraj Saha Roy, and Gerhard Weikum. 2020. PRINCE: Provider-Side Interpretability with Counterfactual Explanations in Recommender Systems. In *Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM '20)*. Association for Computing Machinery, 196–204.

[14] Carlos A. Gomez-Uribe and Neil Hunt. 2016. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Transactions on Management Information Systems* 6, 4 (2016).

[15] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine Based Neural Network for CTR Prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. AAAI Press, 1725–1731.

[16] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4 (2015).

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV) (ICCV '15)*. 1026–1034.

[18] Yu-Chieh Ho, Yi-Ting Chiang, and Jane Yung-Jen Hsu. 2014. Who Likes It More? Mining Worth-Recommending Items from Long Tails by Modeling Relative Preference. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining (WSDM '14)*. 253–262.

[19] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM '08)*. 263–272.

[20] Yejin Kim, Kwangseob Kim, Chanyoung Park, and Hwanjo Yu. 2019. Sequential and Diverse Recommendation with Long Tail. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. 2740–2746.

[21] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015*.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12)*. 1097–1105.

[23] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* 7 (2003), 76–80.

[24] Julian McAuley and Jure Leskovec. 2013. Hidden Factors and Hidden Topics: Understanding Rating Dimensions with Review Text. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys '13)*. 165–172.

[25] Georgina Peake and Jun Wang. 2018. Explanation Mining: Post Hoc Interpretability of Latent Factor Models for Recommendation Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*. 2060–2069.

[26] Steffen Rendle. 2010. Factorization Machines. In *Proceedings of the 2010 IEEE International Conference on Data Mining (ICDM '10)*. 995–1000.

[27] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. 1135–1144.

[28] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann Machines for Collaborative Filtering. In *Proceedings of the 24th International Conference on Machine Learning (ICML '07)*. 791–798.

[29] Ying Shan, T. Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. 2016. Deep Crossing: Web-Scale Modeling without Manually Crafted Combinatorial Features. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. 255–262.

[30] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic Attribution for Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*. 3319–3328.

[31] Nava Tintarev and Judith Masthoff. 2012. Evaluating the effectiveness of explanations for recommender systems: Methodological issues and empirical studies on the impact of personalization. *User Modelling and User-Adapted Interaction* 22, 4-5 (Oct. 2012), 399–439.

[32] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *Proceedings of the ADKDD'17 (ADKDD'17)*.

[33] Yichao Wang, Xiangyu Zhang, Zhirong Liu, Zhenhua Dong, Xinhua Feng, Ruiming Tang, and Xiuqiang He. 2020. Personalized Re-ranking for Improving Diversity in Live Recommender Systems. arXiv:cs.IR/2004.06390

[34] Mark Wilhelm, Ajith Ramanathan, Alexander Bonomo, Sagar Jain, Ed H. Chi, and Jennifer Gillenwater. 2018. Practical Diversified Recommendations on YouTube with Determinantal Point Processes. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM '18)*. 2165–2173.

[35] Mi Zhang and Neil Hurley. 2008. Avoiding Monotony: Improving the Diversity of Recommendation Lists. In *Proceedings of the 2008 ACM Conference on Recommender Systems (RecSys '08)*. 123–130.

[36] Yongfeng Zhang, Guokun Lai, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. 2014. Explicit Factor Models for Explainable Recommendation Based on Phrase-Level Sentiment Analysis. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '14)*. 83–92.

[37] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep Interest Network for Click-Through Rate Prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*. 1059–1068.

[38] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. 2005. Improving Recommendation Lists through Topic Diversification. In *Proceedings of the 14th International Conference on World Wide Web (WWW '05)*. 22–32.