applied sciences

MDPI

*Article*

# MovieDIRec: Drafted-Input-Based Recommendation System for Movies

**Hyeonwoo An [1], Daeyeol Kim [1], Kwangkee Lee [1,*] and Nammee Moon [2,*]**

[1] Tvstorm, Sunghyun Building, 255 Hyorung-ro, Secho-gu, Seoul 13875, Korea; grabro@tvstorm.com (H.A.); wagon0004@tvstorm.com (D.K.)
[2] Department of Computer Engineering, Hoseo University, Asan 31499, Korea
* Correspondence: kklee@tvstorm.com (K.L.); nammee.moon@gmail.com (N.M.)

**Abstract:** In a DNN-based recommendation system, the input selection of a model and design of an appropriate input are very important in terms of the accuracy and reflection of complex user preferences. Since the learning of layers by the goal of the model depends on the input, the more closely the input is related to the goal, the less the model needs to learn unnecessary information. In relation to this, the term Drafted-Input, defined in this paper, is input data that have been appropriately selected and processed to meet the goals of the system, and is a subject that is updated while continuously reflecting user preferences along with the learning of model parameters. In this paper, the effects of properly designed and generated inputs on accuracy and usability are verified using the proposed systems. Furthermore, the proposed method and user–item interaction are compared with state-of-the-art systems using simple embedding data as the input, and a model suitable for a practical client–server environment is also proposed.

**Keywords:** MovieDIRec; drafted-input; personalized recommendation system



**Citation:** An, H.; Kim, D.; Lee, K.; Moon, N. MovieDIRec: Drafted-Input-Based Recommendation System for Movies. *Appl. Sci.* **2021**, *11*, 10412. https://doi.org/10.3390/app112110412

Academic Editor: Vincent A. Cicirello

Received: 8 October 2021
Accepted: 4 November 2021
Published: 5 November 2021



## 1. Introduction

Recently, various studies related to recommendation systems have been actively conducted. There are two main themes. The first is methods to solve the fundamental problems of the recommendation system such as the first rater, the cold start problem, overspecialization, and protection of user privacy [1–3], and the second is the improvement of the accuracy of the recommendation system [4–9].

Currently, various methods combining a DNN (Deep Neural Network) with collaborative filtering, content-based filtering, etc., including pure DNN-based methods, are being studied and advanced to improve accuracy [2,10–12]. Most of the methods of recent research are methods of inferring preference trends using similar users, such as collaborative filtering, based on vectors embedding user–item interaction. In this case, the model learns the user–item preference distribution. However, since the data describing the item are limited to the embedding vector dimension, there is a tendency to generalize the user's complex preferences. On the other hand, a method using content-based filtering is also being actively studied. The study mainly solves the cold-start problem or the first-rater problem and is introduced as a solution with improved accuracy compared to pure content-based filtering.

The method we propose is an approach using a DNN, which uses the Drafted-Input data with potential meanings to omit the inaccurate and generalized semantic transformation process from the model and induces learning only the necessary meanings. Certain parts of the drafting process of Drafted-Input clearly fall under feature engineering or pre-processing. However, it means more than preprocessing, in that it is continuously updated by learning and each role exists within the model network. Drafted-Input is data processed by analyzing/selecting related meta data according to the purpose pursued by

the model, as well as the model. However, the reason for the new definition of "Drafted-Input" without using terms such as "Add-hoc", "meta data", or "Preprocessed data" is as follows:

- It is the subject of training and is input data that are continuously updated through learning once they are created ≈ preliminary and updatable object.
- These are the input data selected to describe the target in terms of the goal of the model ≈ select for a certain purpose.

In this paper, we propose a movie recommendation system that focuses on the latter, improving accuracy, and also providing an appropriate configuration to apply it in a practical client–server environment. The reason why the scope is limited to movies is that the proposed method has the characteristic of operating only in one domain, and the data are relatively open compared to multiple domains. In this paper, movie data of IMDB and rating data of MovieLens-20m were used [13]. We propose two types of DNN-based models that learn through the user's implicit feedback and provide personalized recommendations; the first is a model that is as rough as possible so that it is easy to understand and modify concepts and can be applied in a centralized environment. Second, the Auto-Encoder concept is applied to the first model in consideration of the practical environment so that the server and the client divide and share the operation. It is a model that reduces the traffic for inference as much as possible.

Our Contributions. The major contributions are summarized as follows:

- Verify the effect of Drafted-Input of movie/user on training and recommendation accuracy;
- Propose an inference resource distribution method based on Auto-Encoder considering the client–server environment;
- Propose a method to personalize by paying attention to specific preference features of items in the network using User's weights extracted through a specific method.

## 2. Related Works

### 2.1. DNN-Based Recommendation System

So far, various approaches using a DNN have been proposed in the field of recommendation systems. A well-known example is the recommendation system of the YouTube platform [11]. Due to the vast amount of content on YouTube, a lot of calculations are required for recommendations. Therefore, the method that YouTube has chosen is to apply the Candidate model that filters out candidates from a huge amount of content and the ranking model that infers ranking among the filtered content. It is to infer the user's action using a model similar to the candidate model through hundreds of videos that are impressions from the candidate model. YouTube uses implicit feedback based on whether the user watched or not, instead of explicit feedback such as 'like' and 'dislike', which are functions provided by the platform. This is because, in general, explicit feedback is very sparse, and more user history can be utilized using implicit feedback. Furthermore, YouTube took a method of solving the cold-start problem to some extent by including the user's geographic information or demographic information in the input features.

Another approach that is rapidly emerging is a method using a Variational Auto Encoder (VAE) [4,6–8]. Dissimilar to the dimensionality reduction in the input, which is one of the purposes of the commonly used Auto Encoder, a VAE aims to generate new data using the learned distribution. To apply this to the recommendation system, these studies suggest improvement methods such as applying the reparameterization trick or modifying the loss function of the existing VAE to effectively tune the parameters. A VAE has the characteristic of being able to solve problems of linear latent factor models (e.g., Matrix Factorization), such as overfitting caused by sparse data or slowing down due to model size, by applying the concept of the multinomial distribution, which was not mainly used in the recommendation system. However, as mentioned above, if a single shared model is used, distribution learning using embedding vectors tends to generalize user preferences. In addition, it is also impossible to reflect the negative experience to the user's preference by using a vector that maps interaction to 0 and 1 for the distribution

calculation. Due to these characteristics, the results are reflected insensitively to the user's dislike when recommending.

### 2.2. TF-IDF

The term Frequency–Inverse Document Frequency (TF-IDF) is one of the text vectorization techniques and is a method of weighing the importance of each word in a document through word frequency and inverse document frequency. When the document is $\boldsymbol{d}$ and the word is $\boldsymbol{t}$, the tf-idf vectorization can be expressed as the following expression:

$$tfidf(t, d, D) = tf(t, d) \times idf(t, D) \tag{1}$$

Here, $tf(t, d)$ can be seen as the frequency of the term in the document, and $idf$ is calculated as follows:

$$idf(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|} \tag{2}$$

This property can prevent frequently included words such as "this" and "the" from affecting the vector expressing the meaning of the document, although it contains little meaning.

### 2.3. Truncated-SVD

Singular Value Decomposition (SVD) means decomposing a given $m \times n$ matrix $A$ as follows.

$$A = U\Sigma V^T \tag{3}$$

In the above equation, $\boldsymbol{U}$ is an $\boldsymbol{m} \times \boldsymbol{m}$ orthogonal matrix obtained by the eigen decomposition of $\boldsymbol{AA^T}$. It is called a left singular vector of $\boldsymbol{A}$, and $\boldsymbol{V}$ is an $\boldsymbol{n} \times \boldsymbol{n}$ type right singular vector obtained by the eigen decomposition of $\boldsymbol{A^TA}$. Finally, $\boldsymbol{\Sigma}$ is a rectangular diagonal matrix of the form $\boldsymbol{m} \times \boldsymbol{n}$ whose diagonal elements are the square roots of the eigenvalues of $\boldsymbol{U}$ and $\boldsymbol{V}$. The one method is called Full-SVD, and there are Compact-SVDs in which a portion having singular values of zeros is reduced and a Thin-SVD method in which only rows excluding a diagonal matrix are reduced. However, it is difficult to use in a recommendation system environment with sparse data.

Another method is the Truncated-SVD method, which extracts and uses only the top $\boldsymbol{t}$ singular values of the SVD. This is aimed at approximation rather than restoration, and by setting $\boldsymbol{t}$ at a level that does not negatively affect the approximation, a reduced $\boldsymbol{U\Sigma V^T}$ is obtained, and a compressed vector as much as $\boldsymbol{t}$ can be obtained through this. Because of these characteristics, it can be used in a more suitable way for sparse data. Both SVD and t-SVD are methods frequently used as matrix factorization techniques for collaborative filtering, and studies using them are still being actively conducted [14–18].

### 2.4. Auto Encoder

Dissimilar to the VAE described above, an Auto Encoder is generally used for dimensionality reduction. The model consists of an encoder model that compresses input features and a decoder model that approximates the original by expanding the compressed code information. The Auto Encoder has the characteristic that an SVD is a linear dimensionality reduction algorithm, whereas the Auto Encoder is a non-linear based algorithm, so it can operate smoothly even for complex data compression. Methods using the data compression and approximation characteristics of an Auto Encoder in the recommendation system have been actively proposed [19,20]. These methods aim to create a new latent vector for a user or item, and it was confirmed that they generate a vector of better quality than the traditional generation methods. However, in terms of performance, the overall performance was lower than the recent state-of-the-art systems using a VAE.

## 3. Our Approach

In this chapter, our proposed DNN-based approaches are described in detail. As stated in the introduction, we propose two models: the MovieDIRec, which is rough and

easily configured, and MovieDIRec+, which can be applied to a practical environment by applying an Auto Encoder to MovieDIRec. Furthermore, the Drafted-Input creation method and outline proposed in this paper are described in detail.

### 3.1. Drafted-Input

Drafted-Input is created through a series of preprocessing processes by combining the user's rating vector with the movie's metadata obtained through IMDB. Drafted-Input is data that describe the object in more detail than a simple embedding vector and is composed of data closely related to user preferences. We used director, actor, genre, release date, votes, plot, etc., from the movie information provided by IMDB, and the Drafted-Input Data generated through it consisted of the following: $\boldsymbol{M}$: Data that describe the Movie and consist of a Director, Actor, Genre, Story, and Popularity. $\boldsymbol{U_W}$: User's preferred weights corresponding to each feature of $\boldsymbol{M}$ except for popularity. $\boldsymbol{U_P}$: User's preferred weight for popularity. $\boldsymbol{U_F}$: User Features that express user preferences through the t-SVD method.

The draft process to create Drafted-Input is carried out before learning and when new movies and users are introduced. In this paper, TF-IDF and t-SVD were used during the draft process. According to [6], t-SVD, a type of matrix factorization method, has the following disadvantages: (1) Speed decrease due to model size. (2) Not applicable to new user/item. (3) A large amount of user feedback is concentrated on well-known content, so overfitting is easy.

However, in the proposed system, the draft process, train, and runtime processes were separated, so the speed of the t-SVD model did not affect the runtime. In addition, it was possible to apply a new user through the draft process that could be connected to provisioning. Lastly, even if it was a new item, the Actor and Director features created by the t-SVD were shared and learned between movies, so they could be directly applied to new movies.

#### 3.1.1. Movie Features

Movie Features $\boldsymbol{M}$ are a feature for describing a movie and are information related to preferences. Among them, Director, Actor, and Popularity, except for Genre and Story, which are fixed features, can be interpreted as continuously changing features, which are defined as features that need to be updated by user feedback. We converted the rating data into rating data corresponding to each Actor/Director, differentially applied weights set appropriately, and compressed them through the t-SVD to transform them into matrices with potential meaning. In the case of popularity $\boldsymbol{P}$, it consisted of a single scalar value, and a normalized rank according to the number of ratings per published date was used.

It should be noted that, in the case of $\boldsymbol{P}$, it required a different treatment from other features of $\boldsymbol{M}$. Popularity is a scalar value indicating the popularity of the movie, and it was judged that separate treatment was necessary because it was the only verified value directly related to the quality of a movie among features and an important value that affects the overall preference. Therefore, the user's preferred weight corresponding to popularity was also treated independently.

In the case of Genre, vectorization was performed through TF-IDF. The reason that simple count vectorization was not applied is that count vectors contain very sparse information and the genre distribution tends to be very different.

In the case of the story, the following pre-processing was performed in advance and, then, vectorization was performed:

1. Remove numbers and symbols that are not related to the story;
2. Remove person names and attach NER(Named Entity Recognition) tags using the pre-trained Bert model [21];
3. Remove movies with five or fewer descriptive words;
4. Only words included in the movie plot were left, and TF-IDF was conducted only for words that appeared three or more times in all movies.

While genre and story features were mapped 1:1 with a movie, the created actor and director features were shared to all movies, and they were mapped as N:N, because a movie often includes multiple actors/directors. Therefore, in Drafted-Input, the actor and director features were replaced with two features weighted and averaged according to the written order. When updating, the gradient for the weighted averaged feature was obtained, so the update was performed by applying it to the target features at once. Figure 1 is a visualization of the process of Drafted-Input for the movie being input to the model.

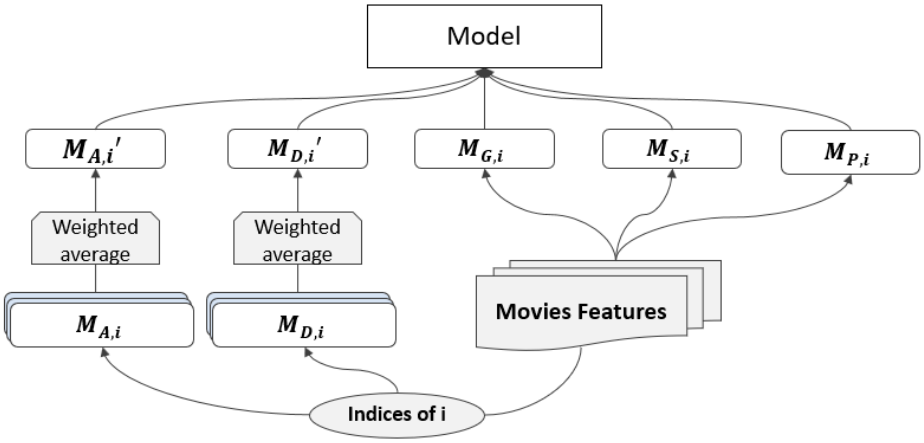



**Figure 1.** The process of Drafted-Input for Movie being input to the model. $i$ is the index for the target movie.

#### 3.1.2. User Features

User features consist of $U_W$, $U_P$, and $U_F$. This is information expressing user preference. $U_W$ is a preference weight array corresponding to (actors, directors, genre, story), respectively, and $U_P$ is a single weight corresponding to popularity. Lastly, $U_F$ is a feature that directly expresses the user rather than the previous features and was obtained by using t-SVD in the User × Movie rating matrix. Since these features all contain changing preferences, they must be continuously updated not only in Train, but also in Runtime.

$U_W$ should express how much each target movie feature affects the user's preference. We tested the following methods to obtain the user's weight. First, we tried to obtain the user's preference bias by using the characteristic of the L2 norm, in which the result value increased as the outlier was larger. In this case, a sparse count vector set $E$, not Movie Features $M$, was used for the weight calculation. Set $E$ had a size unique to each number of features: 151,859 actors, 48,459 directors, 28 genres, and 12,624 words of stories. For user $u$, $R_u$, which is a rating set for specific movies I of the user, exists, and $E_{w,\,I}$ corresponding to four types of metadata $W$ also exists. The preference weight $U_w$ was generated as follows:

$$V = \left\{ e \odot R_u^T \,\middle|\, e \in E \right\},\ N = \{ ||v||_2 \mid v \in V \},\ U_{w,i} = \frac{N_i}{max(N)} \tag{4}$$

However, the above formula tended to significantly lower the values of other features when the user had a large bias towards a particular feature. This was due to the characteristic of L2-norm, whose value increased exponentially according to the size of the outlier, and we tri to lead so that the value was not greatly influenced by each feature by adjusting the scale of the value through the equation below:

$$A_i = \frac{\sum_x^{V_i} x^2}{\sum_{x \in V_i | x \rangle 0} 1},\ U_{w,i} = \frac{A_i}{max(A)} \tag{5}$$

As a result of applying the above formula, the difference according to the scale of the value dropped significantly, but in the case of features such as actors and stories, embedded items with countless but small values entered the average element and tended to lower

the result value. Therefore, we chose the coefficient of variation as a method to more appropriately detect the bias extracted by outliers without being affected by the scale between features. Since the coefficient of variation uses the standard deviation of the target vector, the scale became more uniform than the methods using the square. In addition to the process of obtaining the variance in the vector, the process of dividing by the arithmetic mean was included, so that the bias toward the outlier could be obtained more flattened. The method was as follows:

$$G_i = \frac{std(V_i)}{avg(V_i)}, \quad U_{w,i} = \frac{G_i}{max(G)} \tag{6}$$

In the case of $\boldsymbol{U_P}$, it was created as an average of differentially applied popularity, just such as how to create actors and director features. It was judged that this method was suitable for users to explain the evaluation tendency according to popularity.

If the specific user's $\boldsymbol{U_P}$ was $\boldsymbol{U_{P,u}}$, the movie's popularity was $\boldsymbol{M_p}$, the rated item set was $\boldsymbol{I}$, and the weighted rating set was $\boldsymbol{R}$, then $\boldsymbol{U_{P,u}}$ was obtained as follows:

$$U_{P,u} = \frac{\sum_i^I R_{u,i} * M_{p,i}}{|R_u|} \tag{7}$$

$\boldsymbol{U_P}$ represents how much the user depends on the popularity of a movie, and by using this, the degree of attenuation and amplification according to popularity can be adjusted. That is since users with high $\boldsymbol{U_P}$ tend to give high scores to movies with high popularity, the influence on preference should be amplified. On the other hand, a person with a low $\boldsymbol{U_P}$ is a person who has a negative view of most movies and should try to attenuate the influence on preference. Considering this, if there was $\boldsymbol{P_i}$, which is the popularity of specific movie $\boldsymbol{i}$, and $\boldsymbol{U_{P,u}}$, which is the $\boldsymbol{U_P}$ of a specific user $\boldsymbol{u}$, the above amplification and attenuation effect could be achieved just by multiplying $\boldsymbol{P_i} \times \boldsymbol{U_{P,u}}$ to the previous layer's output.

Because of these characteristics of $\boldsymbol{U_{P,u}}$, it is important to guide the user to input the preferred movie at a higher rate than the unfavorable movie in the provisioning stage where the user's explicit feedback is provided.

### *3.2. Proposed Methods: MovieDIRec and MovieDIRec+*

The first model proposed is shown in the left of Figure 2. The output layer of the model was a Softmax classifier that predicted implicit feedback. Implicit feedback was implemented by converting four or more points into positive in the rating data and binarizing them. Therefore, in the final layer of the model, Softmax performed binary classification to predict the user's positive interaction. The model consisted of fully connected layers that simply compressed or expanded the input, and multiply and concatenation layers that connected $\boldsymbol{U}$ and $\boldsymbol{M}$.

Drafted-Input, which refers to the preferences of actors, directors, and users, must change over time. Therefore, Drafted-Inputs except for $\boldsymbol{M_G}$, $\boldsymbol{M_S}$, and $\boldsymbol{M_P}$ were included in the learning target together with the model parameters during training. As shown in Figure 1, $\boldsymbol{M_A\prime}$ and $\boldsymbol{M_D\prime}$ were features combined into one dimension by weighting all actors/directors included in the target movie. Therefore, when updating, all $\boldsymbol{M_A\prime}$ and $\boldsymbol{M_D}$ included in the target movie were updated using the gradient of $\boldsymbol{M_A\prime}$ and $\boldsymbol{M_D\prime}$ as it was.

We constructed a model network so that as $\boldsymbol{U_P}$ and $\boldsymbol{U_W}$ had higher values, more attention was paid to the $\boldsymbol{M}$ features corresponding to $\boldsymbol{U_P}$ and $\boldsymbol{U_W}$. The features that were paid attention to according to the user's weight were finally combined with the user's latent meaning $\boldsymbol{U_F}$ to interpret the preference, and, at this time, it played a role in expanding the part so that the model could focus on the user's interest.

MovieDIRec+ in the right of Figure 2 was the second model proposed in this paper. Additionally, it is a model configured so that the client and server can share the recommendation load in a practical environment. As shown in Figure 3, by placing an Auto-Encoder

in the MovieDIRec model, the encoding part of Movie features could be configured as the Server-side and the decoding part as the Client-side.

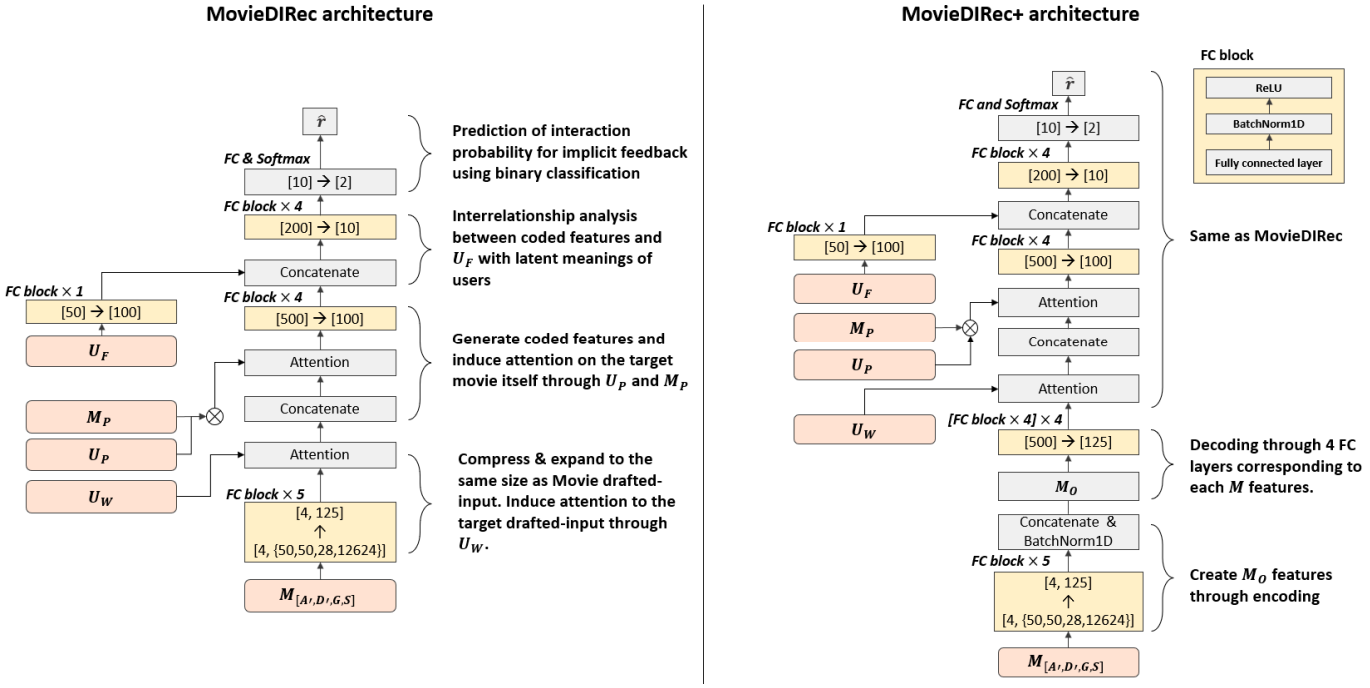



**Figure 2.** MovieDIRec (**left**) and MovieDIRec+ (**right**) model architecture. Yellow boxes mean FC layers, and red boxes mean Drafted-Input.

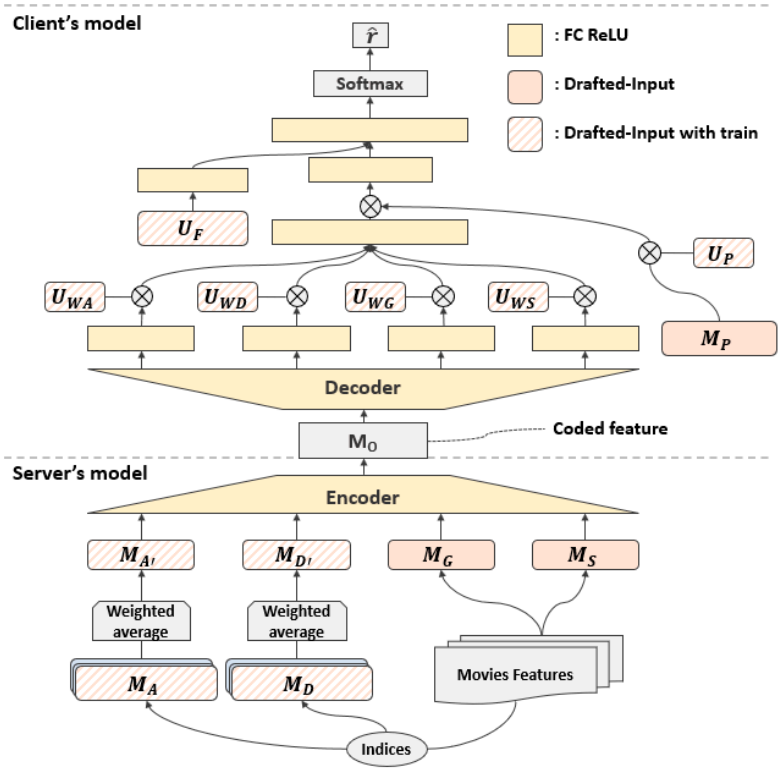



**Figure 3.** MovieDIRec+ model overview. Yellow boxes mean FC blocks, and red boxes mean Drafted-Input. Among the red boxes, the boxes drawn with an oblique line are Drafted-Inputs that are learning objects.

This configuration had the following advantages in a client–server environment; First, coded $\boldsymbol{M_o}$ could be used as a compressed feature for a movie, and the processing speed was improved as the input was reduced and the encoder layers were omitted. If the $\boldsymbol{U_F}$ existed in the client environment, the client could proceed with inference only through $\boldsymbol{M_o}$, which is a coded feature, and $\boldsymbol{M_P}$, which is a scalar value. Second, learning can be distributed by a separate client–server model. This means split learning, and the client model performed the same function as performing batch learning based on the user simply by passing the gradient for $\boldsymbol{M_o}$ to the server-side. Finally, by using compressed information, a candidate group could be derived by methods such as nearest neighbor search, which can dramatically improve speed [11].

However, according to [22–24], the matrix factorization method did not satisfy the triangle inequality caused by the dot product. Since a matrix composed of t-SVD was included among the components, it was necessary to check that the similarity check between the transformed vectors was valid. There was one interesting point here. Figure 4 is a graph that visualizes cases in which the triangle inequality was checked and unsatisfied for all pairs by sampling each epoch for 50 random $\boldsymbol{M_o}$ as the experimental results. As shown in the graph, it was seen that the $\boldsymbol{M_o}$ feature was updated in a direction that satisfied the triangle inequality by a nonlinear transformation. This was a more meaningful result as a test using a model with fixed t-SVD features, except for input training.

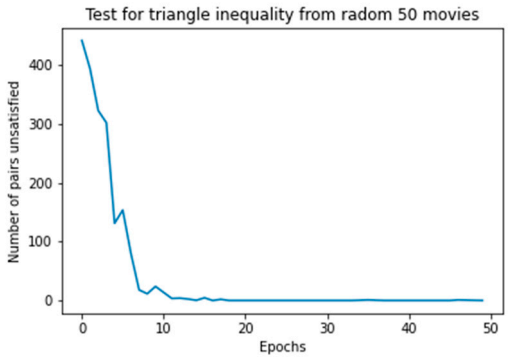



**Figure 4.** The triangle inequality check test. We sampled 50 points randomly at each epoch and checked all pairs were satisfied triangle inequality. The *y*-axis is the number of unsatisfied pairs. The *x*-axis is the number of each epoch.

## 4. Experiments

In this chapter, we compared and verified the learning results of MovieDIRec and MovieDIRec+ models with various state-of-the-art recommendation systems.

### *4.1. Metric*

As a metric for performance evaluation, it was decided to use *Recall@k* and *NDCG@k*, which are ranking-based metrics commonly used in most approaches to compare with various state-of-the-art recommendation systems [4–8,22,25]. Before verification, the probabilities predicted by the implicit positive of the Softmax output were sorted and ranked for verification.

*Recall@k* for verification was performed as follows: $\hat{\boldsymbol{r}}$ means the set predicted for implicit feedback as the sorted Softmax output $\hat{\boldsymbol{r}}$ is greater than 0.5. $R$ is the ground truth value, which is a set of users' implicit feedback. Finally, $\mathbb{I}[\,]$ means indicator function.

$$Recall@k = \frac{\sum_{i=1}^{k} \mathbb{I}\left[\hat{r}'_i \in R\right]}{|R|} \tag{8}$$

*NDCG* (Normalized Discounted Cumulative Gain) is a metric mainly used in the ranking-based system. Since a metric of the same scale was required for comparison, *NDCG* was normalized by dividing *DCG* by *IDCG*, which was the *DCG* value of the ideal

result. The expression of $DCG@k$ for implicit feedback composed of the binary class used in the proposed method was as follows:

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{\log(i+1)}, \quad rel_i = 2^{\mathbb{I}[\hat{r}_i' \in R]} - 1 \tag{9}$$

### 4.2. Setup

The output of the model was composed of a Softmax output layer as a binary probability predicting implicit feedback. Binary cross entropy was used as the loss function. The parameters of the model were updated by Adam and, in the case of the input train, the updates were applied by SGD. The learning rate of both was set equal to 0.0001, Adam's epsilon was set to $1 \times 10^{-8}$, and betas were set to (0.9, 0.999).

As comparison methods, there were records verified using $Recall@k$ and $NDCG@k$ metric using ML-20M dataset for accuracy comparison between methods, and systems with high accuracy were selected.

- VASP [4]: The author proposed a model ensembled by element-wise multiplication of NEASE and FLVAE as a VAE-based Top-N recommendation system.
- RaCT [5]: An efficient and scalable learning-to-rank algorithm was proposed by borrowing the actor–critic idea of reinforcement learning to approximate the ranking metric.
- RecVAE [6]: The authors followed the Mult-VAE model using multinomial distribution instead of Gaussian and Bernoulli distribution, which is generally used as the likelihood function in VAE, but improved the performance by modifying the Evidence Lower Bound (ELBO) formula and detailed architecture.
- CML [22]: By combining metric learning algorithms with collaborative filtering, the authors proposed a method to learn using similarity between user–user and user–item, and achieve significant speedup and approximate nearest-neighbor search with a slight decrease in accuracy.

### 4.3. Dataset

In the case of the Rating Dataset, the preprocessing was performed as follows, considering the negative impact on learning: (1) Filtering to have at least 5 ratings per user among 20 million data of ML-20m. (2) Among the filtered data, the movie to be evaluated consisted of only movies included in the IMDB dataset. The data were finally filtered to 51,869 users, 4714 movies, and 6,429,862 ratings. (3) Finally, the distribution was as follows, so that there was no overlapping rating in each process.

- D@1: All rating data;
- D@2, 3: Data(D@2) randomly extracted from 10 ratings per user to make Drafted-Input in D@1 and remaining data(D@3);
- D@4, 5: Data(D@4) extracted from D@3 of 10,000 Held-out users and the rest of the data (D@5);
- D@6, 7: Data(D@6) extracted at a rate of 10% from D@5 for training and data(D@7) extracted from 20% of the remaining data for testing for input training;
- D@8: Data extracted at a rate of 12% from D@4 for validation/test.

When 80% was used as train data, it had a density of about 2.6%, which was 9 times higher than 0.28%, which was the average density value of train data processed in comparative methods to compare. For an accurate comparison, the density of D@6, the train data, were extracted at a rate of 10% to lower the density to 0.26%.

On the other hand, in the case of test data D@4, the $Recall@k$ and $NDCG@k$ metrics used for comparison produced different results depending on the target average number of ratings and the ratio of positives, so it was essential to set them to the same level as the comparison methods. Therefore, it was extracted at a rate of 12% to approximate 14, which was the average size of validation of the comparison methods.

The steps for verification and comparison were as follows: First, we created Drafted-Input using D@2. Afterward, the entire model was trained using Drafted-Input and D@6. Finally, validation was carried out through D@8, and in the case of the input train model, the effect of the learned Drafted-Input had to be utilized, so it was tested through D@7.

### *4.4. Results and Analysis*

Table 1 is the data setting of the baselines and the proposed system. In the baselines, the density of the entire dataset was similar, so no separate processing was required, whereas the distribution of the proposed method was greatly changed by filtering by IMDB movies. Therefore, we matched the target density and the number of interactions per user in the test through a fixed ratio.

**Table 1.** Comparison table with data environment of baselines. % Density Train means the density of the training dataset in the corresponding methods. #Interaction Test means the average number of ratings for each user used in the test stage. #User, #Movie, and #Rating are the number of preprocessed datasets in each method. % Train/Test is the ratio of the train (and valid) set/test set, including Held-out data.

| | **%Density Train** | **#Interaction Test** | **#User** | **#Movie** | **#Rating** | **%Train/Test** |
|---|---|---|---|---|---|---|
| VASP [4] | 0.29% | 14.63 | 136,677 | 20,108 | 10M | 80%/20% |
| RaCT [5] | - | - | 136,677 | 20,108 | 10M | - |
| RecVAE [6] | 0.28% | 14.61 | 136,677 | 20,720 | 9,990,682 | 80%/20% |
| CML [22] | 0.29% | 15.31 | 129,797 | 20,709 | 9,939,873 | 80%/20% |
| Ours | 0.26% | 14.87 | 51,869 | 4714 | 6,429,862 | 10%/12% |

Table 2 is a comparison table for metrics with target baselines. For a more detailed comparison, we divided the two models into four models with or without input training. Each metric was calculated using D@7 and D@8, respectively, and the models were trained only by 10 epochs using D@6 in consideration of overfitting. The numbers written in the metric columns were calculated for all users and averaged.

**Table 2.** Comparison table for each metric. * is a symbol for models that were trained with Drafted-Input.

| | **%Density Train** | ***NDCG*@10** | ***NDCG*@100** | ***Recall*@20** | ***Recall*@50** |
|---|---|---|---|---|---|
| VASP [4] | 0.29% | - | 0.448 | 0.414 | 0.552 |
| RaCT [5] | - | - | 0.403 | 0.403 | 0.543 |
| RecVAE [6] | 0.28% | - | 0.442 | 0.414 | 0.553 |
| CML [22] | 0.29% | 0.5301 | - | - | 0.4665 |
| MovieDIRec | 0.26% | 0.6113 | 0.4567 | 0.5223 | 0.5319 |
| MovieDIRec+ | | 0.6246 | 0.4790 | **0.5532** | **0.5634** |
| MovieDIRec * | | 0.6136 | 0.4789 | 0.5189 | 0.5342 |
| MovieDIRec+ * | | **0.6535** | **0.5132** | 0.5049 | 0.5276 |

Comparing the metric values between the proposed models showed different performances depending on the environment of the model. In the case of the models undergoing input training, the *NDCG* values increased, but the *Recall* values tended to decrease. We repeated a lot of learning to interpret this phenomenon and came to the following conclusions: First, as a result of examining the output of the models subjected to Drafted-Input learning, it was observed that the positive prediction including false positive and true positive was reduced by about 10%. In addition, *NDCG* is a metric that is more sensitive to ranking than *Recall*, and Recall tends to score high when there are many positive predic-

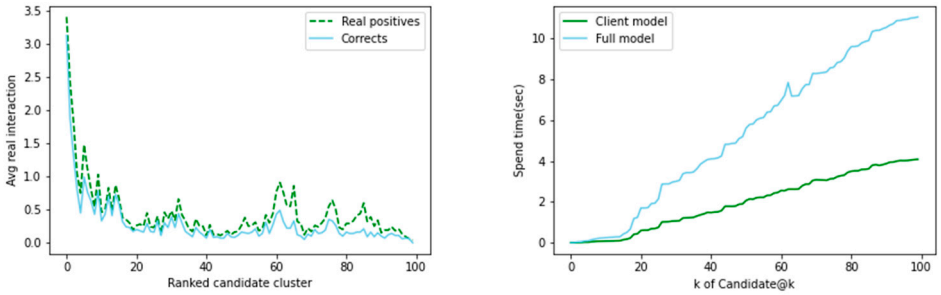
tions. Therefore, if *NDCG* scored high in distribution with a reduced positive prediction, it was interpreted as improved in terms of recommendation accuracy.

Table 3 is a metric table that assumes a rich rating data environment without matching the density with baselines. For this, we separately created an 80% train set and a 20% validation set and trained them for 30 epochs. It could be seen that the more data, the higher the learning progress. In addition, even in the environment of Table 3, it was observed that, as in Table 2, there was a tendency to become more personalized when input training was carried out.

**Table 3.** Metric table of models that did train/test in a ratio of 80:20 without matching the density.

| | %Density Train | *NDCG*@10 | *NDCG*@100 | *Recall*@20 | *Recall*@50 |
|---|---|---|---|---|---|
| MovieDIRec | 3.85% | 0.7305 | 0.5908 | 0.5669 | 0.6384 |
| MovieDIRec+ | | 0.7305 | 0.5910 | **0.5710** | **0.6443** |
| MovieDIRec * | | 0.7522 | **0.6368** | 0.5516 | 0.5851 |
| MovieDIRec+ * | | **0.7528** | 0.6293 | 0.5540 | 0.5972 |

Figure 5a is a graph of the candidate group test result of MovieDIRec+, which consisted of the following; First, a compressed $M_o$ was generated for all movies using a pre-trained server model, and 100 clusters and centroid vectors representing each cluster were generated through a Gaussian mixture. Afterwards, preference inference was performed on all centroid vectors for 10,000 Held-out users who were not used in the train, and the results were sorted by each user. Finally, inferences about the movies in the cluster were performed for each ranked cluster.




(**a**) (**b**)

**Figure 5.** (**a**): Recommendation result graph according to clusters ranked by user. The *x*-axis is the preference ranking of centroids of clusters inferred by each user, and the *y*-axis means the average number of real positive interactions. (**b**): This is a graph measuring the consumed time to recommend a candidate group for a specific user according to a change in k using MovieDIRec+. Candidate@k means that the top k among the candidates for the user was used for recommendation. *x*-axis means k, and *y*-axis means time (seconds).

As a result of the test, positive interactions were gathered in a large proportion in the clusters ranked at the top. This candidate group recommendation had a great advantage in terms of speed. Table 4 below summarizes the speed comparison of the entire model and the client's model for a specific user, including whether or not the candidate group recommendation was applied. The test environment was conducted in the same server for an accurate comparison of speed, and the target movie was 91,514 movies including unrated movies. In the case of the candidate group recommendation, it was conducted through Candidate@10.

**Table 4.** Speed test by scenario. From left to right, full movie inference scenario using MovieDIRec+ full model. Full movie inference scenario using only the client model of MovieDIRec+. Top 10 candidate movie inference scenarios in full model. Top 10 candidate inference scenarios in client model. As for the input size, there is a slight difference between input vectors for each movie, so the approximation is expressed through ≈.

| | **Total + Full Model** | **Total + Client Model** | **Candidate@10 + Full Model** | **Candidate@10 + Client Model** |
|---|---|---|---|---|
| Spend time | 5.1121 s | 2.9961 s | 0.2566 s | **0.1278 s** |
| vs. Full model | - | ↓41.39% | ↓94.98% | **↓97.50%** |
| Input size (91,514 movies) | ≈1.9 GB | ≈183 MB | ≈190 MB | **≈19 MB** |

Analyzing the table, the candidate process and split model approach were very efficient methods in terms of load sharing. By limiting the candidate group to 10, the number of inference objects was reduced by 10 times, but the inference speed was increased by more than 20 times compared to the comparative model. This was not due to the reduced number of inference objects, but was judged to be a load due to the huge input data handling that occurred during the entire reasoning. Figure 5b is a graph measuring the consumed time for inference by increasing the k of Candidate@k from 1 to 100 using it as a related visualization graph. As a result, the second column of Table 4 took longer than the 2.9961 s, which was the time taken for the client model that did not use the candidate. This was a phenomenon caused by the operation of indexing the movies in each cluster together with the load caused by the huge input data handling of the total inference.

## 5. Conclusions

In this paper, we proposed MovieDIRec and MovieDIRec+ using Drafted-Input defined by us. The model prevented overfitting towards identity by configuration and enabled personalized recommendations according to user characteristics. The proposed systems were compared with state-of-the-art methods by dividing the input training case and the without input training case, and it was confirmed that they received a high score.

The proposed system was a system that operated specifically for the Movie domain. However, we judged that if the characteristics of Drafted-Input were well generated and trained, the same performance could be achieved in any domain. In particular, in terms of distribution using a split model, our approach could dramatically reduce the load on the server, and even if device learning was not performed, it would be possible to recommend at a breakthrough speed in fields that require real-time recommendation.

We plan the future work for the system proposed in this paper as follows: Combined with meta learning, it will be improved to uniquely update the client-side model in a federated environment. It was judged that this could provide an advantage in terms of personalization and user satisfaction by providing adaptation through a few-shot explicit feedback. In addition, we plan to add a configuration that allows the training of the model to largely reflect the user's recent preferences. Similar to YouTube's mechanism, changing preferences can be tracked if the user's recent interactions contribute significantly to updating user features.

**Author Contributions:** Conceptualization, H.A.; methodology, H.A.; software, H.A.; validation, H.A.; formal analysis, H.A.; investigation, H.A.; data curation, H.A.; writing—original draft preparation, H.A. and D.K.; writing—review and editing, K.L. and N.M.; visualization, H.A.; supervision, H.A.; funding acquisition, K.L. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) grant funded by the Korean Government (MSIT) under grant 2021-0-00900.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The ml-20m dataset is available at https://grouplens.org/datasets/movielens/20m/ (accessed date 1 October 2016). The IMDB dataset is available at https://datasets.imdbws.com/and https://www.kaggle.com/rmisra/imdb-spoiler-dataset (accessed date 1 October 2021).

**Conflicts of Interest:** The authors declare no conflict of interest.

## References


1. An, H.-W.; Moon, N. Design of recommendation system for tourist spot using sentiment analysis based on CNN-LSTM. *J. Ambient. Intell. Humaniz. Comput.* **2019**, 1–11. [CrossRef]
2. Wei, T.; Wu, Z.; Li, R.; Hu, Z.; Feng, F.; He, X.; Sun, Y.; Wang, W. Fast Adaptation for Cold-start Collaborative Filtering with Meta-learning. In Proceedings of the 2020 IEEE International Conference on Data Mining (ICDM), Sorrento, Italy, 17–20 November 2020; IEEE: Piscataway, NJ, USA, 2020.
3. Ammad-Ud-Din, M.; Ivannikova, E.; Khan, S.A.; Oyomno, W.; Fu, Q.; Tan, K.E.; Flanagan, A. Federated collaborative filtering for privacy-preserving personalized recommendation system. *arXiv* **2019**, arXiv:1901.09888. Available online: https://arxiv.org/abs/1901.09888 (accessed on 1 September 2021).
4. Vančura, V.; Kordík, P. Deep variational autoencoder with shallow parallel path for top-N recommendation (VASP). *arXiv* **2021**, arXiv:2102.05774. Available online: https://arxiv.org/abs/2102.05774 (accessed on 2 September 2021).
5. Lobel, S.; Li, C.; Gao, J.; Carin, L. Towards Amortized Ranking-Critical Training for Collaborative Filtering. *arXiv* **2019**, arXiv:1906.04281. Available online: https://arxiv.org/abs/1906.04281 (accessed on 2 September 2021).
6. Shenbin, I.; Alekseev, A.; Tutubalina, E.; Malykh, V.; Nikolenko, S.I. RecVAE: A new variational autoencoder for top-n recommendations with implicit feedback. In Proceedings of the 13th International Conference on Web Search and Data Mining, Houston, TX, USA, 3–7 February 2020.
7. Steck, H. Embarrassingly shallow autoencoders for sparse data. In Proceedings of the World Wide Web Conference, San Francisco, CA, USA, 13–17 May 2019.
8. Liang, D.; Krishnan, R.G.; Hoffman, M.D.; Jebara, T. Variational autoencoders for collaborative filtering. In Proceedings of the 2018 World Wide Web Conference, Lyon, France, 23–27 April 2018.
9. Kim, J.; Park, J.; Shin, M.; Lee, J.; Moon, N. The Method for Generating Recommended Candidates through Prediction of Multi-Criteria Ratings Using CNN-BiLSTM. *J. Inf. Process. Syst.* **2021**, *17*, 707–720.
10. Paul, C.; Adams, J.; Sargin, E. Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, 15–19 September 2016.
11. Vartak, M.; Thiagarajan, A.; Miranda, C.; Bratman, J.; Larochelle, H. A Meta-Learning Perspective on Cold-Start Recommendations for Items. 2017. Available online: https://papers.nips.cc/paper/7266-a-meta-learning-perspective-on-cold-start-recommendations-for-items.pdf (accessed on 5 September 2021).
12. Bobadilla, J.; Alonso, S.; Hernando, A. Deep Learning Architecture for Collaborative Filtering Recommender Systems. *Appl. Sci.* **2020**, *10*, 2441. [CrossRef]
13. Harper, F.M.; Konstan, J.A. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.* **2015**, *5*, 1–19. [CrossRef]
14. Vozalis, M.G.; Margaritis, K.G. Using SVD and demographic data for the enhancement of generalized Collaborative Filtering. *Inf. Sci.* **2007**, *177*, 3017–3037. [CrossRef]
15. Huseyin, P.; Du, W. SVD-based collaborative filtering with privacy. In Proceedings of the 2005 ACM Symposium on Applied Computing, Santa Fe, NM, USA, 13–17 March 2005.
16. Barathy, R.; Chitra, P. Applying matrix factorization in collaborative filtering recommender systems. In Proceedings of the 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 6–7 March 2020; IEEE: Piscataway, NJ, USA, 2020.
17. Lourenco, J.; Varde, A.S. Item-Based Collaborative Filtering and Association Rules for a Baseline Recommender in E-Commerce. In Proceedings of the 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, 10–13 December 2020; IEEE: Piscataway, NJ, USA, 2020.
18. Yan, C.; Zhang, Y.; Zhong, W.; Zhang, C.; Xin, B. A truncated SVD-based ARIMA model for multiple QoS prediction in mobile edge computing. *Tsinghua Sci. Technol.* **2021**, *27*, 315–324. [CrossRef]
19. Zhuang, F.; Zhang, Z.; Qian, M.; Shi, C.; Xie, X.; He, Q. Representation learning via Dual-Autoencoder for recommendation. *Neural Netw.* **2017**, *90*, 83–89. [CrossRef] [PubMed]
20. Yuan, F.; Yao, L.; Benatallah, B. Adversarial collaborative auto-encoder for top-n recommendation. In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; IEEE: Piscataway, NJ, USA, 2019.
21. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805. Available online: https://arxiv.org/abs/1810.04805 (accessed on 10 September 2021).

22. Hsieh, C.K.; Yang, L.; Cui, Y.; Lin, T.Y.; Belongie, S.; Estrin, D. Collaborative metric learning. In Proceedings of the 26th International Conference on World Wide Web, Perth, Australia, 3–7 April 2017.
23. Kulis, B. Metric Learning: A Survey. *Found. Trends Mach. Learn.* **2013**, *5*, 287–364. [CrossRef]
24. Xing, E.; Jordan, M.; Russell, S.J.; Ng, A. Distance metric learning with application to clustering with side-information. *Adv. Neural Inf. Process. Syst.* **2002**, *15*, 521–528.
25. Daeryong, K.; Suh, B. Enhancing VAEs for collaborative filtering: Flexible priors & gating mechanisms. In Proceedings of the 13th ACM Conference on Recommender Systems, Copenhagen, Denmark, 16–20 September 2019.