VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

SEMANTIC DEEP LEARNING MODELS FOR USER BEHAVIOR PREDICTION

By

Do Pham Minh Thu

Student ID: MITIU19006

Supervisor: Dr. Nguyen Thi Thanh Sang

A thesis submitted to the School of Computer Science and Engineering

in partial fulfillment of the requirements for the degree of

Master of Information Technology Management

Ho Chi Minh City, Vietnam

2021

SEMANTIC DEEP LEARNING MODELS
FOR USER BEHAVIOR PREDICTION

In partial fulfillment of requirements of the Degree of

MASTER OF INFORMATION TECHNOLOGY MANAGEMENT

In IT Management

By

Do Pham Minh Thu

ID: MITIU19006

International University – Vietnam National University HCMC

September 2021

Under the guidance and approval of the committee, and approved by all its members, this thesis has been accepted in partial fulfillment of the requirements for the degree

Approved by:

…….	…….
Chairperson	Committee member
…….	…….
Dr. Nguyen Thi Thanh Sang (Supervisor)	Committee member
…….	…….
Committee member	Committee member

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

I would like to give my profound gratitude to everyone, who helped me to complete this thesis.

First of all, I would like to express my sincere gratitude and deep regard to my supervisor Dr. Nguyen Thi Thanh Sang for her encouragement, trust, and especially her full support and expert guidance throughout the journey from my undergraduate degree up to now.

I would also express my appreciation to all lecturers at the School of Computer Science and Engineering (SCSE) and all visiting lecturers to provide me useful knowledge, support, and encourage me during the MITM program. I also send my thanks to all staff at International University especially at SCSE and the Office of Graduate Program for their help.

I wish to thank all my fellow students especially Ms. Nguyen Ngoc Tram Anh and Mr. Nguyen Trung Nghia because for their support and advice.

Last but not least, I want to give my special thanks to my family for their unconditional love and encouragement.

ABBREVIATION

CBF	Content-based Filtering
CF	Collaborative Filtering
CNN	Convolutional Neural Network
DL	Deep Learning
GMF	Generalized Matrix Factorization
GRU	Gated Recurrent Unit
LSTM	Long-short Term Memory
MF	Matrix Factorization
MLP	Multi-layer Perceptron
NCF	Neural network-based Collaborative Filtering
NeuMF	Neural Matrix Factorization
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
RS	Recommendation System/ Recommender System

ABSTRACT

With the development of the entertainment industry, Recommendation Systems with the purpose of giving personalized suggestions for customers to enhance their user experiences are becoming more and more essential. Deep learning techniques have been studied and applied widely in recommendation systems. However, the lack of semantic relationships between objects leads to inaccurate predictions in terms of semantic. This study aims to propose semantic deep learning models to predict user behavior by combining the domain knowledge base and user interaction data with the integration of deep learning techniques and semantic inference. Firstly, an ontology-based knowledge base is constructed to describe the semantic relationships between objects. Then, the combination of the semantic inference on Ontology and deep learning techniques is applied to user behavior information in order to make predictions. To evaluate the performance of the proposed method, the MovieLens dataset, which is the public and popular dataset, is used to conduct experiments.

CHAPTER 1: INTRODUCTION

With the explosive increase of information, recommendation systems, or recommender systems (RSs) with the purpose is matching user's needs with the items based on their past interactions, plays a more and more important role in a variety of areas such as e-commerce, e-learning, entertainment and so on. The film industry becomes more and more popular makes the need for movie RSs is increasing. Traditional techniques used in RS can be classified into content-based filtering and collaborative filtering [1]. The content-based filtering technique makes a recommendation based on the similarity between the user profile and the item's attributes. This approach can make recommendations with new items and does not require a large number of users. However, it faces the problem of content analysis limitation and over-specialization. On the other hand, the collaborative filtering technique is based on the user behavior (ratings, clicks, comments…) to make suggestions. The recommendations generated can be of different types with the interacted items. This technique also has the limitation that is the cold-star problem and data sparsity.

With the development of deep learning (DL), it is widely applied in many fields especially in RSs. Personalized recommendation techniques are known as: [2] based on ratings, products being viewed, watching time to improve the accuracy. The application of classification machine learning algorithms and collaborative filtering methods [3] can greatly improve the accuracy of the real system. In addition, deep learning models based on tree structure [4] have also effectively combined user behavior information and generated recommendations with more accuracy. Based on the above studies, collaborative filtering is often highly effective when combined with deep learning. And hybrid methods can take advantage of the combination of methods and reduce the costs. The method of deep learning and semantic analysis have been studied; however, it has been applied. In this thesis, semantic deep learning models are proposed to predict user behavior.

The work of this thesis can be divided into three main parts:

- Building knowledge base: in this part, we consider information can be divided into three parts: domain knowledge, textual knowledge, and visual knowledge.
    - For domain knowledge, ontology is used to represent semantic knowledge about related objects and relationships. Then a random walk algorithm and

word embedding method are used to extract semantic representation from the domain knowledge.

  - For the textual knowledge, we use movie summaries to represent the textual knowledge and the embedding method (doc2vec) is used to get the item's textual representation.
  - For the visual knowledge, a movie's poster is used for visual knowledge representation then a CNN pre-trained model is used to extract the feature from the movie poster.

- Building the user behavior analytic model: The semantic knowledge inference on the knowledge base combines with the sequential preference learned from sequential data are the inputs at the collaborative filtering to make predictions.

The rest of this report is organized as follows. **Chapter 2** includes the literature review while the proposed framework will be presented in **Chapter 3**. **Chapter 4** mentions the Implementation while **Chapter 5** presents the Experiment results. Finally, the conclusion and future work are mentioned in **Chapter 6**.

CHAPTER 2: LITERATURE REVIEW

This chapter provides a review of relevant literature to the field of study and some related works. **Section 2.1** provides a brief introduction to RSs. Two main categories of RSs are Content-based Filtering and Collaborative Filtering are also mentioned in this part. The next subpart provides some recent works using Deep Learning techniques - a new trend in RSs. Lastly, some proposed methods that enhancing semantic in the recommendation model are introduced. **Section 2.2** reviews the traditional technique in CF, that is Matrix Factorization. An improvement that combines Matrix Factorization and Multi-layer Perceptron will be presented in this part. **Section 2.3** deals with some specific DL techniques applied in RSs. Firstly, the RNN architecture with its specialized versions LSTM and GRU are mentioned to handle the sequential data. Then, the word embedding methods with word2vec and doc2vec models are presented to apply for textual data. Lastly, for the image data, CNN architecture and its pre-trained model are mentioned especially for feature extraction. **Section 2.4** is dedicated to Ontology, Ontology embedding, and some current works that using ontology to enhance the semantic in RSs.

2.1 Recommendation Systems

2.1.1 Recommendation Systems – an introduction

Due to the rapid development of entertainment platforms especially online movie watching services, RSs with the aim is predicting the preference of specific users on an item, have become more and more popular. RSs bring benefits for both users and service providers [5][6][7]. On the site of users, RSs can help them to reduce the time for finding and choosing items, it leads to the improvement of decision making and the enhancement of user experience. For service providers, the better the user experience, the more revenue they get.

To make suggestions, the RS needs at least one of the following components:

- Item profile: includes the attributes of the item. In the field of movie, the item profile contains the information of movie such as title, directors, actors, genres, and so on
- User profile: it can be the user's personal information or the user-item interaction. The user-item interaction can be explicit feedback (ratings, likes/dislikes, comments…) or implicit feedback (clickstream, purchase history, mouse movements…).

2.1.2 Techniques used in RS

Recommendation techniques can be classified into three main categories that are content-based filtering (CBF), collaborative filtering (CF), and hybrid filtering [1].

Content-based Filtering generates recommendations based on the item profile by finding the most related item to the positively interacted item. So the main task in CBF is calculating the similarity between objects. The advantages of this method are new products can be proposed and do not require a large number of users to achieve high accuracy. CBF also needs a short time to adapt when users change their preferences. However, because this approach depends on the item profile, a rich item description is required to make a recommendation, which calls limited content analysis [1]. One more problem of basing on item's attributes is the over-specialization [1] which means that users will receive recommendations that overwhelmingly fall into their familiar regions. It makes our recommendations cannot be diversified.

Collaborative Filtering, on the other hand, does not pay attention to item profile. The data needed for this technique is the user-item interaction data, then the similarities between user preferences are used to make a recommendation [1]. This approach has two categories that are memory-based and model-based [8]. In the memory-based method, the recommendations are generated in two ways through the item-based approach and user-based approach. While the user-based CF, the interest of user ***u*** in item ***i*** is based on similar users (call neighbors), the interest of user u in item i in item-based CF is predicted through user feedback of items that are similar to i. Correlation-based and cosine-based are the two most popular similarity measures used in CF. With the model-based method, which is most popular in CF techniques, models built based on user-item interaction data are used for making recommendations. The model is built using Machine Learning or Data mining techniques [9] such as Clustering techniques [10], Association Rule [11], Decision tree [12], Support Vector Machine [13], Bayesian Classifiers [14], and so on. CF methods can provide serendipitous recommendations which means that the recommended items are completely different from interacted items. However, this method also has limitations, two of them are the cold-star problem and data sparsity. Cold-star problem [15] is the problem that faces new users and new items. In the case of the new user, the system does not know any information about his/her preference. Similar to that, users have not rated for the new items, so the system cannot give the new item as recommendations. On the other hand, the number of users who give feedback to items

is usually small, and not all items are rated by users, which leads to the lack of information in user-item interactions. This problem is called data sparsity [16].

In addition, there is a **hybrid approach** which is a combination of different methods to making an enhanced recommendation[17]. The main idea is that combining different methods can inherit the advantages and suppress the drawbacks of individual ones. Different methods can be used to build hybrid RSs [18], such as feature combination, feature augmentation, cascading, switching method, and so on.

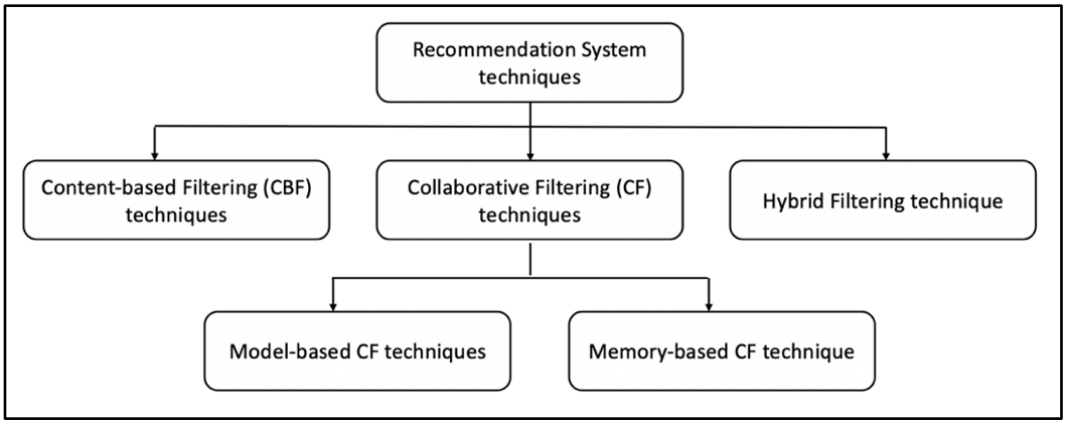

Figure 1: Techniques in Recommendation Systems

2.1.3 Deep Learning – the new trend in RS techniques

In recent years, the development of deep learning especially neural network-based models has brought successful results in many areas such as image processing, natural processing, speech recognition, and also recommendation systems.

Some recent works show that **Recurrent Neural Network** [19] works well for session-based RSs and sequential RSs. In the new recommendation model proposed by Song [20], they presented a multi-rate Long-Short Term Memory (LSTM) that considers not only long-term static but also short-term user preference. Smirnova et al. [21] showed that the combination between historical interactions and context-aware using conditional RNNs gives a better recommendation. Hidasi et al. [22] proposed a GRU4Rec model for session-based recommendations. With the input is the 1-of-N encoding vector represented for the actual state of the session, the output is the likelihood of the next item in the session. To make the training more effective, the authors proposed the session-parallel mini-batches algorithm and sampling method for the output.

Convolution Neural Network (CNN) is widely applied in RSs to extract features. He et al. [23] proposed a model by integrated visual Bayesian personalized ranking with visual features extracted via CNNs. Nguyen et al. [24] proposed a CNNs-based personalized tag recommendation model by integrating visual features of images extracted using convolutional and max-pooling layers with user information. Lee et. al [25] improved the performance of RS by using the CNN model to extract audio and musical features.

Many works use Multilayer Perceptron (MLP) [26], [27], Autoencoder based CF [28], [29], Neural Attention-based model [30]–[32] and so on, demonstrate the effectiveness of DL techniques in RSs.

2.1.4 Semantic enhancement in RSs

Recent works [33]–[35] show that the enhancing of the knowledge base in RSs can boost performance.

Zhang et.al [33] proposed the recommendation model with the integration of collaborative filtering with item's semantic representation from the knowledge base including structural content, textual content, and visual content. Item's structural knowledge encoded by TransR [36], the textual and visual feature vectors extracted with the autoencoder architecture respectively.

Yang et al. [34] proposed the model that movie/user representations are incorporated various feature vectors: knowledge embedding from knowledge graph and tag embeddings from tag corpus. To be more specific, a movie's embedding is the combination of entity embedding and context embedding extracted from a knowledge graph, and the user's embedding is the average of his/her historical favorite movie's knowledge embedding. Moreover, tag embeddings encode co-occurrence relationships between tags indicating more correlation between users and movies.

Huang et. al [35] proposed the novel knowledge-enhanced sequential recommender that integrating an RNN-based network with a Key-value Memory Network (KV-MN) [37]. The RNN-based network is used to capture sequential preference and KV-MN is for capturing the attribute-based user preference.

2.2 Collaborative Filtering in RSs

2.2.1 Traditional Collaborative Filtering: Matrix Factorization

Matrix factorization [38], [39] is an approach that characterizing users and items by vectors of latent factors interpreted from user-item interaction to explain the ratings. Some common matrix factorization models used in RSs are Single Value Decomposition [40], Probabilistic Matrix Factorization [41], and Non-negative Matrix Factorization [42]. Matrix factorization is an unsupervised learning method for latent variable decomposition. The main idea behind this approach is that in RSs, there exist latent features that represented the relationship between users and items. To be more specific, from the rating matrix R ($n \times m$) which contains ratings from *n* users for *m* items, this method is decomposing matrix R into two matrices P and Q. Matrix P which has ($n \times f$) dimensions where f is the number of latent factors, contains latent vectors of users. Similar to that, matrix Q which has ($m \times f$) dimensions is used to represent the items. The dot product of matrix P and the transpose of matrix Q is closely approximating the matrix R.

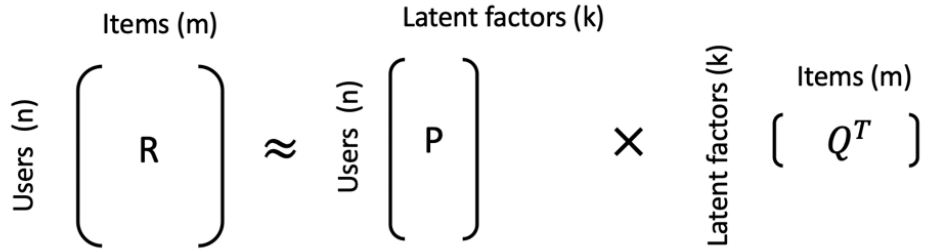

Figure 2: Decomposing rating matrix into latent factors matrices

Each item is associated with the vector $q \in Q$ while each user is associated with the vector $p \in P$. To predict the preference of user *u* on item *i*, we take the dot product between p_u and q_i^T:

$$\hat{r}_{ui} = p_u \times q_i^T$$

Stochastic Gradient Descent and Alternating Least Square [38] are two common methods to minimize the loss function.

2.2.2 Multi-layer Perceptron

Multi-layer Perceptron [43] is a feed-forward neural network that includes three types of layers: input layer to receive the signal, hidden layers to do computations, and output layer to predict or make a decision about the input (Figure 3).

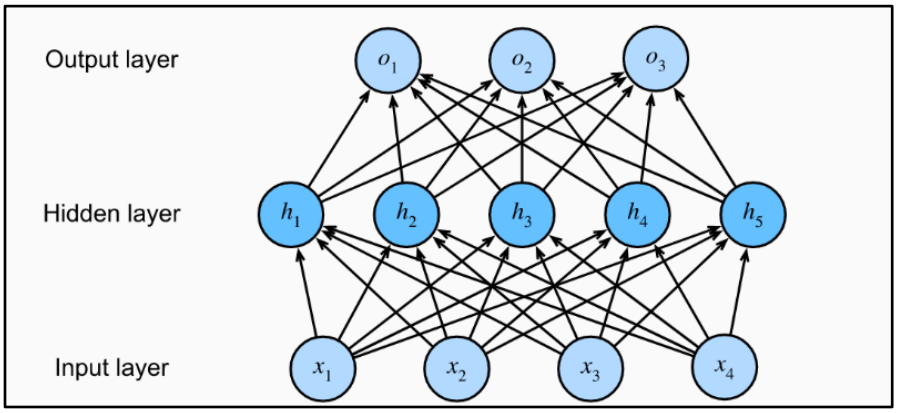

Figure 3: An example of MLP [43]

Each node in the layer is called a unit, so units corresponding to the input layer, hidden layer, output layer are called input unit, hidden unit, and output unit. In the forward pass, the signal from the input layer goes through the hidden layers to the output layers. The output of each unit (except the input unit) is calculated using the activation function.

$$a^{(l)} = f\,(W^{(l)T} a^{(l-1)} + b^{(l)})$$

where $a^{(l)}$ is the output of layer l, $W^{(l)}$ is the matrix weight of layer l to represent the connection between layer l-1 and layer l, $b^{(l)}$ is the biases of layer l and $f(.)$ is the activation function. Some common activation functions, such as, rectified linear units (ReLU) [44], tanh function and sigmoid function [45].

The important thing in training the MLP network is that weight adjusting. The back-propagation algorithm [46] is the training algorithm commonly used in MLP networks. There are two phases: propagation (forward) as mentioned above, and back-propagation (backward). In the back-propagation, the loss function is calculated and the weights are updated with the purpose is minizine the loss function. This process is repeated until meeting the acceptable error or exceeds the defined maximum number of loops, also known as epochs.

2.2.3 Neural Collaborative Filtering

He et al. [26] proposed the **Neural Collaborative Filtering** that used MLP to learn user-item interactions. The architecture of NCF is shown in Figure 4 which includes:

- Input Layer: contains the one-hot vectors to identify users and items
- Embedding Layer: projects the one-hot vector to an embedding vector which is also the latent vector of user/item.
- Neural CF layers use the Multi-layer neural architecture to model the user and item latent features. The input of one layer is the output of the previous layer. The final output layer is the predicted score. The user and item identities are used as the input features, forming the binarized vector with one-hot encoding.

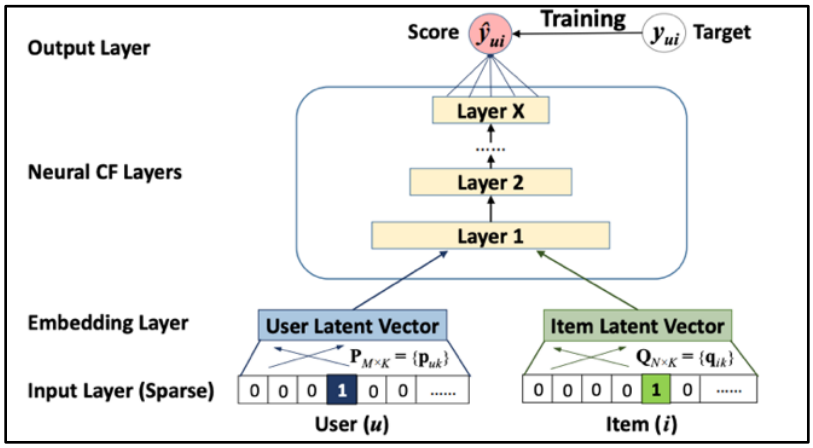

Figure 4: Neural Collaborative Filtering framework [26]

The predicted value at NCF is calculated as follows: $\hat{y}_{ui} = f\ (P^T v_u^U, Q^T v_i^I \mid P, Q, \theta_f)$ where P is the latent factor matrix for users, Q is latent factor matrix for items, $f(.)$ is MLP and θ_f are the model's parameters

Neural Matrix Factorization is the combination between generalized matrix factorization and Multi-layer Perceptron. GMF uses the linear kernel to model user-item interaction, while the nonlinear interactions are modeled using multi neural layers of MLP, then the concatenation of those outputs is used to calculate prediction scores. The NeuMF architecture is shown in Figure 5:

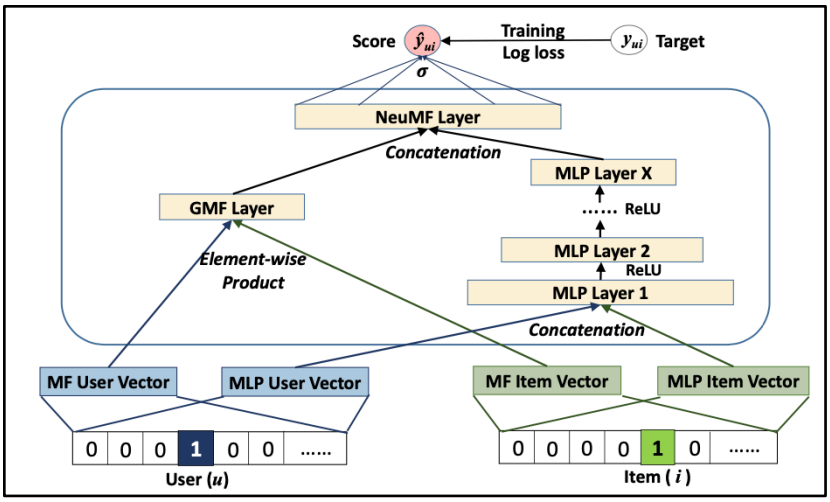

Figure 5: Neural Matrix Factorization architecture [26]

To be more specific, the user and item embeddings in GMF and MLP are different to make sure that the optimal embeddings are learned independently. GMF layer takes the element-wise product of MF user vector and MF item vector as input, while the input of MLP layer is the concatenation of MLP user vector and MLP item vector. Finally, the output of GMF and MLP are concatenated to become an input of NeuMF layer.

2.3 Deep Learning in RSs

2.3.1 For sequential data: RNN with its specialized versions LSTM and GRU

Recurrent Neural Network [19] has proved its high performance in modeling sequential data. The internal hidden state which is the function of the actual input and the previous value of the hidden state to summarize the processed sequence makes the difference between RNN and feedforward layer. The hidden state value in RNN is computed using activation function $f(.)$ as: $h_t = f(Wx_t + Uh_{t-1} + b)$ where x_t is the input at time step t, h_{t-1} is the state at the previous timestep, W is the matrix of weights between the input and the hidden layer, U is the recurrent weights matrix between the hidden layers at adjacent timesteps, and b is the bias.

Recurrent layers are trained using backpropagation through time [47], propagating a sequence forward through the RNN for T steps and computing gradient concerning the weights for all T steps. The multiple RNN layers can be used after one another to process data. The output of the previous hidden state is the input of the next hidden state.

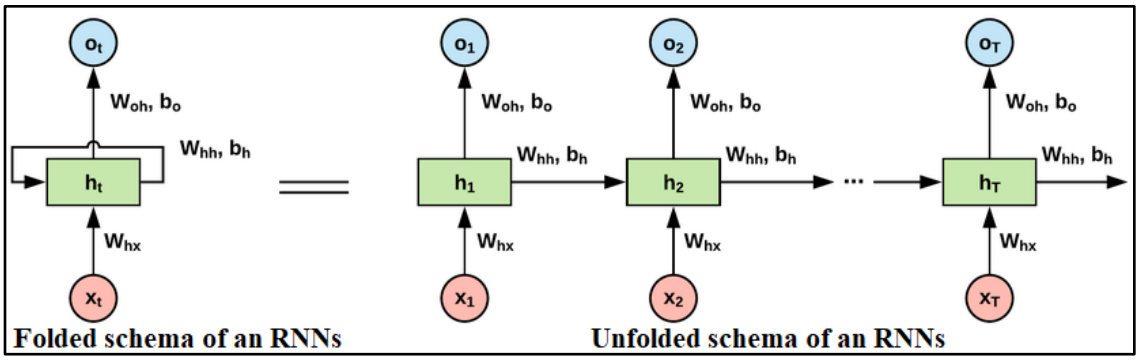

Figure 6: Recurrent Neural Network Schema [19]

Traditional RNNs are not good to capture the long-term dependencies caused by the vanishing/exploding gradients [48] due to the recursion in the structure. To overcome this problem, Gated Recurrent Unit and Long-short Term Memory which are the variants of RNN, are introduced.

Long short-term memory [49] is a specific version of RNN that gives higher accuracy than RNN when processing transitory sequences and long-term dependencies. It has the form of a chain that contains repeating modules (Figure 7), each module is a four-neural-network-layers.

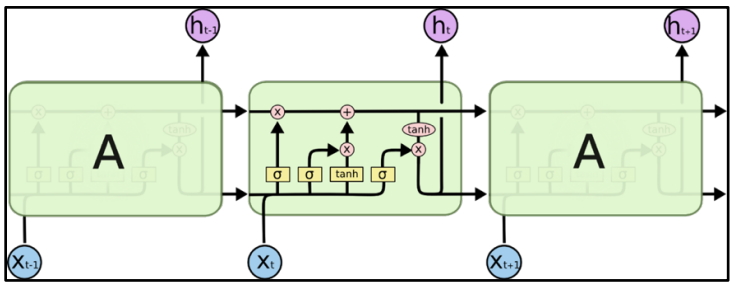

Figure 7: The LSTM architecture [49]

In figure 7, the output of one node to the inputs of the other node is a vector carried in each line. The yellow boxes are the activation function used by the neural network layer which is usually sigmoid and tanh nonlinear functions. The pink circle is the pointwise operation of vectors such as addition, scalar multiplication. The vector's content is being copied and goes to varying locations is represented by the line forking. Integration is denoted by the lines merging.

The core idea of LSTM is a cell state which is the horizontal line running through the top as shown in Figure 8. The LSTM has the ability to add, remove and regulate the flow of information through the cell state using gates. Each gate includes the sigmoid activation function

and pointwise multiplication operation. The sigmoid layer returns the value from zero to one, in which zero means removing all and one means keeping all.

Steps in LSTM:

- Step 1: Forget gate layer determines which information can go through the cell state using the sigmoid function. The input is h_{t-1} and x_t; the output is the value in the range [0,1] with 0 means remove all and 1 means keep all.

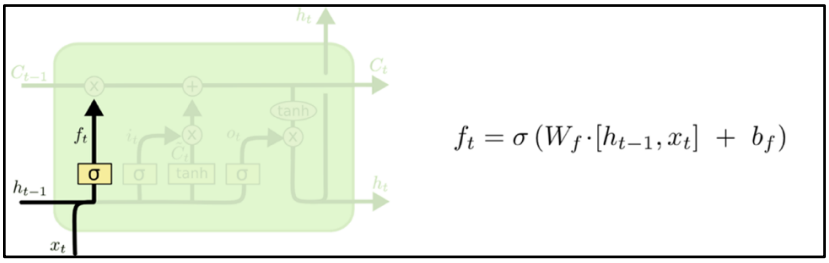

Figure 8: Forget get layer in LSTM

- Step 2: In this step, information that should be stored is decided by two parts. Firstly, the sigmoid hidden layer known as the input gate layer determines how much value should be updated. Then, the tanh hidden layer generates a new state vector $\tilde{C}_t$

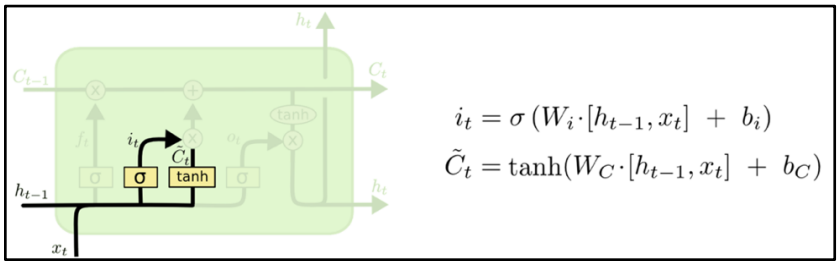

Figure 9: Update value of state vector using input gate layer and the tanh hidden layer

- Step 3: Update the cell state value from C_{t-1} to C_t.

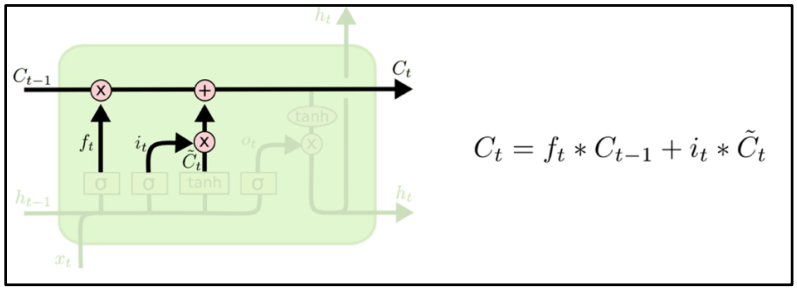

Figure 10: Update value of the cell state

- Step 4: The final output will be filtered using the sigmoid layer and tanh function.

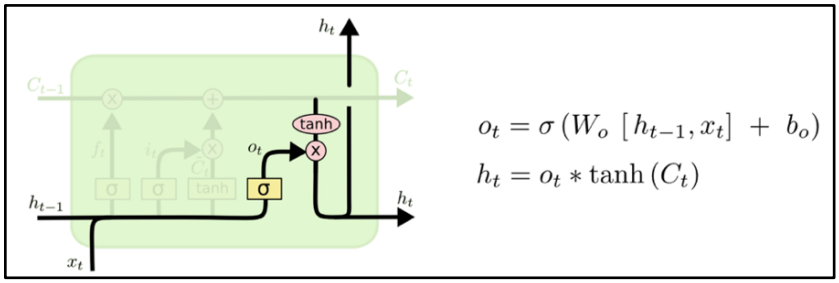

Figure 11: Determine the final output of the LSTM module

Gated Recurrent Unit [50] is also a specific version of RNN which has a similarity in design with LSTM. GRU components include two gating signals called reset gate and update fate, current memory content, and final memory at the current time step.

- Update gate z_t: decide how much information should be passed to the next step. The input x_t and the output of the previous unit h_{t-1} will multiply with corresponding weights. The sum of two values is put into a sigmoid function, then the output will be in the range [0,1] with the meaning is similar to LSTM.

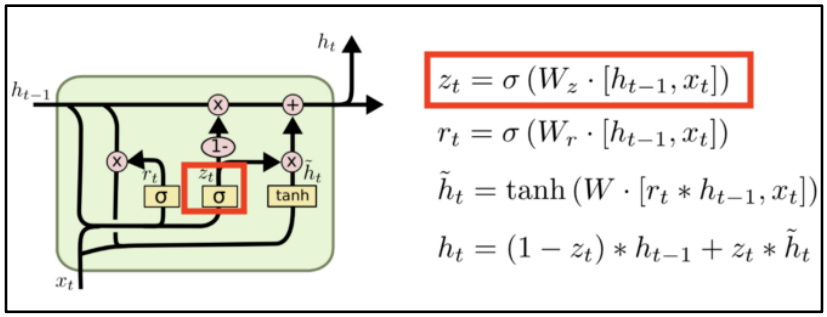

Figure 12: Update gate mechanism in GRU

- The reset gate is used to decide how much past information should be forgotten. The formula is similar to update gate but with different weights and usage.

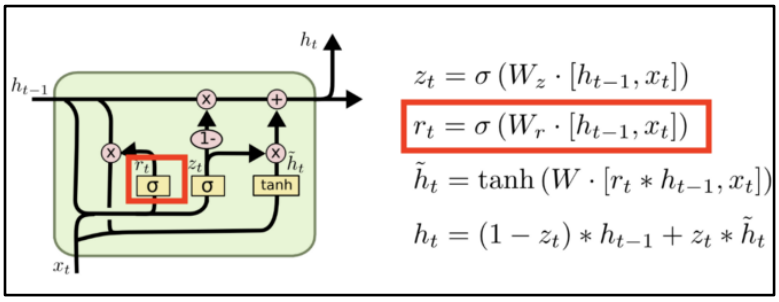

Figure 13: Reset gate mechanism in GRU

- Current memory content: The input x_t and the element-wise multiplication of r_t and h_{t-1} (to pass only relevant past information) will multiply with corresponding weights. The sum of two values is put into a tanh function.

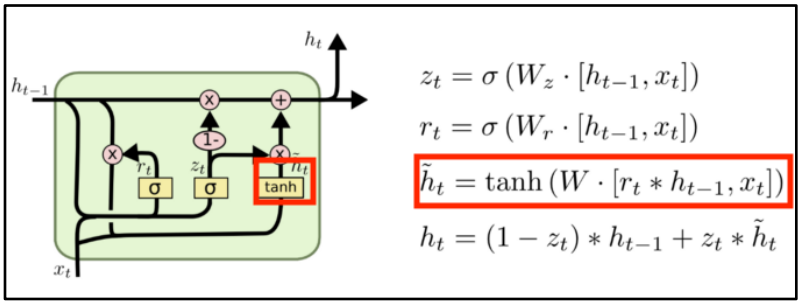

Figure 14: Current memory content mechanism in GRU

- Final memory at current time step: The update gate z_t hold the key point in this process. If the z_t is close to 1 means a big portion of the current content will be kept. At the same time (1- z_t) is close to 0 which means that the past information should be ignored and vice versa.

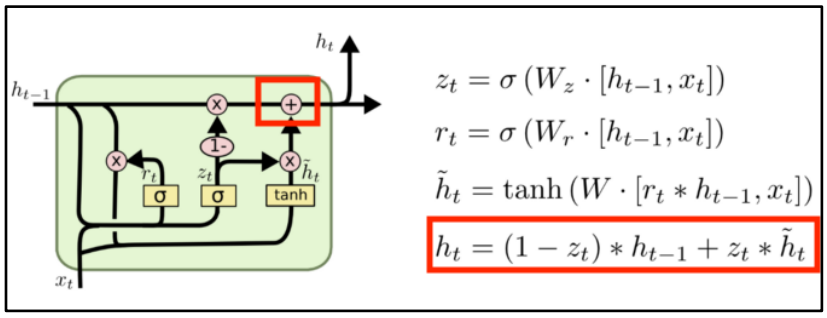

Figure 15: Final memory at current time step mechanism in GRU

The GRU's structure is simpler than LSTM (one gate less than LSTM) so it can save time because of the reduction in matrix multiplication. But in the empirical research [51][52] in the Natural Language Processing field, GRU gives better performance with long text and small datasets while LSTM performs better in other scenarios. However, the GRU values are close to LSTM values but lesser run time.

2.3.2 For textual data: word2 and doc2vec

The first thing when working with textual data is converting strings to the number or text vectorization. **One-hot encoding** is the simplest way to encode the word. The vector of each word has a size equals to the length of vocabulary and the value at the position corresponding to the word is equal to 1. The larger the vocabulary size, the larger the one-hot vector size, then it leads to a large number of memories to store those vectors. One more thing is that two words that have similar meanings should have similar representations. However, the one-hot vector can not capture any relationship between words. **Word embedding** is introduced to solve this problem. The job of word embedding is projecting words into vector space to represent them in a lower-dimensional space and make sure that, the closer the meaning between words, the closer their vector representations are.

Tomas Mikolov et al. [53] proposed the model **Word2vec** which is the effective word embedding using a two-layer neural network. Word2vec takes the text corpus as input and produces feature vectors for each word in that corpus. Word2Vec is a feed-forward and fully connected neural network with two important models inside are Continuous Bags of Words (CBOW) and Skip-gram. While the Skip-gram model tries to predict the neighbors or context based on a target word, the CBOW model tries to predict the word given its neighbors or text. So, these models are similar except for swapping the input and output. CBOW treats an entire context as one observation while skip-gram treats each context-target pair as a new observation. So that is why CBOW is better for smaller datasets and skip-gram works well with the larger ones.

The architecture of the word2vec model can be described as follow:

- The input is the one-hot vector(s) with the size is vocabulary size V
- The hidden layer is the fully connected layer. The weight between the input layer and hidden layer is the matrix W (V x N with N is hidden layer size) and the activation function is linear.
- The output layer produces the probability of the target word(s). The weight between the hidden layer and output layer is W' (N x V) and the activation function is the softmax function.

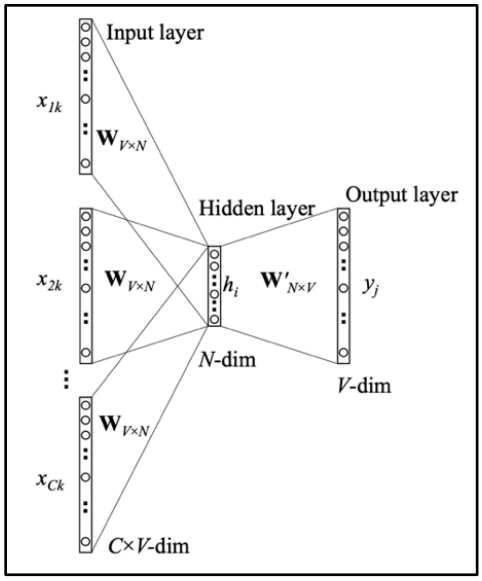

Figure 16: Continuous Bag-of-word architecture [54]

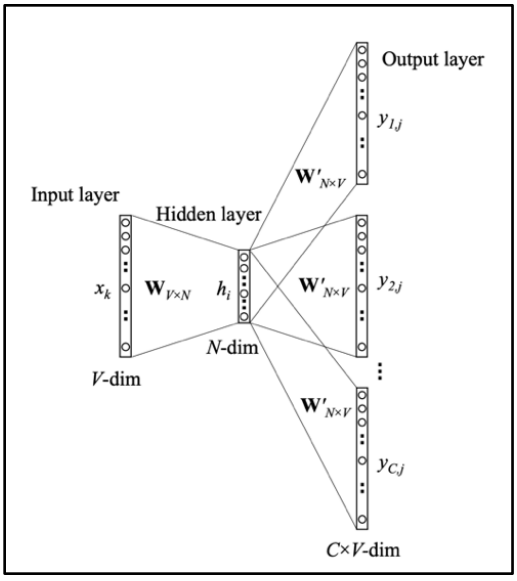

Figure 17: Skip-gram architecture [54]

Doc2vec [55] proposed by Le and Mikolov in 2014 is an extension of word2vec that is applied to produce vector representations of text such as sentences, paragraphs, and documents. Doc2vec also has two approaches: distributed memory (DM) model which has the same idea as the word2vec CBOW model, and distributed bag-of-words (DBOW) model which works similarly to the word2vec skip-gram model. The DBOW model does not care about the word order, faster training and does not use local context. It inserts a "word" ParagraphID which is represented the trained paragraph. While in the DM model, a paragraph is treated as a word and joins this word to the sentence.

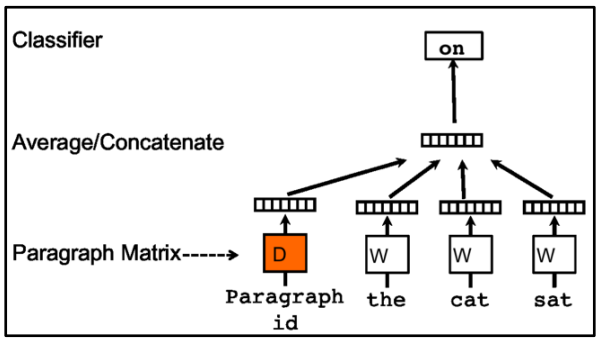

Figure 18: doc2vec distributed memory model [55]

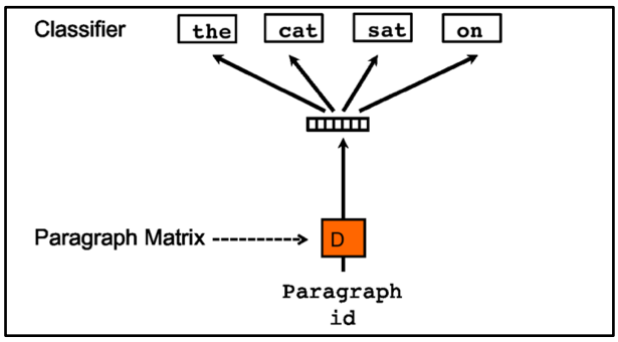

Figure 19: doc2vec distributed bag-of-words model [55]

Before putting the corpus to embedding model, some text pre-processing techniques [56][57] are applied, for example:

- *Lowercasing* all the words is applied to make the consistency in the expected output. For example, the two words "computer" and "Computer" are treated as different in the word embedding method. However, these two words are similar.
- *Tokenization*: is the process that breaks down text into units such as characters, words, or n-grams. All text vectorization processes consist of applying some tokenization scheme.
- *Stemming* is used to reduce inflection in words. It works by cutting off the end or the beginning of the word based on the list of common prefixes and suffixes. For example, the words "walked", "walking", "walks" could all be reduced to their root form "walk". The "root" in this case may be a canonical form of the original word. Porter stemming algorithm [58], Lancaster stemming algorithm [52], and Snowball Stemming algorithm [60] are commonly stemming algorithms.
- *Stop Word Removal:* Stop words are the frequently appeared words in the sentences but are meaningless. Some examples are "a", "the", "is", "and", and so on. These words have high weights in the sentences but do not contribute to the context or content. So, to focus on the important words, those stop words should be removed.
- *Lemmatization* has a similar idea to stemming, which transforms words into their root form. The difference is that lemmatization does not chop things off like stemming, it transforms to actual root using dictionary mappings and some special rule-based approach [61].

Basing on the word2vec and doc2vec, many concepts are produced such as seq2vec [62], item2vec [63], node2vec [64], and so on. Kimothi et.al proposed the seq2vec method to encode the biological sequence into vector space. Each sequence is considered as a document, then the doc2vec model is used to learn the sequence representation. Barkan et. al proposed the item2vec model for item-based collaborative filtering that considers each user sequence as a sentence and each item as a word. Node2vec uses the DeepWalk algorithm[65] to “walk” through the nodes, and the Skip-gram model to generate embeddings from the walk.

2.3.3 For image data: Convolutional Neural Network and pre-trained models

A convolutional neural network (CNN) [19], [66] which is successfully applied to image recognition, object recognition, and audio processing, is a feed-forward neural network that has at least one layer that uses convolution operation instead of matrix multiplication. A typical CNN which has three components: convolution layer, pooling layer, and fully connected layer, is recurrences of a stack of several convolution layers and pooling layer, followed by one or more fully connected layers.

In the **convolution** layer, a kernel is applied across the input, calculates the element-wise product between each element in the kernel and input tensor and their summation is the output value in the corresponding position in the feature map (output tensor). The key point is that the kernels are shared across all image positions.

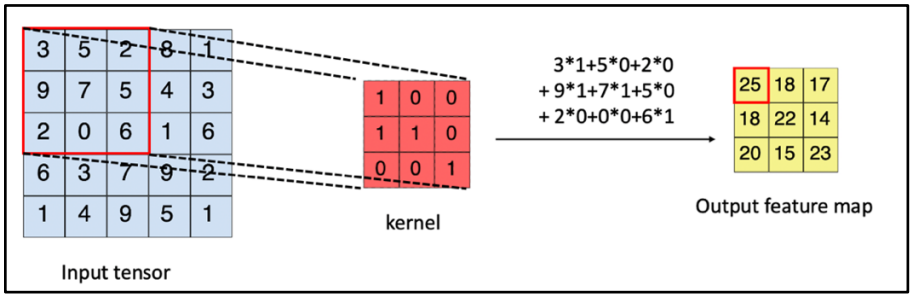

Figure 20: Example of the convolution operation

However, the center of each kernel cannot overlap the outermost elements of the input tensor. The size of the feature map is reduced compared to the input tensor. To solve this problem, a technique called padding is introduced. The most common padding technique is zero padding, in which each size of the input tensor is added with zeros (Figure 24).

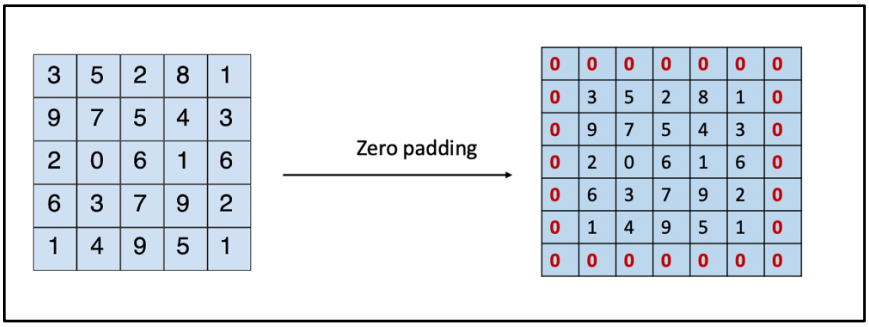

Figure 21: Input tensor after applying zero padding

Stride is the parameter that determines the amount of movement. For example, if the stride is 1, it means that the kernel moves one pixel or unit at a time. The commonly used stride in CNN is 1.

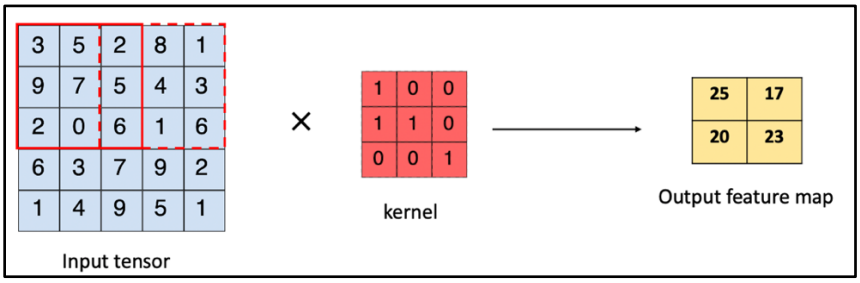

Figure 22: Convolution with stride is 2

To introduce nonlinear to the model, the nonlinear is used to transform the convolved feature. ReLU is widely used in CNN, which is return values of x if x >0, otherwise return 0: $\boldsymbol{f(x) = max(0, x)}$.

Pooling reduces the dimensionality of the feature maps but still keeping critical feature information to decrease processing time. Max pooling is the most popular operation used in this layer. A filter slides over the feature map and outputs the maximum value in each patch.

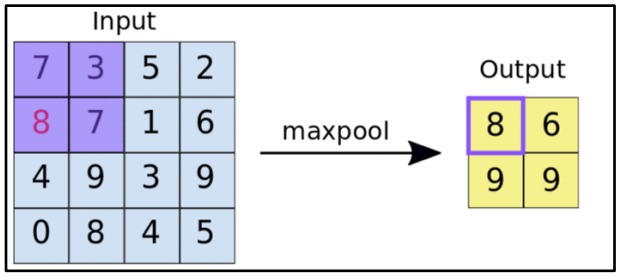

Figure 23: Max pooling with filter size 2x2 and stride is 2

In the end, one or more fully connected layers are used to make predictions.

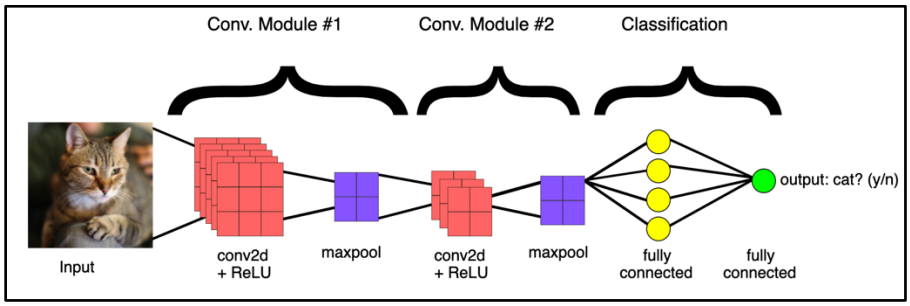

Figure 24: An example of CNN model for classification problem

Some most popular CNN architectures:

- VGG16 [67] has 13 convolution layers and three fully-connected layers. VGG16 introduced the concept of block which consists of repeating CNN layers. This is the first architecture that stacking multiple convolution layers before the max-pooling layer. ReLU is an activation function and the kernel size is 3x3. Starting from VGG16, a common pattern for CNN is using the block of the form [2 convolution layer + pooling layer]. The number of parameters in VGG16 is 138 million.

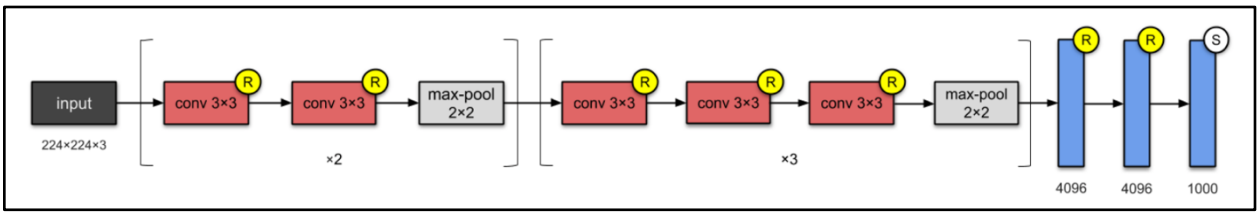

Figure 25: VGG16 Architecture

- The Inception-V1 [68] can answer the big question about the most suitable size of kernels in CNN. They proposed the Inception block architecture. In the inception block, 4 parallel branches use filters with sizes 1x1, 3x3, 5x5 are applied to extract a variety of features. The inception block is repeated 7 times and the total of layers in Inception-V1 is 22 but the number of parameters is down to 5 million.

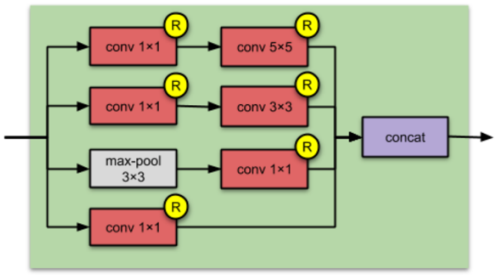

Figure 26: Inception block in Inception-V1 Architecture

- Previous architectures often improve accuracy by increasing the depth of the CNN network. But experiments show that up to a certain depth threshold, the accuracy of the model will saturate and even backfire and make the model less accurate. When traversing too many layers of depth can cause original information to be lost, Microsoft researchers solved this problem on **ResNet** [69] by using a shortcut connection. The main contribution of ResNet is a residual block or identity block. Instead of fitting using underlying mapping, they let layers a residual mapping. As shown in Figure 30,

the formulation of $F(x) + x$ can be realized by feed-forward neural networks with shortcut connections. In shortcut connections, it performs identity mapping, and outputs are added to the output of stacked layers.

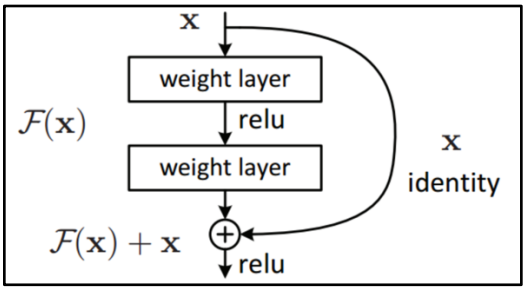

Figure 27: Identity block in ResNet Architecture

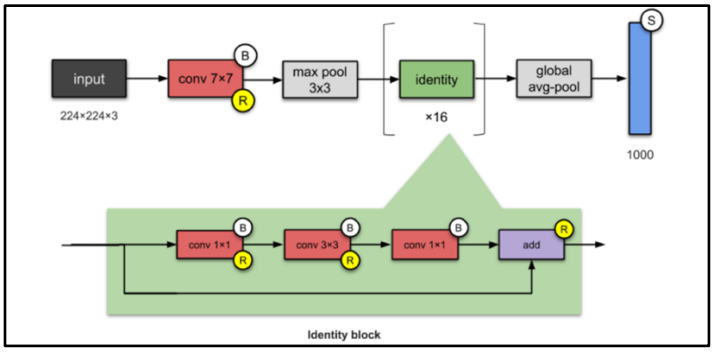

Figure 28: ResNet Architecture

- Having a similar idea with ResNet, DenseNet [70] uses a dense network of shortcut connections to link blocks together. From the input x, we apply the sequence of consecutive mappings with the complexity is increases. In DenseNet, they do not add x directly to f(x), the output of each mapping will be concatenated. Next, the transition layer which is a combination of convolution layer and max-pooling is applied to reduce data dimensions.

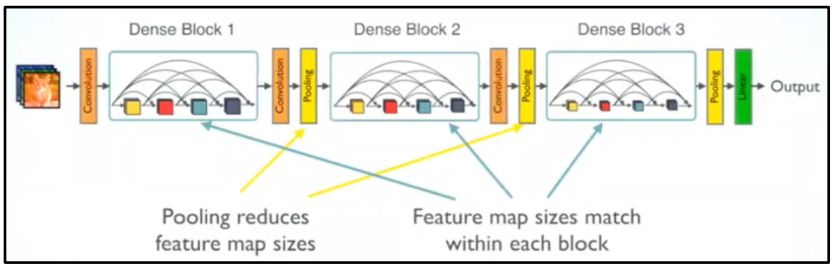

Figure 29: DenseNet Architecture

2.4 Ontologies

2.4.1 Ontology – an introduction

With the increase in using Semantic Web, Ontologies which can be seen as an abstraction in data modeling [71], is used as a method to share definitions and rules in domain knowledge. It serves the same purpose for data models such as database schema but in a higher level of abstraction. In the database, a tuple in the table must follow constraints such as datatypes constraint, primary and foreign keys, or check constraints. Similar to that, ontology must conform to the axioms expressed in that ontology. In comparison with regular databases, the expressiveness in ontology is higher and closer to first-order logic, makes ontology is considered to be at the semantic level.

The ontologies can be used as a description of knowledge about a specific domain. The vocabulary provided by ontology describes the domain with the meaning of terms and the relationship between terms. The main components in ontology include axioms, classes, individuals, relations, attributes [72]

- Individuals (instances), which is the basic component is ontology, are specific objects such as books, movies, or abstract things like numbers and words.
- Class (or concept) is a set of different objects which shared common characteristics. Each class can consist of individuals, other classes, or a combination.
- The attribute is used to describe the objects in the ontology. Each attribute includes name and value.
- Relation describes the relationship between objects in the ontology.
- Rules (Constraint) bound the value in the relationship and attributes.
- Axioms are statements that are true in a specific domain.

2.4.2 Web Ontology Language (OWL)

Resource Description Framework (RDF) [73], [74] is the fundamental semantic model used to encode and share data. The ideas of RDF are:

- An entity, a concept, or a class uses a Uniform Resource Identifier (URI) as a global identifier across system or domain boundary.
- Information is encoded by a triple-based statement. The form <s, p, o> is used to represent the binary relations, which means that the resource s holds the relation p with

the resource o. In RDF, ***p*** is called the property of ***s***, ***s*** is the domain of ***p*** and ***o*** is the range of ***p.*** For example, <ex: Software Engineering, ex:isMainCourse, ex: Computer Science> to encode the relationship in bachelor program. Each triple is an annotated edge where the start node, edge, and end node are labeled with URIs or text strings.

To describe the vocabularies (terms) used in statements, the extension provided by RDF vocabulary description language - RDF Schema is used. It also offers simple object-oriented modeling such as class and relation hierarchy. **Ontology Web Language** (OWL) [75] offers more facilities to express semantic and define vocabularies than RDF Schema. OWL adds more vocabulary to describe properties and classes.

2.4.3 DeepWalk algorithm

A graph is also an effective method in the representation of objects and their relationship. **Graph embedding** has the same idea as word embedding, denotes the entire graph, subgraph, individual nodes, and also edges into vector. One of the embedding strategies is using deep learning methods. The idea is to use random walks combined with the neural network to train the embeddings. The number and size of the walk on the graph are independent of the number of nodes and relationships in the graph, so it is easy to scalable.

Random Walk algorithm [76], [77] is a random process that provides random paths generated from a graph. It simply says that start from one node, the next node is reached random or based on provided probability distribution. By using random walks can gives two benefits. First, several random walkers with different threads or processes can simultaneously explore different parts of the graph. Second, based on the obtained information from the random walk, we can make small changes in the graph structure.

DeepWalk [65] is the combination of random walk and Skip-gram model for graph embedding. Firstly, DeepWalk generates a corpus D using random walks on the graph. Then skip-gram word2vec model is applied on D to generate embedding vectors.

2.4.4 Ontology in RSs

Ontology is used to integrate heterogeneous information to make recommendations. The ontology helps to model the long and short terms of user preference [78], [79] by using the concepts of domain ontology with the items that the user interacted with. In news feed recommendations [73],[74], ontology stores the concepts and relationships between the items, then recommendations

are made by providing the concept for new contents and determining the semantic relations between concepts and terms. Fraihat et. al [81] proposed a semantic recommendation system for e-learning to give personalized learning objectives for students and new resources to enhance syllabuses. This system using semantic relations and ontology reasoning for the keyword query. Ibrahim et. al [82] proposed a recommendation model for making personalized course suggestions using ontology to integrate information from multiple sources (course information, student information, and career information) and build an ontology mapping between the user profile and item profile.

CHAPTER 3: PROPOSED FRAMEWORK

This section describes an approach to build semantic deep learning models for user behavior predictions. **Section 3.1** provides the problem definition with some notations used in this study. The main part of this section is the proposed framework with a detailed explanation in each module will be shown in **Section 3.2.**

3.1 Problem definition

In this study, we build the recommendation model based on explicit feedback and auxiliary knowledge bases.

We introduce some notations used throughout the paper. Let s denote a set of users and $\mathcal{I}$ denote a set of items. In our work, explicit feedback is used which is the rating of the user on the item. The notation r_{ui} is represented for the rating score of user u on item i. From the explicit feedback, we can convert to sequences by sorting the interacted item in time ascending with the format $\{\boldsymbol{i_1^{(u)}}, \boldsymbol{i_2^{(u)}}, \ldots, \boldsymbol{i_t^{(u)}}, \ldots, \boldsymbol{i_{n_u}^{(u)}}\}$ where $i_t^{(u)}$ is the interacted item at time t and n_u is the number of interacted items of user u.

In the auxiliary data domain, we have the knowledge base model which includes three parts: domain knowledge is an Ontology model that represents items, features, and their relations; textual knowledge of movie summaries, and visual knowledge of movie posters. Building a knowledge base can provide deep knowledge and rich semantics on items.

Based on these preliminaries, we are ready to define the recommendation model. Giving the list of interacted items with rating scores of a user, we would like to predict the preference of that user to un-rated items.

3.2 Proposed Framework

The framework which is illustrated in the figure below consists of two main phases. The first one is the generation of embedding vectors from different sources to represent the user and movie. The second one is feeding the user and movie representations into a neural collaborative filtering model to predict the rating score.

To be more specific, we start with the knowledge base representation with domain knowledge using Ontology, textual knowledge and visual knowledge, and sequential learning user behavior using deep learning models such as Doc2Vec or LSTM. Then the embedding vectors for both users and items are the inputs of NeuMF to predict the rating score.

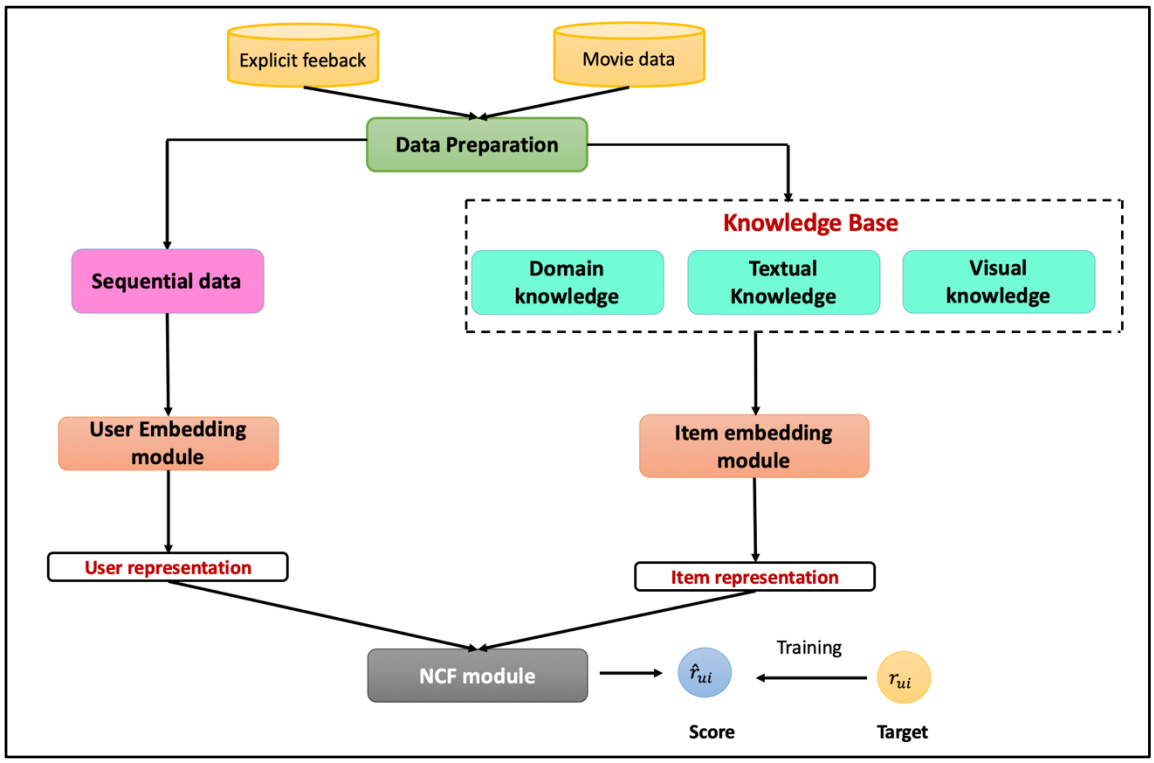

Figure 30: The proposed framework

3.2.1 Data preparation

To build the knowledge base, we need the movie descriptions. In this study, we use the MovieLens dataset [83]. The information of movies in this dataset includes titles and genres. So, to make rich domain knowledge, we need more information. To do that, we combine the information in the MovieLens dataset with the IMDB dataset [84] to make the knowledge base. Each movie will have these features: movieId, title, genres, directors, writers, casts, country (where the movie is produced), runtime, year, movie summary, and poster.

3.2.2 Building knowledge base

In this module, we build the knowledge base model. The information stored in the knowledge base can be divided into three parts: domain knowledge, textual knowledge, and visual knowledge

- **Domain knowledge** is built using Ontology. The needed information is movieId, title, genres, directors, writers, casts, country, runtime, and year. The ontology model for movies recommendation systems is designed as follow:

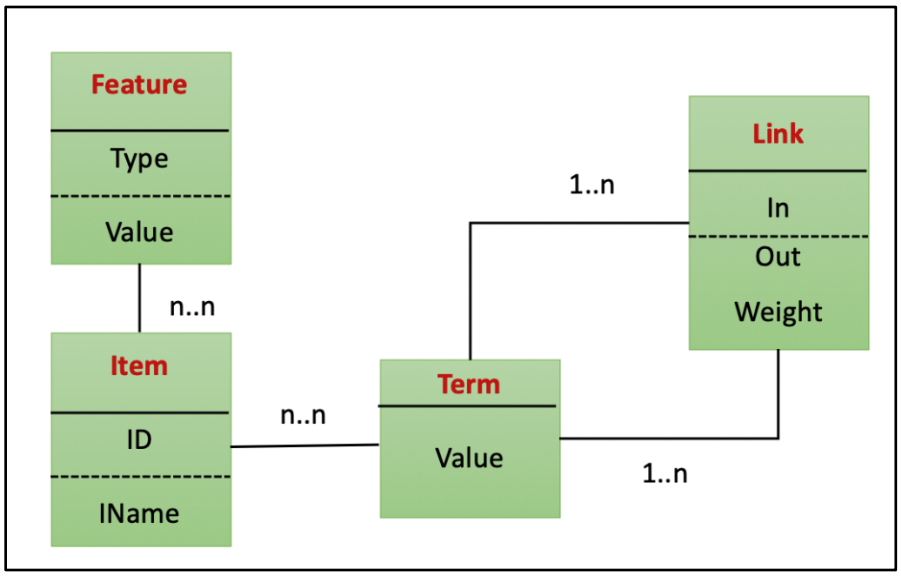

Figure 31: An ontology model for domain knowledge

To be more detailed:

  - *Item* represents the movie (movieId)
  - *Term* represents keyword in the title. A term can collocate with other terms.
  - *Link* represents a connection between 2 terms: in refers to the previous term (in-term), out refers to the next term (out-term), weight is the number of times this link appears.
  - *Feature* which includes genres, directors, writers, casts, runtime, year, country, has two attributes: Type (for example genres) and Value (for example Comedy).

- **Textual Knowledge:** for an item movie, a summary is used to represent the textual knowledge, which usually gives the main topics/content of the movie.
- **Visual Knowledge** is represented by the movie's poster images.

3.2.3 Generate item representation vector

The easiest way to represent an item into a vector is using a one-hot encoding vector. However, one-hot vectors do not include the semantic meaning. Some recent studies [33]–[35] have proved that enhancing semantic in RSs makes better recommendations. In this study, we assume that each item's embedding has three components: an ontology embedding that captures from the domain knowledge, a textual embedding that capture the semantic from the movie's summary, and a visual embedding that is extracted from the movie's poster. The flowchart is shown in the figure below:

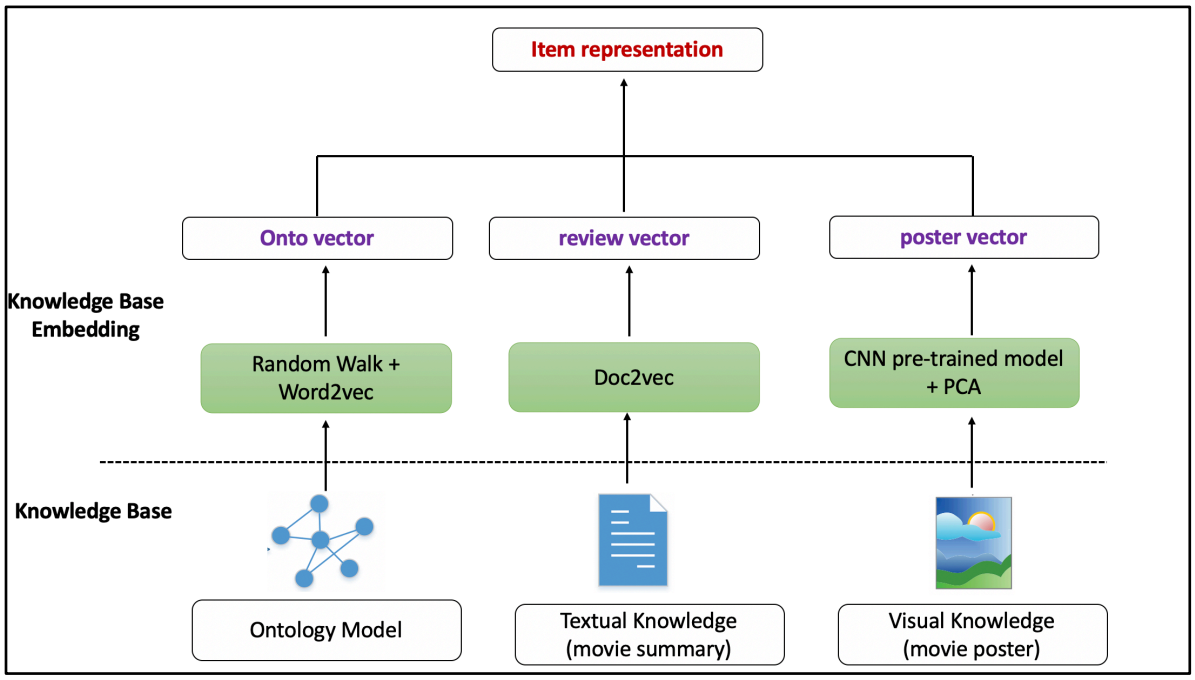

Figure 32: Flow chart for generating item representations

To embedding the **Ontology model**, it is necessary to convert it to plain text before applying the word2vec model. First, we apply the idea of the Random Walk algorithm to generate sentences. The main idea is described as follows:

Firstly, set the number of sentences that will be generated and the maximum number of words in each sentence. Then, randomly walk to a node in {item, term, feature} to generate sentences.

- (1) if walking into an item, randomly walk to one of the features or terms.
- (2) if walking into a feature, randomly walk to one of the items.
- (3) if walking into a term, randomly walk to the next term or an item.

Repeat (1) or (2) or (3) until the length of the sentence is not over the max number of the words in one sentence or the repetition of one node or one edge is over the limitation number. After that, generated sentences are used as input corpus to the Word2Vec model [53] to generate the embedding vector (called onto vector). The onto vector for each movie is the embedding vector generated from the word2vec model for the *item* in the ontology. Note that some text pre-processing techniques such as lowercasing all words, lemmatization, and stop word removal are used before putting them into the Word2Vec model.

For **textual knowledge**, the Doc2Vec model [55] is used to convert movie summary into embedding vector, in this study we call review vector or review2vec. Similar to the above ontology embedding, we apply some text pre-processing techniques before putting them into the Doc2Vec model.

For **visual knowledge**, recent studies [85]–[89] show that CNN can be used as an image feature extractor. When using pre-trained CNN models to extract features, features are usually extracted from fully connected layers before the final classification layer. In this study, we use the DenseNet CNN pre-trained model implemented in Keras [90] to extract the features of the movie poster. Principle Component Analysis (PCA) [91] is also used for feature extraction and dimensionality reduction. It produces a lower-dimensional vector by choosing the dimensions that have most of the variance. In this study, the output vector in the DenseNet CNN model has a size is 1000, so after getting the features extracted from CNN, these features are put into the PCA model to capture the significant features.

3.2.4 Generate user representation vector

Similar to the item modeling, the user modeling aims to represent users into latent vectors based on the historical interactions. We proposed three methods to generate the user representation: sequential preference using the RNN-based model on sequential data, seq2vec using Doc2Vec model on sequential data, and user-review-mean-vector based on the item embedding of user-watched movies.

Sequential preference representation encodes the historical interaction records into hidden state vectors using an RNN-based network. LSTM and GRU have been shown their strong in capturing and characterizing the long temporal dependency in the sequential data.

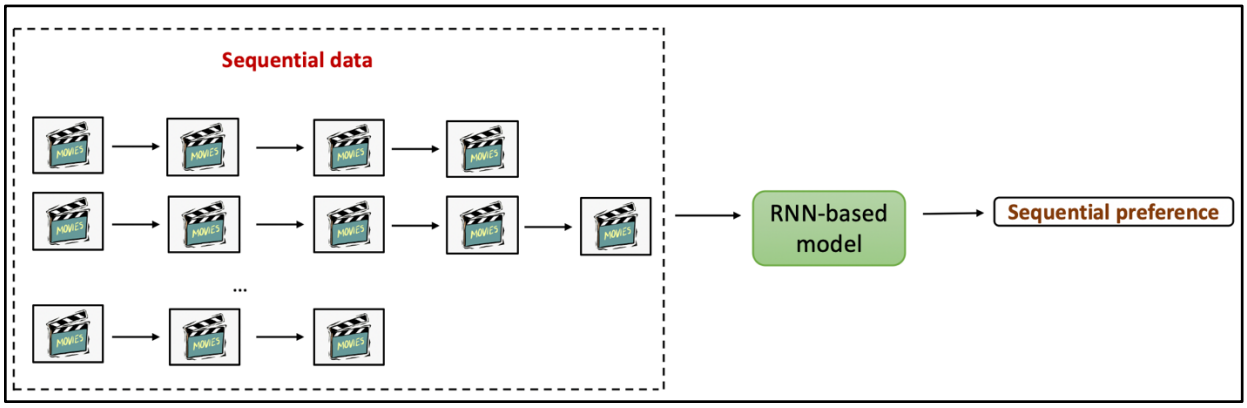

Figure 33: The RNN-based model to capture the user sequential preference

From the explicit feedback (user-item rating scores), we convert it into sequences which is the list of rated movies by time. Let the interaction sequence $\{\boldsymbol{i}_1^{(u)}, \boldsymbol{i}_2^{(u)}, \ldots, \boldsymbol{i}_t^{(u)}, \ldots, \boldsymbol{i}_{n_u}^{(u)}\}$ of user u, the RNN-based model computes the currently hidden state vector h_t^u conditioned on previous state vector h_{t-1}^u as below: $h_t^u = f(h_{t-1}^u, q_i)$, where f(.) is the RNN-based unit, and θ is the RNN-based model parameters. The output h_t^u is the sequential user preference u at time t.

Similar to sequential preference representation using the RNN-based model, the **seq2vec** represented the user interaction sequence using the Doc2Vec model.

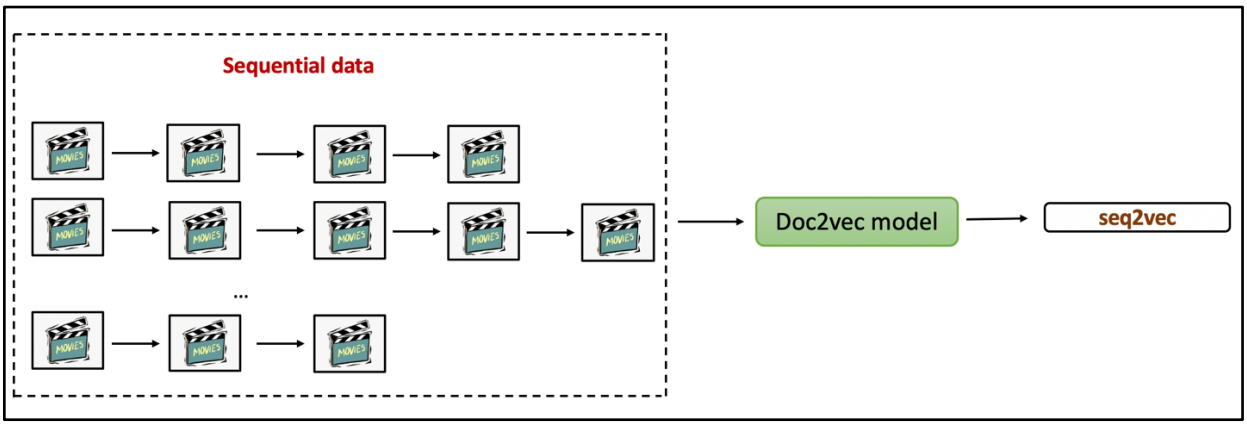

Figure 34: seq2vec using the Doc2Vec model

We also present the user-review-mean-vector which is represented the relationship between user and item based on item embedding. Using the list of favorite movies for each user, the user - review-mean-vector is the average embedding vector of all movies in this list.

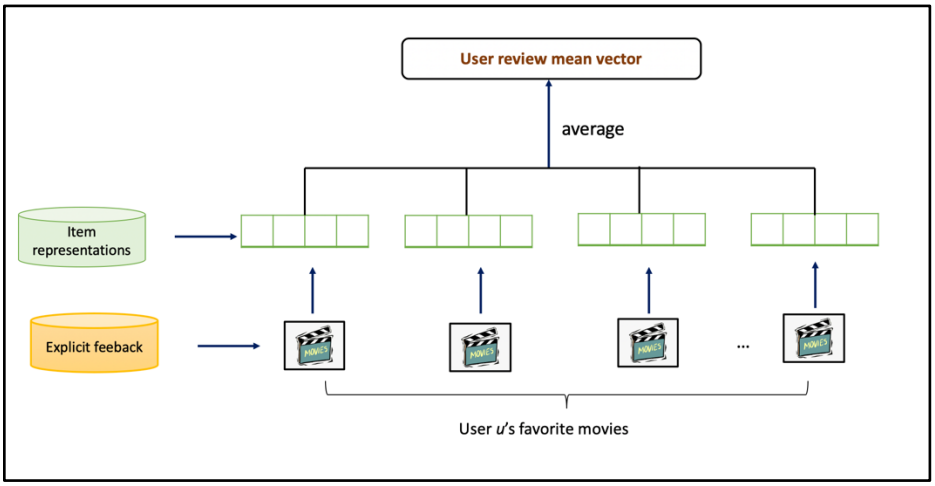

Figure 35: user-review-mean-vector to capture the -relationship between users and items

3.2.5 Using NCF to predict user preference

After getting the latent vectors represented for item and user, the next problem is how to model their interactions based on those representations. In the Neural Collaborative Filtering [26], one-hot vectors represented for users and items are the inputs of the Input layer. Then the Embedding Layer converts one-hot vectors into embedding vectors but it does not have the semantic meaning. To solve this problem, item representation and user representation generated from the two modules above are used as input instead of one-hot vectors.

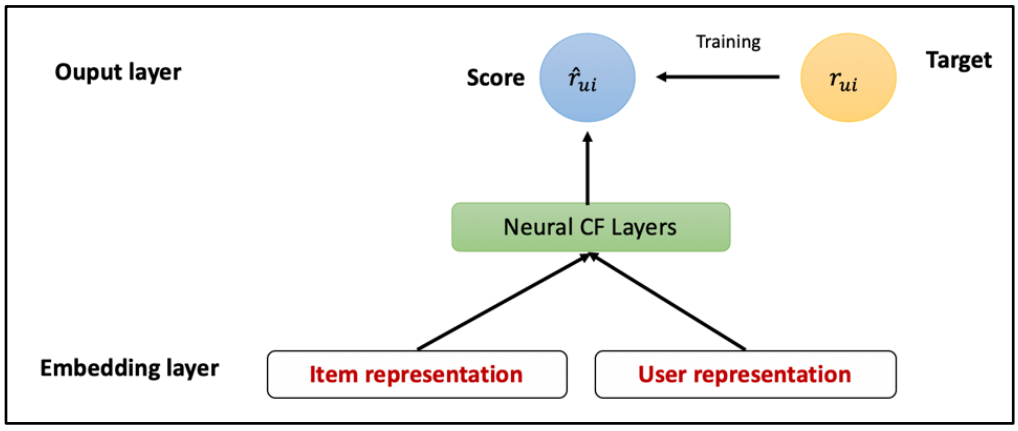

Figure 36: NCF for score predictions

CHAPTER 4: IMPLEMENTATION

This section shows the implementation part basing on the proposed framework. Python and Java are two main programming languages using for implementation. **Section 4.1** describes some tools and API that will be used. Other sections describe the implementations for corresponding modules proposed in Section 3.2.

4.1 Tools and API

4.1.1 Natural Language Toolkit and Gensim

Natural Language Toolkit (nltk) (https://www.nltk.org) is the python platform to work with human language data. It provides text processing libraries for classification, tokenization, stemming, lemmatization, tagging, parsing, semantic reasoning. In this study, we use nltk for text preprocessing includes tokenization, stopwords removal, and lemmatization.

Gensim [92] is the python library for creating and working with unsupervised word embeddings from the text. The library is implemented to be very robust and scalable. It supports many different models such as word2Vec, Doc2Vec, fastText, and Latent Semantic Indexing. The use of Gensim combined with Numpy which is a python library for mathematical computations, lets the efficient in mathematical operations on vectors and matrices. Gensim can be scalable because of its Python's in-built generators and iterations for streamed data-processing. It makes the data set is never actually completely loaded in the RAM.

4.1.2 Keras

Keras [93], [94] is one of the most powerful and easy-to-use python libraries which is built on top of popular deep learning libraries such as TensorFlow [95], Theano [96], and Microsoft Cognitive Toolkit (CNTK) (https://github.com/Microsoft/CNTK) for creating deep learning models. Keras allows the same code to run seamlessly on CPU or GPU. It has built-in support for convolutional networks, recurrent networks, and any combination of both. Keras is also appropriate for building arbitrary deep learning models: multi-input, multi-output, layer sharing, model sharing, and so on.

Keras does not handle low-level operations, it relies on a specialized, well-optimized tensor library. Due to the way of handling problems in a modular way, several backend engines can be plugged seamlessly into Keras. Up to now, Tensorflow (is developed by Google), Theano (s developed by the MILA lab at Université de Montréal), and CNTK (is developed by Microsoft)

are three existing backend implementations. Code written in Keras can be run with any of those backends without changing anything or can switch between backends during development. Keras can run on both CPU and GPU via Tensorflow or Theano or CNTK.

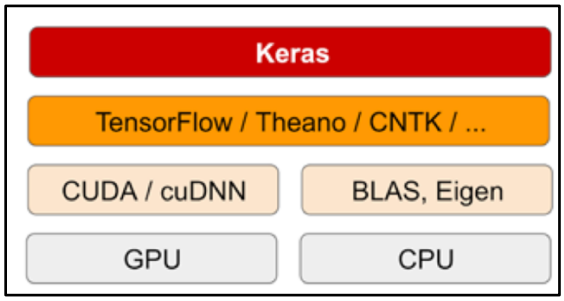

Figure 37: The deep learning software and hardware stack [94]

The neural networks usually contain the following objects:

- Layers are combined to make a network/ model. The layer is a data-processing module with getting one or more tensors as the input and producing one or more tensors as the output.
- The input data and corresponding targets
- The loss function defines the feedback signal during the learning process. It measures the success of the training process.
- The optimizer determines how the network will be updated based on the loss function. It is the specific variant of Stochastic Gradient Descent.

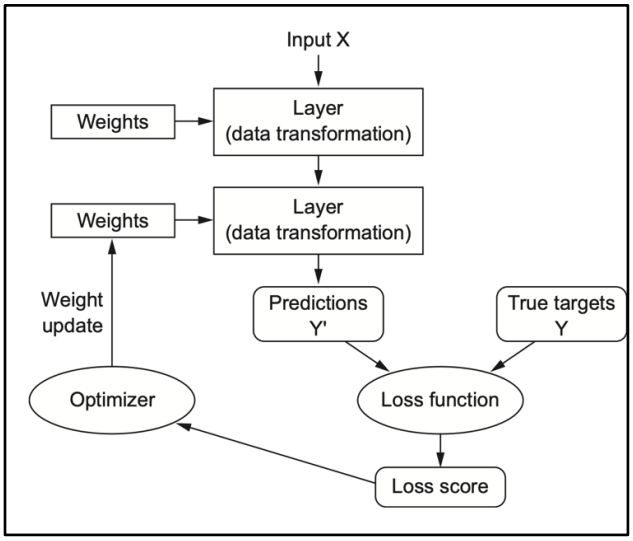

Figure 38: The relationship between the layer, loss function, and optimizer [94]

Some Keras API used in building deep learning models:

- Layers: includes Core layer, convolutional layer, Pooling layer, Recurrent layer, Embedding layer, Merge layers.
    - Core layer: includes commonly used layers
        - Dense layer: is the fully connected layer. Some parameters used in this layer:
            - units: the output's dimension
            - activation: activation function will be applied
            - use_bias (Boolean): whether using bias or not
            - kernel_initializer: the initialization of matrix weights
            - bias_initializer: the initialization of bias vector
            - kernel_regularizer: regularizer for matrix weights
            - bias_regularizier: regularizer for bias vector
            - kernel_constraint, bias_constraint: the constraint functions for weight matrix and bias vector.
        - Activation layer: is used to set the activation function
        - Dropout layer: is used to prevent over-fitting. The important parameter in this layer is the rate which is the fraction of the input units to drop.
        - Flatten layer uses to flatten the input.
        - Input layer uses to initialize the Keras tensor
    - Convolutional layer: is used to build the convolutional network
    - Pooling layer: includes layers using in CNN model such mas MaxPooling, AvergaePooling, GlobalPooling
    - Recurrent Layers: is used to build RNN- based models like GRU, LSTM…
    - Embedding layer: is used to map vector into smaller dimensional space.
    - Merge Layer: supports operations such as add, subtract, multiply, average, concatenate, dot.
- Loss function: computes the quantity that the model should seek to minimize during the training. Some supported loss functions are Mean Square Error, Mean Absolute Error, Categorical Crossentropy, Binary Crossentropy,…

- Activations: the activation functions that Keras supports are linear, softmax, relu, tanh, and sigmoid.
- Optimizer includes Stochastic Gradient Descent optimizer, Adam optimizer, RMSprop optimizer
- Applications: contains some pre-training weight of famous deep learning models, such as, VGG16, VGG19, DenseNet, etc.

4.1.3 Protégé

Protégé [97] (https://protege.stanford.edu/products.php) is the tool for building Ontology. The most prominent function of this software provides the API that ontologies can be used to develop Semantic websites according to W3C OWL.

Protégé provides features to make the effective building and using ontologies in OWL:

- It provides GUI for editing classes, properties, and instances.
- Protégé also supports API for the integration into applications. Protégé OWL also supports three types of reasoning: checking the consistency, classification (subsumption), and instance classification
- Protégé OWL also supports for multiuser to synchronous knowledge entry
- Multiple storage formats are supported such as Clips, XML, REF, and OWL.

4.1.4 OWLAPI

The OWL API [98] is a Java API for working with OWL ontologies. The OWL API supports to manipulation of OWL 2 structures and OWL 2 reasoning. In the OWL API, the ontology is considered as a set of axioms and annotations. The OWL API classifies the entities into classes, object properties, and data properties which are assumed to be disjoint.

The tutorial for working with OWL API can be found in [99].

4.2 Data preparation

In this module, we try to prepare the data for building the knowledge base. As designed above, the knowledge base has three components: domain knowledge, textual knowledge, and visual knowledge. To build the **knowledge base**, we need these features: movieId, title, genres, directors, casts, writers, year, runtime, country, movie summary, and poster. The movie information stored in the MovieLens dataset is movieId, title, and genres. So we use IMDbPY [100] which is a python package for Internet Movie Database to get the missing information. The total number of movies is 62423. Some features such as casts, directors have a number of values much larger than other features. It leads to the vector representations will depend on those features. So, to make fairly contribution of each attribute in the vector representation, we get a max of 8 values for each feature.

```
{'movieId': 1,
 'title': 'Toy Story',
 'genres': 'Animation|Adventure|Comedy|Family|Fantasy',
 'directors': 'John Lasseter',
 'writers': 'John Lasseter|Pete Docter|Andrew Stanton',
 'casts': 'Tom Hanks|Tim Allen|Don Rickles|Jim Varney|Wallace Shawn|John Ratzenberger|Annie Potts|John Morris',
 'runtime': 81,
 'year': 1995,
 'country': 'United States',
 'summary': "A cowboy doll is profoundly threatened and jealous when a new spaceman figure supplants him as top toy
in a boy's room",
 'poster': '/Users/minhthu/Documents/IU/Master/8. Thesis/Poster Embedding/poster/1.jpg'}
```

Figure 39: Example of one movie information

The number of missing values for each attribute are:

Table 1: Statistical of missing values

Attribute	Number of missing values	Percentage
Genres	20	0.03%
Directors	689	1.1%
Writers	3550	5.69%
Casts	1307	2.09%
Country	433	0.69%
Runtime	348	0.56%
Year	0	0%
Summary	1189	1.9%
Poster	1753	2.81%

4.3 Building ontology model

The next step is building domain knowledge using Ontology. Firstly, we will show how Protégé describes some common components in Ontology. Classes are the group of instances, objects having similar characteristics. In Protégé, class owl:Thing is the parent class of all other classes. For the properties, we focus on object properties that are the link between two individuals and datatype properties to demonstrate the connection between individuals and data values. Each object property has its inverse property to illustrate how class A relates to class B and vice versa.

Basing the ontology model described in section 3.2.2, we store it into 3 owl files: OntoModel.owl, ItemTerm.owl, and TermMap.owl. The *OntoModel.owl* stores the Item with Features information, *ItemTerm.owl* stores the Item - Term relationship, *and TermMap.owl* stores the Links between Terms.

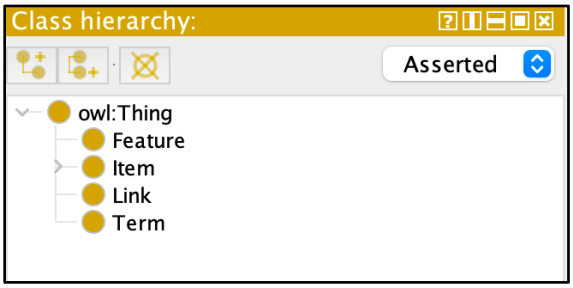

Figure 40: Classes in movie Ontology

- Class *Item* describes the movie
- Class *Feature* includes features for that movie. The features can be genres, writers, directors, casts, runtime, year, and country.
- Class *Term* is the keyword extracted from the movie title. Each word in the title is one term. For example, the movie "Toy Story" has two terms that are "Toy" and "Story".
- Class *Link* represents a connection between 2 terms: in refers to the previous term (in-term), out refers to the next term (out-term), weight is the number of times this link appears. For each term extracted from the title:
    - If the term is the first word in the title, the link is ***S_term***.
    - If the term is the last word in the title, the link is ***term_E***
    - Otherwise, the link is previous term_ term.

  For example, with the title "Toy Story", we have three links: S_toy, toy_story, and story_E.

For the object property:

- *feature_Item* and *item_Feature*: relationships between Item and Feature. Each Item (movie) has Features (casts, directors, genres…) and Feature belongs to Items.
- *item_Term* and *term_Item*: relationships between Term and Item. Each Item (movie) has Terms (keywords in the title) and the Term belongs to Items.
- *IIn* and *IOut*: relationships between Link and Term. Each Link has terms: in-term (IIn) and out-term (IOut).

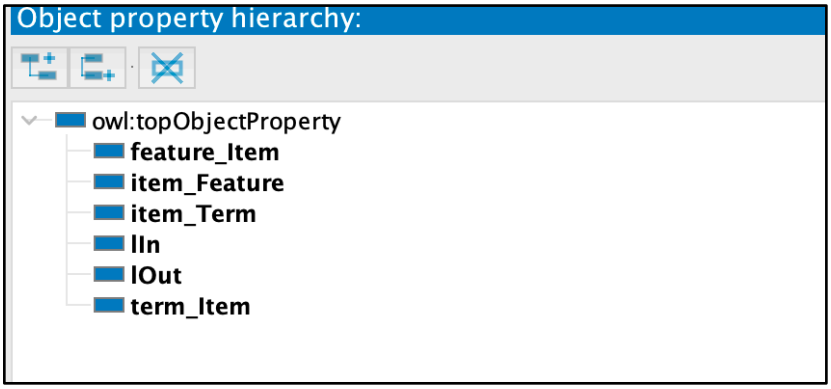

Figure 41: Object properties in movie ontology

Table 2: Domain and Range of object properties in movie ontology

Domain	Object property	Range
Feature	feature_Item	Item
Item	item_Feature	Feature
Item	item_Term	Term
Link	IIn	Term
Link	IOut	Term
Term	term_Item	Item

For the data property:

- The Feature has *fType* is the type of feature (which can be cast, director…) and *fValue* is the value of the feature. For example, actor "Annie Potts" when represented in Feature will have fType is "cast-movie" and fValue is "Annie Potts".
- The Item has *ID* is the ID of Item and *iName* is its name.
- The Term has *termValue* and *termWeight.* For example, 23 movies in the dataset have the word "toy" in the title, so the termValue is "toy" and the termWeight is 23.
- The Link has *linkWeight*. There are 19 movies have title begin with word "toy", so the linkWeight of link "S_toy" is 19.

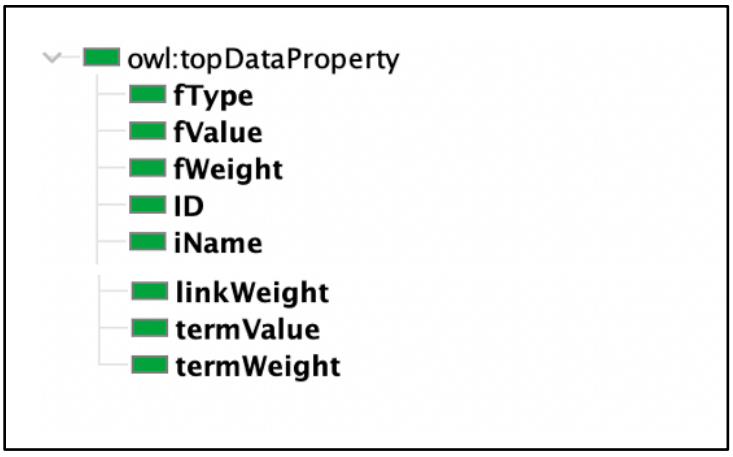

Figure 42: Data properties in movie ontology

Table 3: Domain and Range of data properties in movie ontology

Domain	Data Property	Range
Feature	fType	String
Feature	fValue	String
Item	ID	String
Item	iName	String
Link	linkWeight	int
Term	termValue	String
Term	termWeight	int

From the example movie information (figure 38), its information is represented in the ontology model as follow:

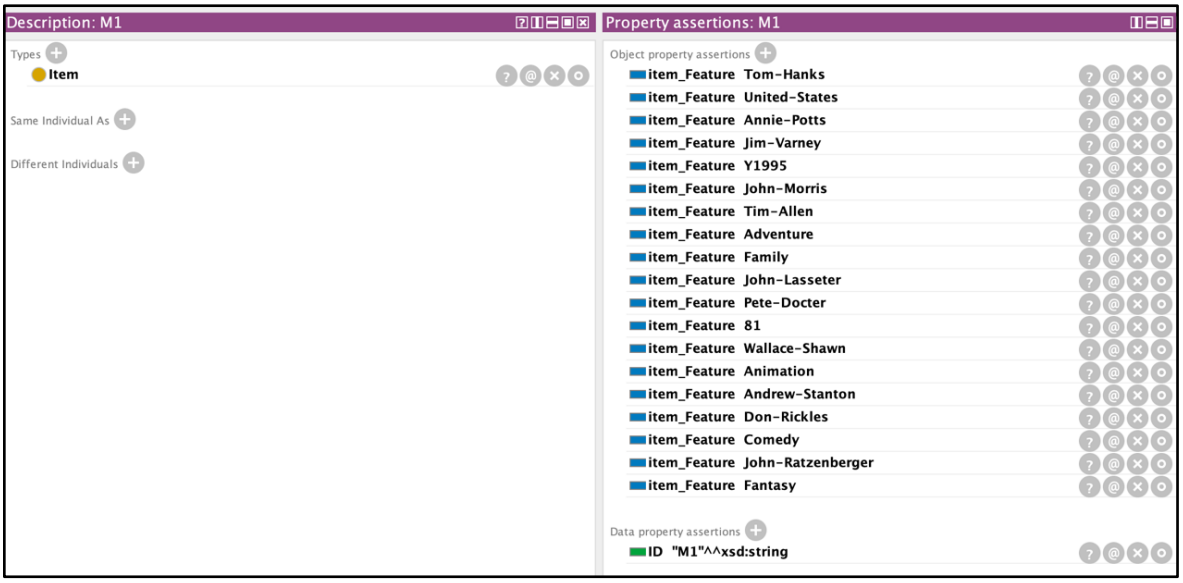

Figure 43: An example of Item individual stored in OntoModel.owl

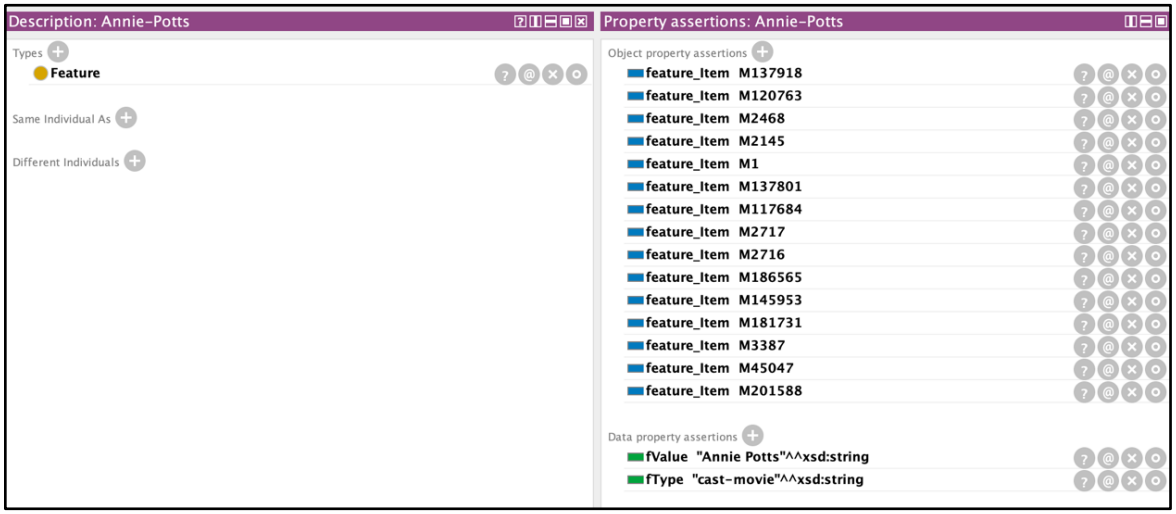

Figure 44: An example of Feature individual stored in OntoModel.owl

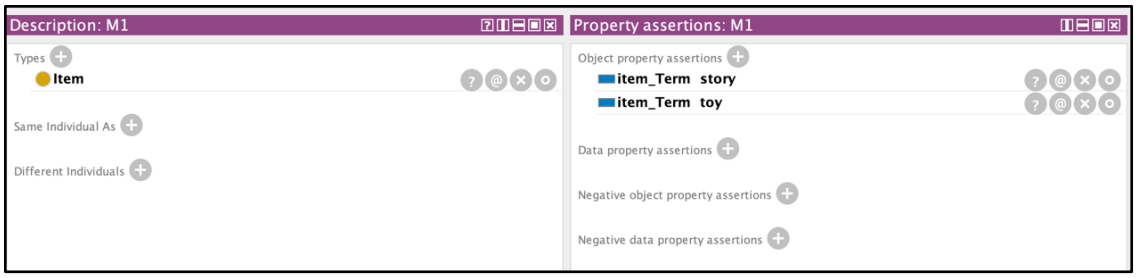

Figure 45: An example of Item individual stored in ItermTerm.owl

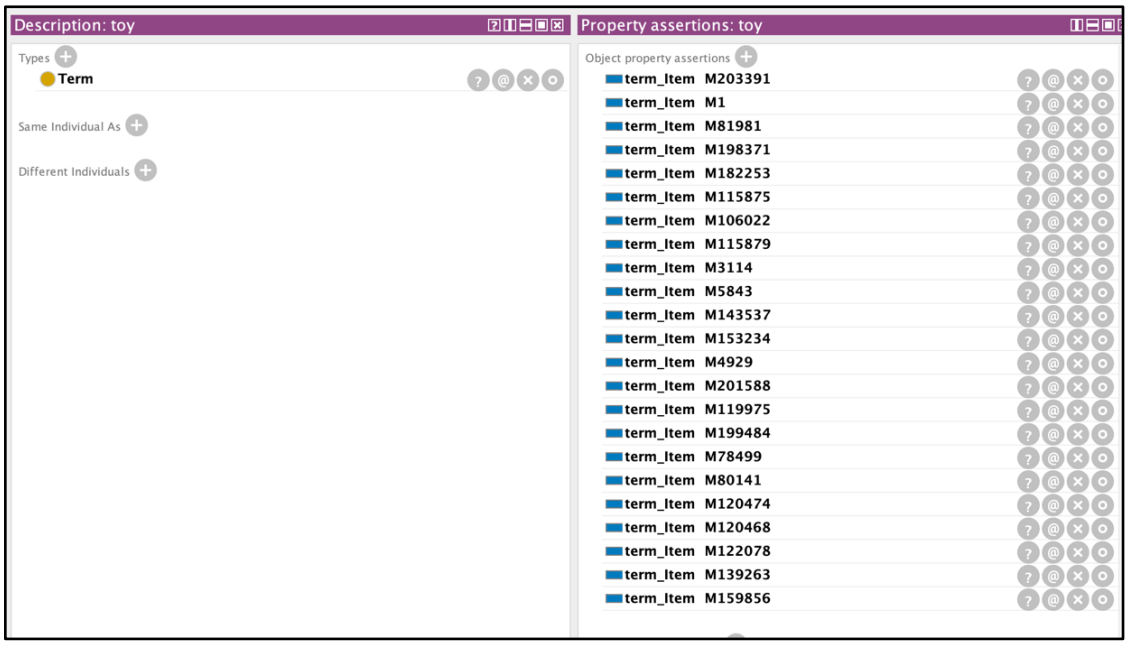

Figure 46: An example of Term individual stored in ItemTerm.owl

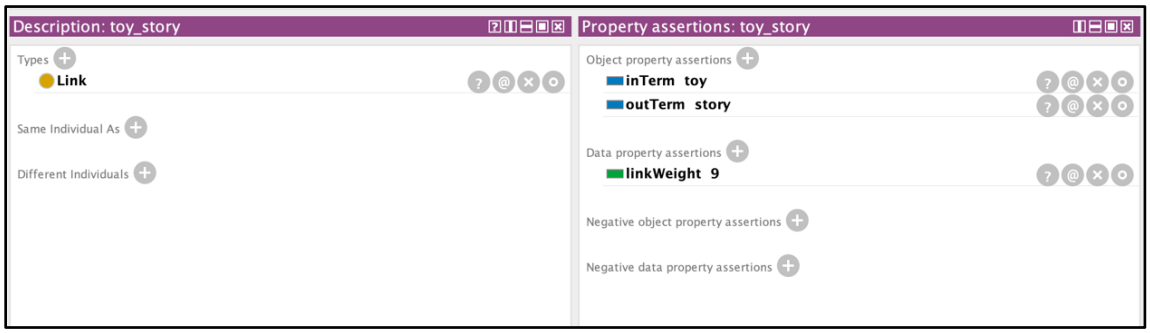

Figure 47: An example of Link individual stored in TermMap.owl

4.4 Generate item embedding vector

After getting the knowledge bases, the next step is the vectorization of the knowledge bases. The item representation is the combination of three embedding vectors: onto vector, review vector, and poster vector.

To create the **onto vector**, there are two steps: (1) generate sentences using a random walk algorithm to walk through nodes in the ontology model, and (2) apply the word2vec model to the generated sentences.

(1) Generate sentences using a random walk algorithm

Step 1: Randomly choosing the starting node

Step 2: Move to the next node based on the conditions:

(a) If the current node is an item, the next node will be a feature or a term

(b) If the current node is a feature, the next node will be an item

(c) If the current node is a term, the next node will be an item or a next term

Repeat (a) or (b) or (c) until the length of the sentence is not over the max number of the words in one sentence or the repetition of one node or one edge is over the limitation number.

```
default: //walking into a term, randomly walk to a next term or an item
    Ontology.TermMap.Term curTerm = reasoner.getTerm(curWord);
    Random ran1 = new Random();
    int next1 = ran1.nextInt( bound: 2);
    if (next1 == 0) { //walk to a term
        Random rTerm = new Random();
        ArrayList alTT = reasoner.getTerms(curTerm);
        if (alTT != null && alTT.size() > 0) {
            curTerm = (Ontology.TermMap.Term) alTT.get(rTerm.nextInt(alTT.size()));
            curWord = reasoner.getTermValue(curTerm);
            objectSel = 2;
        } else curWord = "";
    } else { //walk to an item
        Random rItem1 = new Random();
        ArrayList alTI = reasoner.getItems(curWord);
        if (alTI != null && alTI.size() > 0) {
            Ontology.ItemTermOnto.Item curItem1 = (Ontology.ItemTermOnto.Item) alTI.get(rItem1.nextInt(alTI.size()));
            curWord = reasoner.getItemName(curItem1);
            objectSel = 0;
        } else curWord = "";
    }
```

Figure 48: Moving to the next node when the current node is a term

We set the max number of the words in one sentence as **10**, the number of max repetitions of one node is **5** and the number of max repetitions of a node is **3**. 5 million sentences are generated.

```
M108212 Aymeline Valade 150 M69429 home refugee M127581 poetical Drama
Tamsin Egerton M95223 Thriller M1861 mail order myths M83291 phoenix
Sarah Louise Tyler M206373 rectory 88 M183365 Adam Scott Weissman
hears who dreamland M154560 M194700 Switzerland M109848 Joe Szula M89580
Jemma Kennedy M162340 Drama M187777 agit Sermin Hürmeriç
Clint Bickham M167850 M162718 corpses lie die M70423 time M188359
vasermil M102997 Kobi Maor M178101 Drama M32109 Wadeck Stanczak M125105
gallows M32179 elevator M101264 M152916 Documentary M100046 Y2011 M89945 Ireland
psychonautics M197111 Y2018 M197917 Sharat Katariya M153780 Tom Alter M201636
M200506 doll hell fest M192999 Comedy M135392 Gordon Mitchell M134274
M189983 sweethearts M4639 Catherine Zeta Jones M126164 les M153734 egouts
```

Figure 49: Example of sentences generated from the ontology using the random walk algorithm

(2) Apply word2vec model for vector generation

Some text pre-processing techniques are applied, such as tokenization, removing stopwords, and lemmatization.

```python
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer

filtered_sentences  = []
stop_words=set(stopwords.words("english"))
lem = WordNetLemmatizer()

def sentencePreprocess(sentence):
    tokenized_word=word_tokenize(sentence.lower())

    # REMOVE STOP WORDS
    remove_stopwords = []
    for word in tokenized_word:
        if word not in stop_words:
            remove_stopwords.append(word.lower())

    # LEMMATIZATION
    lemmatized_words = []
    for word in remove_stopwords:
        new_word = lem.lemmatize(word,"a")  #for adjective
        new_word = lem.lemmatize(word,"n")  #for noun
        lemmatized_words.append(lem.lemmatize(new_word,"v"))  #for verb

    return lemmatized_words
```

Figure 50: Text preprocessing using nltk

After that, the preprocessed text is the input of the word2vec model. Some hyperparameters in this model are:

- min_count is the minimum number of times a word must appear in the corpus. Words that appear less than this number in the corpus are ignored.

- window: maximum distance between the current and predicted word within the sentence.
- vector_size: the size (dimension) of the output vector
- alpha: is the learning rate which is the step size for each update of the coefficients.
- sg: is the training algorithm that will be used. If the sg =1, the skip-gram architecture will be used
- workers: number of threads used to train the model
- seed is the seed for the random generator

```
from gensim.models import Word2Vec
model = Word2Vec( min_count = 1, window = 5, vector_size = 50,
                  alpha = 0.025, sg = 1, workers =12, seed =42)
model.build_vocab(filtered_sentences)
model.train(filtered_sentences, total_examples=model.corpus_count, epochs =100)
```

Figure 51: Building word2vec using the gensim library

To create the **review vector**, the doc2vec model is applied. Similar to doc2vec, text pre-processing needs to be done before training the model. Gensim provides an option to train paragraph vectors along with words vectors in the skip-gram fashion (dbow-sg). The hidden-output weights are shared between word vectors and paragraph vectors so word vectors will influence the paragraph vectors to obtain paragraph representations.

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

tagged_data = [TaggedDocument(d, [i]) for i, d in enumerate(list_summaries)]
model = Doc2Vec(vector_size=50, min_count=1,  workers =5, seed = 42, window = 5,
                alpha = 0.025, dbow_words =1 )
model.build_vocab(tagged_data)
model.train(tagged_data, total_examples=model.corpus_count, epochs=50)
```

Figure 52: Building doc2vec model in gensim library

A popular and highly effective approach to deep learning on small image datasets is using the pre-trained network, which is the saved network that is trained on a large dataset typically for a large-scale image classification task. Feature extraction is the approach that uses the representations learned by a pre-trained network to extract features from new samples. To create

the **poster vector**, first, the link of the poster is gotten from imdb API. Then theDense201 CNN pre-trained model is used to extract features from the poster.

```
from tensorflow.keras.preprocessing import image
import numpy as np
from tensorflow.keras.applications.densenet import DenseNet201
from tensorflow.keras.applications.densenet import preprocess_input
import urllib.request
from io import BytesIO
from PIL import Image
from imdb import IMDb

ia = IMDb()  # package for retrieving the data of the IMDb database

def get_image_cover_url(imdb):
    try:
        movie_imdb = ia.get_movie(imdb)     # get the movie giving imdb Id
        url_cover = movie_imdb["full-size cover url"]    # get the url of movie's poster
    except:
        url_cover = ""
    return url_cover

model = DenseNet201(weights='imagenet')     #create the base pre-trained model

def getEmbeddingImage(movieId):
    link = get_image_cover_url(movieId)
    try:
        with urllib.request.urlopen(link) as url:
            img = Image.open(BytesIO(url.read())).resize((224, 224))    #load an image
            x = image.img_to_array(img)     #convert the image pixels to a numpy array
            x = np.expand_dims(x, axis=0)
            x = preprocess_input(x)     #prepare the image for the DenseNet model
            features_extraction= model.predict(x)[0]    #get extracted features
    except:
        features_extraction = []
    return features_extraction
```

Figure 53: Python code to extract features for an image using Dense pre-trained model

Because the output vector size is 1000, so PCA is applied to reduce the dimension.

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

sc = StandardScaler()
images_embedding = sc.fit_transform(images_embedding) # normalize the features

pca = PCA(n_components = 50)
images_embedding = pca.fit_transform(images_embedding)
```

Figure 54: PCA to reduce the poster vector's dimensions

After getting the three-component embedding vectors, the concatenation of them can be used as item vector representation. In the experimental case, we try to test some combinations to find the best one for item representations. For the movies that don't have the summary or the poster image, the embedding vector is the average of corresponding vectors of other movies.

4.5 Generate user embedding vector

This part aims to learn the user latent vectors. There are three approaches to represent users into vectors: sequential preference representation, seq2vec, and user-review-mean-vector. The first step in three cases is defining the list of users' favorite movies. To do that, for each user, the interacted movies are sorted by timestamp, then, movies that have a rating score does not less than the average rating score that the user has rated are seemed to be the user's favorite ones.

4.5.1 Sequential vector representation using the RNN-based model

Sequential preference representation captures the long-term dependency preference over time. The user's favorite sequences are put into the LSTM cell or GRU cell to get the sequential preference.

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# samples that are shorter than the longest item need to be padded with some placeholder value
padded_inputs = keras.preprocessing.sequence.pad_sequences(list_sequences_train, padding ="post")

# inform the part is padding and will be ignored
masking_layer = layers.Masking()

unmasked_embedding = tf.cast(
    tf.tile(tf.expand_dims(padded_inputs, axis=-1), [1,1,1]), tf.float32
)

masked_embedding = masking_layer(unmasked_embedding)

outputs = layers.LSTM(output_size)(masked_embedding)
```

Figure 55: Python code to generate sequential preference using LSTM

When handling the user sequence data, different user sequences have different lengths. To solve this problem, we use Padding and Masking

- Padding: samples that are shorter than the longest one needed to be padded with some placeholder value. The masked steps can be at the start or the end of the sequence.
- Masking: is used to inform the sequence-processing layer which part is the padding so it can ignore them when processing the data.

4.5.2 Seq2vec

Similar to sequential preference representation, but instead of using an RNN-based model, seq2vec uses the Doc2Vec model to put the user session into latent vector space.

```
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

tagged_data = [TaggedDocument(d, [i]) for i, d in enumerate(list_sequences_train)]

model = Doc2Vec(vector_size=output_size, min_count=1,  workers =12, seed = 42, window = 5,
                alpha = 0.025, dbow_words =1 )

model.build_vocab(tagged_data)

model.train(tagged_data, total_examples=model.corpus_count, epochs=50)
```

Figure 56: Seq2vec to capture the user sessions

4.5.3 User-review-mean-vector

User-review-mean-vector representation captures the relationship between user and items over movie embedding vectors. User-review-mean-vector is the average of the item embedding of those favorite movies.

```
import statistics

def getVec(Id, list_Id, list_Vec):
    '''
    Function to get the item_vector of given movie
    Parameters:
            listId: list of movieId
            list_Vec: list of corresponding vectors
       '''
    index = list_Id.index(Id)
    return np.array(list_Vec[index])


def get_user_representation_by_watched_movies (train,userId, list_movieId, list_movieVec):
    '''
    Function to get attribute-based preference of user
    Parameters:
        train: data (in pandas.DataFrame)
        userId: the Id of user
        list_movieId, list_movieVec: list of movieId and list of corresponding vectors
    '''
    movie_vectors = []
    data_user = train.loc[train["userId"]== userId]  # all ratings of given user
    mean_rating = statistics.mean(data_user["rating"].tolist()) #mean value of all rating score that user has rated
    list_favorite_movie = data_user.loc[data_user["rating"]>=mean_rating]["movieId"].tolist() #list favorite movie
    for movie in list_favorite_movie:
        item_vector =getVec(movie, list_movieId, list_movieVec)
        movie_vectors.append(item_vector)
    return sum(movie_vectors)/len(list_favorite_movie)
```

Figure 57: Python code for generating user-review-mean-vector

4.6 Using NCF to predict user preference

After having the user representation and item representation, the last part is building the Neural Collaborative Filtering to model user-item interactions. Traditional NCF uses one-hot vectors in the Input Layer (Figure 4 and 5) and does not focus on the semantic meaning. Instead of using one-hot vector representations, user embedding vectors and item embedding vectors are put into the Input Layer. In the experimental cases, we test three NCF models that are Generalized Matrix Factorization (GMF), Multi-Layer Perceptron (MLP), and Neural Matrix Factorization (NeuMF) for choosing the best model.

- ***GMF***: This model gets the embedding vectors (for both users and items) as inputs. Then the element-wise product of two latent vectors is calculated before putting it to the fully connected layer (Dense Keras layer) to get the predicted score. In three models, activation function *ReLU* [44] is used, and *Mean Squared Error* [101], [102] is used as loss function during the training

```
import numpy as np
from tensorflow.keras import initializers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import  Input, Dense, Multiply
from tensorflow.keras.optimizers import  Adam
import tensorflow as tf
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
# DEFINE MODEL

def get_model_GMF(size):
    '''Function to define the GMF_update model
        Parameters:
            size: the size of latent vectors
        '''
    # Input user and latent vector
    user_latent = Input(shape=(size,),dtype = "float64", name = 'user_input_latent')
    item_latent = Input(shape=(size,), dtype ="float64",name = 'item_input_latent')

    # Element-wise product of user and item embeddings
    predict_vector = Multiply()([user_latent, item_latent])

    # Final prediction layer
    prediction = Dense(1, activation='relu', kernel_initializer=initializers.LecunUniform(),
                       name = 'prediction')(predict_vector)

    model = Model(inputs=[user_latent, item_latent],
                  outputs=prediction)
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

    return model
```

Figure 58: The GMF model using embedding vectors as inputs

- ***MLP***: This model gets the embedding vectors (for both items and users) as inputs. After finding the concatenated vector of two embedding vectors, that vector goes through some hidden layers which are the fully-connected layers (Dense Keras layer) before moving to the last fully-connected layer as the prediction layer.

```
import numpy as np
from tensorflow.keras import initializers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import  Input, Dense, Multiply, concatenate
from tensorflow.keras.optimizers import  Adam
from tensorflow.keras.regularizers import l2
import tensorflow as tf
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

def get_model_MLP( size, layers, reg_layers):
    assert len(layers) == len(reg_layers)
    num_layer = len(layers) #Number of layers in the MLP
    # Input latent vectos
    user_input = Input(shape=(size,), dtype='float64', name = 'user_input')
    item_input = Input(shape=(size,), dtype='float64', name = 'item_input')

    # The 0-th layer is the concatenation of embedding layers
    vector = concatenate([user_input, item_input])

    # MLP layers
    for idx in range(1, num_layer):
        layer = Dense(layers[idx], kernel_regularizer= l2(reg_layers[idx]), activation='relu', name = 'layer%d' %idx)
        vector = layer(vector)

    # Final prediction layer
    prediction = Dense(1, activation='relu', kernel_initializer='lecun_uniform', name = 'prediction')(vector)

    model = Model(inputs=[user_input, item_input],
                  outputs=prediction)

    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')

    return model
```

Figure 59: The MLP using embedding vectors as inputs

- ***NeuMF***: This model is the combination of the two above models (GMF and MLP).
    - The MF part finds the element-wise multiplication of two embedding vectors.
    - The MLP takes the concatenation of two embedding vectors and puts this vector through multiple Dense layers.
    - The concatenation of two vectors from MF part and MLP part is the input of the prediction layer which is a Dense layer.

```
from tensorflow.keras import initializers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import  Input, Dense, Multiply, concatenate
from tensorflow.keras.optimizers import  Adam
from tensorflow.keras.regularizers import l2
import tensorflow as tf
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

def get_model_NeuMF(size, layers, reg_layers):
    assert len(layers) == len(reg_layers)
    num_layer = len(layers) #Number of layers in the MLP
    # Input variables
    user_input = Input(shape=(size,), dtype='float64', name = 'user_input')
    item_input = Input(shape=(size,), dtype='float64', name = 'item_input')

    # MF part
    mf_vector = Multiply()([user_input, item_input]) # element-wise multiply

    # MLP part
    mlp_vector = concatenate([user_input, item_input])
    for idx in range(1, num_layer):
        layer = Dense(layers[idx], kernel_regularizer= l2(reg_layers[idx]), activation='relu', name="layer%d" %idx)
        mlp_vector = layer(mlp_vector)

    # Concatenate MF and MLP parts
    predict_vector = concatenate([mf_vector, mlp_vector])

    # Final prediction layer
    prediction = Dense(1, activation='relu', kernel_initializer='lecun_uniform', name = "prediction")(predict_vector)

    model = Model(inputs=[user_input, item_input], outputs=prediction)
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')
    return model
```

Figure 60: The NeuMF model using embedding vectors as inputs

CHAPTER 5: EXPERIMENTS

5.1 Datasets: MovieLens

This movie dataset has been used to evaluate many recommender systems. We use the ml-25m version which contains 62423 movies and 25000095 ratings given by 162541 users for 59047 movies. Each user has at least 20 ratings.

5.2 Evaluation methods

We evaluate our recommendation model in two cases:

- Firstly, we consider the predicted output as the rating score. In this case, we will use *Mean Square Error* and *Mean Absolute Error* to evaluate the model.
- Secondly, we consider our problem is a classification problem. It means that we predict the user's like or dislike of the movie. In this case, we use *precision, recall, accuracy, and F1-score* for evaluation.

To be more specific, we first present the way to divide the train and test dataset. To make the more realistic recommendation scenario, we try to simulate the data stream. We first sort all interactions in chronological order, the first 80% is used as the train set and the last 20% is used for the test set.

Some evaluation metrics used in this study:

- *Mean Square Error* (MSE) [101] is used to calculate the difference between true values and predicted values of the model for the test set, which is the mean of the squared prediction errors over all instances in the test set.

$$MSE = \frac{\sum_{i=N}^{N}(y_i - \hat{y}_i)^2}{N}$$

Where N is the number of instances in the test set, y_i is the true value and $\hat{y}_i$ is the predicted value.

- *Mean Absolute Error* (MAE) [103] has a similar purpose as MSE. The MAE value is the mean of absolute prediction error over all instances in the test set.

$$MAE = \frac{\sum_{i=N}^{N}|y_i - \hat{y}_i|}{N}$$

Where N is the number of instances in the test set, y_i is the true value and $\hat{y}_i$ is the predicted value.

- Confusion matrix [104] is used to describe the performance of classification problems. Four important in the confusion matrix are true positive, true negative, false positive, and false negative. The meaning of these concepts in our study case are:
  - True positive: the user liked the movie and the model gave a true prediction
  - True negative: the user disliked the movie and the model gave a true prediction
  - False positive: the user disliked the movie, but the model predicts "like"
  - False negative: the user liked the movie, but the model predicts "dislike"

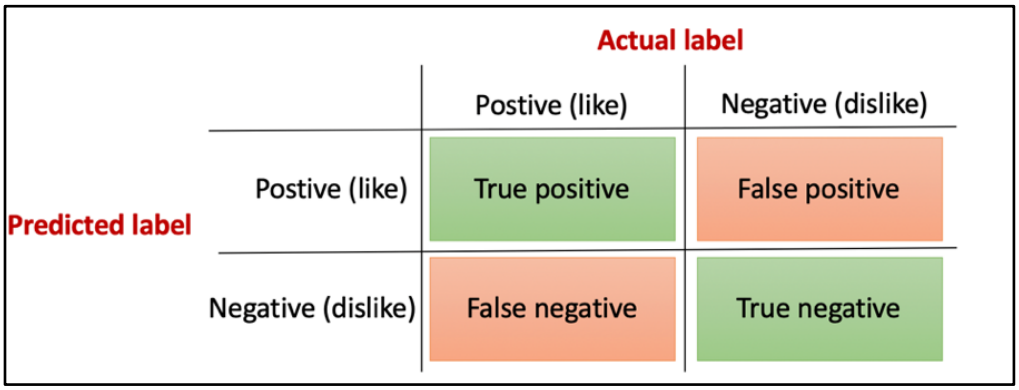

Figure 61: The confusion matrix in classification problems performance [104]

Basing the confusion matrix, we have evaluations such as:

- *Accuracy* is the proportion of correct predictions

$$Accuracy = \frac{true\ postive\ +\ true\ negative}{true\ positive\ +\ true\ negative\ +\ false\ positive\ +\ false\ negative}$$

- *Precision* is the proportion of positive prediction that was actually correct

$$Precision = \frac{true\ positve}{true\ positve +\ false\ positive}$$

- *Recall* is the proportion of positive instances was identified correctly

$$Recall = \frac{true\ positive}{true\ positive\ +\ false\ negative}$$

- *F1-score* is the harmonic mean of precision and recall. A good classification model will have high F1-score

$$F1 - score = \frac{2 \times precision \times recall}{precision\ +\ recall}$$

- Criteria in choosing the best recommendation model are:
  - All liked movies are identified means that the recall is high.
  - All dislike-movies should not appear in recommendation lists means that the precision is high

5.3 Experimental Cases

Basing the way generating embedding vectors, we have the following experimental cases:

- Using neural collaborative filtering models with one-hot vector representation for user and item.
- Using neural collaborative filtering models with inputs being the embedding vectors of users and items.
  - Item representation is created by:
    - (1) Using only the onto vector
    - (2) The concatenation of onto vector and review vector
    - (3) The concatenation of three vectors (onto, review and poster)
  - User representation is created by:
    - (4) Sequential preference learned from RNN-based network
    - (5) user-review-mean-vector
    - (6) seq2vec model using Doc2Vec

5.4 Experimental Results

As mentioned in the section of evaluation metrics, we consider two cases:

- The class is numeric which means that the output is the rating score.
- The class is nominal which means that the output is like (1) or dislike (0). To convert from the rating score feedback to the nominal value, we follow the rule: if the rating score is less than 2.5, the class label is 0 (dislike), otherwise is 1 (like)

Evaluation metrics:

- Numeric class: MAE and MSE are used to calculate the difference between the true rating score and the predicted rating score.
- Nominal class: Precision, recall, accuracy, and F1-score are used to determine the classification performance.

5.4.1 Case 1: Using Neural Collaborative Filtering with one-hot vector representation for user and item.

Three considering algorithms are Generalized Matrix Factorization (GMF), Multi-layer Perceptron (MLP), and Neural Matrix Factorization (NeuMF). The parameters from the three models are inherited from Xiangnan He et. al [26].

Table 4: Evaluation results of Case 1

Model	Training time (s)	Numeric class (rating score)		Nominal class (like/dislike)			
		MAE	MSE	Precision	Recall	Accuracy	F1
GMF	6912.2	0.6185	0.6756	0.9111	0.9586	0.8827	0.9342
MLP	22231.5	0.6160	0.6760	0.9128	0.9530	0.8809	0.9326
NeuMF	18960	**0.6126**	**0.6607**	**0.9129**	**0.9569**	**0.8833**	**0.9344**

As can be seen from table 4, model NeuMF gives the best results in both cases.

5.4.2 Case 2: Movie embedding vectors and user-review-mean-vector are the inputs of neural collaborative filtering models

In this case, instead of using one-hot vectors as input, the embedding vectors are used. The user-review-mean-vector is represented for the user. Basing on the way to generate item embedding vectors, our experimental design includes two cases.

- ***Case 2.1*:** Movie embedding vectors are **onto vectors** and user embedding vectors are the **user-review-mean-vector**

Table 5: Evaluation results of Case 2.1

Model	Training time (s)	Numeric class (rating score)		Nominal class (like/dislike)			
		MAE	MSE	Precision	Recall	Accuracy	F1-score
GMF	238.4	0.8280	1.0977	0.8643	**0.9998**	0.8642	**0.9271**
MLP	322.8	0.7388	0.9162	0.8802	0.9773	0.8654	0.9262
NeuMF	1151.3	**0.7329**	**0.9101**	**0.8819**	0.9762	**0.8665**	0.9267

In the numeric class, the NeuMF gives the best results.

In the case of nominal class, NeuMF gives the best precision and accuracy, while the GMF model gives the best recall. The GMF model returns the highest F1-score, so it is chosen as the best classification model in this case.

- ***Case 2.2:*** Movie embedding vectors are the concatenation of **onto vector and review vector**, and embedding vectors are the **user-review-mean-vectors**

Table 6: Evaluation results of case 2.2

Model	Training time (s)	Numeric class (rating score)		Nominal class (like/dislike)			
		MAE	MSE	Precision	Recall	Accuracy	F1-score
GMF	1179.2	0.8259	1.0930	0.8643	**0.9995**	0.8640	**0.9270**
MLP	799.8	0.7475	0.9172	**0.8833**	0.9692	0.8628	0.9243
NeuMF	1617.5	**0.7372**	**0.9123**	0.8823	0.9749	**0.8659**	0.9263

In the numeric class, the NeuMF model gives the lowest MAE and MSE, so it is chosen as the best model in this case.

In the nominal class, the MLP model gives the best precision, the GMF gives the best recall and the NeuMF gives the best accuracy. The GMF model returns the highest F1-score, so it is chosen as the best classification model in this case.

5.4.3 Case 3: Movies embedding vectors and seq2vec are inputs of neural collaborative filtering models.

In this case, the seq2vec – the result of applying the Doc2Vec model on sequential data. Basing on the way movie embedding vectors are generated, we have 3 small experiment cases.

- ***Case 3.1***: Movie embedding vectors are the **onto vectors**, and user embedding vectors are the **seq2vec** (apply doc2vec model in the sequential data)

Table 7: Evaluation results of case 2.3

Model	Training time (s)	Numeric class (rating score)		Nominal class (like/dislike)			
		MAE	MSE	Precision	Recall	Accuracy	F1-score
GMF	243	0.8293	1.1193	0.8643	**1**	0.8543	0.9272
MLP	1950.3	**0.6556**	**0.7415**	0.9085	0.9504	**0.8744**	**0.9290**
NeuMF	2133.4	0.6585	0.7474	**0.9089**	0.9483	0.8732	0.9282

In the numeric class, the MLP model gives the lowest MAE and MSE. So, it is chosen as the best model.

For the nominal class, the NeuMF model gives the best precision, the GMF model gives the best recall while the MLP model gives the best accuracy. The MLP model returns the highest F1-score, so it is chosen as the best classification model in this case.

- ***Case 3.2***: Movie embedding vectors are the concatenation of **onto vector and review vector**, while the user embedding vectors are the **seq2vec** (apply doc2vec to sequential data)

Table 8: Evaluation results for case 3.2

Model	Training time (s)	Numeric class (rating score)		Nominal class (like/dislike)			
		MAE	MSE	Precision	Recall	Accuracy	F1-score
GMF	768.9	0.8278	1.1092	0.8643	**0.9999**	0.8642	0.9272
MLP	2224.1	0.6570	**0.7473**	**0.9111**	0.9431	0.8713	0.9269
NeuMF	1821.8	**0.6564**	0.7492	0.9084	0.9484	**0.8728**	**0.9280**

In the numeric class, the MLP gives the best MSE value while MLP gives the lowest MAE. Since the MSE value is the first criterion, the MLP model is chosen as the best model.

In the nominal class, the MLP gives the best precision, GMF gives the best recall while NeuMF returns the best accuracy. The NeuMF model returns the highest F1-score, so it is chosen as the best classification model in this case.

- ***Case 3.3:*** Movie embedding vectors are the concatenation of **onto vector, review vector, and poster vector** while the user embedding vectors are the **seq2vec** (apply doc2vec to sequential data)

Table 9: Evaluation results for case 3.3

Model	Training time (s)	Numeric class (rating score)		Nominal class (like/dislike)			
		MAE	MSE	Precision	Recall	Accuracy	F1-score
GMF	821.4	0.8285	1.1101	0.8643	**0.9998**	0.8641	0.9271
MLP	1793.1	**0.6617**	**0.7522**	0.9084	0.9480	**0.8724**	**0.9278**
NeuMF	1981.2	0.6648	0.7585	**0.9143**	0.9312	0.8651	0.9227

In the numeric class, the MLP model returns the best performance.

In the nominal class, the NeuMF model gives the highest precision, the GMF model returns the best recall while the highest accuracy is returned by the MLP model. The MLP model returns the highest F1-score, so it is chosen as the best classification model in this case.

5.4.4 Case 4: Movies embedding vectors and sequential preference (apply RNN-based model in sequential data) are inputs of neural collaborative filtering models

In this case, movie vectors are onto vectors and user vectors are sequential preference vectors learned from the RNN-based network. Basing on the type of the RNN- based network used, there are two small cases are designed.

- ***Case 4.1***: Movie embedding vectors are **onto vectors** and user vectors are sequential vectors learned using the **LSTM** model.

Table 10:Evaluation results for case 4.1

Model	Training time (s)	Numeric class (rating score)		Nominal class (like/dislike)			
		MAE	MSE	Precision	Recall	Accuracy	F1-score
GMF	768.0	0.8236	1.1031	0.8643	**0.9998**	0.8641	**0.9271**
MLP	1518.5	0.7424	**0.9214**	**0.8792**	0.9782	0.8650	0.9261
NeuMF	1487.8	**0.7380**	0.9225	0.8781	0.9809	**0.8658**	0.9267

For the task of predicting rating scores, the NeuMF model returns the lowest MAE while the MLP model gives the smallest MSE. In this case, the MLP model is chosen as the best model.

For the like/dislike binary classification task, the MLP model gives the highest precision, the GMF model returns the highest recall, while the highest accuracy is returned by the NeuMF model. The GMF model returns the highest F1-score, so it is chosen as the best one.

- ***Case 4.2***: Movie embedding vectors are **onto vectors** and user vectors are sequential vectors learned using the **GRU** model.

Table 11: Evaluation results the case 4.2

Model	Training time (s)	Numeric class (rating score)		Nominal class (like/dislike)			
		MAE	MSE	Precision	Recall	Accuracy	F1-score
GMF	259	0.8193	1.0911	0.8643	**1.0**	0.8643	0.9272
MLP	2728.9	0.7409	0.9190	0.8752	0.9872	**0.8673**	**0.9278**
NeuMF	2925	**0.7399**	**0.9180**	**0.8773**	0.9829	0.8664	0.9271

For the task of predicting rating scores, the NeuMF model returns the lowest MAE and MSE and is chosen as the best one.

For the like/dislike binary classification task, the NeuMF model gives the highest precision, the GMF model returns the highest recall, while the highest accuracy is returned by the MLP model. The MLP model returns the highest F1-score, so it is chosen as the best classification model.

5.4.5 Case 5: Movies embedding vectors, seq2vec (apply Doc2Vec in sequential data), and user-review-mean-vector are inputs of neural collaborative filtering models.

In case 5, the input of neural collaborative filtering models includes three vectors: movie embedding vector (onto vector), the seq2vec for user representation, and the relationship between user and items represented by user-review-mean-vector.

Table 12: Evaluation results for case 5

Model	Training time (s)	Numeric class (rating score)		Nominal class (like/dislike)			
		MAE	MSE	Precision	Recall	Accuracy	F1-score
GMF	462.9	0.8301	1.1123	0.8643	**1.0**	0.8643	**0.9272**
MLP	1867.5	0.6848	0.8031	0.9052	0.9499	**0.8707**	0.9270
NeuMF	5154.8	**0.6842**	**0.8017**	**0.9093**	0.9417	0.8684	0.9252

The NeuMF model returns the best performance in predicting rating scores.

In the nominal class, the NeuMF gives the best precision, GMF returns the best recall while MLP shows its best performance in accuracy metric. The GMF model returns the highest F1-score, so it is chosen as the best classification model in this case.

The summarization of performance for each type of collaborative filtering model:

- Generalized Matrix Factorization (GMF)

Table 13: The performance summarization for the GMF model

Case	Training time (s)	Numeric class (rating score)		Nominal class (like/dislike)			
		MAE	MSE	Precision	Recall	Accuracy	F1-score
1 (Original)	6912.2	**0.6185**	**0.6756**	**0.9111**	0.9586	**0.8827**	**0.9342**
2.1	238.4	0.8280	1.0977	0.8643	0.9998	0.8642	0.9271
2.2	1179.2	0.8259	1.0930	0.8643	0.9995	0.8640	0.9270
3.1	**243.0**	0.8293	1.1193	0.8643	**1.0**	0.8543	**0.9272**
3.2	768.9	0.8278	1.1092	0.8643	**0.9999**	0.8642	**0.9272**
3.3	821.4	0.8285	1.1101	0.8643	0.9998	0.8641	0.9271
4.1	768	0.8236	1.1031	0.8643	0.9998	0.8641	0.9271
4.2	259.0	**0.8193**	**1.0911**	0.8643	**1.0**	**0.8643**	**0.9272**
5	462.9	0.8301	1.1123	0.8643	**1.0**	**0.8643**	**0.9272**

As can be seen from table 13:

- *In the predicting rating score task*, the original model (using one-hot vectors for inputs) gives the lowest MAE and MSE.
- *In the like/dislike classification task*, the original model gives the highest precision and accuracy, while the model in case 3.1 (the onto vector is represented for a movie and the seq2vec is represented for a user), case 4.2 (the onto vector is represented for the movie, and the sequential preference learned from GRU) and case 5 (three vectors are used as inputs) gives the highest recall (100%). The original model returns the highest F1-score (0.9342), while the highest F1-score that semantic models can achieve is 0.9272.
- *For the training time,* the original model takes more time for training than the others. The average training time for semantic GMF models is 592.6 seconds, while the original model requires 6912.2 seconds for training (11.7 times slower).

- Multi layers Perceptron (MLP)

Table 14: The performance summarization for the MLP model

Case	Training time (s)	Numeric class (rating score)		Nominal class (like/dislike)			
		MAE	MSE	Precision	Recall	Accuracy	F1-score
1 (Original)	22231.5	**0.6160**	**0.6760**	**0.9128**	0.9530	**0.8809**	**0.9326**
2.1	**322.8**	0.7388	0.9162	0.8802	0.9773	0.8654	0.9262
2.2	**799.8**	0.7475	0.9172	0.8833	0.9692	0.8628	0.9243
3.1	1950.3	**0.6556**	**0.7415**	0.9085	0.9504	**0.8744**	**0.9290**
3.2	2224.1	0.6570	0.7473	**0.9111**	0.9431	0.8713	0.9269
3.3	1793.1	0.6617	0.7522	0.9084	0.9480	0.8724	0.9278
4.1	1518.5	0.7424	0.9214	0.8792	**0.9782**	0.8650	0.9261
4.2	2728.9	0.7409	0.9190	0.8752	**0.9872**	0.8673	0.9278
5	1867.5	0.6848	0.8031	0.9052	0.9499	0.8707	0.9270

As can be seen from table 14:

- *In the predicting rating score task*, the original model (using one-hot vectors for inputs) gives the lowest MAE (0.6160) and MSE (0.6760). The best semantic model (MLP) has the MAE and MSE values are 0.6556 and 0.7415.
- *In the like/dislike classification task*, the original model gives the highest precision and accuracy, while the model in case 4.2 (the onto vector is represented for a movie and the sequential preference learned from the GRU model is represented for the user) gave the highest recall (98.72%). The original model returns the highest F1-score, while the best sematic MLP model (case 3.1 – movie vectors are onto vectors and user vectors are seq2vec vectors) gives the F1-score value is 0.9290.
- *For the training time,* the original model takes more time for training than the others. The average training time for semantic MLP models is 1650.6 seconds, while the original model requires 22231.5 seconds for training (13.5 times slower).

- Neural Matrix Factorization (NeuMF)

Table 15: The performance summarization for the NeuMF model

Case	Training time (s)	Numeric class (rating score)		Nominal class (like/dislike)			
		MAE	MSE	Precision	Recall	Accuracy	F1-score
1 (Original)	18960	**0.6126**	**0.6607**	**0.9129**	0.9569	**0.8833**	**0.9344**
2.1	1151.3	0.7329	0.9101	0.8819	0.9762	0.8665	0.9267
2.2	1617.5	0.7372	0.9123	0.8823	0.9749	0.8659	0.9263
3.1	2133.4	0.6585	**0.7474**	0.9089	0.9483	**0.8732**	**0.9282**
3.2	1821.8	**0.6564**	0.7492	0.9084	0.9484	0.8728	0.9280
3.3	1981.2	0.6648	0.7585	**0.9143**	0.9312	0.8651	0.9227
4.1	1487.8	0.7380	0.9225	0.8781	**0.9809**	0.8658	0.9267
4.2	2925	0.7399	0.9180	0.8773	**0.9829**	0.8664	0.9271
5	5154.8	0.6842	0.8017	0.9093	0.9417	0.8684	0.9252

As can be seen from table 15:

- *In the predicting rating score task*, the original model (using one-hot vectors for inputs) gave the lowest MAE (0.6126) and MSE (0.6607). The best semantic NeuMF model has the MAE and MSE values are 0.6585 and 0.7474.
- *In the like/dislike classification task*, the original model gave the highest accuracy. The model in case 4 (the onto vector is represented for a movie and the sequential preference learned from the GRU model is represented for the user) gave the highest recall (98.29%), and the model in case 3.3 (the concatenation of onto vector, review vector and poster vector is represented for an item and the seq2vec is represented for a user) return the highest precision (91.43%). The original model returns the highest F1-score (0.9344), while the best semantic NeuMF model that can achieve the F1-score value is 0.9282.
- *For the training time,* the original model takes more time for training than the others. The average training time for semantic NeuMF models is 2284.1

seconds, while the original model requires 18960 seconds for training (8.3 times slower).

The summarization of the best model in each case.

- For the predicting rating score task

Table 16: Performance summarization for the predicting rating score task

Case	Model name	Training time (s)	MAE	MSE
1 (Original)	NeuMF	18960	**0.6126**	**0.6607**
2.1	NeuMF	**1151.3**	0.7329	0.9101
2.2	NeuMF	1617.5	0.7372	0.9123
3.1	MLP	1950.3	**0.6556**	**0.7415**
3.2	MLP	2224.1	0.6570	0.7473
3.3	MLP	1793.1	0.6617	0.7522
4.1	MLP	**1518.5**	0.7424	0.9214
4.2	NeuMF	2925.0	0.7399	0.9180
5	NeuMF	5154.8	0.6842	0.8017

  - *For the comparison between all models:*
    - *Performance:* The original model gives the lowest MSE (0.6607) and MAE (0.6126). The best sematic NCF model in case 3.1 (onto vectors are represented for movies and seq2vec vectors are represented for users) achieves the MSE and MAE values are 0.7415 and 0.6556.
    - *For the training time:* The average training time for semantic models is 2291.8 seconds, which is 8.3 times faster than the original model. The best semantic NCF model in the numeric class requires 1950.3 seconds, which is 9.7 times faster than the original.
  - *For the comparison between semantic models:*
    - The semantic NeuMF model gives the best performance in case 2 (the user review mean vectors are represented for users), 4.2 (the sequential preferences learned from GRU are represented for users), and 5 (using three embedding vectors as inputs), while the semantic MLP model returns the best performance in case 3 (using seq2vec vectors are represented for users),

case 4.1 (the sequential preferences learned from LSTM are represented for users), and also in all semantic models. In case 3, case 3.1 (onto vectors are represented for movies) gives the best performance.

- For the like/dislike binary classification task

Table 17: The performance summarization for the like/dislike classification task

Case	Model name	Training time (s)	Precision	Recall	Accuracy	F1-score
1 (Original)	NeuMF	18960	**0.9129**	0.9569	**0.8833**	**0.9344**
2.1	GMF	238.4	0.8643	**0.9998**	0.8642	0.9271
2.2	GMF	1179.2	0.8643	0.9995	0.8640	0.9270
3.1	MLP	1950.3	**0.9085**	0.9504	**0.8744**	**0.9290**
3.2	NeuMF	1821.8	0.9084	0.9484	0.8728	0.9280
3.3	MLP	1793.1	0.9084	0.9480	0.8724	0.9278
4.1	GMF	**768**	0.8643	**0.9998**	0.8641	0.9271
4.2	MLP	2728.9	0.8752	0.9872	0.8673	0.9278
5	GMF	**462.9**	0.8643	**1.0**	0.8643	0.9272

  - *For the comparison between all models:*
    - *For the performance:* The original model returns the highest precision, accuracy, and F1-score, while the model in case 5 (using three embedding vectors as inputs) gives the best recall. If we pay attention to the classification task, the original model is the best one since it provides the highest F1-score (0.9344). In RSs, the first criterion is the model does not miss any movies that users like, which leads to high recall. Basing on that criteria, the semantic NCF model in case 5 is the best one since it can detect 100% of movies that users like.
    - *For the training time:* The average training time for semantic models is 1367.8 seconds, which is 13.9 times faster than the original model. The NCF semantic model provides the highest recall (in case 5) requires 462.9 seconds, which is 41 times faster than the original, while the best semantic classification model (case 3.1) requires 1950.3 seconds, which is 9.7 times faster than the original.

- *For the comparison between semantic models:*
  - *Overall performance*: The MLP model in case 3.1 returns the best F1-score, which is also the best classification model, while the model in case 5 gives the highest recall, which can be chosen as the best recommendation model.

- *Conclusion for movie/user representations*:
  - *Movie representation*: As can be seen from the result of case 2 and case 3 in table 16 and 17, semantic NCF models using **onto vectors** for movie representations return a better result than other combinations.
  - *User representation:*
    - As can be seen from the result of cases 2, 3, and 4 in table 16 and 17, semantic NCF models using **seq2vec** (apply Doc2Vec in the sequential data) for user representations produce the best performance.
    - From the results in cases 4.1, and 4.2, sequential preferences learned from the GRU network return better performance in both experimental cases.

Conclusion:

- **The original model:**
  - Cons: they provide best performances in the predicting rating score task (0.6126 for MAE, and 0.6607 for MSE), and in the like/dislike classification task (0.9344 for F1-score).
  - Pros: Since original models give their outperform in the performance criteria, original models have some disadvantages:
    - *More memory requires in the training phase*: Since their inputs are one-hot encoding vectors, the size of those vectors is equal to the number of users/ items in the training dataset, if we have a large number of users and movies, the required memory to store those vectors are huge.
    - *Time for training is too large*: the GMF model requires 6912.2 seconds, the MLP requires 22231.5 seconds, and the NeuMF needs 18960 seconds for training.

- *Cold-start problem*: Since those models using the identity of a user and an item as the input, then transforming it into a one-hot encoding vector, so it faces the cold-start problem.

- **The semantics models**
  - Cons: Some limitations of original NCF models that can be solved when using semantic NCF models are:
    - *Less memory for storing user/item representation vectors*: Since the dimensional of embedding vectors are too smaller than the one-hot encoding vectors, less memory is required for storing those embedding vectors
    - *Less training time*: Semantic GMF, MLP, and NeuMF models need on average 592.6 seconds, 1650.6 seconds, and 2284.1 seconds respectively in the training phase, which are corresponding 11.7 times, 13.5 times, and 8.3 times faster than original models.
    - *Handling the cold-start problem*: By using embedding vectors to represent users and items, the cold-start problem can be solved. On the side of the problem of the new item, its information is put into knowledge bases then its embedding vector is generated. On the other hand, new user vector representation can be the average of all user vectors, so the new user problem is handled.
  - Pros: Those semantic NCF model's performance is worse than original models on the side of predicting rating score and like/dislike classification.
    - In the task of *predicting rating score*, MAE and MSE values of the best original model are 0.6126 and 0.6676, while those values of the best semantic model are 0.6556 and 0.7415.
    - In the task of *like/dislike classification*, the best original model gives the precision, recall, accuracy, and F1-score values are 0.9129, 0.9569, 0.8833, and 0.9344, while those values of the best semantic model are 0.9085, 0.9504, 0.8744, 0.9290.
    - However, *in making recommendations,* the recall value is more important than the precision value. High recall means that the model can detect all

liked movies of users. In this criterion, the best recall value of original models is 0.9586, while this value in semantic models is 1.0, which means that it can point out all movies that users like.

- From the experiment results, we can see that the accuracy of semantic models is quite near to the original model's performance. The performance of semantic neural collaborative filtering models depends on the quality of user/item representation vectors. To increase their performance, the embedding vectors need to be improved.

CHAPTER 6: CONCLUSION AND FUTURE WORKS

The neural network-based collaborative filtering proposed by Xiangnan He et. al [26] gave a good performance on RSs. However, it faces the cold-start problem and lack of semantic meaning. In this thesis, we proposed semantic enhancing in deep learning models for recommendation systems. Two phases are building the knowledge base and training neural collaborative filtering models. Firstly, the knowledge base for the movie domain is built, then deep learning techniques are used to embed them into vector space. Next, the sequential data is converted into latent vectors using deep learning models such as LSTM, Doc2Vec. Finally, user and item latent vectors are fed into neural collaborative filtering to predict the user preference. From the experimental results, models using semantic embedding vectors as inputs require less memory and faster training than models using a one-hot vector as input. Their performance in the predicting rating score (numeric class) is worse than the original model (using one-hot vectors). However, in the user preference classification (like/dislike), they gave a better recall, which is more important in recommendation since we do not want to miss any movies the user may like. One more important thing is that the proposed method can solve the cold-start problem.

Some works are planned to do in the future:

- Improve the embedding vectors: research and apply some techniques, such as OWL2Vec* [105] for ontology embedding, the seq2vec model proposed by Kim et. al [106] for sequential representation.
- Apply the proposed method for implicit feedback for the next item prediction.

REFERENCES

[1] Lops, P., De Gemmis, M., Semeraro, G., *Recommender Systems Handbook*. Springer, Boston, 2011.

[2] L. J. Grace, V. Maheswari, and D. Nagamalai, "Efficient Personalized Web Mining: Utilizing the Most Utilized Data," in *International Conference on Advances in Computing and Information Technology*, 2011, pp. 418–426.

[3] A. Sattar, M. Ghazanfar, and M. Iqbal, "Building Accurate and Practical Recommender System Algorithms Using Machine Learning Classifier and Collaborative Filtering.," *Arab. J. Sci. Eng. Springer Sci. Bus. Media BV*, vol. 42, no. 8, 2017.

[4] H. Zhu *et al.*, "Learning tree-based deep model for recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1079–1088.

[5] P. Pu, L. Chen, and R. Hu, "A user-centric evaluation framework for recommender systems," in *Proceedings of the fifth ACM conference on Recommender systems*, 2011, pp. 157–164.

[6] B. Pathak, R. Garfinkel, R. D. Gopal, R. Venkatesan, and F. Yin, "Empirical analysis of the impact of recommender systems on sales," *J. Manag. Inf. Syst.*, vol. 27, no. 2, pp. 159–188, 2010.

[7] R. Hu and P. Pu, "Acceptance issues of personality-based recommender systems," in *Proceedings of the third ACM conference on Recommender systems*, 2009, pp. 221–224.

[8] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowl.-Based Syst.*, vol. 46, pp. 109–132, 2013.

[9] X. Amatriain, A. Jaimes, N. Oliver, and J. M. Pujol, "Data mining methods for recommender systems," in *Recommender systems handbook*, Springer, 2011, pp. 39–71.

[10] G. Linden, B. Smith, and J. York, "Amazon. com recommendations: Item-to-item collaborative filtering," *IEEE Internet Comput.*, vol. 7, no. 1, pp. 76–80, 2003.

[11] K. Yi, T. Chen, and G. Cong, "Library personalized recommendation service method based on improved association rules," *Libr. Hi Tech*, 2018.

[12] M. F. Adak and M. Uçar, “A Book Recommendation System Using Decision Tree-based Fuzzy Logic for E-Commerce Sites,” in *2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2021, pp. 1–5.

[13] X. Wang, F. Luo, C. Sang, J. Zeng, and S. Hirokawa, “Personalized movie recommendation system based on support vector machine and improved particle swarm optimization,” *IEICE Trans. Inf. Syst.*, vol. 100, no. 2, pp. 285–293, 2017.

[14] V. Kant, T. Jhalani, and P. Dwivedi, “Enhanced multi-criteria recommender system based on fuzzy Bayesian approach,” *Multimed. Tools Appl.*, vol. 77, no. 10, pp. 12935–12953, 2018.

[15] J. Gope and S. K. Jain, “A survey on solving cold start problem in recommender systems,” in *2017 International Conference on Computing, Communication and Automation (ICCCA)*, 2017, pp. 133–138.

[16] G. Guo, H. Qiu, Z. Tan, Y. Liu, J. Ma, and X. Wang, “Resolving data sparsity by multi-type auxiliary implicit feedback for recommender systems,” *Knowl.-Based Syst.*, vol. 138, pp. 202–207, 2017.

[17] R. Burke, “Hybrid recommender systems: Survey and experiments,” *User Model. User-Adapt. Interact.*, vol. 12, pp. 331–370, 2002.

[18] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 6, pp. 734–749, 2005.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[20] Y. Song, A. M. Elkahky, and X. He, “Multi-rate deep learning for temporal recommendation,” in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, 2016, pp. 909–912.

[21] E. Smirnova and F. Vasile, “Contextual sequence modeling for recommendation with recurrent neural networks,” in *Proceedings of the 2nd workshop on deep learning for recommender systems*, 2017, pp. 2–9.

[22] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, “Session-based recommendations with recurrent neural networks,” *ArXiv Prepr. ArXiv151106939*, 2015.

[23] R. He and J. McAuley, “VBPR: visual bayesian personalized ranking from implicit feedback,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016, vol. 30, no. 1.

[24] H. T. Nguyen, M. Wistuba, J. Grabocka, L. R. Drumond, and L. Schmidt-Thieme, “Personalized deep learning for tag recommendation,” in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2017, pp. 186–197.

[25] G. Lee, J. Jeong, S. Seo, C. Kim, and P. Kang, “Sentiment classification with word localization based on weakly supervised learning with a convolutional neural network,” *Knowl.-Based Syst.*, vol. 152, pp. 70–82, 2018.

[26] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, “Neural collaborative filtering,” in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.

[27] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, “DeepFM: a factorization-machine based neural network for CTR prediction,” *ArXiv Prepr. ArXiv170304247*, 2017.

[28] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, “Autorec: Autoencoders meet collaborative filtering,” in *Proceedings of the 24th international conference on World Wide Web*, 2015, pp. 111–112.

[29] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester, “Collaborative denoising auto-encoders for top-n recommender systems,” in *Proceedings of the ninth ACM international conference on web search and data mining*, 2016, pp. 153–162.

[30] J. Chen, H. Zhang, X. He, L. Nie, W. Liu, and T.-S. Chua, “Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention,” in *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, 2017, pp. 335–344.

[31] B. Hu, C. Shi, W. X. Zhao, and P. S. Yu, “Leveraging meta-path based context for top-n recommendation with a neural co-attention model,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1531–1540.

[32] Q. Liu, Y. Zeng, R. Mokhosi, and H. Zhang, “STAMP: short-term attention/memory priority model for session-based recommendation,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1831–1839.

[33] F. Zhang, N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma, "Collaborative knowledge base embedding for recommender systems," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 353–362.

[34] D. Yang, Z. Guo, Z. Wang, J. Jiang, Y. Xiao, and W. Wang, "A knowledge-enhanced deep recommendation framework incorporating gan-based models," in *2018 IEEE International Conference on Data Mining (ICDM)*, 2018, pp. 1368–1373.

[35] J. Huang, W. X. Zhao, H. Dou, J.-R. Wen, and E. Y. Chang, "Improving sequential recommendation with knowledge-enhanced memory networks," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2018, pp. 505–514.

[36] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion," 2015.

[37] A. Miller, A. Fisch, J. Dodge, A.-H. Karimi, A. Bordes, and J. Weston, "Key-value memory networks for directly reading documents," *ArXiv Prepr. ArXiv160603126*, 2016.

[38] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[39] K. Yeturu, "Machine learning algorithms, applications, and practices in data science," in *Handbook of Statistics*, vol. 43, Elsevier, 2020, pp. 81–206.

[40] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Application of dimensionality reduction in recommender system-a case study," Minnesota Univ Minneapolis Dept of Computer Science, 2000.

[41] A. Mnih and R. R. Salakhutdinov, "Probabilistic matrix factorization," in *Advances in neural information processing systems*, 2008, pp. 1257–1264.

[42] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[43] M. W. Gardner and S. R. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," *Atmos. Environ.*, vol. 32, no. 14–15, pp. 2627–2636, 1998.

[44] A. F. Agarap, “Deep learning using rectified linear units (relu),” *ArXiv Prepr. ArXiv180308375*, 2018.

[45] B. Karlik and A. V. Olgac, “Performance analysis of various activation functions in generalized MLP architectures of neural networks,” *Int. J. Artif. Intell. Expert Syst.*, vol. 1, no. 4, pp. 111–122, 2011.

[46] R. Hecht-Nielsen, “Theory of the backpropagation neural network,” in *Neural networks for perception*, Elsevier, 1992, pp. 65–93.

[47] J. Guo, “Backpropagation through time,” *Unpubl Ms Harbin Inst. Technol.*, vol. 40, pp. 1–6, 2013.

[48] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 6, no. 02, pp. 107–116, 1998.

[49] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[50] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *ArXiv Prepr. ArXiv14123555*, 2014.

[51] S. Yang, X. Yu, and Y. Zhou, “LSTM and GRU neural network performance comparison study: Taking Yelp review dataset as an example,” in *2020 International workshop on electronic communication and artificial intelligence (IWECAI)*, 2020, pp. 98–101.

[52] A. N. Shewalkar, “Comparison of rnn, lstm and gru on speech recognition data,” 2018.

[53] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[54] X. Rong, “word2vec parameter learning explained,” *ArXiv Prepr. ArXiv14112738*, 2014.

[55] Q. Le and T. Mikolov, “Distributed representations of sentences and documents,” in *International conference on machine learning*, 2014, pp. 1188–1196.

[56] S. Kannan *et al.*, “Preprocessing techniques for text mining,” *Int. J. Comput. Sci. Commun. Netw.*, vol. 5, no. 1, pp. 7–16, 2014.

[57] M. Rajman and R. Besançon, “Text mining: natural language techniques and text mining applications,” in *Data mining and reverse engineering*, Springer, 1998, pp. 50–64.

[58] P. Willett, “The Porter stemming algorithm: then and now,” *Program*, 2006.

[59] R. Hooper and C. Paice, “The Lancaster stemming algorithm,” *Univ. Lancester*, 2005.

[60] M. F. Porter, “Snowball: A language for stemming algorithms.” 2001.

[61] J. Plisson, N. Lavrac, and D. Mladenic, “A rule based approach to word lemmatization,” in *Proceedings of IS*, 2004, vol. 3, pp. 83–86.

[62] D. Kimothi, A. Soni, P. Biyani, and J. M. Hogan, “Distributed representations for biological sequence analysis,” *ArXiv Prepr. ArXiv160805949*, 2016.

[63] O. Barkan and N. Koenigstein, “Item2vec: neural item embedding for collaborative filtering,” in *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2016, pp. 1–6.

[64] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks,” in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[65] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: Online learning of social representations,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[66] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights Imaging*, vol. 9, no. 4, pp. 611–629, 2018.

[67] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *ArXiv Prepr. ArXiv14091556*, 2014.

[68] C. Szegedy *et al.*, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[69] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[70] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[71] T. Gruber, "Ontology.," *Encycl. Database Syst.*, vol. 1, pp. 1963–1965, 2009.

[72] B. Smith, "Ontology," in *The furniture of the world*, Brill, 2012, pp. 47–68.

[73] E. Miller, "An introduction to the resource description framework," *Bull. Am. Soc. Inf. Sci. Technol.*, vol. 25, no. 1, pp. 15–19, 1998.

[74] J. Z. Pan, "Resource description framework," in *Handbook on ontologies*, Springer, 2009, pp. 71–90.

[75] D. L. McGuinness and F. Van Harmelen, "OWL web ontology language overview," *W3C Recomm.*, vol. 10, no. 10, p. 2004, 2004.

[76] F. Spitzer, *Principles of random walk*, vol. 34. Springer Science & Business Media, 2013.

[77] F. Xia, J. Liu, H. Nie, Y. Fu, L. Wan, and X. Kong, "Random walks: A review of algorithms and applications," *IEEE Trans. Emerg. Top. Comput. Intell.*, vol. 4, no. 2, pp. 95–107, 2019.

[78] J. Kang and J. Choi, "An ontology-based recommendation system using long-term and short-term preferences," in *2011 International Conference on Information Science and Applications*, 2011, pp. 1–8.

[79] I. Cantador, A. Bellogín, and P. Castells, "Ontology-based personalised and context-aware recommendations of news items," in *2008 IEEE/WIC/ACM international conference on web intelligence and intelligent agent technology*, 2008, vol. 1, pp. 562–565.

[80] W. IJntema, F. Goossen, F. Frasincar, and F. Hogenboom, "Ontology-based news recommendation," in *Proceedings of the 2010 EDBT/ICDT Workshops*, 2010, pp. 1–6.

[81] S. Fraihat and Q. Shambour, "A framework of semantic recommender system for e-learning," *J. Softw.*, vol. 10, no. 3, pp. 317–330, 2015.

[82] M. E. Ibrahim, Y. Yang, D. L. Ndzi, G. Yang, and M. Al-Maliki, "Ontology-based personalized course recommendation framework," *IEEE Access*, vol. 7, pp. 5180–5199, 2018.

[83] "MovieLens," *GroupLens*, Sep. 06, 2013. https://grouplens.org/datasets/movielens/ (accessed Jul. 24, 2021).

[84] “IMDb.” https://www.imdb.com/interfaces/ (accessed Jul. 24, 2021).

[85] S. Chaib, H. Yao, Y. Gu, and M. Amrani, “Deep feature extraction and combination for remote sensing image classification based on pre-trained CNN models,” in *Ninth International Conference on Digital Image Processing (ICDIP 2017)*, 2017, vol. 10420, p. 104203D.

[86] M. S. Hasan, “An application of pre-trained CNN for image classification,” in *2017 20th International Conference of Computer and Information Technology (ICCIT)*, 2017, pp. 1–6.

[87] U. K. Lopes and J. F. Valiati, “Pre-trained convolutional neural networks as feature extractors for tuberculosis detection,” *Comput. Biol. Med.*, vol. 89, pp. 135–143, 2017.

[88] T. S. Nazare, R. F. de Mello, and M. A. Ponti, “Are pre-trained CNNs good feature extractors for anomaly detection in surveillance videos?,” *ArXiv Prepr. ArXiv181108495*, 2018.

[89] S. Rajaraman *et al.*, “Pre-trained convolutional neural networks as feature extractors toward improved malaria parasite detection in thin blood smear images,” *PeerJ*, vol. 6, p. e4568, 2018.

[90] K. Team, “Keras documentation: DenseNet.” https://keras.io/api/applications/densenet/#densenet201-function (accessed Jul. 26, 2021).

[91] H. Abdi and L. J. Williams, “Principal component analysis,” *Wiley Interdiscip. Rev. Comput. Stat.*, vol. 2, no. 4, pp. 433–459, 2010.

[92] R. Rehurek and P. Sojka, “Software framework for topic modelling with large corpora,” 2010.

[93] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[94] F. Chollet, *Deep learning with Python*. Simon and Schuster, 2017.

[95] M. Abadi *et al.*, “Tensorflow: A system for large-scale machine learning,” in *12th ${$USENIX$}$ symposium on operating systems design and implementation (${$OSDI$}$ 16)*, 2016, pp. 265–283.

[96] J. Bergstra *et al.*, “Theano: a CPU and GPU math expression compiler,” in *Proceedings of the Python for scientific computing conference (SciPy)*, 2010, vol. 4, no. 3, pp. 1–7.

[97] J. H. Gennari *et al.*, “The evolution of Protégé: an environment for knowledge-based systems development,” *Int. J. Hum.-Comput. Stud.*, vol. 58, no. 1, pp. 89–123, 2003.

[98] M. Horridge and S. Bechhofer, “The owl api: A java api for owl ontologies,” *Semantic Web*, vol. 2, no. 1, pp. 11–21, 2011.

[99] N. Matentzoglu and I. Palmisano, “An Introduction to the OWL API,” *URL Httpsyllabus Cs Manch. Ac Ukpgt2018COMP62342introduction-Owl-Api-Msc Pdf*, 2016.

[100] “IMDbPY.” https://imdbpy.github.io/ (accessed Jul. 24, 2021).

[101] C. Sammut and G. I. Webb, Eds., “Mean Squared Error,” in *Encyclopedia of Machine Learning*, Boston, MA: Springer US, 2010, pp. 653–653. doi: 10.1007/978-0-387-30164-8_528.

[102] C. J. Willmott and K. Matsuura, “Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance,” *Clim. Res.*, vol. 30, no. 1, pp. 79–82, 2005.

[103] C. Sammut and G. I. Webb, Eds., “Mean Absolute Error,” in *Encyclopedia of Machine Learning*, Boston, MA: Springer US, 2010, pp. 652–652. doi: 10.1007/978-0-387-30164-8_525.

[104] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks,” *Inf. Process. Manag.*, vol. 45, no. 4, pp. 427–437, 2009.

[105] J. Chen, P. Hu, E. Jimenez-Ruiz, O. M. Holter, D. Antonyrajah, and I. Horrocks, “OWL2Vec*: Embedding of OWL ontologies,” *Mach. Learn.*, pp. 1–33, 2021.

[106] H. J. Kim, S. E. Hong, and K. J. Cha, “seq2vec: Analyzing sequential data using multi-rank embedding vectors,” *Electron. Commer. Res. Appl.*, vol. 43, p. 101003, 2020.

SEMANTIC DEEP LEARNING MODELS FOR USER BEHAVIOR PREDICTION

ORIGINALITY REPORT

17%	10%	9%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to International University - VNUHCM Student Paper	1%
2	www.duo.uio.no Internet Source	1%
3	link.springer.com Internet Source	1%
4	junyelee.blogspot.com Internet Source	<1%
5	towardsdatascience.com Internet Source	<1%
6	arxiv.org Internet Source	<1%
7	Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, Wei-Ying Ma. "Collaborative Knowledge Base Embedding for Recommender Systems", Proceedings of the 22nd ACM SIGKDD International Conference	<1%

on Knowledge Discovery and Data Mining - KDD '16, 2016
Publication

8	insightsimaging.springeropen.com Internet Source	<1%
9	Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, Edward Y. Chang. "Improving Sequential Recommendation with Knowledge-Enhanced Memory Networks", The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval - SIGIR '18, 2018 Publication	<1%
10	"PRICAI 2018: Trends in Artificial Intelligence", Springer Science and Business Media LLC, 2018 Publication	<1%
11	export.arxiv.org Internet Source	<1%
12	Deqing Yang, Zikai Guo, Ziyi Wang, Juyang Jiang, Yanghua Xiao, Wei Wang. "A Knowledge-Enhanced Deep Recommendation Framework Incorporating GAN-Based Models", 2018 IEEE International Conference on Data Mining (ICDM), 2018 Publication	<1%
13	Submitted to University of Hong Kong Student Paper	

		<1%
14	pure.uva.nl Internet Source	<1%
15	Thi Thanh Sang Nguyen, Pham Minh Thu Do. "Classification optimization for training a large dataset with Naïve Bayes", Journal of Combinatorial Optimization, 2020 Publication	<1%
16	"Proceedings of the Future Technologies Conference (FTC) 2020, Volume 1", Springer Science and Business Media LLC, 2021 Publication	<1%
17	Sabih Ahmad Khan, Hsien-Tsung Chang. "Comparative analysis on Facebook post interaction using DNN, ELM and LSTM", PLOS ONE, 2019 Publication	<1%
18	Submitted to Imperial College of Science, Technology and Medicine Student Paper	<1%
19	dr.ntu.edu.sg Internet Source	<1%
20	www.scopus.com Internet Source	<1%

21	"Knowledge Science, Engineering and Management", Springer Science and Business Media LLC, 2019 Publication	<1 %
22	medium.com Internet Source	<1 %
23	theses.gla.ac.uk Internet Source	<1 %
24	Pan Wang, Xingquan Zuo, Congcong Guo, Ruihong Li, Xinchao Zhao, Chaomin Luo. "A multiobjective genetic algorithm based hybrid recommendation approach", 2017 IEEE Symposium Series on Computational Intelligence (SSCI), 2017 Publication	<1 %
25	Submitted to University of Dundee Student Paper	<1 %
26	Submitted to Liverpool John Moores University Student Paper	<1 %
27	Submitted to University of Edinburgh Student Paper	<1 %
28	thesai.org Internet Source	<1 %
29	Submitted to De Montfort University Student Paper	<1 %

30	Thi Thi Zin, Pyke Tin, Hiromitsu Hama. "Chapter 29 Deep Learning Model for Integration of Clustering with Ranking in Social Networks", Springer Science and Business Media LLC, 2017 Publication	<1 %
31	"Malware Analysis Using Artificial Intelligence and Deep Learning", Springer Science and Business Media LLC, 2021 Publication	<1 %
32	Submitted to Royal Holloway and Bedford New College Student Paper	<1 %
33	SpringerBriefs in Computer Science, 2015. Publication	<1 %
34	hdl.handle.net Internet Source	<1 %
35	"Web and Big Data", Springer Science and Business Media LLC, 2020 Publication	<1 %
36	Submitted to Glasgow Caledonian University Student Paper	<1 %
37	Recommender Systems Handbook, 2015. Publication	<1 %
38	www.cs.uoregon.edu Internet Source	<1 %

39	Submitted to Facultad Latinoamericana de Ciencias Sociales (FLACSO) - Sede Ecuador Student Paper	<1 %
40	Submitted to University of Leicester Student Paper	<1 %
41	arizona.openrepository.com Internet Source	<1 %
42	waseda.repo.nii.ac.jp Internet Source	<1 %
43	www.mdpi.com Internet Source	<1 %
44	"Web-Age Information Management", Springer Science and Business Media LLC, 2013 Publication	<1 %
45	Tong Wei, Christophe Roche, Maria Papadopoulou, Yangli Jia. "Using ISO and Semantic Web standard for building a multilingual terminology e-Dictionary: A use case of Chinese ceramic vases", Journal of Information Science, 2021 Publication	<1 %
46	krex.k-state.edu Internet Source	<1 %
47	www.eecs.qmul.ac.uk Internet Source	<1 %

48	www.hindawi.com Internet Source	<1%
49	www.slideshare.net Internet Source	<1%
50	dl.acm.org Internet Source	<1%
51	github.com Internet Source	<1%
52	"Advances in Multimedia Information Processing – PCM 2018", Springer Science and Business Media LLC, 2018 Publication	<1%
53	N. Nikzad–Khasmakhi, M.A. Balafar, M. Reza Feizi–Derakhshi. "The state-of-the-art in expert recommendation systems", Engineering Applications of Artificial Intelligence, 2019 Publication	<1%
54	Ravi Nahta, Yogesh Kumar Meena, Dinesh Gopalani, Ganpat Singh Chauhan. "Embedding metadata using deep collaborative filtering to address the cold start problem for the rating prediction task", Multimedia Tools and Applications, 2021 Publication	<1%

55	Shuai Zhang, Lina Yao, Aixin Sun, Yi Tay. "Deep Learning Based Recommender System", ACM Computing Surveys, 2019 Publication	<1 %
56	Submitted to University of Southampton Student Paper	<1 %
57	etd.lsu.edu Internet Source	<1 %
58	www.researchsquare.com Internet Source	<1 %
59	"Recent Advances on Soft Computing and Data Mining", Springer Science and Business Media LLC, 2017 Publication	<1 %
60	Amol C. Adamuthe, Sneha Jagtap. "Comparative Study ofConvolutional Neural Network with Word Embedding Technique for Text Classification", International Journal of Intelligent Systems and Applications, 2019 Publication	<1 %
61	Submitted to University College London Student Paper	<1 %
62	Submitted to University of Bath Student Paper	<1 %
63	Submitted to University of Hertfordshire Student Paper	<1 %

64	Submitted to University of Westminster Student Paper	<1 %
65	staff.fnwi.uva.nl Internet Source	<1 %
66	Submitted to De La Salle University Student Paper	<1 %
67	Quangui Zhang, Li Wang, Xiangfu Meng, Keda Xu, Jiayan Hu. "A Generic Framework for Learning Explicit and Implicit User-Item Couplings in Recommendation", IEEE Access, 2019 Publication	<1 %
68	Submitted to University of Hull Student Paper	<1 %
69	Submitted to University of Reading Student Paper	<1 %
70	Xiaoyu Du, Xiangnan He, Fajie Yuan, Jinhui Tang, Zhiguang Qin, Tat-Seng Chua. "Modeling Embedding Dimension Correlations via Convolutional Neural Collaborative Filtering", ACM Transactions on Information Systems, 2019 Publication	<1 %
71	gnosis.library.ucy.ac.cy Internet Source	<1 %
	ojs.academypublisher.com	

72	Internet Source	<1%
73	papers.www2017.com.au.s3-website-ap-southeast-2.amazonaws.com Internet Source	<1%
74	www.groundai.com Internet Source	<1%
75	www.politesi.polimi.it Internet Source	<1%
76	"Artificial Intelligence Applications and Innovations", Springer Science and Business Media LLC, 2019 Publication	<1%
77	Gabriel A. Pinheiro, Johnatan Mucelini, Marinalva D. Soares, Ronaldo C. Prati, Juarez L. F. Da Silva, Marcos G. Quiles. "Machine Learning Prediction of Nine Molecular Properties Based on the SMILES Representation of the QM9 Quantum-Chemistry Dataset", The Journal of Physical Chemistry A, 2020 Publication	<1%
78	hagan.okstate.edu Internet Source	<1%
79	hal.inria.fr Internet Source	<1%

80	manning-content.s3.amazonaws.com Internet Source	<1 %
81	Benjamin A. Kwapong, Richard Anarfi, Kenneth K. Fletcher. "Chapter 3 Collaborative Learning Using LSTM-RNN for Personalized Recommendation", Springer Science and Business Media LLC, 2020 Publication	<1 %
82	Guangli Li, Jianwu Zhuo, Chuanxiu Li, Jin Hua, Tian Yuan, Zhenyu Niu, Donghong Ji, Renzhong Wu, Hongbin Zhang. "Multi-modal Visual Adversarial Bayesian Personalized Ranking Model for Recommendation", Information Sciences, 2021 Publication	<1 %
83	Qika Lin, Yaoqiang Niu, Yifan Zhu, Hao Lu, Keith Zvikomborero Mushonga, Zhendong Niu. "Heterogeneous knowledge-based attentive neural networks for short-term music recommendations", IEEE Access, 2018 Publication	<1 %
84	Recommender Systems Handbook, 2011. Publication	<1 %
85	Shaoyun Shi, Min Zhang, Yiqun Liu, Shaoping Ma. "Attention-based Adaptive Model to Unify Warm and Cold Starts Recommendation", Proceedings of the 27th ACM International	<1 %

	Conference on Information and Knowledge Management - CIKM '18, 2018 Publication	
86	Submitted to University of Wales, Bangor Student Paper	<1 %
87	Zhao Huang, Pavel Stakhiyevich. "A Time-Aware Hybrid Approach for Intelligent Recommendation Systems for Individual and Group Users", Complexity, 2021 Publication	<1 %
88	brage.bibsys.no Internet Source	<1 %
89	escholarship.org Internet Source	<1 %
90	opus4.kobv.de Internet Source	<1 %
91	workshop2015.iwslt.org Internet Source	<1 %
92	www.ijert.org Internet Source	<1 %
93	www.shichuan.org Internet Source	<1 %
94	Jin Huang, Zhaochun Ren, Wayne Xin Zhao, Gaole He, Ji-Rong Wen, Daxiang Dong. "Taxonomy-Aware Multi-Hop Reasoning Networks for Sequential Recommendation",	<1 %

	Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining - WSDM '19, 2019 Publication	
95	Lei Mei, Pengjie Ren, Zhumin Chen, Liqiang Nie, Jun Ma, Jian-Yun Nie. "An Attentive Interaction Network for Context-aware Recommendations", Proceedings of the 27th ACM International Conference on Information and Knowledge Management - CIKM '18, 2018 Publication	<1 %
96	Smart Innovation Systems and Technologies, 2016. Publication	<1 %
97	bura.brunel.ac.uk Internet Source	<1 %
98	coek.info Internet Source	<1 %
99	ijnaa.semnan.ac.ir Internet Source	<1 %
100	www.di.uniba.it Internet Source	<1 %
101	www.gelbukh.com Internet Source	<1 %
102	www.thinkmind.org Internet Source	<1 %

103	www.warse.org Internet Source	<1 %
104	"Computational Data and Social Networks", Springer Science and Business Media LLC, 2020 Publication	<1 %
105	"Digital Heritage. Progress in Cultural Heritage: Documentation, Preservation, and Protection", Springer Science and Business Media LLC, 2016 Publication	<1 %
106	"Intelligent Data Engineering and Automated Learning – IDEAL 2018", Springer Science and Business Media LLC, 2018 Publication	<1 %
107	"Knowledge Discovery, Knowledge Engineering and Knowledge Management", Springer Science and Business Media LLC, 2015 Publication	<1 %
108	"Smart Computing and Communication", Springer Science and Business Media LLC, 2018 Publication	<1 %
109	F.O. Isinkaye, Y.O. Folajimi, B.A. Ojokoh. "Recommendation systems: Principles,	<1 %

methods and evaluation", Egyptian Informatics Journal, 2015
Publication

110 Lei Zheng, Vahid Noroozi, Philip S. Yu. "Joint Deep Modeling of Users and Items Using Reviews for Recommendation", Proceedings of the Tenth ACM International Conference on Web Search and Data Mining - WSDM '17, 2017 <1%
Publication

111 Liang Chen, Angyu Zheng, Yinglan Feng, Fenfang Xie, Zibin Zheng. "Chapter 28 Software Service Recommendation Base on Collaborative Filtering Neural Network Model", Springer Science and Business Media LLC, 2018 <1%
Publication

112 Lizi Liao, Xiangnan He, Hanwang Zhang, Tat-Seng Chua. "Attributed Social Network Embedding", IEEE Transactions on Knowledge and Data Engineering, 2018 <1%
Publication

113 Peng Liu, Lemei Zhang, Jon Atle Gulla. "Dynamic attention-based explainable recommendation with textual and visual fusion", Information Processing & Management, 2020 <1%
Publication

114	Submitted to University of Sheffield Student Paper	<1%
115	Submitted to University of Warwick Student Paper	<1%
116	Weihua Yuan, Hong Wang, Baofang Hu, Lutong Wang, Qian Wang. "Wide and deep model of multi-source information-aware recommender system", IEEE Access, 2018 Publication	<1%
117	Weihua Yuan, Hong Wang, Xiaomei Yu, Nan Liu, Zhenghao Li. "Attention-based context-aware sequential recommendation model", Information Sciences, 2020 Publication	<1%
118	Xiangnan He, Tat-Seng Chua. "Neural Factorization Machines for Sparse Predictive Analytics", Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '17, 2017 Publication	<1%
119	Yiteng Pan, Fazhi He, Haiping Yu. "A Novel Enhanced Collaborative Autoencoder with Knowledge Distillation for Top-N Recommender Systems", Neurocomputing, 2018 Publication	<1%

120	ciit.finki.ukim.mk Internet Source	<1 %
121	cse.iitk.ac.in Internet Source	<1 %
122	deepai.org Internet Source	<1 %
123	docplayer.net Internet Source	<1 %
124	eprints.ucm.es Internet Source	<1 %
125	espace.library.uq.edu.au Internet Source	<1 %
126	iariajournals.org Internet Source	<1 %
127	jultika.oulu.fi Internet Source	<1 %
128	ksco.info Internet Source	<1 %
129	mafiadoc.com Internet Source	<1 %
130	ntit-conf.com Internet Source	<1 %
131	orca.cf.ac.uk Internet Source	<1 %

132	people.engr.tamu.edu Internet Source	<1 %
133	preserve.lehigh.edu Internet Source	<1 %
134	proceedings.mlr.press Internet Source	<1 %
135	rd.springer.com Internet Source	<1 %
136	silo.pub Internet Source	<1 %
137	tel.archives-ouvertes.fr Internet Source	<1 %
138	workshop2017.iwslt.org Internet Source	<1 %
139	www.act.edu.in Internet Source	<1 %
140	www.aimspress.com Internet Source	<1 %
141	www.gipp.com Internet Source	<1 %
142	www.tandfonline.com Internet Source	<1 %
143	"Information Retrieval", Springer Science and Business Media LLC, 2018	<1 %

Publication

144 Le Wu, Peijie Sun, Richang Hong, Yong Ge, Meng Wang. "Collaborative Neural Social Recommendation", IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2018 <1%
Publication

145 Zhu Sun, Qing Guo, Jie Yang, Hui Fang, Guibing Guo, Jie Zhang, Robin Burke. "Research commentary on recommendations with side information: A survey and research directions", Electronic Commerce Research and Applications, 2019 <1%
Publication

146 "Technology Trends", Springer Science and Business Media LLC, 2019 <1%
Publication

147 Hongkui Tu, Jiahui Wen, Aixin Sun, Xiaodong Wang. "Joint Implicit and Explicit Neural Networks for Question Recommendation in CQA Services", IEEE Access, 2018 <1%
Publication

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		