Ontology based hierarchical recommendation model on Deep Neural Networks

Xinghao Li[1 [0000−0003−0733−9341]], Armin Haller[1 [0000−0003−3425−0780]], and Yingnan Shi[1 [0000−0003−1694−3401]]

Australian National University, Canberra ACT 2600, Australia
{xinghao.li,armin.haller,yingnan.shi}@anu.edu.au

Abstract. Traditional algorithms in recommender systems aim to solve the recommendation problem by finding the latent connections between users and items. They mainly consider the information provided by the input datasets without exploring the inherent connections between items such as taxonomic and hierarchical relationships in a specific field. In this paper, we propose an extension to an existing deep neural network algorithm, by using an associated domain ontology as auxiliary information to support the training of the model, called Ontology-based Neural Collaborative Filtering (ONCF). Specifically, our method exploits the hierarchical properties of the item set as defined by an ontology and integrates the structural information into the training process of the deep neural network. Consequently, our algorithm is not only able to predict the exact item but also offers a recommendation to a specific class of candidate items based on the ontological relations in the knowledge graph. We demonstrate our OCNF model's comparable accuracy in terms of the quality of predictions, while at the same time having a lower computational complexity to recommend potential matches.

Keywords: Recommender System · Knowledge Graph · Deep Neural Networks.

1 Introduction

Recommendation systems are an essential part of internet media, personalised services and the advertising industry and have a significant impact on the popularity, turnover and profit of electronic businesses, which, for example, is evident in the case of Amazon.com, the earliest adopter of a recommendation engine [16]. Effective recommendation systems typically have three major parts: a comprehensive dataset such as the histories of user activities, an effective algorithm to intelligently process the information, and an accurate prediction engine to output intuitive and reasonable recommendations. A user's behaviour history is the fundamental knowledge source of a recommendation system that provides the raw data about the interactions between users and items. The recommendation algorithm is a comprehensive engine that preprocesses the original data, analysis the latent correspondences of the entities and produces those models with an appropriate data structure. The prediction engine is used to transform the machine-readable models into human-understandable outcomes.

The general idea to solve a recommendation problem is to find the correlations between users and items. If the user-item relationships are stored in a matrix, one possible

solution is to find a matrix decomposition [11, 14], with the general idea to extract the characteristics of users and items from their interactions. However, those methods have been shown to not work well with sparse matrices, and the dataset must be preprocessed to handle the missing relationships [11].

Deep Neural Networks (DNN) have been introduced as an alternative model to find latent correspondences. Prior research [18] has shown that DNNs work well on finding non-linear relationships between the input data and the output data. A Neural Collaborative Filtering (NCF) approach proposed by He et al. [9] is one of the state-of-art algorithms to solve the recommendation problem with the support of DNN. It combines general matrix factorisation and multi-layer perceptions in the DNN model and achieves significant improvement on the performance over traditional linear models. However, like the traditional matrix decomposition method, their algorithm heavily relies on the density of the dataset. There are also other content-based recommender systems based on Convolutional Neural Networks [20]. However, each of them only focuses on a single domain, such as processing videos, audios or images. In summary, there are two major shortcomings in existing algorithms. First, they require the dataset to be dense and comprehensive, i.e., each item must have a considerable number of mappings to the user set. Otherwise, the entire model may face an underfitting problem. Besides, they are unable to capture the semantic properties among the entities. Finally, there is a lack of assorted prediction algorithms for those models to offer the most relevant results.

In the context of recommender systems, an ontology could help us to discover the inherent properties and relationships among the items and enhance the performance of a recommender model with the auxiliary information from a knowledge graph in the specific field. Gruber [7] originally defined an ontology as an "explicit specification of a conceptualization". There are many models to formally define an ontology. In this paper, we focus on ontologies expressed in the Resource Description Framework (RDF). The Resource Description Framework Schema (RDFS) and the Web Ontology Language (OWL) are two data modelling languages to describe an RDF model.

In our model, we use the corresponding ontology of the item-set as the supplementary source to retrieve the connections among the item set. The ontology provides the semantic correlations of the items that are potentially helpful in the training and prediction procedures. For instance, in the use case described in this paper, a food ontology [5] is used that contains a class hierarchy of the ingredients that humankind uses in their recipes. In our algorithm, we design a DNN-based learning model that identifies the best-matched class, while we then perform a further prediction of the best match subclass, if it exists, based on the calculated statistical confidence level of all subclasses. The algorithm addresses the first problem we identified above, i.e., the sparsity of data since we can merge sparse items based on their taxonomical classifications known in the ontology. To address the second issue, i.e., the introduction of new items, our algorithm only needs to update the corresponding confidence level unless the new item is completely disjoint to all existing classes.

The remainder of the paper is structured as follows. In Section 2, we discuss related work of recommendation algorithms and analyse their limitations in certain use cases. In Section 3 we will introduce, describe and analyse our algorithm. In Section 4.2, we

evaluate our algorithm on a sparse dataset, i.e., a recipe dataset. We conclude the paper in Section 5 with a brief summary and potential improvements and future work.

2 Related Work

The traditional approach to deal with the recommendation problem is to use a singular value decomposition (SVD) to decompose the user-item matrix [15]. However, in the context of product recommendations where this was first introduced, a user can only buy a limited variety of items, and this results in a correlation matrix that is sparse and cannot be directly used in the algorithm. One possible approach is to fill in the matrix with some noise, but this significantly increases the complexity of computations and potentially distorts the dataset. There are many SVD-based machine learning recommender algorithms such as the latent factor models (LFM), which is a combination of SVD and machine learning[2]. It simply introduces a matrix of user features and another of item features. Then the model is trained by minimising the root-mean-square deviation between the ground truth and the product of the corresponding elements in two matrices. It is usually accompanied with an L-2 regularisation to avoid the overfitting problem. However, like traditional SVD, it assumes the user-item correlations are linear, which is not always true, depending on the use case.

The adaption of Deep Neural Network is another approach to tackle the limitations of a linear model. He et al. [9] propose a model that is a combination of matrix factorisation and multi-layer perceptions with the capability to learn non-linear underlying features. Zhang et al. [19] developed another approach using a DNN. It applies a quadratic polynomial regression when initialising the input layer of the multi-level perceptions instead of using a plain one-hot encoding of the user and item set. Such an approach considers both linear and non-linear features that potentially exist in the correlations. However, they only explore the explicit relationships that are present in the given user history. All of the correlations within users, within items and between users and items are learnt from the training process of the neural network. The performance of such models is significantly limited when the history dataset is too sparse, or if correlations exposed are inconsistent with the ground truth.

3 Methodology

In this paper, we focus on the hierarchical information, that is, the subclass hierarchy in the ontology. Our data flow is comprised of four major components, 1) raw data processing, 2) ontology matching, 3) deep neural network construction and 4) prediction. In the raw data processing stage, we extract the key information from the "user-item" interactions in plain text, which are ingredients in each recipe in the context of this paper. In the second step, we match the extracted "items" to the corresponding class(es) in the selected ontology. Additionally, we reconstruct a tree data structure to represent the matching, which is compatible with the deep neural network in the next stage. The deep neural network takes the input of the user-item interactions in terms of the corresponding class, as well as the hierarchical information in the reconstructed ontology tree, to train the parameters in the network. Then we are able to perform the predictions

for future use based on the ontology tree and the neural network model. In the following sections, we provide more details on each of the four above steps.

3.1 Raw Data Processing

The simplest datasets used for a recommendation system contains two sets, a user set U and an item set I. In addition, there are mappings M between a user and items such that $u \in U$ and $i \in I$. Usually, the potential items a user is interested in are recommended, so each user is treated as a container that includes a set of items that the user is interested in, such that for each user u, we have $E(u) = \{i | (u, i) \in M\}$.

Our model explores the intermediate correlations among the entities in the datasets using an appropriate domain ontology. In the user-item problem, it is hard to find the natural relationships (the correlation that is already known) of a user, but we can infer the connections among items from the ontology-based knowledge graph about the corresponding item sets. The recipe dataset we used in this paper is compatible to the generalised user-item model, i.e., we treat each recipe as a "user", and its corresponding ingredients are the "items".

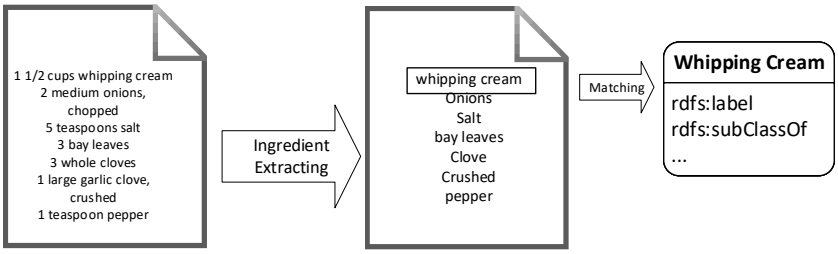


Fig. 1: Data processing and Ontology Matching

Specifically, we extracted the ingredients from the plain-text recipe based on the CRF Ingredient Phrase Tagger [6, 12], and then use the popular NLP library nltk [3] in combination with custom-rules to find the closest matching class. An example is shown in the first part of Figure 1, where the ingredient "whipping cream" is separated from the original recipe from its quantity and unit.

3.2 Ontology Matching and Reconstruction

The ontology used in our model is in the form of triples (s, p, o), where s refers to the subject, p refers to the predicate, and o refers to the object. At this stage we only consider predicates of type `rdfs:subClassOf` and `owl:equivalentClass`, where rdfs and owl denote the RDFS namespace and the OWL namespace, respectively. In this section, we discuss two stages, the ontology matching and the ontology reconstruction.

Ontology Matching Ideally we had access to a dataset that encodes recipes using an ontology, in this case, the FoodOn. However, since to the best of our knowledge, there is no dataset available that has recipes and their ingredients encoded using the FoodOn ontology [5], we have to map each ingredient from our chosen recipe dataset to instances in the FoodOn ontology. This ontology matching preprocess which matches the ingredients of each recipe in the form of plain text to the corresponding class in ontology, can be done with any existing state-of-the-art ontology matching tool such as those based on statistical analysis [13] or machine learning [4]. However, to achieve a close to 100% precision (trading off recall), we have developed custom rules for this specific task and our ontology, and the results from our empirical survey indicate a precision of about 85%.

The evaluation of the performance of the ontology matching algorithm we used in our paper was performed through an evaluation survey. We invited 6 people with proficient English skill, which is sufficient to distinguish the common English words and suitable for the vocabulary in our selected ontology and dataset. Each survey contains 200 randomly selected candidates from a pool of 7,059 matching entries processed by our ontology matching algorithm. In the survey, participants are offered a unique order of matchings, including the original textual data and the generated matching to the (label of) corresponding ontology. The evaluation of the matching accuracy is assessed on a 5-point Likert scale: Perfect Match, Partial Match, Neutral, Weak Match, and No Match.

According to the human evaluation of our ontology matching algorithm, when participants can only make a binary decision that it is a match or not a match, the accuracy is about 84.5%. While if they are provided with an instruction to make them in the 5-point Likert scale, the accuracy is 85.5% if we consider the matching marked as level 4 (Partial Match) and level 5 (Perfect Match) as a match. From the evaluation, we observed a relatively acceptable accuracy that will not generally become the bottleneck of the overall model performance. Additionally, an improvement of the matching accuracy may provide a better result for our predictive performance of our model.

Ontology Reconstruction The structural patterns in ontologies may differ from domain to domain; therefore we need a normalising algorithm to transform ontologies so that they share a standard structural pattern. The development of the algorithm is very difficult due to numerous unexpected structures. In this paper, we briefly introduce some of the techniques to normalise the ontology, to fit it to our neural network model in the following section.

We extract the triples from the original ontology file into a set of 3-tuples in the form of (s,p,o) as explained in Section 3.2. Then we restructure them based on the extracted triples by reducing the complexity of the ontology, such as removing classes with only one child, consolidate equivalent classes and reshape the graph into an ontology tree. Those procedures significantly reduce the depth of the matched ingredients and reduce the potential error in the training process. Specifically, it mostly solves the issue that two "similar" ingredients are matched to two equivalent classes or to a class that derives from other classes. Besides, it reduces the number of sparse classes in the ontology that are seldom matched. However, in some cases, we need to add extra classes to the existing ontology to ensure the compatibility and accuracy of our deep neural network

model. Here we discuss two examples of the most common cases. Figure 2 illustrates the case of classes containing multiple paths from the root (usually owl:Thing, but may be different for a specific domain). If we only allow selecting one path, its ancestors (i.e., super-classes) may not be accurate to the ground truth in the context of the corresponding recipe. Therefore we need to duplicate each path to make sure they have equal contribution to the learning process in our model. Figure 3 demonstrates the case that some classes have shallow hierarchical properties. When this happens, we duplicate the leaf class to all lower layers down to the original deepest leaf in the ontology.

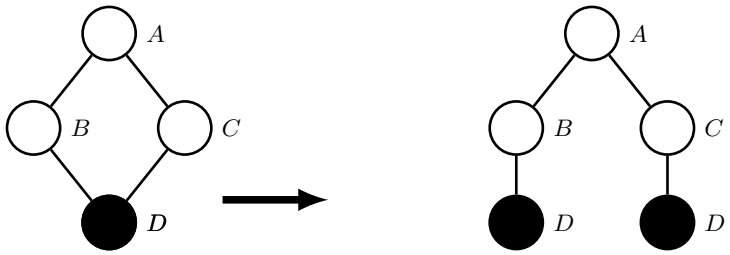


Fig. 2: Dealing with Multiple Paths. Multiple paths to a single class will be duplicated. We process this from the top to the bottom. This solves the selection problem about which path to include, if we only allow one path as an input to our DNN model.

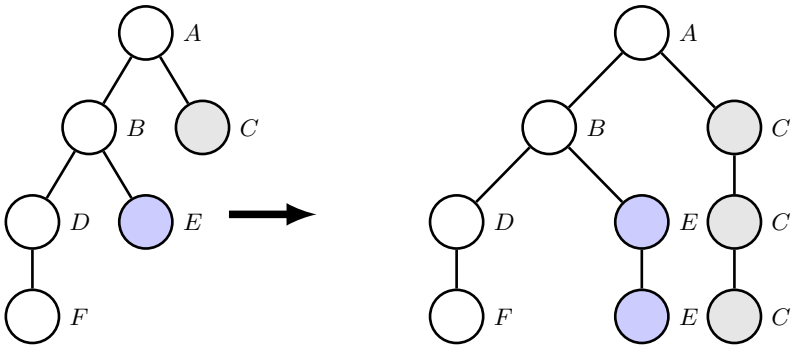


Fig. 3: Dealing with Shallow Class Hierarchy. Missing classes will be filled with duplicate leaf node(s) of that path. This solves the vector filling problem when there are insufficient nodes in a specific depth.

3.3 Ontology-based Neural Collaborative Filtering (ONCF)

To begin with, we introduce the Neural Collaborative Filtering (NCF) [9], which is a Deep Learning based algorithm that combines both the concepts of matrix factorisa-

tion and multi-layer perceptions. It mainly focuses on implicit feedback with the binary representation of the interaction between users and items. Our model is based on the traditional 4-layer neural collaborative filtering algorithm, which contains input layer, vectorisation layer, DNN layer and output layer. As an improvement, we integrate the hierarchical information into the traditional DNN model, and we mainly focus on the implicit feedback with the binary representation of the interaction between users and items. Here, the user and item are two abstract concepts. We consider a "user" as a container, and the corresponding items are its contents that belong to, or have a direct relationship to the container. For example, a recipe in this context is a "user", while its ingredients are the "items". Hence, the implicit interaction between the user i and the item j is denoted as:

$$y_{ui} = \begin{cases} 1 & recipe\ u\ contains\ ingredient\ i \\ 0 & otherwise \end{cases} \tag{1}$$

Our model contains four sublayers: Input layer, embedding layer, DNN layer and output layer

Input Layer The input layer is the vector representation of the user and item. It has three parts: the user vector U_i, and a group of the item vectors I_j.

The user sub-vector U_i is the one-hot encoding of each user. For every item j, its sub-vector I_j as we discussed in the previous section, is the concatenation of the sub-vectors of all its super classes in the ontology tree including itself. Such that:

$$I_j = \{i_{j,1}, i_{j,2}, \ldots, i_{j,d}\}, where\ i_{j,d} = onehot(S_{j,d}) \tag{2}$$

And $S_{j,d}$ is the superclass of item j in the ontology tree such that

$$S_(j,d) = \{s | j \in s, distance(j,s) = d\} \tag{3}$$

The input layer can then be formulated as a set of $d+1$ sub-vectors:

$$U_{input} = U_i \tag{4}$$

and

$$I_{input} = I_j = \{i_{j,1}, i_{j,2}, \ldots, i_{j,d}\} \tag{5}$$

where d is the overall depth of the subclass hierarchy.

Vectorisation Layer The vectorisation layer, or embedding layer, converts the vectors in the input layer to dense representations of the features of the user-item pair. The output vector of the original one-hot encoding has a dimension which is equivalent to the size of the item set. The transformation between the original encoding to a dense vector significantly reduces the memory and time in the training stage. In our model, we designed 2-level embedding layers. This layer transforms each one-hot vector in x_{input} into a corresponding dense vector such that the outputs are e_u for the user vector and e_i for the item vector.

DNN Layer and Output Layer This is the core part of our model. We first concatenate the outputs of the vectorisation layer, so that the input of the first DNN layer becomes:

$$x_1^{in} = concat(e_u, e_i) \tag{6}$$

The general form of each DNN layer is:

$$x_i^{out} = activation(W_i x_i^{in} + + b_1) \tag{7}$$

Here, W_n denotes the weight matrices, and b_n are the corresponding bias vectors. If we define $T_{n,n+1}(x_n) = activation(W_n x_n + b_n)$ to be the transition function from the $n - th$ to $n + 1$-th layer, we have:

$$x_{n+1}^{in} = x_n^{out} = T_{n,n+1}(x_n^{in}) \tag{8}$$

The output layer converts the output of the last DNN layer into a scalar score, which represents the correspondence between the specific pair of a user and an item in a numerical form, such that:

$$\hat{y_{ui}} = activation(Ex_n) \tag{9}$$

where E is the edge weight, and X_n is the last output of the DNN layer.

Cost Function In our case, the classes are in a hierarchy and we cannot assume them to be mutually exclusive. Therefore, we updated the cost function from the cross entropy to use "progressive target values" instead of binary values. For example, each class has the target value

$$y_{ui} = \frac{1}{\alpha^{d_i}} \tag{10}$$

where d_i is the distance to the matched class from the ontology matching algorithm as described in the ontology matching section, α is a parameter that represents the level to penalise inaccurate predictions (i.e. its super-classes instead of itself).

The likelihood function then becomes:

$$p(y, y^*, y^-) = \prod_{(u,i) \in y} \hat{y_{ui}} \prod_{(u,i) \in y^*} |(y_{ui} - \hat{y_{ui}})| \prod_{(u,i) \in y^-} (1 - \hat{y_{ui}}) \tag{11}$$

Where y^* denotes the super-classes of the matched class. This partially resolves the issues that if the algorithm matches to one of its super-classes, the evaluation module counts it as wrong. In cases where the model predicts one of the intermediate classes, we can do further predictions based on the confidence level derived from the statistics data of the training dataset. Specifically, for a partially positive item that does not exactly match the ground truth, but has some correspondence in terms of the number of common super-classes, we will assign a positive mark, instead of treating it as incorrect and give it a zero score. For example, consider the matched class is "Fuji Apple" with a full score value of 1, its super-class "Apple" has a score value of 0.37, and its further super-class "Fruit" has a score value of 0.14. To derive the objective function, we take

the negative logarithm, then we have

$$\begin{aligned} L(y, y^*, y^-) &= \log p(y, y^*, y^-) \\ &= -\sum_{(u,i)\in y} \log \hat{y_{ui}} - \sum_{(u,i)\in y^*} \log |(y_{ui} - \hat{y_{ui}})| - \sum_{(u,i)\in y^-} \log (1 - \hat{y_{ui}}) \end{aligned} \tag{12}$$

If we constrain $y_{ui} \in (0, 1)$, the objective function thus becomes

$$\begin{aligned} & L(y, y^*, y^-) \\ &= -\sum_{(u,i)\in y\cup y^*\cup y^-} y_{ui} \log (1 - |y_{ui} - \hat{y_{ui}}|) + (1 - y_{ui}) \log (1 - |y_{ui} - \hat{y_{ui}}|) \\ &= -\sum_{(u,i)\in y\cup y^*\cup y^-} \log (1 - |y_{ui} - \hat{y_{ui}}|) \end{aligned} \tag{13}$$

4 Prediction and Evaluation

In this section, we discuss the evaluation results of our model. In addition, we also introduce an innovative prediction algorithm that takes advantages of the class hierarchy information. We start this section from the idea of the prediction mechanism, followed by the evaluation for both the accuracy of our model and the general performance of the prediction algorithm.

4.1 Prediction

From the previous section, we are able to calculate the scores for a given user and item from the model we generated by our OCNF algorithm. The calculation takes advantage of the hierarchical structure of the ontology. We know that the cost function adjusts the loss according to the distance between the node in the path and the matched class. If the model is trained well, the scores for a given user and the intermediate "item" are progressively increasing along the path from the root to the correct predication in the reconstructed ontology tree. Formally, in the tree-structured ontology which captures the subclass hierarchy, we define the nodes along the path from the root to the best result as n_k, where k refers to the distance to the root, we have:

$$y(n_k) < y(n_{k+1}), k \in [0, d-1] \tag{14}$$

where d is the distance between the node of the expected prediction and the root. In this case, we can recursively choose the children with the highest score until we find the local maximum. Thus, we have the initial prediction Algorithm 0.

However, the accuracy of the model may be affected by a variety of factors such as the quality of the ontology matching process (as discussed above), and the completeness of the datasets. Such errors may result in the score of the optimal prediction to fluctuate, which in turn deteriorates the performance of our algorithm. Thus, we adjust the basic Algorithm 0 as follows:

Algorithm 1 Improved Prediction

Input: root r, shift s, tolerance k
$children = root.getChildren()$
$highest\_score = 0$
$curr\_node =$ r
$scores =$[]
$expanded =$ []
repeat
  N = children.pop().getChildren()
  for $n \in N$ **do**
    score[n] = getScore(n)
  end for
  max_score = $\max(score)$
  $d \leftarrow$ distance from $curr\_node$ to its deepest descendant
  $threshold_d = max\_score_d - (\frac{s}{\alpha_i^d})$
  selected = $\{n | n \in N, getScore(n) > threshold_d\}$
  for $child \in selected$ **do**
    expanded.append(child)
    children.append(child)
  end for
until !$children.isEmpty()$
sorted = expanded.sort(key = lambda x : x.getScore())
Return: $sorted[1:k]$

1. Following Algorithm 0 above to find the "best" prediction with the corresponding score (in each layer).
2. Apply an error shift to the maximum score in each layer such that

$$threshold_d = max\_score_d - (\frac{s}{\alpha_i^d}) \tag{15}$$

   Where d_i is the current distance from the leaf with the maximum score (as predicted from Algorithm 1 above, instead of the global maximum). And s is the shift parameter which represents the sensitivity of the tolerance to the potential error.
3. Expand the tree with all scores larger than or equal to this threshold score in each layer. Then perform step 1 in each of the layers.
4. Rank them in terms of the score and find the best K predictions.

The improved algorithm 1 adapts the general behaviour of the score value for relevant and irrelevant classes but is able to allow tolerances to potential training errors, which are inevitable, by expanding more nodes instead of just exploring the one with the highest rank from the scoring engine that takes the recipe and ontology ID as input and outputs the corresponding score from the OCNF model described above. In fact, in the ideal case where Algorithm 0 is applicable, the overall prediction complexity will be in $O(d)$, where d is the depth of the ontology, while we can usually achieve $O(\log d)$ for most of the ontologies. With the redundancy check, the complexity increases to $O(K^d)$, but it is still much smaller than a brute-force algorithm in the consideration

that we usually select a small K, and the depth d will be limited during the ontology reconstruction process.

Algorithm 1 resolves the accuracy problems with an acceptable decrease of the performance by a branch extension, however, the accuracy is still limited by the sparsity of the ontology. If the reconstructed hierarchical tree is too sparse, the algorithm may waste a considerable time on the unnecessary branches, and we cannot ignore the risk that the predictor gets trapped in a local optimum. To deal with this issue, we also consider two adjustments. First, we prune the unnecessary branches such that the total matched descendant is less than K. We set $K = 2$ to eliminate the branches with 0 or 1 matched descendant as they lack deterministic information, but it can be adjusted to a higher value for more complicated and more sparse datasets. Besides, we adapt the idea of decision trees to amend the order of the nodes and layers that the branches with the highest information gain to the top, to reduce the uncertainty of the prediction by selecting the most correct super-class(es) at the beginning of the algorithm. This approach enhances the speed, as well as further reducing the risk for the predictor to be trapped in a local optimum, or goes to the wrong branch.

4.2 Evaluation

In this section, we discuss the evaluations of our OCNF model, along with the analyses of the general performance of the prediction algorithm provided in section 4.1.

Datasets and Ontology In our evaluations, we focus on two major datasets, "ChoseMyPlate" [1] from USDA, and MovieLens 25M dataset [8]. For each of the datasets, we need at least one base ontology, which contains the relative hierarchical or connectivity information among all entities (ingredients for the USDA and movies for the MovieLens dataset).

The first dataset we use is the "ChoseMyPlate" recipe dataset from the U.S. Department of Agriculture, which contains about 6,000 unique recipes. Each recipe contains a set of ingredients used to prepare the meal. The motivation is to adapt the inherent properties and relationships among the ingredients from the ontology and try to make a recommendation of the most relevant ingredient for the given recipe. The corresponding ontology we selected is the "foodon" food ontology [5] which contains about 9,000 individual food ingredients. We also validate our model over the ontology created within Wikidata [17]. For Wikidata, we considered the sub-ontology whose root ancestor is 'food ingredient' (i.e. `http://www.wikidata.org/entity/Q25403900`) or 'food' (i.e. `http://www.wikidata.org/entity/Q2095`), or both. We selected about 4,000 food ingredients in the tailored Wikidata ontology we used in this paper.

The second dataset is the MovieLens 25M dataset with about 2,500,000 ratings for 62,000 movies by 162,000 users. The corresponding ontology is also tailored from the Wikidata dump with extractions about entities which are related to movies. We selected major properties including but not limited to the genre, country, year (grouped in 5) up to 4 Star actors, directors and the distributing company(s). We reconstructed the graph based ontology to a tree model in terms of the divisibility, which is evaluated according

to the information gain. The properties with higher divisibility will be placed at the top. For the MovieLens dataset,

Evaluation Settings The model evaluation assesses the basic performance of the model in comparison with the original neural collaborative filtering algorithm. We adopt the popular top-K evaluation with the leave-one-out mechanism [9]. The test data contains one positive sample, that is, one ingredient belongs to the specific recipe, while 99 negative samples are not in the recipe. Since our algorithm adapts the hierarchical properties of the ingredients in the ontology, we changed the policy for generating negative samples to ensure they are also not in the same path (not an ancestor) of the selected positive ingredient. Finally, we rank the scores for the samples and validate if the positive ingredient is in the samples of the top-K highest scores.

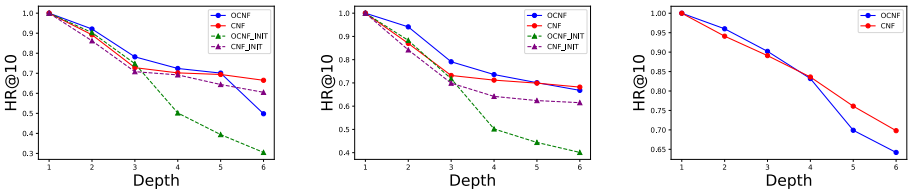
Baseline According to the motivation of our model, we will evaluate the model performance in comparison to the original neural collaborative filtering [9]. We calculated the Hit Ratio (HR) and Normalized Discounted Cumulative Gain (NDCG) [10] with the error-tolerance parameter set to $K = 10$. The Hit Ratio metric intuitively measures if the ground truth falls into the top K rankings according to the score from the output layer of the neural network model. The Normalized Discounted Cumulative Gain considers the ranking information with a higher value for the upper ranking of the ground truth. The value is higher if the ground truth is at the top 1 rather than at a middle or bottom rank. The baseline we used in our evaluations are the results from the CNF model, which are marked in red lines.

Performance Analyses The first set of figures illustrates the comparison of the HR and NDCG between the original neural collaborative filtering model and our OCNF model. The first experiment evaluates the *plain performance* that is based on the exact accuracy without considering the semantic relevance. For example, for the ground truth "Fuji Apple", a "Fruit" will not be considered as a hit. For HR, the valid hit must be the *exact match* from the prediction output to the "ground truth". In addition, when calculating the NDCG, the relevance score is binary, in which 1 denotes a hit and 0 otherwise.

Figure 4 and Figure 5 illustrates the comparison of the hit rates and NDCG of K = 10 with respect to the depth of layers for the initial settings that use the non-reconstructed ontology (dashed lines) and re-constructed ontology (solid lines). In the first case, if a class contains multiple paths, a random one will be chosen, and if a class does not contain the maximum possible hierarchical information, we use a dummy vector of all zeros to represent the missing class vectors. For the MovieLens dataset, we do not have such comparison because the tailored ontology is not originally in a hierarchical structure, and we reconstructed the ontology tree based on our customised measurement mentioned in previous sections. The baseline indicates the corresponding performance of the original CNF algorithm without any information supported by the ontology. In the plots, the x-axis represents the number of iterations used for training the model, and the y-axis is the corresponding performance in our test dataset. From the plot, it is noteworthy that the accuracy drops in the bottom layers. In our selected datasets, our model works well in distinguishing the classes with shallower hierarchical levels.

However, when dealing with the bottom layer, the exact accuracy drops below the base-line. The solid lines indicate the HR and NDCG performance when K = 10 with the ontology reconstruction. We can also clearly notice the performance improvement of the ontology reconstruction in the USDA dataset, which proves the effectiveness of the reconstruction mechanisms as described in Section 3.2.

By observing the comparison between our OCNF model and the baseline for the MovieLens dataset, we notice the similar behaviours: the hit rate of our model is also slightly better at the first 4 layers, but also drops for lower layers. This issue may be caused by the quality of the hierarchical modelled ontology, which is created by converting the graph based knowledge graph. The result may get better for a natively hierarchical based knowledge domains.

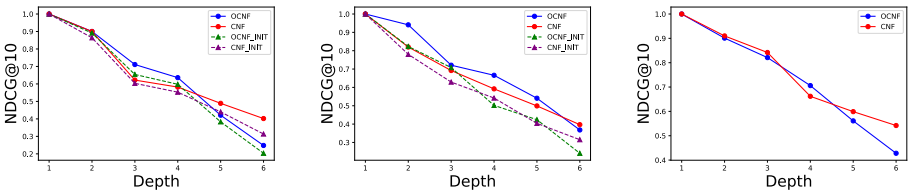


(a) HR@10 USDA-foodon (b) HR@10 USDA-WikiData (c) HR@10 MovieLens

Fig. 4: HR@10 with 3 dataset settings

There are two observations to explain the behaviour about the accuracy drop in the USDA dataset. First, for the USDA dataset, most of the ingredients contain limited hierarchical information, and those ingredients that are part of a shallow class hierarchy do not contribute to the model during the training process. Additionally, our model is structurally more complex so as to integrate ontological information. The enlarged complexity leads to additional problems such as over-fitting. Additionally, in the USDA-Recipe dataset, classes with the depth of 4-8 are much denser than top and bottom classes, which leads to a trade-off between the better result focusing on the upper classes and the averaged overall behaviour. We observed a better result for the Wikidata ontology, which indicates the performance may be highly influenced by the overall structure of the ontology. It also leads to potential future work to "normalise" an ontology that minimises the overall effect caused by the irregular structure of the ontology, such as sporadic deep branches, instead of its content. Additionally, we leave the solution to the potential over-fitting problem as a part of our future work to dynamically adjust the complexity of the model based on the distributions of the dataset and the corresponding ontology. At the current stage, our algorithm successfully predicts the upper-classes, and one of our hypothesis is that for an ontology with a shallow hierarchical structure, we are able to achieve a better result than the base-line, even at the bottom layers. The performance of our model is comparable to the traditional non-ontology based algorithm in general, but we have the benefit to get better prediction accuracy for the classes

or entities in the higher level. In other words, our result is better if we are more interested in the generalisation of an item, instead of the item itself.



(a) NDCG@10 USDA-foodon (b) NDCG@10 USDA-WikiData (c) NDCG@10 MovieLens

Fig. 5: HR@10 with 3 dataset settings

Prediction Evaluation In addition to the model evaluation, we also need to assess the corresponding prediction algorithm. Our prediction algorithm adopts the hierarchical relationship of the food ingredients. However, the original CNF model does not have a corresponding prediction algorithm. Instead of directly comparing the performance, we designed two evaluations based on a "random-selecting algorithm" that randomly selects a number of ingredients from the solution space, as well as the complete search based on a "brute-force" algorithm that calculates the score of all the potential ingredient in the ontology. The evaluation contains two major parts. The first test is based on the original Algorithm 0 without any shifts and adjustments of the threshold. The second test is based on the improved Algorithm 1. The third test is based on the reordered tree according to the information gain in each level. We will compare the results with different values of k and s for the Alg0 and Alg1. The dataset used for the prediction evaluation is slightly different from what we used for the model evaluation. We strip out the unused hierarchical information in the vector representations for the ingredient when searching a specific layer (zero out all deeper layers), and use the prediction algorithm mentioned in the technical detail section to generate the top N = 10 relevant ingredients according to the rank of the scores. Again, if the ground truth lies in the top N predictions, we consider it as a hit. We stop expanding the nodes when the remaining children consists less than 10% of the original entities, and check if the item (or its superclass if the expansion stops) contains in the top N predictions. All tests are performed in single thread with the Intel(R) Core(TM) i9-9880H CPU.

Table 1 shows the evaluation results of the prediction algorithm, the three numbers represents the USDA-Foodon, USDA-Wikidata and MovieLens dataset. The parameters were set to balance the speed and accuracy by limiting the total expansions to 10% of the overall entities. Among all combinations of k and s, we are able to achieve a hit rate of about 50%, while the speed is about 10 times faster than the brute force algorithm. If we require more accuracy, we could increase the value of k, and achieve a hit rate around 70% with a speed of about 5 times faster than the brute-force algorithm. The

Performance Evaluation				
	Brute-Force	Alg0	Alg1	Alg-DT
Accuracy	0.905/0.898/0.847	0.347/0.408/0.302	0.568/0/609/0.566	0.582/0.626/0.594
Time Consumption (sec)	10.54/6.82/825.01	1.25/0.71/100.26	1.36/0.78/91.24	1.22/0.68/86.75

Table 1: Prediction Evaluations

precision is already a huge improvement over the random guess, and the speed is much faster than the brute-force algorithm. The actual deployment of the prediction engine contains a trade-off between the accuracy and the speed.

With the adaption of the decision tree reordering mechanism, we can achieve a further improved performance, as indicated in Table. However, this structural and expanding order amendment works differently in different datasets. We noticed that it grants huge improvements for the USDA datasets with both the foodon and WikiData ontology, whose entity sparsity is higher, but it does not provide a noticeable difference for the MovieLens dataset with dense entities included.

5 Conclusion and Future work

In this paper, we introduced an improvement to the traditional DNN based model for recommender systems that integrates the hierarchical information of an ontology, along with a template compatible algorithm that helps the system to provide actual predictions. Comparing to the traditional model that only considers the dataset without any external information related to the knowledge domain, our model can be used on sparse data and with new data added without reconstructing the entire model. Additionally, our model behaves well when focusing on the recommendation results of the categorical classes instead of the exact item. However, the performance of our model is tightly bound to the data quality and the structure of the corresponding ontology. Results show that our model is successful for predicting shallow layers but still requires more adjustments to give an exactly accurate prediction.

In future works, we will formalise the ontology normalisation algorithm and the fast-prediction algorithm. In addition, we want to generalise our model to consider other relations than only the hierarchical structure.

References

1. Choosemyplate. `https://www.choosemyplate.gov/`, accessed: 2019-06-01
2. Agarwal, D., Chen, B.C.: Regression-based latent factor models. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 19–28. ACM, New York, NY, USA (2009)
3. Bird, S., Klein, E., Loper, E.: Natural Language Processing with Python. O'Reilly Media, Inc., 1st edn. (2009)

4. Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Ontology Matching: A Machine Learning Approach, pp. 385–403. Springer Berlin Heidelberg (2004)
5. Dooley, D.M., Griffiths, E.J., Gosal, G.S., Buttigieg, P.L., Hoehndorf, R., Lange, M.C., Schriml, L.M., Brinkman, F.S.L., Hsiao, W.W.L.: Foodon: a harmonized food ontology to increase global food traceability, quality control and data integration. npj Science of Food **2**(1), 23 (2018)
6. Erica Greene, A.M.: CRF ingredient phrase tagger. `https://github.com/nytimes/ingredient-phrase-tagger` (2016), accessed: 2019-06-01
7. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition **5**(2), 199–220 (1993)
8. Harper, F.M., Konstan, J.A.: The movielens datasets: History and context. ACM Trans. Interact. Intell. Syst. **5**(4) (Dec 2015). https://doi.org/10.1145/2827872, `https://doi.org/10.1145/2827872`
9. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., Chua, T.S.: Neural collaborative filtering. In: Proceedings of the 26th International Conference on World Wide Web. pp. 173–182 (2017)
10. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of ir techniques. ACM Trans. Inf. Syst. **20**(4), 422–446 (Oct 2002)
11. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (Aug 2009)
12. Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the Eighteenth International Conference on Machine Learning. pp. 282–289. ICML '01, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
13. Ochieng, P., Kyanda, S.: A statistically-based ontology matching tool. Distributed and Parallel Databases **36**(1), 195–217 (2018)
14. Salakhutdinov, R., Mnih, A.: Probabilistic matrix factorization. In: Proceedings of the 20th International Conference on Neural Information Processing Systems. pp. 1257–1264. NIPS'07, USA (2007)
15. Sarwar, B.M., Karypis, G., Konstan, J.A., Riedl, J.: Application of dimensionality reduction in recommender system - a case study (2000)
16. Smith, B., Linden, G.: Two decades of recommender systems at amazon.com. IEEE Internet Computing **21**(3), 12–18 (May 2017)
17. Vrandeĉić, D., Krötzsch, M.: Wikidata: A free collaborative knowledgebase. Communications of the ACM **57**(10), 78–85 (2014)
18. Wang, H., Wang, N., Yeung, D.Y.: Collaborative deep learning for recommender systems. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1235–1244. KDD '15, ACM, New York, NY, USA (2015)
19. Zhang, L., Luo, T., Zhang, F., Wu, Y.: A recommendation model based on deep neural network. IEEE Access **6**, 9454–9463 (2018)
20. Zhang, S., Yao, L., Sun, A., Tay, Y.: Deep learning based recommender system: A survey and new perspectives. ACM Comput. Surv. **52**(1), 5:1–5:38 (Feb 2019)