EFFECTS OF NONLINEAR FUNCTIONS ON KNOWLEDGE GRAPH CONVOLUTIONAL NETWORKS FOR RECOMMENDER SYSTEMS WITH YELP KNOWLEDGE GRAPH

Xing Wei and Jiangjiang Liu

Department of Computer Science, Lamar University, Beaumont, USA

ABSTRACT

Knowledge Graph (KG) related recommendation method is advanced in dealing with cold start problems and sparse data. Knowledge Graph Convolutional Network (KGCN) is an end-to-end framework that has been proved to have the ability to capture latent item-entity features by mining their associated attributes on the KG. In KGCN, aggregator plays a key role for extracting information from the high-order structure. In this work, we proposed Knowledge Graph Processor (KGP) for pre-processing data and building corresponding knowledge graphs. A knowledge graph for the Yelp Open dataset was constructed with KGP. In addition, we investigated the impacts of various aggregators with three nonlinear functions on KGCN with Yelp Open dataset KG.

KEYWORDS

Recommender Systems, Knowledge Graph, Activation Function.

1. INTRODUCTION

In 2019, the number of internet users reached 7.71 billion [1], and 2.5 quintillion bytes of data was being created every day [2]. It becomes more and more challenging for people and companies to cope with such dramatic data explosion. For example, Netflix, the online streaming-service provider, offers more than 10 thousand movies and TV shows from which users can choose. The traditional search method only displays the sorted list of items relating to the search key word and cannot provide specific items in different users' interests, so the users may not find the items they really want. The heavy information overload has become a major problem for both consumers and providers of the online content industry.

To better deliver content to users, one of the practical strategies is personalization. As a successful solution, the recommender systems have been playing a vital and indispensable role in Web applications, ranging from search engines and E-commerce to social media sites and online streaming services. Almost every online content provider has applied a recommender system.

The traditional recommendation methods, such as Collaborative Filtering (CF) and content-based recommendation, have achieved good performance on rating data. However, despite CF's effectiveness and universality, its ability on modeling side information, such as item attributes and user profiles [3], suffers from the cold start problem, which happens when items added to the catalogue have either none or very little interactions and consequently process sparse data where

David C. Wyld et al. (Eds): CCSIT, SIPP, PDCTA, AISC, NLPCL, BIGML, NCWMC - 2021
pp. 185-199, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.110715

users and items have few interactions. As a common solution for those problems, model-based methods are designed to transform user ID, item ID, and the side information into a generic feature vector that compensates for the sparse data and improves the recommendation performance, such as matrix factorization [4], factorization machine (FM) [5], and Wide & Deep [6].

However, these methods only view each interaction between entities as an independent data instance rather than linked data with relations. This makes them insufficient to distill attribute-based collaborative signals from the collective behaviors of users. As we can see in Figure 1, there is an interaction between User John and Movie m2, which is directed by Director D1, and Director D1 directed Movie m2 and Movie m4. CF methods can only determine the similarity of users who also watched Movie m2, such as User David and User Paul. Model-based methods find the similar items Movie m21 and Movie m2 by the same attributes of Actor A1 and Director D1.As we can see, based on these two types of information, not only recommendation can be generated, but also a high-order relationship can be found, which connects user and item with one or multiple linked attributes.

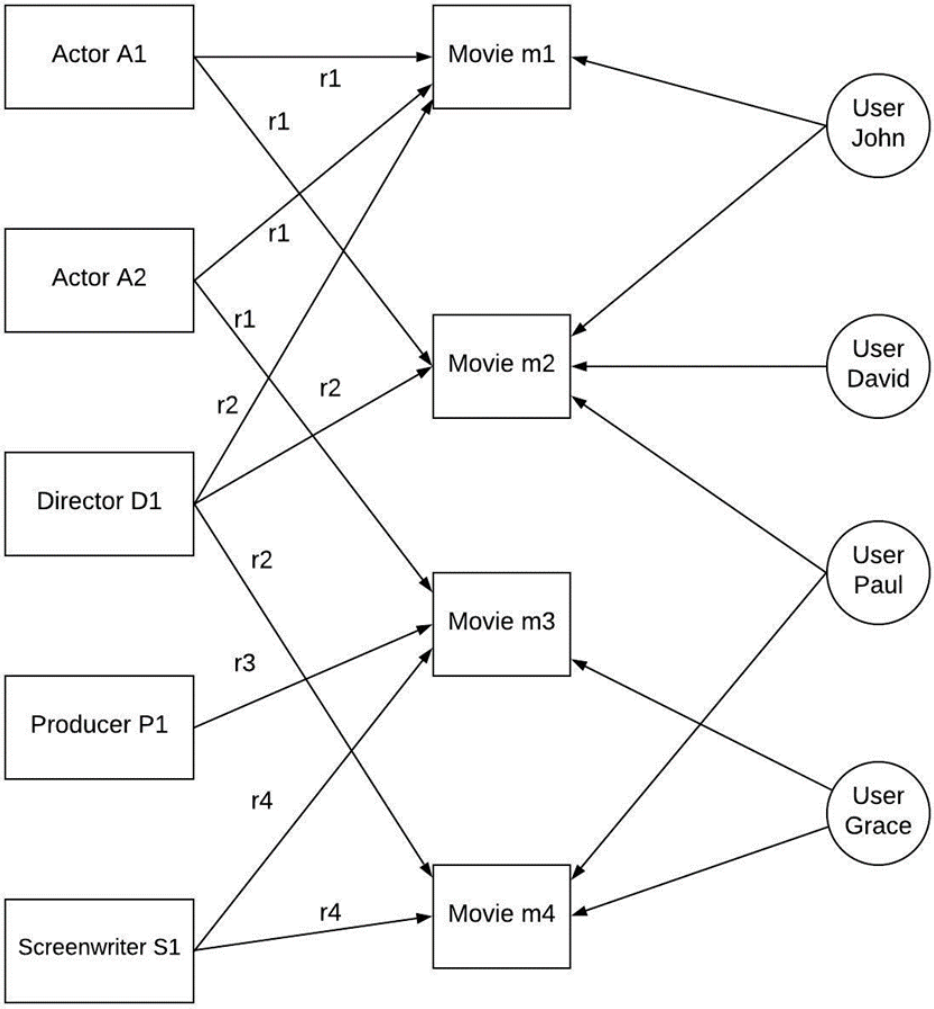

Figure 1. A movie example of knowledge graph. Relations r1: actor of, r2: director of, r3: producer of, r4: screenwriter of.

Therefore, to fulfill the shortage of traditional recommendation methods, graphsare a solution to collaborating with side information. Knowledge graphs (KGs) are composed of structured information of the real world, which links attributes themselves as well as with users and items. Typically, a knowledge graph is a directed heterogeneous graph in which each edge is represented as a triple (head entity, relation, tail entity), indicating that two entities are connected by a specific relation, e.g., (George Lucas, film, director, Star Wars). In general, because we can process the information from the high-order structure of the knowledge graph, by comparing with the traditional model method, the knowledge graph related method is advanced in dealing with cold start problems and sparse data.

Several recent efforts have applied knowledge graphs on the recommendation system. For example, knowledge graph embedding methods, which transfer entities and relations to low-dimensional representation vectors [21], and graph algorithm-based methods, which exploit the latent information from users, items, and the relations in between them by treating knowledge graph as a high-order structure information network [22]. KGCN is an end-to-end framework that captures latent item-entity features by mining their associated attributes from the high-order relationships on the KG. Additionally, an aggregator plays an important role in the KGCN. In the field of graph-related neural networks, aggregators widely operate with nonlinear functions such as ReLU [7], Leaky ReLU [8], and Tanh [9].

1.1. Our Contributions

To build a knowledge graph from scratch often requires tremendous time and effort, which is an obstacle for most researchers studying knowledge graphs. The process of adopting knowledge graphs for business analytics to support decision making is even more challenging and time consuming.

To tackle this challenge, we propose an efficient tool, Knowledge Graph Processor (KGP), with a user-friendly interface that can easily transfer the raw dataset to a knowledge graph format dataset. We built a Yelp knowledge graph from a Yelp Dataset by using KGP.

We also conducted analysis on aggregators used in the Knowledge Graph Convolutional Network, with several widely adopted nonlinear functions and achieved significant improvement, compared to the original KGCN with ReLU aggregators.

We released the code of KGP and Yelp Knowledge Graph datasets (knowledge graphs). The source code and the dataset are available at https://github.com/XingWeiLamar/KGP.

2. RELATED WORK

2.1. Knowledge Graph

A knowledge graph is a multi-relational graph constructed by entities (nodes) and relations (different types of edges). Each edge represents a triple of the form (head entity, relation, tail entity). Recently, the knowledge graph (KG) has been rapidly applied to various applications.

Large-scale knowledge graphs for academic and commercial purposes, such as NELL, DBpedia, Google Knowledge Graph, and Microsoft Satori, have become the core data structure of many practical applications from named entity disambiguation [20] and information extraction [17] to search engines [18] and question/answer systems [10].

Traditional recommendation techniques, such as collaborative filtering, usually represent customer-items interaction as an N-dimensional vector, then model their interaction by specific techniques, such as inner product or neural networks. However, CF methods usually suffer from limited performance when user-item interactions are very sparse, and they perform poorly when processing new products and users. To address those limitations, a common paradigm is to turn the user-item interaction into a more feature-based scenario, where attributes of users and items are transformed into the model as vectors to remedy the sparsity and perform better in cold start scenarios. [6]

2.2. Recent Knowledge Graph Related Recommender System

The successful applications of knowledge graphs in a wide variety of tasks have inspired researchers to study KG on improving the performance of recommendation systems. In comparison with knowledge-free methods, applying knowledge graphs on recommender system gains advantages in three ways: 1) discovering latent connections among items of knowledge graphs by the semantic relatedness among items; 2) exploring users' interests to increase the diversity of the recommendation from the varied types of relations; and 3) improving the exam inability of recommender systems due to the connections of user's historically-liked and recommended items in the knowledge graph.

There are three types of knowledge-aware recommender systems, based on current research. First, embedding-based methods [19] pre-process a knowledge graph with knowledge graph embedding algorithms then process user entity embedding into recommendation. Embedding-based methods can easily utilize KGs to fulfill the needs of recommender systems. However, the knowledge graph embedding algorithms are designed to model rigorous semantic relatedness [13], which can perform much better on graph applications (e.g., link prediction) rather than recommendation systems. Furthermore, embedding-based methods do not usually do end-to-end way training. Second, path-based methods [11] provide instruction for recommendation from discovered patterns of connections among entities in a KG. It is barely applied in practical terms from a cost and effectiveness perspective because meta-paths/meta-graphs need to be manually designed. Third, hybrid methods [12] combine the former two methods and learn user-item embeddings by extracting the structure of knowledge graphs.

2.3. Nonlinear Functions

For Neural Networks, the purpose of the nonlinear activation function is to introduce non-linearity into the output. A neural network without a nonlinear activation function is essentially a linear regression model. The activation function does the non-linear transformation to the input-making so it is capable of learning and performing more complex tasks.

The most common nonlinear activation function is ReLU, which is defined as: $\sigma(x) = \max\{0, x\}$. Hence, whenever x is negative, the function returns 0. When X is positive, it returns x. Note that this function is not differentiable at 0, and hence the back propagation will fail at this point. Nonetheless, in general, because of the precision issues of floating-point numbers in computers, this situation barely appear in reality, and ReLU as defined above works well in practice.

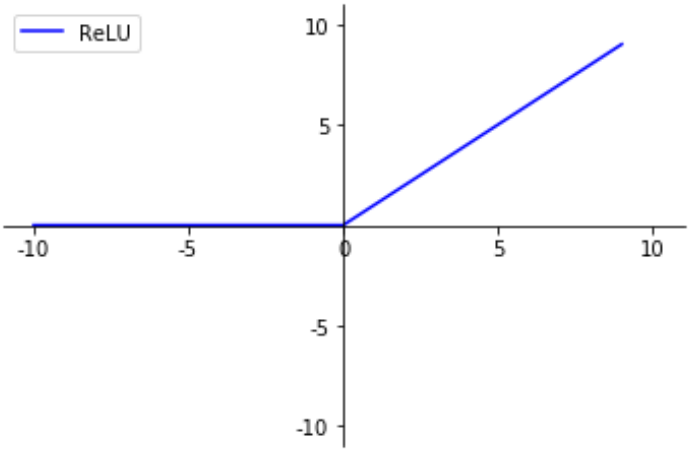

Figure 2. ReLU

One of the variations of ReLU is Leaky-ReLU [1], which is given by: $\sigma(x) = x$ if $x > 0$ and $a \cdot x$ otherwise, in which a is an adjustable constant. Instead of defining the ReLU function as 0 for negative values of x, it is defined as an extremely small linear component of x. By making this small modification, the gradient of the left side of the graph comes out to be a non-zero value. Hence, we would no longer encounter dead neurons in that region.

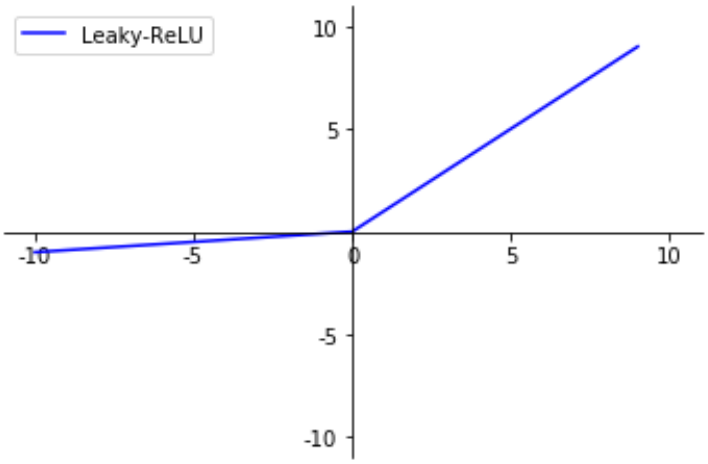

Figure 3. Leaky-ReLU

Exponential Linear function is also a special case of ReLU. The function is given by: $\sigma(x) = x$ for $x > 0$ and $a \cdot (e^{x} - 1)$ for $x<0$, where a is a hyper-parameter to be learned from the data. Unlike the Leaky-ReLU and parametric ReLU functions, instead of a straight line, ELU uses a log curve for defining the negative values.

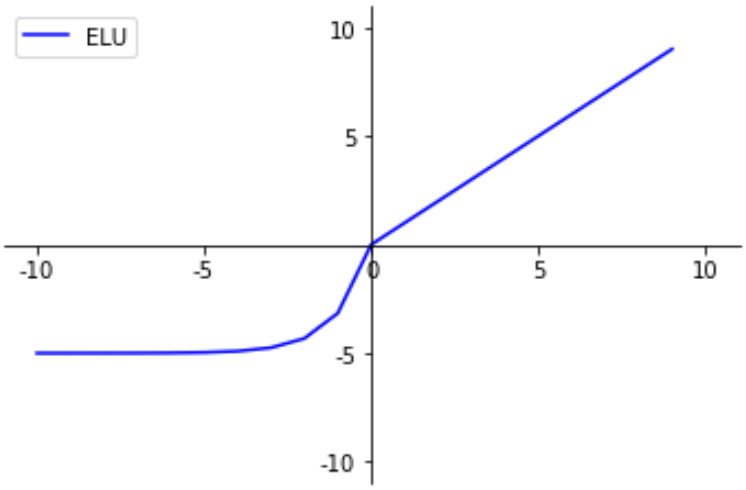

Figure 4. ELU

3. PROPOSED WORK

3.1. Knowledge Graph Processor

Data analysts often spend most of their time on data pre-processing. Data scientists mostly spend 80 percent of their time on finding, cleansing, and organizing data. [16] In the experiment, we utilized the Yelp dataset as a benchmark, which is publicly accessible and varies in terms of domain, size, and sparsity.

Therefore, to conquer this problem, we developed an easy-to-use tool, Knowledge Graph Processor, KGP, which is able to pre-process the raw dataset efficiently and effectively. The workflow chart of KGP is shown in Figure 5. First, KGP detects the format of the dataset. If it is stored in JSON format, KGP will transfer the JSON file to CSV format. Next, KGP performs a data cleaning function to remove the duplicated values and null value in the dataset. Then, users will input the relation's name and assign the head and tail of the knowledge graph triplet by selecting the column number in the dataset, so KGP can extract the selected data from those columns and construct them with relation named as a triplet. After users finish the triplets building, KGP extracts those triplets and reconstructs them into a CSV file and assigns an index value for each item. Finally, after users input a user-item interaction file, which usually is the file stores rating information, the KGP transforms the rating file from explicit feedback to implicit feedback. (explanation is in section 3.2)

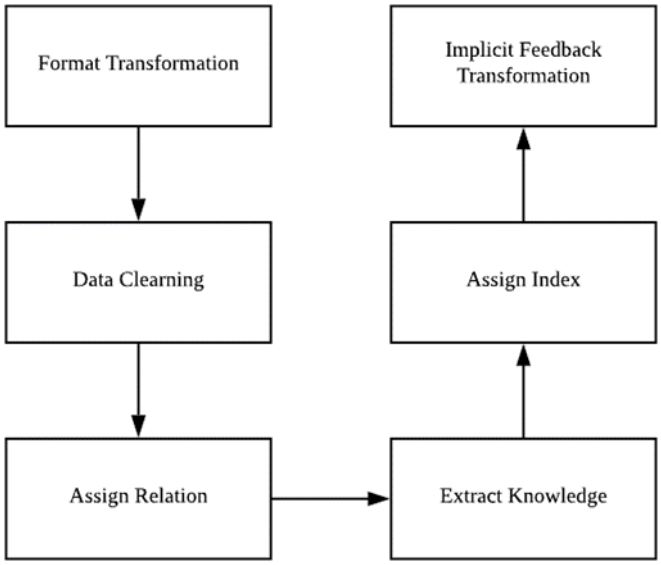

Figure 5. Working Flow of KGP

3.2. Yelp Knowledge Graph with KGP

By using KGP, we constructed a Yelp knowledge graph from Yelp open dataset[1] which contains over 6 million reviews, nearly 20 thousand businesses. The local businesses such as restaurants, bars, spas and barber shops are viewed as items.

Beside the item-to-item interactions, we extracted item knowledge from the local business information network (e.g., categories, locations, and attributes) as knowledge graph data. To ensure the knowledge graph quality, we pre-processed the three knowledge graph parts by filtering out entities with frequency lower than 10, and retaining the relations appearing in at least 50 triplets. The basic statistics of the datasets are presented in Table 1.

To reduce the noise deviation and test the performance of the model in a different sparsity, we use 10-core, 20-core, and 30-core settings to ensure that each item has at least 10 interactions, 20 interactions, and 30 interactions in the knowledge graph.

The original rating dataset is explicit feedback, in which the rated stars are from 0-5, with 5 is the best and 0 as the worst. From the rating stars, we can directly understand how much the user likes or dislikes the business. However, the practical data behaviors are implicit feedback, which are more complicated and important. We transform the explicit feedback into implicit feedback where the entry will be marked with 1, indicating that the user rated the item positively, with the positive rating threshold as 4, and sample an unwatched set marked as 0 for each user.

3.3. Problem Formulation

In this study, there is a set of *M*of users $U = \{u_1, u_2..., u_m\}$ and a set of *N* of items $I = \{i_1, i_2, ...,i_n\}$. The user-item interaction matrix $P \in R^{m \times n}$ defined as users' implicit feedback, which $y_{ui}=1$indicates that user *u* engages with item *v*, such as, browsing and clicking, if not $y_{ui}=0$. Moreover, the knowledge graph G is defined as entity-relation-entity triples *(h,r,t)*. Here $h \in E$,

[1]https://www.yelp.com/dataset

$r \in R$, and t∈E infer the head, relation, and tail. *E* and *R* are the set of entities and relations in the knowledge graph. For example, the triple (Star Wars: The Rise of Skywalker, movie; director; J.J. Abrams) indicates the fact that J.J. Abrams is the director of the movie *Star Wars: The Rise of Skywalker*. In our scenarios, an item $i \in I$ correspond to one entity $e \in E$. For instance, in movie recommendations, the knowledge also contains the item "Star Wars: The Rise of Skywalker" as an entity.

The KGCN model is to predict whether user *u* has potential interest in new item *i* by given the user-item matrix *P* and the knowledge graph *G*. We have a prediction function $f_{ui}=F(u,v|\Theta,Y,G)$, where P_{ui} denotes the probability that user *u* will engage with item *i*, and Θ denotes the model parameters of function *F*.

3.4. Analysis of Three Aggregators for Knowledge Graph Convolutional Networks

Knowledge Graph Convolutional Networks [14] is an end-to-end framework that explores users' preferences on knowledge graph for recommender systems. The Architecture KGCN is shown as Figure 6, where the first KGCN layer captures the high-order structural proximity by aggregating each entity's representation and its neighborhood representation into a single vector. Then the Learning Layer will take the H-order entity representation fed into a function $R^d x R^d \rightarrow R$ to predict probability.

First, consider a candidate pair of user *u* and item(entity) *v*. *N(v)* is denoted as the set of entities directly connected to *v*, and $r_{ei,ej}$ is denoted as the relation between entity e_i and e_j. The function: $R^d x R^d \rightarrow R$ is used to calculate the score between a user and a relation:

$$\pi_v^u = g(u,r) \qquad (1)$$

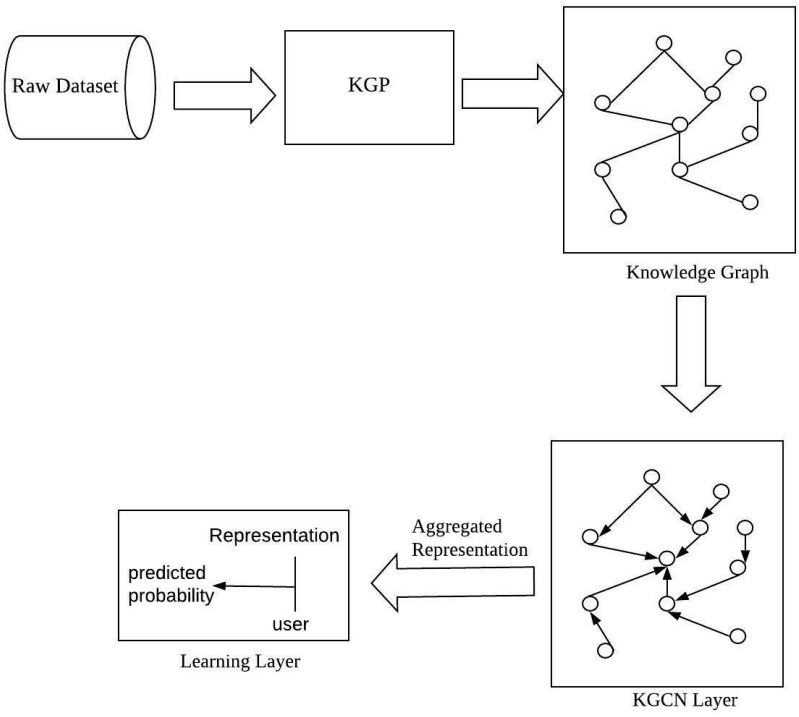

Figure 6. KGP+KGCN Architecture

In the formula, $u \in R^d$ and $r \in R^d$ are the representations of user u and relation r, and d is the dimension of representations. The function is to compute the importance between relation r and user u. For instance, a user might have more interests in restaurants in specific categories and another user may be more concerned about whether or not the restaurant is kid-friendly.

The below linear function combines v's neighborhood to capture the topological proximity structure of item v.

$$V^u_{N(v)} = \sum_{e \in N(v)} \tilde{\pi}^u_{r_{v,e}} e' \qquad (2)$$

$\tilde{\pi}^u_{r_{v,e}}$ is the normalized user-relation score:

$$\tilde{\pi}^{u}_{r_{v,e}} = \frac{\exp(\pi^{u}_{r_{v,e}})}{\sum_{e \in N(v)} \exp(\pi^{u}_{r_{v,c}})} \qquad (3)$$

The e denotes the representation of entity e. User-relation scores perform as personalized filters in the formula 2 because we aggregate the neighbors with bias, with respect to these user-relation scores.

Next, KGCN layer aggregates the entity representation v and its neighborhood representation $V^u_{S(v)}$ into a single vector by aggregators. ReLU is the default aggregator used in KGCN.

Aggregators perform a significant role in the KGCN, and different non-linear functions can affect the output of the KGCN. In this study, we analyze the effectiveness of three aggregators with the following different nonlinear function.

$$\mathrm{agg}_{\mathrm{Sum}} = \sigma(w \cdot (v + v^{u}_{s(v)}) + b) \qquad (4)$$

$$\mathrm{agg}_{\mathrm{concat}} = \sigma(w \cdot \mathrm{concat}(v, v^{u}_{s(v)}) + b) \qquad (5)$$

$$\mathrm{agg}_{\mathrm{neighbor}} = \sigma(w \cdot v^{u}_{s(v)} + b) \qquad (6)$$

where σ denotes non-linear functions, w and b are transformation weight and bias.

The KGCN layer extends 1-order entity representation to h-order entity representation by iterating the KGCN layer multiple times, so the algorithm can explore the user's interests more comprehensively. To achieve that, the KGCN layer first propagates the initial representation of each entity to its neighbors and aggregates the vectors to receive the 1-order representations. Then it repeats the procedure and aggregates the 1-order representation and its neighbors to obtain the 2-order representations. The combination of an entity and its neighbors up to h hops away is the h-order representation of an entity.

The KGCN learning algorithm is as the following. The algorithm receives a pair of users and items then calculates the receptive field M of v layer by layer. Next the aggregation repeats H times: in iteration h, it calculates the neighborhood representation of each entity $e \in M[h]$ then aggregates it with its own representations $e^u[h-1]$ to achieve the representation to be used in the next iteration. Last, the final H-order entity representation vu will be pass to the prediction function f: $R^d \times R^d \rightarrow R$ with user representation u.

$$\hat{y}_{uv} = f(u, v^u) \quad (7)$$

4. EXPERIMENTS

4.1. Datasets

Yelp open dataset contains over 6 million reviews, nearly 20 thousand businesses.

MovieLens-20M is a widely used benchmark dataset for movie recommendation, which contains over 20 million explicit ratings (rating from 1-5). The knowledge graph of MovieLens-20M is pre-constructed by Wang with Microsoft Satori [14].

Table 1. Statistics of the datasets

	10-Core	20-Core	30-Core	MovieLens-20M
Number of Users	44837	44784	16295	138159
Number of Items	71822	26613	2550	16954
Number of interactions	876829	320872	24611	13501622
Number of Entities	70534	25509	1822	102569
Number of Relations	34	34	33	32
Number of KG Triples	1268941	627552	60463	499474

4.2. Data Pre-processing

We built a Yelp Knowledge Graph from a Yelp open dataset. To reduce the noise deviation and test the performance of the model in different sparsity, we use 10-core, 20-core, and 30-core settings to ensure that each item has at least 10 interactions, 20 interactions, and 30 interactions in the knowledge graph. Please see the statistics of each datasets in Table 1.

4.3. Nonlinear Function Setting

In the experiment, to achieve the best performance, we set a as 0.2 for the Leaky ReLU. For the ELU a is 1.

4.4. Baseline

We tested the dataset with 3 baselines, in which the first baseline is a well-known KG-free baseline, the second one is a KG-aware method, and the third one is the KGCN with ReLU function.

LibFM [7] is a software implementation for factorization machines that features stochastic gradient descent (SGD) and alternating least squares (ALS) optimization as well as Bayesian inference using Markov Chain Monte Carlo (MCMC).

RippleNet [15] is a memory-network-like approach propagating user preferences over the entities in KG and iteratively extending a user's potential interest along links in the KG to provide recommendation.

4.5. Experiments Setup

In KGCN, the function g and f are set as inner product, σ is non-linear function for non-last-layer aggregator, and tanh for last-layer aggregator. We applied hyper-parameters as follows:

Each hyper-parameter is determined by the best Accuracy (AUC) on the corresponding dataset. For each dataset, we randomly select 80% of rating interaction history as the training set and 20% as the evaluation set, as well as a test set, respectively. Each experiment is repeated three times, and the average outcome is reported. The performance of each method is evaluated by two evaluation criteria: AUC and F1, which are applied to evaluate the click through rate (CTR) prediction. The test set was applied in training model to predict each interaction.

For the baseline, we use the Python vision of LibFm, Pylibfm to test the dataset. The number of factors is 10, the number of training epochs is 50, and the initial learning rate is 1×10-4. For the RippleNet, dimension of embeddings is 8, number of hops is 2, learning rate is 0.02, l2 weight is 1×10-7, batch size is 1024. For the KGCN ReLU λ is 3×10-8, μ is 0.003, batch size is 1024, H is 8, d is 64, S is 8.

Table 2. Basic Hyper Parameter Setting for KGCN

	10-Core	**20-Core**	**30-Core**	**MovieLens-20M**
Dimension of embeddings(d)	**128**	**64**	**128**	**32**
Neighbor Sampling size(S)	**8**	**8**	**8**	**4**
Depth of receptive field(H)	**2**	**2**	**2**	**2**
L2 Regularize Weight(λ)	2×10^{-9}	2×10^{-9}	3×10^{-5}	2×10^{-7}
Learning Rate(μ)	3×10^{-3}	3×10^{-3}	6×10^{-4}	2×10^{-2}
Batch Size	**1024**	**1024**	**1024**	**65536**

5. RESULT

The results of click through rate are presented in Table 3. We have the following observations:

Among all the models, the KGCN Leaky ReLU-sum achieves the best performance on the average of the three datasets.

In general, comparing with KG free model LibFM, we find that the improvements of KGCN Leaky ReLU-sum on 10-core and 20-core are 15% and 13% higher than 30-core. This demonstrates that KGCN Leaky ReLU-sum can well addres0.80.s sparse scenarios, since the 10-core and 20-core datasets are sparser than 30-core.

RippleNet shows strong performance compared with the KG-free baseline, because RippleNet also uses a multi-hop neighborhood structure, which also captures the proximity information from the KG.

Table 3. The results of AUC and F1 in CTR prediction

	10-Core		20-Core		30-Core		MovieLens-20M	
	AUC	F1	AUC	F1	AUC	F1	AUC	F1
PyLibFm	0.715	0.674	0.723	0.671	0.702	0.655	0.955	0.903
RippleNet	0.903	0.863	0.897	0.855	0.721	0.639	0.968	0.912
KGCN-Leak-ReLU-Sum	0.914	0.854	0.896	0.835	0.791	0.674	0.979	0.934
KGCN-Leak-ReLU-Concat	0.913	0.850	0.903	0.841	0.768	0.662	0.978	0.933
KGCN-Leak-ReLU-Neighbor	0.838	0.774	0.814	0.753	0.484	0.595	0.977	0.932
KGCN-ELU-Sum	0.906	0.842	0.895	0.833	0.754	0.629	0.977	0.931
KGCN-ELU-Concat	0.902	0.840	0.903	0.841	0.708	0.725	0.978	0.932
KGCN-ELU-Neighbor	0.835	0.755	0.814	0.753	0.665	0.582	0.976	0.932
KGCN-ReLU-Sum	0.912	0.851	0.891	0.833	0.765	0.650	0.978	0.932
KGCN-ReLU-Concat	0.912	0.852	0.905	0.842	0.773	0.666	0.977	0.931
KGCN-ReLU-Neighbor	0.838	0.779	0.814	0.751	0.454	0.495	0.977	0.932
KGCN-Leak-ReLU Average	0.888	0.826	0.871	0.810	0.681	0.644	0.978	0.933
KGCN-ELU Average	0.881	0.813	0.871	0.809	0.709	0.645	0.977	0.932
KGCN-ReLU Average	0.887	0.827	0.870	0.809	0.664	0.604	0.977	0.932

Among the three original KGCN aggregators, KGCN Sum and KGCN Concat perform significantly better than KGCN-Neighbour, and KGCN Neighbour shows a clear gap on 30-Core setting dataset; that may be because the KGCN Neighbour only aggregated the neighborhood's representation and does not include the information from the entity itself.

The last three lines of Table 3 represent the average performance of each non-linear function implementing on the three aggregators. We can see that Leaky ReLU function obtains the best results for all three datasets, which may be because Leaky ReLU overcomes the dead ReLU problem. When the representation values are negative, the Leaky ReLU returns a small fraction;in contrast, the ReLU always return 0. ELU function performs the worst among the three functions, which may be because for ELU, $\sigma(x) = x$ when $x > 0$ and $a \cdot (e^{x} - 1)$ when $x < 0$, KGCN is lack of ability to learn and update the value of a when the input value is negative.

We conducted experiments on how hyper parameter influences the performance of a KGCN-Leaky ReLU-Sum model. In Table 4, we can see the influence of the neighbor sampling size from 2 to 64. The model performs best when S=8. This is because a too small S does not have enough capacity to incorporate neighborhood information, while a too large S is prone to be misled by noises.

Table 4. AUC Result of Leaky ReLU-Sum KGCN with Different Neighbour Sampling Size S

S	2	4	8	16	32	64
10-Core	0.886	0.895	0.913	0.910	0.912	0.912
20-Core	0.872	0.886	0.903	0.903	0.899	0.892
30-Core	0.770	0.764	0.769	0.757	0.792	0.789

We can see the influence of depth of receptive field H on the model by setting H from 1 to 4 in Table 5, which illustrates that the model reacts sensitively on the variation of H.

The model performs best when H=2 and collapse dramatically when H is greater than 2, which may be because a larger H brings massive noises to the model and the too-long relation-chain makes little sense when inferring inter-item similarities.

Table 5. AUC Result of Leaky ReLU-Sum KGCN with Different Depth of Receptive Field H

H	**1**	**2**	**3**	**4**
10-Core	**0.878**	**0.904**	**0.504**	**0.505**
20-Core	**0.875**	**0.897**	**0.507**	**0.507**
30-Core	**0.785**	**0.770**	**0.382**	**0.385**

Table 6 displays the effects of dimension of embedding d on performance of the model. The result is intuitive: performance improves dramatically as the d increases; when d is 128, the model achieves the best performance, since the larger d can include more information of users and entities. When d is greater than 128, the model is drawn back by overfitting.

Table 6. AUC result of KGCN with different dimension of embedding D

D	**4**	**8**	**16**	**32**	**64**	**128**	**256**
10-Core	**0.820**	**0.858**	**0.893**	**0.900**	**0.903**	**0.914**	**0.903**
20-Core	**0.824**	**0.827**	**0.886**	**0.892**	**0.890**	**0.894**	**0.898**
30-Core	**0.780**	**0.817**	**0.806**	**0.753**	**0.751**	**0.785**	**0.773**

6. CONCLUSION AND FUTURE WORKS

In this paper, we proposed an efficient and effective data preprocessing and knowledge graph generation tool, Knowledge Graph Processor (KGP). By using the KGP, we constructed a knowledge graph for a Yelp dataset. Our proposed KGP can process JSON, CSV, and text files. More features could be supported, such as automatically extracting information from well-build knowledge graph database.

By testing KGCN on the Yelp knowledge graph and MovieLens-20M dataset with Leaky ReLU, ELU, and ReLU non-linear functions, Leaky ReLU is able to improve performance of the original KGCN for recommendation systems. The reason is because Leaky ReLU has an advantage on overcoming dead ReLU problem as well as its robust performance when the input values are negative.

We point out two avenues for future work: 1) The knowledge graph's content and quality can significantly affect the performance of KGCN. An interesting direction of future research is to quantify the quality of the knowledge graph dataset; 2) We investigated the influence of the nonlinear function in the first layer of the aggregator. Future work could explore the impact of the different nonlinear functions on the second layer, and the impact of the optimizer is also a valuable direction to study.

REFERENCES

[1] Jeannie Dougherty. (2019). Internet growth + usage stats 2019: Time online, devices, users. https://www.clickz.com/internet-growth-usage-stats-2019-time-online-devices-users/235102/

[2] Charlotte Johnson. (2019) How much Data is Produced every Day 2019? https://www.the-next-tech.com/blockchain-technology/how-much-data-is-produced-every-day-2019/

[3] Xiang Wang, Xiangnan He, Fuli Feng, Liqiang Nie, and Tat-Seng Chua. (2018). "TEM: Tree-enhanced Embedding Model for Explainable Recommendation", WWW, ppl 1543–1552

[4] Yehuda Koren, Robert Bell and Chris Volinsky. (2009, August). Matrix Factorization Techniques for Recommender Systems. IEEE 2009

[5] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. (2011, July). Fast context-aware recommendations with factorization machines. SIGIR'11.

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu. (2016). "Wide & Deep Learning for Recommender Systems", DLRS@RecSys. ppl 7–10.

[7] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. (2017)." Non-Local Neural Networks", arXiv preprint arXiv:1711.07971, vol. 10.

[8] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. (2018). "Graph attention networks", The 6th International Conferences on Learning Representations

[9] Yujia Li, Daniel Tarlow, Marc Brockschmidt, Richard Zemel. (2016). "Gated Graph Sequence Neural Networks", arXiv: Learning, 2016.

[10] Xiao Huang, Jingyuan Zhang, Dingcheng Li, Ping Li. (2019). "Knowledge Graph Embedding Based Question Answering", WSDM '19: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining (January), ppl 105–113. https://doi.org/10.1145/3289600.3290956

[11] Binbin Hu, Chuan Shi, Wayne Xin Zhao, and Philip S Yu. (2018). "Leveraging Metapath Based Context for Top-N Recommendation With A Neural Co-Attention Model", the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, ppl 1531–1540

[12] Zhu Sun, Jie Yang, Jie Zhang, Alessandro Bozzon, Long-Kai Huang, and Chi Xu. (2018). "Recurrent knowledge graph embedding for effective recommendation", the 12th ACM Conference on Recommender Systems. ACM, 297–305

[13] Joan Bruna, Wojciech Zaremba, Arthur Szlam, Yann LeCun. (2014). "Spectral Networks and Locally Connected Networks on Graphs", arXiv:1312.6203v3

[14] Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo, (2019, May). "Knowledge Graph Convolutional Networks for Recommender Systems", WWW. ACM ISBN 978-1-4503-6674-8/19/05

[15] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo, (2018). "RippleNet: Propagating User Preferences on the Knowledge Graph for Recommender Systems", CIKM. ppl 417–426

[16] Crowd Flower. (2016). 2016 Data science report, https://visit.figure-eight.com/rs/416-ZBE142/images/ CrowdFlower_DataScienceReport_2016.pdf

[17] Amit Singhal. (2012). "Introducing The Knowledge Graph: Things, Not Strings," https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html

[18] Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S. Weld. (2011). "Knowledge-based weak supervision for information extraction of overlapping relations," Proc. 49th Annu. Meeting Assoc. Comput. Linguistics, ppl 541–550

[19] Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y Chang. (2018). "Improving Sequential Recommendation with Knowledge-Enhanced Memory Networks", the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. ACM, 505–514

[20] Sherzod Hakimov, Sherzod Hakimov, Salih Atilay Oto and Erdogan Dogdu (2012). "Named entity disambiguation using linked data," Proc. 9th Extended Semantic Web Conf.

[21] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. (2014). "Personalized entity recommendation: A heterogeneous information network approach", the 7th ACM International Conference on Web Search and Data Mining. ACM, ppl 283–292.

[22] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. "Metagraph based recommendation fusion over heterogeneous information networks", the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, ppl 635–644.

AUTHORS

Xing Wei is a graduate students in the department of Computer Science at Lamar University. His research interests focus on data analytics for recommender systems.

Jiangjiang (Jane) Liu is a professor in the department of Computer Science at Lamar University. Her research interests are on data analytics, cloud computing, and high performance computing.