Personalized Recommendations using Knowledge Graphs: A Probabilistic Logic Programming Approach

Rose Catherine
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA, USA
rosecatherinek@cs.cmu.edu

William Cohen
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA, USA
wcohen@cs.cmu.edu

ABSTRACT

Improving the performance of recommender systems using knowledge graphs is an important task. There have been many hybrid systems proposed in the past that use a mix of content-based and collaborative filtering techniques to boost the performance. More recently, some work has focused on recommendations that use external knowledge graphs (KGs) to supplement content-based recommendation.

In this paper, we investigate three methods for making KG based recommendations using a general-purpose probabilistic logic system called ProPPR. The simplest of the models, `EntitySim`, uses only the links of the graph. We then extend the model to `TypeSim` that also uses the types of the entities to boost its generalization capabilities. Next, we develop a graph based latent factor model, `GraphLF`, which combines the strengths of latent factorization with graphs. We compare our approaches to a recently proposed state-of-the-art graph recommendation method on two large datasets, Yelp and MovieLens-100K. The experiments illustrate that our approaches can give large performance improvements. Additionally, we demonstrate that knowledge graphs give maximum advantage when the dataset is sparse, and gradually become redundant as more training data becomes available, and hence are most useful in cold-start settings.

Keywords

Knowledge Graph based Recommendations; Probabilistic Logic Programming; Graph based Matrix Factorization

1. INTRODUCTION

Recommendation is usually social or content-based, with social methods best for problems with many users and relatively few items (e.g., movie recommendation for Netflix) and content-based best on cold start or "long tail" settings. An important research problem is how to leverage external knowledge for improving content-based recommendations. Content features are usually available for both users and items. For users, these are typically their demographics. For items like movies, these may include the actors, genre, directors, country of release, etc. and for items like restaurants, these may include the location, cuisine, formal vs. casual, etc. Although many methods have been proposed in the past that use content, not many use the interconnections between the content itself and external knowledge sources, which we refer to as a knowledge graph (KG).

However, in recent work [27], the authors propose a metapath based method to infer users' preferences to items that have not been explicitly rated by them, and showed state-of-the-art results on two large real-world datasets. In this paper, we show how KG recommendations can be performed using a general-purpose probabilistic logic system called ProPPR[26]. We formulate the problem as a probabilistic inference and learning task, and present three approaches for making recommendations. Our formulations are not identical to that proposed in [27], but are similar in spirit. We show that a number of engineering choices, such as the choice of specific metapaths and length of metapaths, can be eliminated in our formalism, and that the formalism allows one to easily explore variants of the metapath approach. Our best methods outperform the latter.

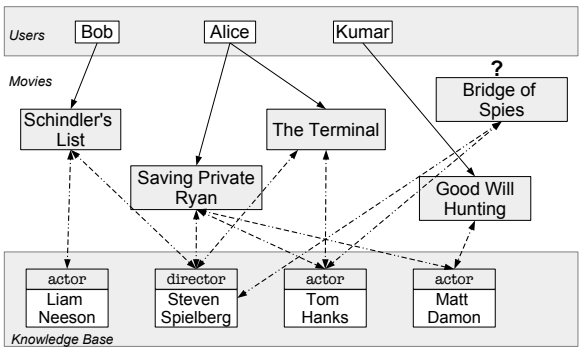

Figure 1: Example of Movie Recommendation

A typical movie recommendation example is depicted in Figure 1 where users watch and/or rate movies, and content about the movies are available in a database. The links in the figure show the content associated with each of the movies as well as the movies that each of the users watched. `EntitySim`, the simplest of the methods proposed in this paper, learns users' preferences over the content, and leverages the link structure of the KG to make predictions. In some cases, it is also possible to obtain a *type* information

RecSys '16, September 15-19, 2016, Boston , MA, USA

DOI: http://dx.doi.org/10.1145/2959100.2959131

about the content. For example, `Tom Hanks` is of type `Actor`. `TypeSim`, the second method, builds on `EntitySim` but additionally models the type popularities and type similarities. It is closest to the method of [27] in that it uses the type information of the nodes to prioritize certain paths in the graph. `GraphLF`, the most complex of the models proposed in this paper, is a latent factorization model over the knowledge graph, and is type agnostic i.e. it does not required typed KGs. By comparing our methods to the published results of the state-of-the-art method that uses KGs in generating recommendations [27], we show that our methods are able to achieve a large improvement in performance.

We also study the behavior of the methods with changing dataset densities and show that at higher densities, just the graph link structure suffices to make accurate recommendations and the type information becomes redundant. Additionally, our experiments show that at much larger densities, the knowledge graph itself becomes redundant and simple methods can perform as well or better. However, in sparse datasets with few training examples, the knowledge graph is a valuable source of information.

The rest of the paper is organized as follows: Section 2 discusses the prior relevant work. The approaches proposed in this paper are detailed in Section 3 followed by experiments and a discussion of results in Section 4. In Section 5 we conclude and give future directions of research.

2. RELATED WORK

Recommendation systems have been popular for a long time now and are a well researched topic. However, there has not been much effort directed at using external KGs for improving recommendations.

A recent method [27], called `HeteRec_p`, proposed the use of KGs for improving the recommender performance. Since this method is the present state-of-the-art against which we compare our approaches, we detail it in Section 2.1. Another link-based method [28] proposed by the same authors precedes [27], and learns a global model of recommendation based on the KG, but does not attempt to personalize the recommendations.

Another recent effort at using multiple sources of information is the HyPER system [16], where the authors show how to recommend using a Probabilistic Soft Logic framework [2]. They formulate rules to simulate collaborative filtering (CF) style recommendation. For instance, in the case of user-based CF, the rule is of the form, $SimilarUsers(u_1, u_2) \wedge Rating(u_1, i) \Rightarrow Rating(u_2, i)$, where $SimilarUsers$ indicate if the users u_1 and u_2 are similar using a k-nearest neighbor algorithm, computed offline using different similarity measures like Cosine, Pearson etc. If the friend-network of users is available, then they leverage it using the rule $Friends(u_1, u_2) \wedge Rating(u_1, i) \Rightarrow Rating(u_2, i)$. If other rating prediction algorithms like Matrix Factorization (MF) are available, then they induce an ensemble recommender using the rules $Rating_{MF}(u, i) \Rightarrow Rating(u, i)$ and $\neg Rating_{MF}(u, i) \Rightarrow \neg Rating(u, i)$. Eventually, during the training phase, they learn a weight per rule using the PSL framework, which is later used for predicting the ratings in the test set. Similar to HyPER is the approach proposed in [11] that uses Markov Logic Networks, and that proposed in [6] that uses Bayesian networks to create a hybrid recommender. Like these methods, we also base our methods on a general-purpose probabilistic reasoning system. However, we differ from these methods in our focus on using external knowledge in recommendations.

Prior research has previously proposed using various kinds of special purpose or domain–specific knowledge-graphs. In [12], the authors proposed to use a trust-network connecting the users especially for making recommendations to the cold-start users. The latter is matched up against the network to locate their most trusted neighbors, whose ratings are then used to generate the predictions. Another popularly used network is the social network of the users. Prior work like [14] and [9] among various other similar approaches use the social connection information of the users to find similar users or "friends" in this case, and use their ratings to generate recommendations for the former.

2.1 HeteRec_p

`HeteRec_p` [27] aims to find user's affinity to items that they have not rated using *metapaths*. Metapaths describe paths in a graph through which two items may be connected. In the example of Figure 1, an example metapath would be `User` $\rightarrow$ `Movie` $\rightarrow$ `Actor` $\rightarrow$ `Movie`. Given a graph/network schema $G_T = (A, R)$ of a graph G where A is the set of node types and R is the set of relations between the node types A, then, metapaths are described in the form of $P = A_0 \xrightarrow{R_1} A_1 \xrightarrow{R_2} A_2 \ldots \xrightarrow{R_k} A_k$ and represent a path in G_T, which can be interpreted as a new composite relation $R_1 R_2 \ldots R_k$ between node-type A_0 and A_k, where $A_i \in A$ and $R_i \in R$ for $i = 0, \ldots k$, $A_0 = dom(R_1) = dom(P)$, $A_k = range(R_k) = range(P)$ and $A_i = range(R_i) = dom(R_{i+1})$ for $i = 1, \ldots, k-1$. For the specific purpose of recommending on user-item graphs, `HeteRec_p` uses metapaths of the form $user \rightarrow item \rightarrow * \rightarrow item$. Given a metapath P, they use a variant of PathSim [24] to measure the similarity between user i and item j along paths of the type P, a method the authors refer to as *User Preference Diffusion*. For each P, they use the user preference diffusion to construct the diffused user-item matrix $\tilde{R}_P$. Let $\tilde{R}^{(1)}, \tilde{R}^{(2)}, \ldots \tilde{R}^{(L)}$ be the diffused matrices corresponding to L different metapaths. Each such $\tilde{R}^{(q)}$ is then approximated as $\hat{U}^{(q)} \cdot \hat{V}^{(q)}$ using a low-rank matrix approximation technique. Then, the global recommendation model is expressed as: $r(u_i, v_j) = \sum_{q \in L} \theta_q \ \hat{U_i}^{(q)} \cdot \hat{V_j}^{(q)}$ where, θ_q is a learned weight for the path q. To personalize recommendations, they first cluster the users according to their interests. Then, the recommendation function is defined as: $r^*(u_i, v_j) = \sum_{k \in C} sim(C_k, u_i) \sum_{q \in L} \theta_q^{(k)} \hat{U_i}^{(q)} \cdot \hat{V_j}^{(q)}$ where, C represents the user clusters and $sim(C_k, u_i)$ gives a similarity score between the k^{th} cluster center and user i. Note that the θ_q is now learned for each of the clusters. This formulation of the rating prediction function is similar to [5]. Although `HeteRec_p` performed well on the recommendation tasks, the algorithm needs several hyper-parameters that need to be determined or tuned, like choosing the specific L metapaths from a potentially infinite number of metapaths, and the number of clusters. It also requires a rich KB with types for entities and links.

3. PROPOSED METHOD

3.1 Preliminaries

In this paper, we use the term *entity* as a generic term to denote a word or a phrase that can be mapped onto a knowl-

edge base or an ontology. Since the knowledge bases used in this paper are based on structured data, the mapping is straightforward. However, when using a generic knowledge base like Wikipedia[1], Yago [23] or NELL [21], one might require a wikifier or an entity linker [17].

A Knowledge Graph (KG) is a graph constructed by representing each item, entity and user as nodes, and linking those nodes that interact with each other via edges. In [27], the authors emphasize that their method is tailored to Heterogenous Information Networks (HIN), which are essentially KGs but with type mapping functions for entities and links, where there is more than one type (heterogenous). Otherwise, the network becomes homogenous. This is in contrast to prior works that use graphs or networks of only one type of nodes, like say, a friend network. A KG, as referred to in this paper, is therefore a relaxed version of a HIN where the types of entities and links may or may not be known. We assume that the nodes are typically heterogenous even if their type information is missing. If the types are unknown, then only the two methods `EntitySim` and `GraphMF` are applicable. However, if the knowledge graph is indeed an HIN, then all three methods apply.

Similar to [27], we too use binary user feedback. i.e. the rating matrix entry R_{ij} for user i and item j is 1 if a review or another form of interaction like clicking or liking is available. Otherwise, R_{ij} is 0.

3.2 Running Example

For example, consider a movie recommendation task, similar to that of [27], where we are tracking three users `Bob`, `Alice` and `Kumar`. From historical records, we know that `Alice` has watched `Saving Private Ryan` and `The Terminal`, both of which have `Steven Spielberg` as the `Director` and `Tom Hanks` as an `Actor`, as specified by the knowledge base. The knowledge base may also provide additional content like plot keywords, language and country of release, awards won etc. Similarly, we also know the movies that were watched in the past by `Bob` and `Kumar`. In addition to watching, we could also include user's actions such as "reviewing" or "liking", if available. Given the past viewing history of users, we may want to know the likelihood of them watching a new movie, say `Bridge of Spies`. This scenario is graphically represented in Figure 1. Although in this particular case, the movie-entity graph is bipartite, it is also common to have links between movies themselves like say, `Finding Nemo` and `Finding Dory` where the latter is a sequel to the former, or between entities themselves like for example, `Tom Hanks` and `Best Actor Academy Award`.

3.3 Recommendation as Personalized PageRank

Similar to the Topic Sensitive PageRank proposed in [10] and the weighted association graph walks proposed in [4], imagine a random walker starting at the node `Alice` in the graph of Figure 1 (ignore the direction of the edges). At every step, the walker either moves to one of the neighbors of the current node with probability $1 - \alpha$ or jumps back to the start node (`Alice`) with probability α (the reset parameter). If repeated for a sufficient number of times, this process will eventually give an approximation of the steady-state probability of the walker being in each of the nodes. However, since we need only the ranking of the movies and not the other entities like actors and directors, we consider only those nodes corresponding to the movie nodes being tested, and sort them according to their probability to produce a ranked list of movie recommendations.

In the above method, there is no control over the walk. The final outcome of the walk is determined by the link structure and the start node alone. However, recent research has proposed methods to learn how to walk. Backstrom et. al in [3] showed how a random walker could be trained to walk on a graph. This is done by learning a weight vector w, which given a feature vector ϕ_{uv} for an edge in the graph $u \rightarrow v$, computes the edge strength as $f(w, \phi_{uv})$, a function of the weight and the feature vectors, that is used in the walk. During the training phase, learning w is posed as an optimization problem with the constraint that the PageRank computed for the positive example nodes is greater than that of the negative examples. In our case, the positive examples would be those movies that the user watched, and negative examples would be those that the user did not watch or give an explicit negative feedback.

3.4 Learning to Recommend using ProPPR

ProPPR [26], which stands for **Pro**gramming with **P**ersonalized **P**age**R**ank, is a first-order probabilistic logic system which accepts a *program* similar in structure and semantics to a logic program [18] and a set of *facts*, and outputs a ranked list of entities that *answers* the program with respect to the facts. ProPPR scores possible answers to a query based on a Personalized PageRank process in the proof graph (explained below) for the query. Below, we show how it can be used for the task of learning to recommend.

For the recommendation task, the first step is to find a set of entities that each user is interested in, from their past behaviors. We call this set the `seedset` of the user because it will later seed the random walk for that user. For this, we use the ProPPR program of Figure 2. The first rule states that the entity `E` belongs to the `seedset` of user `U` if `U` has reviewed `M` which is linked to entity `X` and `X` is related to `E`. Further, two entities are defined to be related if they are the same (Rule 2), or if there is a link between `X` and another entity `Z` which is related to `E` (Rule 3). This last rule is recursive. The `link` and the *type* (`isEntity`, `isItem` and `isUser`) information forms the knowledge graph in our case. Sample entries from the knowledge graph in the ProPPR format are shown in Figure 3. To find the seed set for `Alice`, we would issue the query $Q =$ `seedset(Alice, E)` to ProPPR.

$$\texttt{seedset}(\texttt{U},\texttt{E}) \leftarrow \texttt{reviewed}(\texttt{U},\texttt{M}), \texttt{link}(\texttt{M},\texttt{X}), \texttt{related}(\texttt{X},\texttt{E}), \texttt{isEntity}(\texttt{E}). \quad (1)$$

$$\texttt{related}(\texttt{X},\texttt{X}) \leftarrow \texttt{true}. \quad (2)$$

$$\texttt{related}(\texttt{X},\texttt{E}) \leftarrow \texttt{link}(\texttt{X},\texttt{Z}), \texttt{related}(\texttt{Z},\texttt{E}). \quad (3)$$

Figure 2: Seed Set generation

`link(Bridge of Spies, Tom Hanks)`	`isEntity(Tom Hanks)`
`link(Tom Hanks, Saving Private Ryan)`	`isEntity(Matt Damon)`
`link(Saving Private Ryan, Matt Damon)`	`isItem(Bridge of Spies)`

Figure 3: Example entries from the knowledge graph

ProPPR performs inference as a graph search. Given a program LP like that of Figure 2 and a query Q, ProPPR

[1]https://en.wikipedia.org

starts constructing a graph G, called the *proof graph*. Each node in G represents a list of conditions that are yet to be proved. The root vertex v_0 represents Q. Then, it recursively adds nodes and edges to G as follows: let u be a node of the form $[R_1, R_2, ..., R_k]$ where R_* are its predicates. If ProPPR can find a fact in the database that matches R_1, then the corresponding variables become bound and R_1 is removed from the list. Otherwise, ProPPR looks for a rule in LP of the form $S \leftarrow S_1, S_2, ..., S_l$, where S matches R_1. If it finds such a rule, it creates a new node with R_1 replaced with the body of S as, $v = [S_1, S_2, ..., S_l, R_2, ..., R_k]$, and links u and v. In the running example, v_0 is `seedset(Alice, E)` which is then linked to the node v_1 = `[reviewed(Alice,M), link(M,X), related(X,E), isEntity(E)]` obtained using Rule 1. Then, ProPPR would use the training (historical) data for `reviewed` to substitute `Saving Private Ryan` and `The Terminal` for `M` creating two nodes v_2 and v_3 as `[link(Saving Private Ryan,X), related(X,E), isEntity(E)]` and `[ link(The Terminal,X), related(X,E), isEntity(E)]` respectively. ProPPR would proceed by substituting for `X` from the knowledge graph and `related(X,E)` using the rules and so on until it reaches a node whose predicates have all been substituted. These nodes are the answer nodes because they represent a complete proof of the original query. The variable bindings used to arrive at these nodes can be used to answer the query. Examples would be:

$$\texttt{seedSet}(\texttt{Alice}, \texttt{E} = \texttt{TomHanks})$$
$$\texttt{seedSet}(\texttt{Alice}, \texttt{E} = \texttt{StevenSpielberg})$$

Note that such a graph construction could be potentially infinite. Therefore, ProPPR uses an approximate grounding mechanism to construct an approximate graph in time $O(\frac{1}{\alpha\epsilon})$, where ϵ is the approximation error and α is the reset parameter. Once such a graph has been constructed, ProPPR runs a Personalized PageRank algorithm with the start node as v_0 and ranks the answer nodes according to their PageRank scores.

The output of the program of Figure 2 is a ranked list of entities for the user `U` and the first K of these will be stored as `U`'s seed set. Note that the Personalized PageRank scoring will rank high those entities that are reachable from the movies that the user reviewed through multiple short paths, and rank low the entities that are either far and/or do not have multiple paths leading to them.

$$\texttt{reviewed}(\texttt{U},\texttt{M}) \leftarrow \texttt{seedset}(\texttt{U},\texttt{E}), \texttt{likesEntity}(\texttt{U},\texttt{E}), \texttt{related}(\texttt{E},\texttt{X}), \texttt{link}(\texttt{X},\texttt{M}), \texttt{isApplicable}(\texttt{U},\texttt{M}). \quad (4)$$
$$\texttt{likesEntity}(\texttt{U},\texttt{E}) \leftarrow \{\texttt{l}(\texttt{U},\texttt{E})\}. \quad (5)$$

Figure 4: `EntitySim`: ProPPR program for finding movies that a user may like using similarity measured using the graph links

After generating the seed set for each user, the next step in recommendation is to train a model and then use it to make predictions. As one method, we use the ProPPR program of Figure 4. It states that the user `U` may like a movie `M` if there is an entity `E` belonging to `U`'s seed set, and `U` likes `E`, and `E` is related to another entity `X`, which appears in the movie `M` (Rule 4). The predicate `isApplicable` controls the train and the test information for each user. During training, it lists the positive and negative training examples and during the test phase, that for the test, for each user. The predicate `related` is defined recursively as before. For the definition of the predicate `likesEntity`, note the term `{l(U,E)}`. This corresponds to a feature that is used to annotate the edge in which that rule is used. For example, if the rule is invoked with `U = Alice` and `E = Tom Hanks`, then the feature would be `l(Alice, Tom Hanks)`. In the training phase, ProPPR learns the weight of that feature from the training data. During the prediction phase, ProPPR uses the learned weight of the feature as the weight of the edge. Note that these learned weights for each user-entity pair are not related to the ranking obtained from the seed set generation program of Figure 2, because these weights are specific to the prediction function.

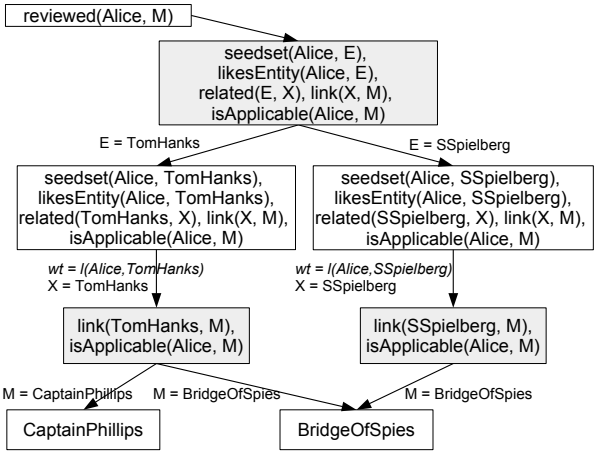

Figure 5: Sample grounding of the `EntitySim` ProPPR program

During the training phase, ProPPR grounds the program similar to that for the seed set generation discussed earlier. A sample grounding for `EntitySim` is depicted in Figure 5 where `Bridge Of Spies` and `Captain Phillips` are in the set of test examples for `Alice`. We may have other test examples for `Alice`, but if they are not provable using the rules (or beyond a certain approximation error), they will not be present in the grounding. ProPPR then follows a procedure similar to that proposed by Backstrom et. al in [3], to train the random walker. This is done by learning a weight vector $\mathbf{w}$, which given a feature vector $\mathbf{\Phi}_{uv}$ for an edge in the graph $u \rightarrow v$, computes the edge strength as $f(\mathbf{w}, \mathbf{\Phi}_{uv})$, a function of the weight and the feature vectors. i.e. the probability of traversing the edge $P(v|u) \propto f(\mathbf{w}, \mathbf{\Phi}_{uv})$. Our method uses $f(\mathbf{w}, \mathbf{\Phi}_{uv}) = e^{\mathbf{w}\cdot\mathbf{\Phi}_{uv}}$.

During the training phase, learning of $\mathbf{w}$ is posed as an optimization problem with the constraint that the PageRank computed for the positive example nodes is greater than that of the negative examples, as below:

$$-\sum_{k=1}^{m}\Big(\sum_{i=1}^{I_m}\log \mathbf{p}[u_i^{k+}] + \sum_{j=1}^{J_m}\log(1-\mathbf{p}[u_j^{k-}])\Big) + \mu\|\mathbf{w}\|_2^2 \quad (6)$$

where, $\mathbf{p}$ is the PageRank vector computed with the edge weights obtained with $\mathbf{w}$. The optimization function of Equation 6 used by ProPPR is the standard positive-negative

log loss function instead of the pairwise loss function used in [3]. To learn $\mathbf{w}$, we use AdaGrad [8] instead of the quasi-Newton method used in [3] and SGD used in [26]. The initial learning rate used by AdaGrad and the regularization parameter μ are set to 1. For a thorough description of ProPPR, we refer the reader to [26] and [25].

3.5 Approach 2: TypeSim

The `EntitySim` method uses only the knowledge graph links to learn about user's preferences. However, recall that we are in fact using a heterogenous information network where in addition to the link information, we also know the "type" of the entities. For example, we know that `New York City` is of type `City` and `Tom Hanks` is of type `Actor`. To leverage this additional type information, we extend the `EntitySim` method to `TypeSim` method as shown in Figure 6.

```
reviewed(U,R) ← seedset(U,E), likesEntity(U,E),
                popularEntity(E), related(E,X),
                link(X,R), isApplicable(U,R).        (7)
likesEntity(U,E) ← {l(U,E)}.                         (8)
popularEntity(E) ← entityOfType(E,T),
                   popularType(T){p(E)}.             (9)
popularType(T) ← {p(T)}.                             (10)
typeAssoc(X,Z) ← entityOfType(X,S), entityOfType(Z,T),
                 typeSim(S,T).                       (11)
typeSim(S,T) ← {t(S,T)}.                             (12)
```

Figure 6: TypeSim method for recommendations

`TypeSim` models the general popularity of each of the node types in Rule 10 by learning the overall predictability offered by the type of the entity using `p(T)`. For example, nodes of the type `Actor` may offer more insight into users' preferences than those of type `Country`. Note that, the learned weight is not specific to the user and hence its weight is shared by all the users. Similar to Rule 10, in Rule 9, the model learns the overall predictability offered by the entity itself, independent of the user using `p(E)`. For example, it could be that the movies directed by `Steven Spielberg` are more popular than those by other lesser known directors. `TypeSim` also models a general traversal probability between two types using Rules 11 and 12. For example, `Actor` → `Movie` is generally a more predictive traversal on the graph compared to `Country` → `Movie`. These weights are incorporated into the prediction rule of `EntitySim` as shown in Rule 7.

3.6 Approach 3: GraphLF

One of the most successful types of Collaborative Filtering (CF) are the Latent Factor (LF) models [15]. They try to uncover hidden dimensions that characterize each of the objects thus mapping users and items onto the same feature space for an improved recommendation performance. Koren et al. note in [15] that for movies, latent factors might measure obvious dimensions such as comedy versus drama, amount of action, or orientation to children, as well as less well-defined dimensions such as depth of character development, or quirkiness, or even uninterpretable dimensions. For users, each factor measures how much the user likes movies that score high on the corresponding factor. Singular Value Decomposition (SVD) is one of the more popular methods of generating LF models for recommendation. An SVD method assigns users and items with values along each of the hidden dimensions while minimizing a loss function over the predicted and actual rating matrix.

The main attraction of Collaborative Filtering methods is that they do not require any knowledge about the users or items and predict solely based on the rating matrix. Similarly, the main attraction of Latent Factor based CF models is that they develop a general representation of users and items based on the ratings data that are more generalizable and often indiscernible in the raw data.

```
reviewed(U,R) ← related(U,E), related(E,X), link(X,R),
                isApplicable(U,R).                   (13)
related(U,E) ← seedset(U,E), simLF(U,E).             (14)
related(X,X) ← .                                     (15)
related(X,Y) ← link(X,Z), simLF(X,Z), related(Z,Y).  (16)
simLF(X,Y) ← isDim(D), val(X,D), val(Y,D).           (17)
val(X,D) ← {v(X,D)}.                                 (18)
```

Figure 7: GraphLF method for recommendations

Given that we have access to a KG that connects the items via different entities, the third approach that we propose in this paper, `GraphLF`, integrates latent factorization and graph-based recommendation. The overall ruleset is defined in Figure 7. Its principal rule is the definition of Latent Factor Similarity `simLF` in Rules (17) and (18). Essentially, `simLF` of two input entities `X` and `Y` is measured by first picking a dimension `D`, and then measuring the values of `X` and `Y` along `D`. If there are many dimensions `D` along which the values of both `X` and `Y` are high, then probabilistically, their similarity scores will also be high. The value of an entity `X` along dimension `D`, `val(X,D)` is learned from the data during the training phase, as defined in Rule (18).

Note how the recursive definition of the relatedness of two entities `related(X,Y)` in Rule (16) has now changed to account for their latent factor similarity in addition to the presence of a link between them. Also, the original prediction rule has changed in Rule (13) to use a new relatedness score between the user and the entity. Essentially, the definition of `related(U,E)` in Rule (14) replaces the earlier predicate `likesEntity(U,E)` with the latent factor similarity `simLF(U,E)`, between the user and an entity belonging to their seedset. Therefore, the model no longer learns a weight for each user-entity pair, and instead learns weights for the users and entities separately along each of the dimensions.

It is also important to note that `GraphLF` is type-agnostic unlike `TypeSim` and `HeteRec_p`. Types are not always available, especially for general-purpose graphs like the Wikipedia. Therefore, being type-agnostic is a desirable property and increases its applicability to a wide range of data domains.

3.7 Model Complexity

Let n be the number of users and m, the items. Let e be the number of distinct entities and t be the types. Then the

complexity of the model as characterized by the number of parameters learned for each of the methods proposed in this paper are as below:

- `EntitySim` - $\mathcal{O}(n)$: In this method, we learn one parameter per user-entity pair. However, by virtue of the rules, we constrain the entities to be chosen from the seedset of that user, which is of a constant size c.
- `TypeSim` - $\mathcal{O}(n + e + t^2)$: In addition to those parameters learned for `EntitySim`, it also learns $e + t$ weights for each of the entities and types. Moreover, it also learns the type association between pairs of types leading to an additional t^2 parameters.
- `GraphLF` - $\mathcal{O}(n+m+e)$: For each of the users, entities and items, we learn a constant d number of weights corresponding to the latent dimensions.

In typical domains, we would expect $t \ll m$. Therefore, `EntitySim` is the simplest of the models and `GraphLF` is the more complex of the models proposed in this paper, in terms of the number of parameters.

4. EXPERIMENTS AND RESULTS

4.1 Datasets

To test our proposed methods, we use two well known large datasets:

- **Yelp2013**: This is the 2013 version of the Yelp Dataset Challenge[2] released by Yelp[3], available now at Kaggle[4]. We use this earlier version instead of the latest version from the Yelp Dataset Challenge for the purposes of comparing with the published results of the `HeteRec_p` algorithm. Similar to [27], we discard users with only 1 review entry.

- **IM100K**: This dataset is built from the MovieLens-100K dataset[5] unified with the content — director, actor, studio, country, genre, tag — parsed out from their corresponding IMDb pages[6]. We could not obtain the dataset used in [27], which we will refer to as `IM100K-UIUC`. Our dataset `IM100K*` is a close approximation to it, created from the same MovieLens-100K dataset, but we have recovered the content of 1566 movies of the total 1682 movies compared to 1360 in [27], and have 4.8% more reviews than [27].

For all the datasets, similar to [27], we sort the reviews in the order of their timestamps, and use the older 80% of the reviews as training examples and the newer 20% of the reviews as the test examples. The overall statistics of these datasets, viz. the number of users, the number of items and the total number of reviews/ratings, are listed in Table 1.

4.2 Experimental Setup

We evaluate the performance using the standard metrics, Mean Reciprocal Rank (MRR), and Precision at 1, 5 and 10 (P@1, P@5 and P@10) [19].

In our experiments, we found that any reasonable choice for the seed set size worked well enough. A fixed size serves to constrain the number of parameters learned and hence, the complexity of the model.

[2]https://www.yelp.com/dataset_challenge
[3]http://www.yelp.com/
[4]https://www.kaggle.com/c/yelp-recsys-2013/data
[5]http://grouplens.org/datasets/movielens/
[6]http://www.imdb.com/

Dataset	#Users	#Items	#Ratings
`Yelp`	43,873	11,537	229,907
`IM100K-UIUC`	943	1360	89,626
`IM100K*`	940	1566	93,948

Table 1: Dataset Statistics

In the following sections, we compare our methods to the state-of-the-art method `HeteRec_p` proposed in [27] on the two datasets. We also compare the performance to a Naïve Bayes (`NB`) baseline, which represents a recommender system that uses only the content of the item without the knowledge graph and link information, to make predictions. Naïve Bayes classifiers have been previously shown to be as effective as certain computationally intensive algorithms [22]. For each user, the `NB` system uses the entities of the items in that user's training set as the features to train the model. These are the same entities used by our methods. We use the probabilities output by the classifier to rank the pages. The implementation used is from the Mallet [20] package. Since `HeteRec_p` was shown to be better than Popularity (which shows the globally popular items to users), Co-Click (which uses the co-click probabilities to find similar items), Non–Negative Matrix Factorization[7] and Hybrid-SVM (which uses a Ranking SVM[13] on metapaths) in [27], we refrain from repeating those comparisons again.

4.3 Performance Comparison on Yelp

The performance of the algorithms proposed in this paper as well as the baselines on the `Yelp` data are tabulated in Table 2. It can be seen from the results that our methods outperform `HeteRec_p` by a large margin. For example, `GraphLF` is 126% better on P@1 and `TypeSim` is 89% better on MRR.

Also, we can note that using the type information (`TypeSim`) improves the performance drastically compared to `EntitySim`. For example, P@1 improves by 118% and MRR by 51%. Similarly, we can note that discovering the latent factors in the data (`GraphLF`) also improves the performance tremendously compared to its lesser-generalizable counterpart (`EntitySim`). For example, P@1 improves by 115% and MRR by 37%.

However, there is no clear winner when comparing `TypeSim` and `GraphLF`: while the former scores better on MRR, the latter is better on P@1.

The `NB` baseline's performance is poor, but that was expected since the dataset is extremely sparse.

4.4 Performance Comparison on IM100K

The performance of `HeteRec_p` on the `IM100K-UIUC` dataset, and that of the algorithms proposed in this paper and the Naïve Bayes baseline on the `IM100K*` dataset, are tabulated in Table 3.

As noted in Section 4.1, the `IM100K-UIUC` dataset and the `IM100K*` dataset are slightly different from each other. Therefore, we cannot compare the performance of the methods directly; the most that can be said is that the methods appear to be comparable.

A more interesting and surprising result is that the simplest of the methods, `NB` and `EntitySim`, perform as well or

Method	P@1	P@5	P@10	MRR	Settings
HeteRec_p	0.0213	0.0171	0.0150	0.0513	*published results*
EntitySim	0.0221	0.0145	0.0216	0.0641	$n = 20$
TypeSim	0.0444	**0.0188** [↑ 10%]	**0.0415** [↑ 176%]	**0.0973** [↑ 89%]	$n = 20$
GraphLF	**0.0482** [↑ 126%]	0.0186	0.0407	0.0966	$n = 20$, $dim = 10$
NB	0	0.0012	0.0013	0.0087	

Table 2: Performance comparison on Yelp: The best score for each metric is highlighted in blue and the lowest score in red. [↑ x%] gives the percent increase compared to the corresponding HeteRec_p score

Method	P@1	P@5	P@10	MRR	Settings
HeteRec_p (on IM100K-UIUC)	0.2121	**0.1932**	0.1681	**0.553**	*published results*
EntitySim	0.3485	0.1206	**0.2124** [↑ 26.3%]	0.501 [↓ −9.4%]	$n = 10$
TypeSim	**0.353** [↑ 66.4%]	0.1207 [↓ −37.5%]	0.2092	0.5053	$n = 10$
GraphLF	0.3248	0.1207 [↓ −37.5%]	0.1999	0.4852	$n = 10$, $dim = 10$
NB	0.312	0.1202	0.1342	0.4069	

Table 3: Performance comparison on IM100K (IM100K-UIUC & IM100K*): The best score for each metric is highlighted in blue and the lowest score in red. [↑ x%] gives the percent increase compared to the corresponding HeteRec_p score and [↓ x%], the percent decrease.

better than the more complex TypeSim and GraphMF. In fact, NB outperforms HeteRec_p on the P@1 metric. This leads us to speculate that when there are enough training examples per user and enough signals per item, simple methods suffice. We explore this conjecture more fully below.

4.5 Effect of Dataset Density on Performance

In the previous two sections, we saw how on the Yelp dataset TypeSim and GraphLF performed extremely well in comparison to the EntitySim method, whereas on the IM100K dataset, the latter was as good as or better than the former two. In this section, we explore this phenomenon further.

An important difference between the two datasets is that Yelp is a complete real world dataset with review frequencies exhibiting a power law distribution, while IM100K is a filtered version of a real world dataset counterpart, as noted by the authors of [27]: in IM100K dataset, each user has rated at least 20 movies.

To quantitatively compare their differences, we define the Density of a dataset as $\frac{\#\text{reviews}}{\#\text{users} \times \#\text{items}}$, which is the ratio of filled entries in the rating matrix to its size. Using this definition, the density of Yelp was found to be only 0.00077 whereas that of IM100K* was 0.06382.

To study this further, we created 4 additional datasets from Yelp by filtering out users and businesses that have less than k reviews, where k =10, 25, 50 and 100. The MRR scores of our methods and the NB baseline with varying k is plotted in Figure 8 (with the left y axis). These are with a seedset of size 20. The graph densities at the different k are also plotted in the same Figure 8 (in green with the right y axis). Note that the density increases to 0.11537 at $k = 100$, which is 148 times higher than the density at $k = 2$.

From the figure, we can see that when the dataset is the least dense ($k = 2$), the more complex methods TypeSim and GraphLF perform much better than the simple EntitySim. However, as the density increases with larger k, we

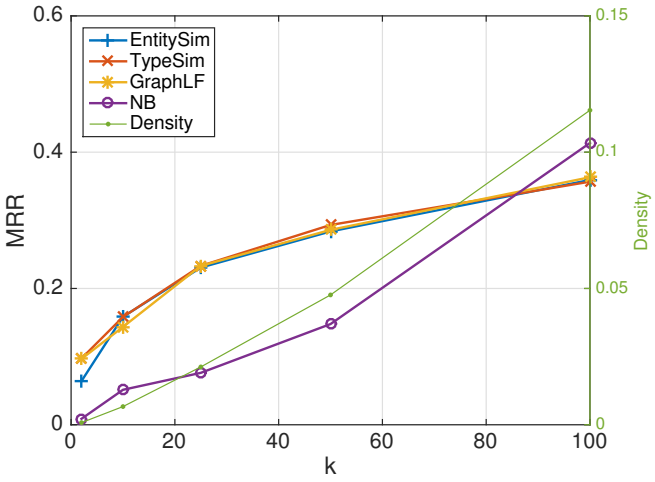

Figure 8: Performance of different methods with varying graph densities on Yelp

can observe that EntitySim eventually equals TypeSim and comes within 1% of that of GraphLF, at $k = 100$. Therefore, we can deduce that, when we have enough training examples and a dense graph connecting them, a simple method like EntitySim that predicts based on just the links in the graph can give good results.

Another observation from Figure 8 is that the NB recommender, whose performance is poor at low graph densities — 633% worse than EntitySim — slowly improves with increasing k to eventually better all the KG based methods at $k = 100$ (14% better than GraphLF). From this, we conjecture that when there are enough training examples per user, we can produce accurate predictions using a simple classi-

fier based recommender like NB. However, at slightly lower densities, like at $k = 50$, the knowledge graph is a valuable source of information for making recommendations, as can be seen from the figure, where NB is 92% below EntitySim at $k = 50$.

5. CONCLUSIONS AND FUTURE WORK

Recommender systems are an important area of research. However, not many techniques have been developed in the past that leverage KGs for recommendations, especially in the case of sparse long-tailed datasets. In this paper, we proposed three methods for performing knowledge graph based recommendations using a general-purpose probabilistic logic system called ProPPR. Our methods use the link structure of the knowledge graph as well as type information about the entities to improve predictions. The more complex of the models proposed in this paper combined the strengths of latent factorization with graphs, and is type agnostic. By comparing our methods to the published results of the state-of-the-art method that uses knowledge-graphs in generating recommendations, we showed that our methods were able to achieve a large improvement in performance.

We also studied the behavior of the methods with changing dataset densities and showed that at higher densities, just the graph link structure sufficed to make accurate recommendations and the type information was redundant. In addition, we showed that in sparse datasets, the knowledge graph is a valuable source of information, but its utility diminishes when there are enough training examples per user.

In the future, we plan to use the sentiments expressed in the reviews to improve the recommendations. Extending the approach to account for the temporal dynamics would also be an interesting direction for research.

Acknowledgments

We would like to thank Kathryn Mazaitis for her help with the ProPPR code. This research was supported in part by Yahoo! through the CMU InMind project [1].

6. REFERENCES

[1] A. Azaria and J. Hong. Recommender system with personality. In *Proc. RecSys*, 2016.

[2] S. H. Bach, M. Broecheler, B. Huang, and L. Getoor. Hinge-loss markov random fields and probabilistic soft logic. arXiv:1505.04406 [cs.LG], 2015.

[3] L. Backstrom and J. Leskovec. Supervised random walks: Predicting and recommending links in social networks. In *Proc. WSDM '11*, pages 635–644, 2011.

[4] M. Brand. A random walks perspective on maximizing satisfaction and profit. In *Proc. SDM*, 2005.

[5] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: Scalable online collaborative filtering. In *Proc. WWW '07*, pages 271–280, 2007.

[6] L. de Campos, J. Fernández-Luna, J. Huete, and M. Rueda-Morales. Combining content-based and collaborative recommendations: A hybrid approach based on bayesian networks. *Int. J. Approx. Reasoning*, 2010.

[7] C. Ding, T. Li, and M. Jordan. Convex and semi-nonnegative matrix factorizations. *IEEE TPAMI*, 2010.

[8] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *JMLR '11*, pages 2121–2159, 2011.

[9] I. Guy, N. Zwerdling, D. Carmel, I. Ronen, E. Uziel, S. Yogev, and S. Ofek-Koifman. Personalized recommendation of social software items based on social relations. In *Proc. RecSys*, 2009.

[10] T. H. Haveliwala. Topic-sensitive pagerank. In *Proc. WWW '02*, pages 517–526, 2002.

[11] J. Hoxha and A. Rettinger. First-order probabilistic model for hybrid recommendations. In *Proc. ICMLA '13*, pages 133–139, 2013.

[12] M. Jamali and M. Ester. Trustwalker: A random walk model for combining trust-based and item-based recommendation. In *Proc. KDD*, 2009.

[13] T. Joachims. Optimizing search engines using clickthrough data. In *Proc. SIGKDD*, 2002.

[14] I. Konstas, V. Stathopoulos, and J. M. Jose. On social networks and collaborative recommendation. In *Proc. SIGIR '09*, pages 195–202, 2009.

[15] Y. Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proc. KDD '08*, pages 426–434, 2008.

[16] P. Kouki, S. Fakhraei, J. Foulds, M. Eirinaki, and L. Getoor. Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems. In *Proc. RecSys '15*, pages 99–106, 2015.

[17] T. Lin, Mausam, and O. Etzioni. Entity linking at web scale. In *Proc. AKBC-WEKEX '12*, pages 84–88, 2012.

[18] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag New York, Inc., 1984.

[19] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[20] A. K. McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[21] T. Mitchell, W. Cohen, E. H. Jr., P. Talukdar, J. Betteridge, A. Carlson, B. Mishra, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *Proc. AAAI*, 2015.

[22] M. Pazzani and D. Billsus. Learning and revising user profiles: The identification ofinteresting web sites. *Mach. Learn.*, 27(3):313–331, June 1997.

[23] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge. In *Proc. WWW*, 2007.

[24] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *PVLDB*, 2011.

[25] W. Y. Wang and W. W. Cohen. Joint information extraction and reasoning: A scalable statistical relational learning approach. In *Proc. ACL 2015*, pages 355–364, 2015.

[26] W. Y. Wang, K. Mazaitis, and W. W. Cohen. Programming with personalized pagerank: A locally groundable first-order probabilistic logic. In *Proc. CIKM '13*, pages 2129–2138, 2013.

[27] X. Yu, X. Ren, Y. Sun, Q. Gu, B. Sturt, U. Khandelwal, B. Norick, and J. Han. Personalized entity recommendation: A heterogeneous information network approach. In *Proc. WSDM*, 2014.

[28] X. Yu, X. Ren, Y. Sun, B. Sturt, U. Khandelwal, Q. Gu, B. Norick, and J. Han. Recommendation in heterogeneous information networks with implicit user feedback. In *Proc. RecSys*, 2013.