Translational Models for Item Recommendation

Enrico Palumbo[1,2,3], Giuseppe Rizzo[1], Raphaël Troncy[2], Elena Baralis[3], Michele Osella[1] and Enrico Ferro[1]

[1] ISMB, Italy,
{palumbo, giuseppe.rizzo, osella, ferro}@ismb.it
[2] EURECOM, France,
raphael.troncy@eurecom.fr
[3] Politecnico di Torino, Italy,
elena.baralis@polito.it

Abstract. Translational models have proven to be accurate and efficient at learning entity and relation representations from knowledge graphs for machine learning tasks such as knowledge graph completion. In the past years, knowledge graphs have shown to be beneficial for recommender systems, efficiently addressing paramount issues such as new items and data sparsity. In this paper, we show that the item recommendation problem can be seen as a specific case of knowledge graph completion problem, where the "feedback" property, which connects users to items that they like, has to be predicted. We empirically compare a set of state-of-the-art knowledge graph embeddings algorithms on the task of item recommendation on the Movielens 1M and on the LibraryThing dataset. The results show that translational models outperform typical baseline approaches based on collaborative filtering and popularity and that the dimension of the embedding vector influences the accuracy of the recommendations.

Keywords: Knowledge Graphs, Recommender Systems, Embedding, Translational models

1 Introduction

Recommender systems are traditionally divided in two families: content-based and collaborative filtering algorithms. Content-based algorithms recommend items similar to the set of items that a user has liked in the past, considering the item content, i.e. its metadata. On the other hand, collaborative filtering algorithms look for users that are similar in terms of item preferences and suggest to a user items that similar users have liked. Hybrid systems attempt to put together the best of both worlds, by combining content-based filtering and collaborative filtering [1]. Knowledge graphs provide an ideal data structure for such systems, as a consequence of their ability of encompassing heterogeneous information, such as user-item interactions and items' relation with other entities, at the same time. Recommender systems leveraging knowledge graphs have shown to be competitive with state-of-the-art collaborative filtering and

to efficiently address issues such as new items and data sparsity [24,16,6,17,18]. In recent years, a great deal of attention has been given to machine learning algorithms able to learn entity and relation vector representations ('embeddings') from knowledge graphs for prediction tasks, such as knowledge graph completion, triple classification, entity resolution [22]. More in detail, translational models, which model relations between entities as translations in a vector space, have shown to be quite accurate at these prediction tasks, while being computationally efficient and scalable to large graphs [3,23,13].
In this paper, we show how translational models can be used to create hybrid recommender systems leveraging knowledge graphs, we evaluate their accuracy and we compare them empirically. More in detail, we address the following research questions:

1. How can translational models for knowledge graph embeddings be used for item recommendation?
2. How do they perform on two standard benchmark datasets and how do they compare with collaborative filtering baselines?
3. How much is the performance affected by the hyper-parameters, such as the embedding size?

We show that, when modelling users and items as entities of a knowledge graph, the item recommendation problem can be seen as a specific case of knowledge graph completion problem, where the "feedback" property has to be predicted. Thus, we compare three popular translational models for knowledge graph embeddings (TransE [3], TransH [23], TransR [13]) on the problem of item recommendation. The evaluation on the Movielens 1M and the LibraryThing dataset shows that: 1) knowledge graph embeddings methods outperform SVD, a matrix factorization collaborative filtering baseline, and the "Most Popular" baseline 2) models such as TransE and TransH obtain better performance with respect to TransR 3) the embedding size affects the accuracy of the recommendations, but TransE performs better than competing methods on the Movielens1M dataset for almost all metrics and dimensions d.

2 Related Work

Knowledge Graph Embeddings: the term knowledge graph embeddings refer to vector representations of entities and/or relations that attempt to preserve the structure and the semantics of the knowledge graph. Comprehensive surveys of machine learning algorithms used to learn features from knowledge graphs are [14,22]. All methods attempt to describe the existing triples in the knowledge graph by learning latent features according to some modeling assumption. RESCAL [15] is a tensor factorization method that explains triples via pairwise interactions of vector representations of entities; NTN (Neural Tensor Network) is an expressive non-linear model that learns representations using neural networks [20]; distance based models, such as the Structured Embeddings (SE) [4], explain triples using a distance in the vector

space. Translational models are a special case of distance-based models that model relations as translations in the vector space and score triples according to a distance function. These models have shown to be computationally efficient and accurate at the same time and are described more in detail Sec. 3, as they are the object the paper.

Recommendations using knowledge graphs: in the past years, several works have shown the effectiveness of external knowledge resources to enhance the performance of recommender systems. In [24,6] the authors start from a graph-based data model including user feedback and item properties to generate personalized entity recommendations. In [16,17] a hybrid graph-based data model is used leveraging Linked Open Data to extract metapath-based feature that are fed into a learning to rank framework. In [19] the authors propose a content-based recommender system that automatically learns item representations using a feature learning algorithm on a knowledge graph and show the effectiveness of the learned representations in an Item-based K-Nearest Neighbor method. In [18], the authors use property-specific knowledge graph embeddings based on node2vec [10] and learning to rank to provide item recommendations. In [25], the authors use knowledge graph embeddings considering structural knowledge (e.g. triples), textual knowledge (e.g. abstract) and visual knowledge (e.g. poster) to derive semantic representation of items for item recommendation.

3 Approach

In this paper, we introduce the definition of knowledge graph, we describe translational models, we show how the problem of item recommendation can be interpreted as a knowledge graph completion problem and how a ranking function for item recommendation can be derived from translational models (Fig. 1).

3.1 Knowledge Graph

We use the definition of knowledge graph given in [18]. A knowledge graph is defined as a set $K = (E, R, O)$ where E is the set of entities, $R \subset ExΓxE$ is a set of typed relations among entities, and O is an ontology, which defines the set of relation types ('properties') $Γ$. Entities include users $u \in U \subset E$ and items $i \in I \subset E \setminus U$. An observed positive feedback between a user and an item[4] is described by a special property, which we name 'feedback'. In this work, the ontology O is represented by the DBpedia ontology [2].

3.2 Translational Models

In order to predict missing relations in a knowledge graph, most algorithms rely on feature learning approaches that are able to map entities and relations into a

[4] Movie ratings are given by users on a 1-5 scale, we assume $r \geq 4$ to be a positive rating.

vector space, generating knowledge graph embeddings. In this work, we compare the following models (known as "translational models"):

- **TransE** [3]: learns representations of entities and relations so that $h + l \approx t$ where $(h, l, t) \in R$ is a triple. h is the 'head' entity, l is the relation and t is the 'tail' entity. The score function for a triple is thus $f(h, l, t) = D(h + l, t)$ where D is a distance function such as the L_1 or the L_2 norm.
- **TransH** [23]: first extension of TransE, enables entities to have different representations when involved in different relations by projecting entities on a hyperplane identified by the normal vector w_l. The score function becomes: $f(h, l, t) = D(h_\perp + l, t_\perp)$, where $h_\perp = h - w_l^T h w_l$ and $t_\perp = t - w_l^T t w_l$ and D is a distance function such as the L_1 or the L_2 norm.
- **TransR** [13]: enables entities and relations to be embedded in a separate vector space through a matrix M_l associated to any relation l that performs projections of vectors from entity to relation space. The score function is: $f(h, l, t) = D(h_l + l, t_l)$ where $h_l = hM_l$ and $t_l = tM_l$ and D is a distance function such as the L_1 or the L_2 norm.

The models are trained through the minimization of a pairwise ranking loss function L that measures the total difference between the scores of 'positive triples' D^+ and 'negative triples' D^-, plus regularization terms such as the margin γ and other constraints:

$$L = \sum_{(h,l,t) \in D^+} \sum_{(h',l,t') \in D^-} max(0, \gamma + f_l(h, t) - f_l(h', t')) \tag{1}$$

Positive triples D^+ are triples of the knowledge graph K, whereas negative triples D^- are obtained by 'corrupting' positive triples replacing the head or tail entities with other entities. Notice that this strategy can produce false negatives, as knowledge graphs are known to be incomplete and missing triples can still be valid facts. In order to reduce this risk, we adopt the strategy described in [23], which considers non-uniform sampling probabilities depending on the type of relation.

3.3 Item Recommendation

The problem of item recommendation is that of ranking a set of N candidate items $I_{candidates} \subset I$ according to what a user may like. More formally, the problem consists in defining a ranking function $\rho(u, i)$ that assigns a score to any user-item pair $(u, i) \in UxI_{candidates}$ and then sorting the items according to $\rho(u, i)$:

$$L(u) = \{i_1, i_2, ..., i_N\} \tag{2}$$

where $\rho(u, i) > \rho(u, i+1)$ for any $i = 1..N-1$. The core idea of using knowledge graph embeddings for item recommendation is that of using the negative score assigned to a triple $f(u, feedback, i)$ as the ranking function $\rho(u, i)$ (Fig. 2). Thus, the approach can be summarized as:

– **Data splitting:** define the set of users' feedback X as a set of triples $(u, feedback, i)$. We split the set of triples X into a X_{train} and X_{test} so that $X = X_{train} \bigcup X_{test}$.
– **Training:** learn the knowledge graph embeddings from K, which includes all the triples in X_{train} as well as other triples describing item content (see how the knowledge graph is built in Sec. 4.1), obtaining vector representations of each $e \in E$ and $r \in R$ (including the 'feedback' property)
– **Testing:** for every $u \in U$, sort every $i \in I_{candidates}$ according to the score assigned to the triple $(u, feedback, i)$ by the trained translational model $\rho(u, i) = -f(u, feedback, i)$

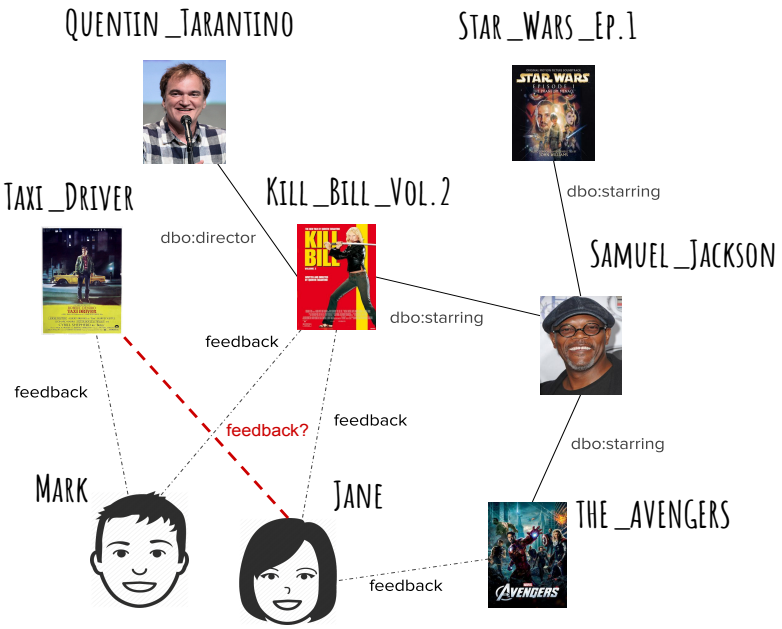


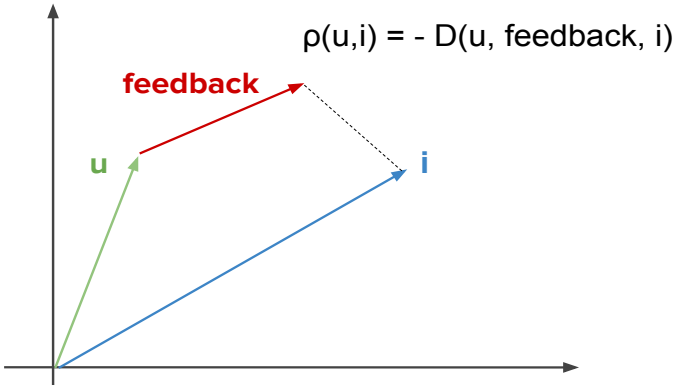
Fig. 1: Recommending items as a knowledge graph completion problem

4 Experimental setup

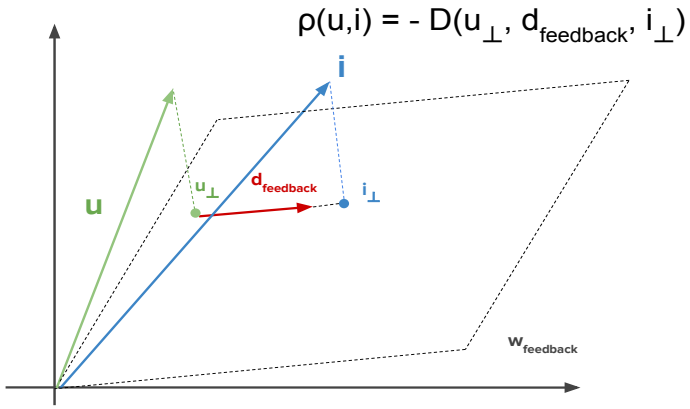
4.1 Knowledge graph construction

The first dataset used for the comparison of the knowledge graph embeddings methods is MovieLens 1M[5]. MovieLens 1M [11] is a well known dataset for the

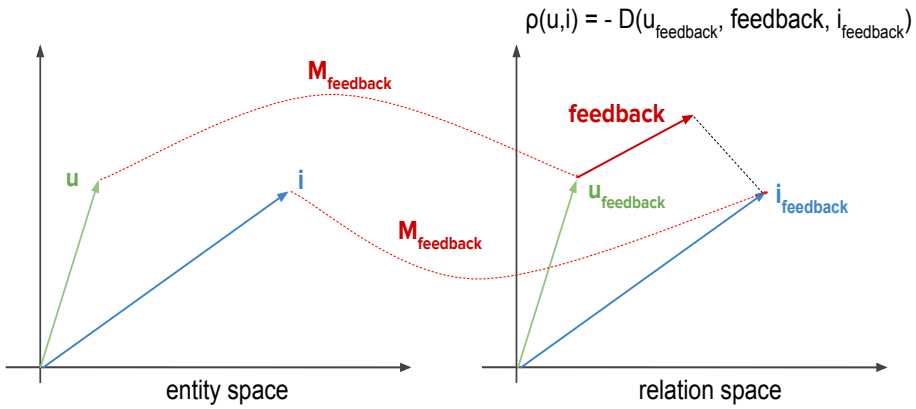
[5] `https://grouplens.org/datasets/movielens/1m/`



(a)



(b)



(c)

Fig. 2: (a): in TransE, user, items and relations are embedded in the same space and the ranking function is defined through the distance between the $u + feedback$ and i (b): in TransH, translations are performed on the hyperplane $w_{feedback}$ and thus the ranking function is defined through the distance between $u_\perp + d_{feedback}$ and $i_\perp$ (c): in TransR, entities and relations are embedded in different vector spaces and thus the ranking function is defined through the distance between $u_{feedback} + feedback$ and $i_{feedback}$

evaluation of recommender systems and it contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users. MovieLens 1M items have been mapped to the corresponding DBpedia entities [17] and we leverage these publicly available mappings to create the knowledge graph K using DBpedia data. Since not every item in the Movielens data has a corresponding DBpedia entity, after this mapping we have 948978 ratings, from 6040 users on 3226 items. We split the data into a training X_{train}, validation X_{val} and test set X_{test}, containing, per each user, respectively 70%, 10% and 20% of the ratings. In order to select the most relevant properties for the knowledge graph construction, we count what are the most frequent properties used in DBpedia to describe the items in the Movielens1M dataset and we sort them according to their frequency. We select a subset of properties of the DBpedia Ontology[6] to create the knowledge graph by sorting them according to their frequency of occurrence and selecting the first K so that the frequency of the K+1 property is less than 50% of the previous one. We obtain: ["dbo:director", "dbo:starring", "dbo:distributor", "dbo:writer","dbo:musicComposer", "dbo:producer", "dbo:cinematography", "dbo:editing"]. In this way, we avoid to select a fixed number of properties and we rely on the actual frequency of occurrence to determine the cut-off. We also add "dct:subject" to the set of properties, as it provides an extremely rich categorization of items. For each item property p, we include in K all the triples (i, p, e) where $i \in I$ and $e \in E$, e.g. (dbr:Pulp_Fiction, dbo:director, dbr:Quentin_Tarantino). We finally add the 'feedback' property, modeling all movie ratings that are $r \geq 4$ in X_{train} as triples $(u, feedback, i)$. The second dataset used for the comparison is LibraryThing[7], which contains book ratings on a 1-10 scale. By using the same mappings [17] used for Movielens1M, we obtain 410199 ratings, given by 6789 users to 9926 items linked to DBpedia. The selected properties according to the strategy described above are: ["dbo:author", "dbo:publisher", "dbo:literaryGenre", "dbo:mediaType", "dbo:subsequentWork","dbo:previousWork", "dbo:series", "dbo:country", "dbo:language", "dbo:coverArtist", "dct:subject"]. We convert ratings into the 'feedback' property when $r \geq 8$ and we apply the same data splitting procedure as for the Movielens1M dataset. Datasets statistics are summarized in Tab. 1. The sparsity of the ratings matrix ρ_r is defined as the ratio between the actual ratings and the number of possible existing user-item interactions:

$$\rho_r = \frac{T}{|U||I|} \tag{3}$$

where T is the number of ratings, $|U|$ is the number of users and $|I|$ is the number of items in the dataset. The sparsity of the knowledge graph ρ_k is defined as the ratio between the number of existing edges and the number of possible existing edges:

$$\rho_k = \frac{M}{N(N-1)} \tag{4}$$

[6] `https://wiki.dbpedia.org/services-resources/ontology`
[7] `https://www.librarything.com`

where $M = |Ex\Gamma xE|$ is the number of edges and $N = |E|$ is the number of entities in the graph.

Dataset	**Type**	**Ratings**	**Users**	**Items**	$\rho_{\mathbf{r}}$	**P**	**N**	**M**	$\rho_{\mathbf{k}}$
Movielens1M	Film	948976	6040	3226	95.130	10	29166	465338	99.945
LibraryThing	Book	410199	6789	9926	99.391	12	35768	283967	99.978

Table 1: Datasets stats. ρ_r represents the sparsity of the ratings matrix, P the number of distinct properties in the knowledge graph, N the number of nodes in the graph, M the number of edges in the graph, ρ_k is the sparsity of the graph.

4.2 Evaluation

We use the evaluation protocol known as AllUnratedItems [21], i.e. for each user we select as possible candidate items all the items either in the training or in the test set that he or she has not rated before in the training set. Items that are not appearing in test set are considered as negative examples, which is a pessimistic assumption, as users may actually like items that they have not seen yet. Scores are thus to be considered as a worst-case estimate of the real recommendation quality that would derive from an online recommendation scenario. We measure standard information retrieval metrics such as P@5, P@10, Mean Average Precision (MAP), NDCG (Normalized Discounted Cumulative Gain) to assess the quality of the ranking function. In addition to these precision-focused metrics, we also measure the serendipity of the recommendations. Serendipity can be defined as the capability of identifying items that are both attractive and unexpected [9]. Ge *et al.* proposed to measure it by considering the precision of the recommended items after having discarded the ones that are too obvious [8]. Eq. 5 details how we compute this metric. The value of *hit* is 1 if the recommended item i is relevant to user u, otherwise it is 0. Differently from the metric of precision, we consider the top-k most popular items always as non-relevant, even if they are included in the test set of user u. Popular items can be regarded as obvious because they are well-known by many users.

$$\mathrm{SER}(k) = \frac{1}{|U| \times k} \sum_{u \in U} \sum_{j=1}^{k} \mathrm{hit}(i_j, u) \tag{5}$$

As baselines, we use state-of-the-art collaborative filtering algorithms based on Singular Value Decomposition [12] and the Most Popular Items recommendation strategy, which simply ranks items based on their popularity (i.e. number of positive ratings). All the baselines have been trained on the user ratings contained

in X_{train} in the original matrix format and tested on X_{test}. The baselines are implemented using the *surprise* python library[8]. The implementation of the translational based embeddings[9] and the script used to compare them[10] are publicly available on Github. The models are compared using default hyper-parameters: $d = 100$, $k = 100$ (TransR), $learning\_rate = 0.001$, $\gamma = 1$, $epochs = 1000$.

5 Results

5.1 Empirical comparison of translational models

In this section, we empirically compare translational models for item recommendation. The results of the evaluation on the Movielens 1M are reported in Tab. 2. The results show that all knowledge graph embeddings algorithms significantly outperform baselines such as SVD, MostPop and Random. At the same time, we observe that the MostPop baseline, although trivial, is able to achieve very good results, outperforming the SVD method. Note that the MostPop is known to be quite effective on MovieLens due to the power-law distribution of user feedback data, i.e. to the fact that most user ratings tend to be concentrated on few very popular items [7]. TransE and TransH obtain the best scores for item recommendations on the Movielens1M dataset. In particular, TransE is the best performing method among translational models, showing that, for the case of Movielens 1M, a simple model with fewer parameters to learn is more effective than more complex ones such as TransR and even TransH. For LibraryThing (Tab. 3), where the graph and ratings are sparser, TransH obtains the best scores, but TransE still outperforms TransR. We also observe that the MostPop and SVD are much less effective on LibraryThing, which is significantly sparser than Movielens1M in terms of ρ_r (Tab. 1).

System	P@5	P@10	MAP	R@5	R@10	NDCG	SER@5	SER@10
TransE	**0.201424**	**0.175066**	**0.13912**	**0.079105**	**0.130375**	**0.466307**	**0.195099**	**0.163758**
TransH	0.200132	0.173493	0.136114	0.077194	0.12898	0.463236	0.191987	0.159172
TransR	0.186424	0.161325	0.127131	0.073067	0.123122	0.454165	0.182185	0.151258
MostPop	0.144603	0.129156	0.092103	0.049231	0.084936	0.406294	0.064669	0.053692
SVD	0.067814	0.0624	0.04267	0.02021	0.037233	0.328776	0.059238	0.047202
Random	0.005762	0.005861	0.008381	0.001456	0.003241	0.245982	0.005629	0.005579

Table 2: Comparison of knowledge graph embeddings and collaborative filtering algorithms on the Movielens1M dataset

[8] http://surprise.readthedocs.io/en/v1.0.2/matrix_factorization.html

[9] https://github.com/thunlp/KB2E

[10] https://github.com/D2KLab/entity2rec/blob/dev/entity2rec/trans_recommender.py

System	P@5	P@10	MAP	R@5	R@10	NDCG	SER@5	SER@10
TransH	**0.10411**	**0.082781**	**0.071303**	**0.063403**	**0.095078**	**0.335357**	**0.101576**	**0.07873**
TransE	0.097187	0.079054	0.067255	0.059799	0.09194	0.329329	0.094388	0.075357
TransR	0.07739	0.06484	0.054964	0.04585	0.072106	0.312535	0.074768	0.061246
MostPop	0.034261	0.029872	0.028128	0.025619	0.04302	0.237088	0.006982	0.005583
SVD	0.012019	0.010149	0.01085	0.010236	0.016683	0.188468	0.009368	0.006864
Random	0.000648	0.000604	0.001665	0.000354	0.000791	0.155435	0.000619	0.000574

Table 3: Comparison of knowledge graph embeddings and collaborative filtering algorithms on the LibraryThing dataset

5.2 Embedding dimension

In this section, we study how the quality of the recommendations varies as we vary the dimension of the embedding vector d. For TransR, we keep the dimension of the relation space $k = d$, as in the default configuration. We conduct the experiment on the Movielens 1M dataset. As we can see from Tab. 4, the best performance is achieved by TransE when $d = 50$, which is slightly better than the default configuration $d = 100$ for all metrics under consideration except for SER@10. In general, TransE appears to perform better than the other approaches for all values of d, except for $d = 20$ where all the models have very similar performance and where for some metrics (e.g. R@5 and R@10) TransR seems to perform slightly better. To better compare the models, in Fig. 3 we depict the evolution of the NDCG as a function of d for the translational models. We observe that when $d = 20$ the models have very similar performance, but in general TransE performs better than TransH and TransR for most values of d. It is also interesting to notice that there is large gap between TransE and the other two methods when $d = 10$, showing that a simpler model can obtain significantly better results when the number of features is small. The peak performance varies: TransE has its peak when $d = 50$, whereas TransH and TransR when $d = 100$. For all models, it seems a good strategy to not increase d over 100. In general, the default configuration of $d = 100$ appears to be justified and sensible, but the picture shows that tuning the embedding dimension parameter to generate item recommendations can still improve the performance in some cases. We leave as future work the study of the importance of other hyper-parameters such as the learning rate or the margin γ of the loss function.

Model	d	P@5	P@10	MAP	R@5	R@10	NDCG	SER@5	SER@10
TransE	10	0.093444	0.08904	0.068161	0.020198	0.039709	0.366541	0.091556	0.085083
TransH	10	0.002649	0.003874	0.029036	0.000473	0.001633	0.303868	0.002649	0.003825
TransR	10	0.018212	0.021589	0.024472	0.005329	0.013912	0.297924	0.018212	0.021589
TransE	20	0.159172	0.144917	0.110341	0.050229	0.091037	0.429045	0.151854	0.132103
TransH	20	0.151854	0.139007	0.106262	0.050553	0.090665	0.425542	0.14394	0.125728
TransR	20	0.15149	0.138808	0.108588	0.052173	0.093729	0.428234	0.148841	0.132815
TransE	30	0.188609	0.168245	0.128919	0.06693	0.117108	0.452667	0.176689	0.150414
TransH	30	0.185728	0.165877	0.126865	0.06692	0.115647	0.4504	0.173377	0.148709
TransR	30	0.175728	0.154272	0.120587	0.062922	0.109542	0.44422	0.169272	0.142616
TransE	**50**	**0.207053**	**0.179834**	**0.143832**	**0.081219**	**0.134856**	**0.471388**	**0.195828**	0.163411
TransH	50	0.192715	0.168891	0.13259	0.072662	0.122104	0.458386	0.183808	0.155298
TransR	50	0.173709	0.152384	0.118384	0.066293	0.113591	0.442525	0.166589	0.14149
TransE	100	0.201424	0.175066	0.13912	0.079105	0.130375	0.466307	0.195099	**0.163758**
TransH	100	0.200132	0.173493	0.136114	0.077194	0.12898	0.463236	0.191987	0.159172
TransR	100	0.186424	0.161325	0.127131	0.073067	0.123122	0.454165	0.182185	0.151258
TransE	200	0.174272	0.152616	0.118552	0.067019	0.108311	0.443624	0.170861	0.143262
TransH	200	0.169768	0.146722	0.11241	0.062785	0.100185	0.436909	0.161689	0.133858
TransR	200	0.145232	0.10856	0.107458	0.062646	0.098634	0.430513	0.143311	0.102401

Table 4: Translational models performance on the Movielens 1M dataset as a function of the embedding dimension d.

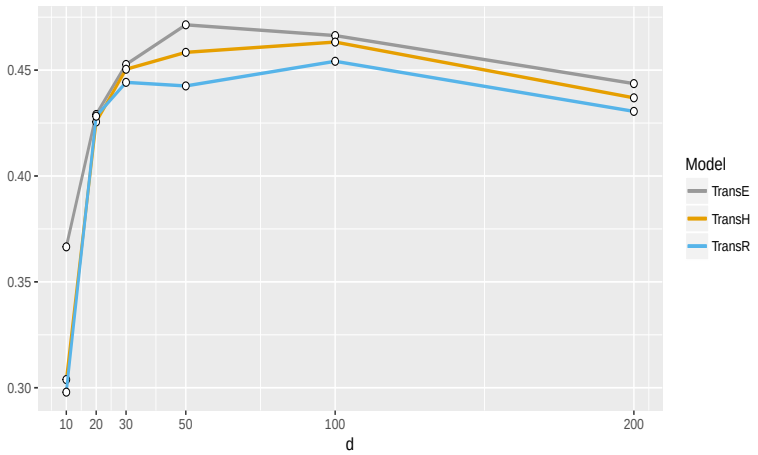


Fig. 3: NDCG of translational models on the Movielens 1M dataset as a function of the embedding dimension d

6 Conclusions

In this work, we have described the application of translational models for item recommendation, addressing the following research questions:
1) How can translational models for knowledge graph embeddings be used for item recommendation?
Item recommendation can be interpreted as a knowledge graph completion problem, where a special property called 'feedback', modeling users preferences for items, has to be predicted. More precisely, translational models can be used to learn knowledge graph embeddings and to score triples to 'complete' the knowledge graph by predicting the feedback property. The score assigned to triples $(u, feedback, i)$ defines a ranking function to perform item recommendation.
2) How do they perform on two standard benchmark datasets and how do they compare with collaborative filtering baselines?
We have evaluated the translational models on the Movielens 1M and the LibraryThing datasets, comparing them to SVD and MostPop observing that: 1) knowledge graph embeddings algorithms outperform traditional collaborative filtering algorithms such as SVD for item recommendation 2) TransE performs better for Movielens1M, whereas TransH has a better accuracy for LibraryThing. TransR, albeit being more complex, has in general a worse performance, except for some specific configurations of the embeddings size and some metrics in which it performs slightly better than the others ($d = 20$ and recall metrics)
3) How much is the performance affected by the hyper-parameters, such as the embedding size?
The embedding size affects the accuracy of the recommendations, but TransE seems better than the other approaches consistently for almost all values of d under consideration and to preserve its quality even with very small sizes such as $d = 10$.
In a future work, we plan to extend this evaluation to other datasets using implicit feedback such as LastFM [5], to include in the evaluation other existing recommender systems based on knowledge graphs such as Sprank [17], RDF2Vec [19] or entity2rec [18], to take into account specific collaborative filtering issues such as new items and data sparsity, perform exhaustive search and optimization of the hyper-parameters of the models and to account for multiple possible interactions between users and items.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE transactions on knowledge and data engineering 17(6), 734–749 (2005)
2. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. In: The semantic web, pp. 722–735. Springer (2007)
3. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Advances in neural information processing systems. pp. 2787–2795 (2013)
4. Bordes, A., Weston, J., Collobert, R., Bengio, Y., et al.: Learning structured embeddings of knowledge bases. In: AAAI. vol. 6, p. 6 (2011)
5. Cantador, I., Brusilovsky, P., Kuflik, T.: 2nd workshop on information heterogeneity and fusion in recommender systems (hetrec 2011). In: Proceedings of the 5th ACM conference on Recommender systems. RecSys 2011, ACM, New York, NY, USA (2011)
6. Catherine, R., Cohen, W.: Personalized recommendations using knowledge graphs: A probabilistic logic programming approach. In: Proceedings of the 10th ACM Conference on Recommender Systems. pp. 325–332. ACM (2016)
7. Cremonesi, P., Koren, Y., Turrin, R.: Performance of recommender algorithms on top-n recommendation tasks. In: Proceedings of the fourth ACM conference on Recommender systems. pp. 39–46. ACM (2010)
8. Ge, M., Delgado-Battenfeld, C., Jannach, D.: Beyond accuracy: Evaluating recommender systems by coverage and serendipity. In: Proceedings of the fourth ACM conference on Recommender Systems. pp. 257–260. ACM Press (2010)
9. de Gemmis, M., Lops, P., Semeraro, G., Musto, C.: An investigation on the serendipity problem in recommender systems. Information Processing & Management 51(5), 695–717 (2015)
10. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 855–864. ACM (2016)
11. Harper, F.M., Konstan, J.A.: The movielens datasets: History and context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5(4), 19 (2016)
12. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer 42(8) (2009)
13. Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: AAAI. vol. 15, pp. 2181–2187 (2015)
14. Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E.: A review of relational machine learning for knowledge graphs. Proceedings of the IEEE 104(1), 11–33 (2016)
15. Nickel, M., Tresp, V., Kriegel, H.P.: A three-way model for collective learning on multi-relational data. In: ICML. vol. 11, pp. 809–816 (2011)
16. Noia, T.D., Ostuni, V.C., Tomeo, P., Sciascio, E.D.: Sprank: Semantic path-based ranking for top-n recommendations using linked open data. ACM Transactions on Intelligent Systems and Technology (TIST) 8(1), 9 (2016)
17. Ostuni, V.C., Di Noia, T., Di Sciascio, E., Mirizzi, R.: Top-n recommendations from implicit feedback leveraging linked open data. In: Proceedings of the 7th ACM conference on Recommender systems. pp. 85–92. ACM (2013)
18. Palumbo, E., Rizzo, G., Troncy, R.: Entity2rec: Learning user-item relatedness from knowledge graphs for top-n item recommendation. In: Proceedings of the Eleventh ACM Conference on Recommender Systems. pp. 32–36. ACM (2017)

19. Rosati, J., Ristoski, P., Di Noia, T., Leone, R.d., Paulheim, H.: Rdf graph embeddings for content-based recommender systems. In: CEUR workshop proceedings. vol. 1673, pp. 23–30. RWTH (2016)
20. Socher, R., Chen, D., Manning, C.D., Ng, A.: Reasoning with neural tensor networks for knowledge base completion. In: Advances in neural information processing systems. pp. 926–934 (2013)
21. Steck, H.: Evaluation of recommendations: rating-prediction and ranking. In: Proceedings of the 7th ACM conference on Recommender systems. pp. 213–220. ACM (2013)
22. Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge graph embedding: A survey of approaches and applications. IEEE Transactions on Knowledge and Data Engineering 29(12), 2724–2743 (2017)
23. Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge graph embedding by translating on hyperplanes. In: AAAI. vol. 14, pp. 1112–1119 (2014)
24. Yu, X., Ren, X., Sun, Y., Gu, Q., Sturt, B., Khandelwal, U., Norick, B., Han, J.: Personalized entity recommendation: A heterogeneous information network approach. In: Proceedings of the 7th ACM international conference on Web search and data mining. pp. 283–292. ACM (2014)
25. Zhang, F., Yuan, N.J., Lian, D., Xie, X., Ma, W.Y.: Collaborative knowledge base embedding for recommender systems. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 353–362. ACM (2016)