Knowledge Graph Construction with External Task

Dongwoo Kim

1 Knowledge Graph Construction with External Task

Many active learning algorithms for knowledge graph construction only focus on obtaining as many positive triples as possible. This is not a crazy idea, however, it does not agree with a practical usage of knowledge graphs. For example, a content provider may want to use the knowledge graph to recommend items that fit to user's preference. This could be achieved by considering the similarity between different items in the knowledge graph based on a history of a user. If the active knowledge graph construction does not care the performance of following tasks, it may end up to discover some facts that do not lead any improvement in recommendation. Therefore, 'good' active construction algorithms should reflect the following usages of the knowledge graph.

A recent study [Zhang et al., 2016] examines how a knowledge graph can be used for a movie recommendation. Their approach resembles a co-factorisation where two different loss functions for user-movie matrix and knowledge graph tensor are used to estimate the latent embeddings of movies and users. Let $u_i \in \mathbb{R}^p$ be a latent embedding of user i, $e_j \in \mathbb{R}^p$ be a latent embedding of entity j, and $R_k \in \mathbb{R}^{p \times p}$ be a latent embedding of relation k. $X \in \mathcal{X}^{U \times E}$, where U is a number of users and E is a number of entities, is a user-movie rating matrix, and $\mathcal{G} \in \{0,1\}^{E \times E \times K}$, where K is a number of relations, is a tensor-represented knowledge graph. We use $g_{jj'k}$ to denote the value of triple $t = \{j, j', k\}$. The goal of co-factorisation is to minimise the following loss

$$\begin{aligned} L_{X,\mathcal{G}}(\mathbf{u}, \mathbf{e}, R) &= L_X(\mathbf{u}, \mathbf{e}) + \lambda L_{\mathcal{G}}(\mathbf{e}, R) & (1) \\ &= \sum_{ij} \ell_X(X_{ij}, u_i^\top e_j) + \lambda \sum_{jj'k} \ell_{\mathcal{G}}(g_{jj'k}, e_j^\top R_k e_{j'}), & (2) \end{aligned}$$

where we use standard bilinear model to score each triple given latent variables. A squared error is typically used as a loss for the matrix factorisation. We may consider the recommendation as a binary classification problem. In this case, we can choose an appropriate loss ℓ_X to optimise. We often measure the performance of these problems using AUC to take into account an imbalanced distribution of labels. In this case, it would be preferable to directly optimise AUC on a training set, or in practice, optimise a surrogate loss. However, the current objective optimise both losses for recommendation and knowledge graph construction, which is not an optimal choice.

One way to back propagate the loss of the classification is to design nested objectives. One possible candidate is

$$L_{X,\mathcal{G}}(\mathbf{u}) = \sum_{ij} \ell(X_{ij}, u_i^\top f_j(\mathcal{G})) \quad (3)$$

where f_j extracts p-dimensional feature of entity j from the entire knowledge graph $\mathcal{G}$. Two questions are naturally occurring from here: 1) How to extract feature of entity j from the entire knowledge graph? 2) How does the back-propagated loss help to construct a KG?

How to model feature function of an entity? The feature function f_j should reflect some local (or global) structure of the knowledge graph. Let $\mathcal{T}_j$ be a set of triples that contain entity j. We **may** also use the bilinear structure to model features as follow:

$$f_j(\mathcal{G}) = \frac{1}{|\mathcal{T}_j|}\left(\sum_{jj'k} R_k e_{j'} + \sum_{j'jk} R_k^\top e_{j'}\right). \quad (4)$$

To compare with the second loss in Equation 2, we only took the structure of knowledge graph (which entities are connected to/from entity j through which relations) to model the feature function.

Which triple should be queried next? Our goal is to minimise the empirical loss in Equation 3. Therefore, next triple t should be chosen to minimise the expected loss

$$\arg\min_t \mathbb{E}_{g_t}\left[\sum_{ij} \ell(X_{ij}, u_i^\top f_j(\mathcal{G} \cup t))\right], \tag{5}$$

or maximise the expected gain

$$\arg\max_t \sum_{ij} \ell(X_{ij}, u_i^\top f_j(\mathcal{G})) - \mathbb{E}_{g_t}\left[\sum_{ij} \ell(X_{ij}, u_i^\top f_j(\mathcal{G} \cup t))\right], \tag{6}$$

where we take an expectation over the possible value of triple i.e. $g_t \in \{0, 1\}$. Again, this requires some model that estimates the probability of triple such as PRESCAL.

Does the domain adaptation or transfer learning help? The domain adaptation often provides a systematic way to transfer knowledge from one domain to the other domain. For example, a model trained on a set of synthesised images with varying font styles is used to identify a house-number on street-view dataset with the domain adaptation [Ganin and Lempitsky, 2015]. In domain adaptation, each dataset comes from different source, but all datasets share the same task i.e. input distribution $p(X)$ changes between different datasets, but label distribution $p(Y|X)$ remains the same. In our case, the domain adaptation does not fit well because we tackle to solve two separate tasks: knowledge graph completion and recommendation. The transfer learning might be more appropriate in this case. The assumption behind transfer learning is that the input distribution $p(X)$ remains the same but the label distribution $p(Y|X)$ changes between different datasets, and typically, it is assumed that we have one data set that is labeled for both problems. The difference between our problem and general transfer learning problems is that the data are i.i.d. distributed in general transfer learning, but our data is not i.i.d. distributed and can be represented by multiple relations. Markov logic networks have been used to tackle the transfer learning problem in relational datasets. For example, [Mihalkova et al., 2007] try to find mapping between relations from two different knowledge graphs via mapping two Markov logic networks. The MLN approach only works for homogeneous datasets does not suit in our case.

References

[Ganin and Lempitsky, 2015] Ganin, Y. and Lempitsky, V. (2015). Unsupervised Domain Adaptation by Backpropagation. In *ICML*.

[Mihalkova et al., 2007] Mihalkova, L., Huynh, T. N., and Mooney, R. J. (2007). Mapping and Revising Markov Logic Networks for Transfer Learning. *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, (July):608–614.

[Zhang et al., 2016] Zhang, F., Yuan, N. J., Lian, D., Xie, X., and Ma, W.-Y. (2016). Collaborative Knowledge Base Embedding for Recommender Systems. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD 2016)*, pages 353–362.