Recommendation System Based on Knowledge Graph Completion with Imbalanced Relation Types

Hao Xu*, Kin-Ping Wong*, Ruhan Li*, Yongkang Zhao*
***Equal contribution**

Abstract

In this paper, we proposed an effective and robust approach to alleviating the impacts of imbalanced relation types in the training and evaluation of recommendation systems based on knowledge graph completion methods. The adjusted loss function, that is focal loss, was proven to alleviate the bias towards dominant relations in the knowledge graph across various model architectures. The success of our approach can be attributed to the fact that, by down-weighting the abundant relations, the models are forced to pay extra attention to the scarce relations. As a result, the trained models are less biased toward abundant relations and more capable of detecting the scarce relations. We believe that this is a valuable property for a knowledge graph-based recommendation system model, where the target relation is often a minority.

Introduction

The recommendation system seeks to infer customers' preferences to deliver a personalized experience by predicting a list of entities (e.g., products or other users) that likely results in a positive interaction, such as a conversion event. An effective recommendation system facilitates users seeking information and benefits content providers with more potential to make profits. Therefore, the applications of recommendation systems have been widely demonstrated in e-commerce websites like Amazon and Alibaba, streaming services like Netflix, Spotify, Pandora, and social networks like Facebook and LinkedIn. Consequentially, recommendation systems are prevalent in today's world.

Most studies of recommendation systems focus on developing machine learning models to fit user behavior data and dismiss the imaginable impact of unbalanced data, which can lead to severe issues. So it is crucial and urgent to explore ways to alleviate or eliminate the impacts of unbalanced data on the effectiveness of these models.

The representative types of implementations of recommendation systems:

1. Matrix completion is a popular approach to personalized recommendations that identify similarities among users from their collective historical choices. While this approach has produced successful results, it is noted that attributes of entities may fail to reveal underlying preferences of customers beyond historical, collective user-item interactions.

2. Graph neural networks can be employed to learn user and item embedding from the topological structure of a bipartite graph and predict interactions between users and items. This approach can be viewed as a generalization of the matrix completion approach as it allows for multi-hop message propagation across nodes.

3. The Knowledge Graph Completion approach expands on the bipartite graph approach by adding additional entities and relations to depict user-item interactions. This additional information delivers a richer data representation of the users and the items, which can be beneficial to the performance of the recommendation system. Recent developments of the graph neural network approach on knowledge graph include KGAT [19], KGNN [7], and CompGCN [15] building on the established line of shallow knowledge graph embedding models (TransE [2], CompleEx [14], TransR [8]) have shown promises in improving the state of the art results on drug-drug interaction problem, node classification, and link prediction problems.

In the knowledge graphs designed for recommendation system models, for every (user, item) pair, there are various relations to expand and enrich the information. See the sample relation graph of Amazon Beauty Product as an example (Figure 1). As shown in Figure 2, the target relation "like" (orange) takes up less than 5% of all observations in the data. This observation poses a relation imbalance issue to model training. Inspired by [6], we propose to derive focal loss functions for each model to assign more weights to poorly trained examples. The

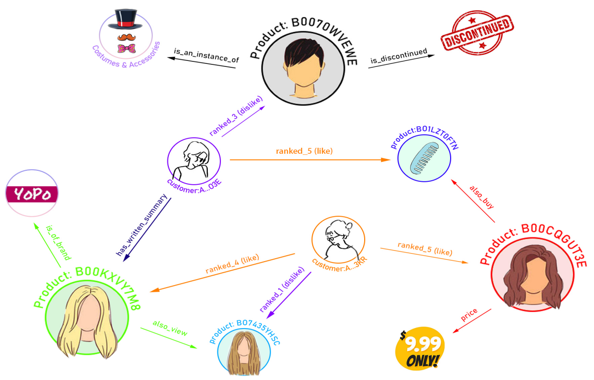


Figure 1: Sample relation graph of Amazon Beauty Product

explicit functional forms and derivations are documented in the appendix. It is worth to mention that KGAT has two components of the loss function (CF and KG losses), we can potentially apply focal loss function adjustment individually.

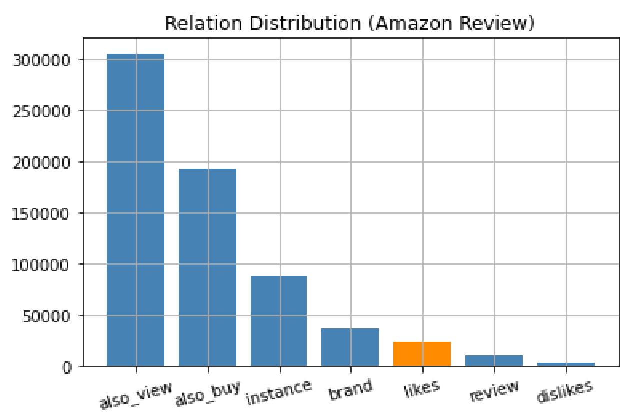
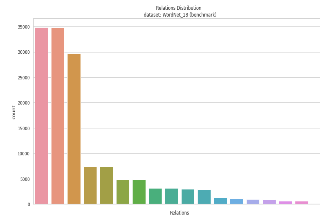


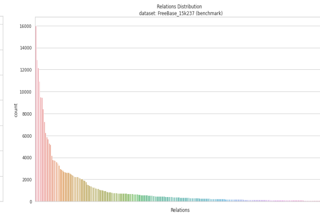
Figure 2: Distribution of Relations

When the relation types are highly imbalanced, it can lead to a bias problem that the recommendation system model is trained to be highly likely to learn these over-represented users/items/relations, reinforce them in the ranked results, and potentially result in systematic discrimination, reducing the visibility of disadvantaged users/items/relations and under-representing the minorities.

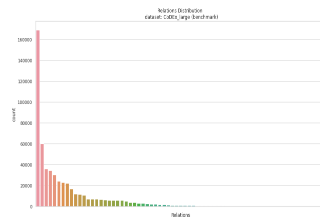
It is common for a knowledge graph to exhibit a long-tailed distribution in relations (Figure 3), with the target relation (positive user-item interaction) at the end of the tail. This arises from a one-to-many correspondence: for each (user, item) pair, there will be multiple other relations connected to each user and item nodes. As a result, the target relation is scarce. Consequently, generic knowledge graph completion algorithms suffer from the imbalance problem in the relation space due to fewer examples of target relations in the graph.



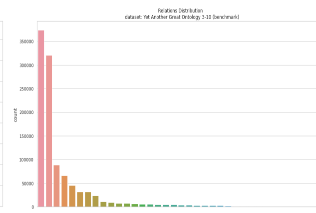
(a) WordNet18



(b) Freebase 15K-237



(c) CoDEx



(d) yago3-10

Figure 3: Selected benchmark datasets relation distributions

Therefore, we aim to develop a general approach to alleviate the impacts of imbalanced relation types within knowledge graphs for recommendation models in a principled, practical, and highly effective way. In our work, an adjusted loss function was derived from the operation on the weights of relations and proposed to alleviate the bias towards overfitting samples that are dominant in terms of relation type in the knowledge graph and under-fitting of samples that are under represented in the graph. Our approach has demonstrated significant improvements across model architectures with knowledge graphs constructed from the Amazon product review data released by Ni et al.[4]

Our loss function adjustment with focal weights aligns with the line of seminal work on "**hard example mining**" ([6], [13], [11]) - instance-level reweighting to focus on the hard samples. To our knowledge, there is not much work documenting its application on the context of graph relations. We have found improvement in predictive power across shallow and deep graph embedding models on both our experiment data and benchmark datasets. We hope that this will shed light on related topics in knowledge graph models.

Approach

In this section, we present a high level overview of the data used and models experimented with, as well as highlight our approaches to the challenges that we encountered during the process.

Data Pipeline

Data Source The raw data is obtained from the Amazon product review dataset released by Ni et al. in 2019 [4].[1] The database includes two data collections: the "**meta**" collection stores product attributes across 39 main categories, and the "**review**" collection includes product review rating, corpus, and review summaries for each product in the meta database. Product items are identified by the Amazon Standard Identification Number (ASIN) in both data collections, while reviewers are identified by an anonymous user ID in "reviews". To conclude, for each product category, we are able to observe product attributes, as well as customer-item interactions in terms of written reviews. Note that we only observe purchase activities in which the customer wrote a product review for their purchase.[2]. Here are excerpts from example documents of a meta and a review raw data:

```
{
  "price": "$9.99",
  "main_cat": "All Beauty",
  "category": [
    "Clothing, Shoes & Jewelry",
    "Costumes & Accessories",
    "Women",
    "Wigs"
  ],
  "asin": "B0002483KI",
  "brand": "Topbuti",
  ...
}
```

```
{
  "overall": 5.0,
  "verified": true,
  "reviewTime": "05 25, 2016",
  "reviewerID": "A0969754FZ",
  "asin": "B0002483KI",
  "reviewText": "With the low price of this wig,
      ↪ I was very impressed.",
  "summary": "Five Stars",
  ...
}
```

To see how a knowledge graph is built from the raw data and to present the list of relations we kept, we extracted a snippet of our data illustrated in Figure 1: here products *"B0070WVEWE"* (the black wig), *"B00KXVY7M8"* (the green wig), and *"B00CQGUT3E"* (the red wig) are three most popular wigs in the dataset. Customers *"A...O3E"* (customer A) and *"A...3KR"* (customer B) are two customers who purchased these three popular products as well as some other closely related products, including wigs of other colors and combs. There are some other relations, specifically, a product can be linked to a brand via the `"brand"` relation; to a category via the `"is_an_instance_of"` relation; to a price via the `"price"` relation; to other products via the `"also_view"` or `"also_buy"` relation, indicating that customers who purchased the product also viewed or bought some other products. Similarly, a customer can be linked to a product via `"wrote_summary"` relation, when they gave a review to the product, or `"ranked_number"` relation, with which the number stands for the number of stars that the customer has given to the product. Since the purpose of the recommendation system is to recommend products to customers that they might like, ranks above 4 are considered as liked, and the rest are classified as disliked. In this sense, the knowledge graph extends the collaborative filtering approach by featuring more types of user-item interactions. In the example, customer B has ranked 5 to the red wig, and the red wig has a `"also_buy"` relation to product *"BO1LZT0FTN"* (comb). Customer A, who bought the green wig as customer B did, has ranked 5 (like) to the comb, therefore, presumably, we would expect a system to recommend the comb to customer B.

A comprehensive visualization for the constructed knowledge graph around the same three products with actual observed number of nodes and edges is included in the Appendix.

Data Selection We select the subset products that belong to the "Beauty" category, one of the smaller datasets. This is mainly driven by storage and computational resource concerns. For model training purposes, we also filter on users and items with at least 5 interactions. As for the train-test split, our training set includes all relations and our test set includes only positive rankings (the like relation) as this is the only relation relevant to a recommender system. Implication of the induced relation class imbalanced will be discussed and addressed in the next section. The recommender systems we considered are transductive in nature, so users and items that appear in the validation and test sets must have appeared in the training set.

Data Format To streamline model training, the raw data needs to be transformed and parsed as a knowledge graph in a tensor format that can be readily loaded and read by machine learning modules such as the `pytorch DataLoader` class.

To elaborate on the data transformation, we invoke a definition of a knowledge graph data point

[1]Data repository at: https://nijianmo.github.io/amazon

[2]Non-purchasers are not qualified to write product reviews on Amazon.

proposed in [3]: A knowledge is defined by an ontology using resource description framework (RDF). Let E and R denote the sets of entities and relations in the knowledge. The RDF representation of the knowledge graph data base consists data points in terms of triples. Each triple (s, p, o) is an ordered set of the following RDF terms: a subject $s \in E$, a predicate $p \in R$, and an object $o \in E$.

For instance, the RDF representation of the raw "meta" and "review" raw data regarding the item `B0002483KI` is:

```
{ #subject, predicate, object
  ["B0002483KI", "price",          "$9.99"],
  ["B0002483KI", "main_cat",       "All Beauty"],
  ["B0002483KI", "instance_of",    "Wigs"],
  ["B0002483KI", "brand",          "Topbuti"],
  ["A0969754FZ", "rank_5",         "B0002483KI"],
  ["A0969754FZ", "wrote_summary", "Five Stars"],
  ...
}
```

The RDF data representation is essentially tabular. Therefore, it would suffice to assign an integer lookup ID for each entity and relation so they can be presented to computational module in terms of a `pytorch Long` tensor.

Implementation

Several knowledge graph completion models were used to approach the user-item prediction problem, including two shallow embedding approaches (TransE [2] and ComplEx [14]), and two graph neural network models (CompGCN [15] and KGAT [19]).

In our data, the target relation is "`like`": a user s is identified to "like" an item o if, s left a rating of at least 4 stars on the product page of o. Ultimately, for each user s in the test set, we want to find items o such that (s, like, o).

Given an RDF triplet (s, p, o) we want to train an encoder model to learn a representation/embedding for each of the entities s, o and the relation p in the d-dimensional vector space, $e_s, e_p, e_o \in \mathbb{R}^d$ such that the relation likelihood can be evaluated using a score function of tuple $\phi(e_s, e_p, e_o)$. A loss function L is then constructed to optimize the model weights.

At the end, fine-tuned models decode ϕ and output a score for each (user, item) pair, indicative of how likely the user will like the item. The scores are then ranked to give K recommendations for each test user. We will discuss performance metrics in the next section.

Shallow Knowledge Graph Embedding (KGE) Approach: TransE and ComplEx In [12], it was noted that some of the older shallow knowledge graph embedding (KGE hereafter) models, when trained and tuned properly, can provide very satisfactory results, even matching state-of-the-art results in knowledge graph completion. We also note that KGE models are much less computational expensive than deep graph networks. So, we start our implementation with two popular KGE approach and will compare performances to see if they provide simpler, but faster and more economic performances than the deep learning counterparts.

In TransE [2], an embedding representation $e_s, e_p, e_o \in \mathbb{R}^d$ is learned for each (s, p, o). The score ϕ of triplet (s, p, o) is simply the vector norm $\|e_s + e_p - e_o\|$, which the model learns to minimize.[3] The intuition for the scoring function is to consider a relation as a translation operator that moves entities in a given direction. This simple setup turns out to be scalable and compatible with a lot of real world relations[4] and makes TransE the state-of-the-art knowledge graph embedder at its time. Nevertheless, we do not anticipate TransE to perform particularly well in recommender systems because the target relation that we seek to predict - positive rating - is essentially one-to-many. For a given customer embedding, e_s, the positive rating embedding e_o learned by a TransE model will always map the customer to the same entity $e_s + e_o$. ComplEx [14] is another shallow knowledge graph embedding model that learns such one-to-many relations.

In ComplEx, the embeddings (e_s, e_p, e_o) learned are in the complex vector space $\mathbb{C}^d$. The score function for the ComplEx encoder is $\phi(s, p, o) = \mathrm{Re}(\langle e_p, e_s, \bar{e_o} \rangle)$, such that the odds of observing the triplet (s, p, o) is given by the sigmoid-activated scores. One can easily see that given an user embedding there can be multiple item embeddings that yield similar scores ϕ. Like TransE, ComplEx also utilizes a contrastive loss with negative sampling.

We implement both TransE and ComplEx on our data using the `LibKGE` package.[5]

Deep Graph Neural Network Approach: CompGCN and KGAT: We refer to TransE and ComplEx as "shallow" embedding methods as they do not involve deep neural networks. Graph neural networks presented below are deep learning extensions of KGE methods that are more expressive and powerful in fitting network patterns arising from multi-hop interactions. We implement two deep learning models for knowledge graph completion:

[3] The loss function here corresponds to the original form given in [2]. When implemented, we apply a sigmoid activation in a pairwise logistic loss formulation for smoothness. Please see appendix.

[4] For example, we can enforce logical constraints such as symmetry to enhance performance

[5] Repository at: https://github.com/uma-pi1/kge/

CompGCN [15] and KGAT [19]. The former is an extension of GCN [5] that learns multiple relations, and latter is an extension of GAT [16] with the explicit goal of modeling user-item interactions. We view both models as extensions of KGE approaches because both models uses elements of established KGE models to learn embedded layer that propagate in the networks. Because KGE models only apply to seen entities in a fixed graph, this implies that all the models we have considered in this project are transductive in nature - that they cannot generalize to new, unseen nodes or edge types. We acknowledge that scaling and inductive learning in knowledge graph completion poses real business importance and there is a line of work about this topic ([17], [9], [18], [20]); but that would be out of the scope for this project.

CompGCN is a graph convolutional network that learns relations between entities in the graph. The model is a generalization of GCN [5] that can learn to combine multiple relations in the graph. Unlike KGAT, CompGCN does not rely on pre-learned KGE models. In CompGCN a score ϕ is computed for the embedded nodes and relations (the authors considered KGE scoring functions from TransE, DistMult, ConvE) and then is propagated across layers. Like GCN [5], the computation graph for a node in each layer is given by its one-hot neighbor. At the end, a sigmoid activation layer is applied to the embedding output of the final layer to represent the likelihood of an observe/corrupt triplet.

KGAT attempts to learn two aspects of the data: a collaborative filtering component and a knowledge graph completion component. The first concerns whether interactions should exists given similarity of historical behavior in the data, to be learned by a GAT [16] instance. The latter concerns message propagation across entities for all observed relations, which is learned by a TransR [8] KGE. Embedding and model weights are trained to optimize an additive composite loss: the collaborative filtering (CF) cross entropy loss for edge prediction (for the target relation class), and a knowledge graph (KG) loss implied by the TransR [8] scoring function $\phi(s,p,o) = \|W_p e_s + e_p - W_p e_o\|^2$.

Both models utilize negative sampling. We implement them using the `CompGCN`[6] and `KGAT-pytorch`[7] modules.

Challenges

As we applied the workflow to train models on our data, we encountered the following empirical challenges:

1. **Users with little activities:** We observe that more than 95% of users present in the review database only wrote one review among all Beauty products. The sparseness of these users would make it difficult for the models to learn their embeddings with adequate statistical confidence. We filtered our training and test data to include users and products with at least 5 interactions.
2. **Memory space and time limitations:** Even with the filter above, the data size is still rather large to train and takes a long time to fine-tune. Another problem that comes with data size is having to train larger models, since graph model scales up as number of nodes and edges increase. A few instances of our attempt to fit the full data could not fit in the GPU memory. In light of the timeline and resources available of this project, we made the following compromises:
   - i Further filtered the data to manageable size,
   - ii Searched over larger learning rates,
   - iii Reduced max epochs limits,
   - iv Decreased model sizes in terms of embedding dimension,
   - v Reduced training time memory requirement by reducing batch sizes.
3. **Hyper-parameter Search:** There are a large number of hyper-parameters to be tuned for each model. Also, the evaluation metrics vary greatly across combinations. To facilitate tuning, we employ Bayesian hyper-parameter optimization techniques (implemented from with `Ax` through `LibKGE` and `optuna`) to effectively search the hyper-parameter space. We documented results across our hyperparameter trials in ML Flow server.[8]
4. **Gradient Explosion:** When the focal loss parameter γ is close to 1, gradients can become very large in magnitude and result in `NaN` values in training loss. To solve the problem, we applied clipping of focal weights - constraining them to lie in $[\varepsilon, 1-\varepsilon]$ - to stabilize learning.[9]

Experiments & Results

Experiments

After configuring the experiment environment stated as above, experiments were carried out in two stages: 1. to set baseline for each model, each of TransE, ComplEx, CompGCN, and KGAT were fitted with the same training data and validated with validation dataset. many iterations of hyper-parameters tuning were performed to achieve representative results for each model. Finally, evaluated each tuned

[6]Repo at: https://github.com/malllabiisc/CompGCN
[7]Repo at: https://github.com/LunaBlack/KGAT-pytorch
[8]http://52.53.202.9:5000
[9]We take ε to be 10^{-6}.

model with test set as baseline. 2. to compare our proposed method, we adjusted loss functions of all TransE, ComplEx, CompGCN, and KGAT models, then proceeded to tuning hyper-parameters for the adjusted model algorithms with the same manner performed in the first stage. Finally, evaluated these new models with the same test set for comparison.

Hyperparameter search over the candidate grid points was facilitated using Bayesian optimization techniques [1]. The sets of grid points for tuned hyperparameters for each model are presented in the Appendix. We use mean reciprocal rank (MRR) as the validation metric and criterion for model selection.

Results

We present the results from models fine-tuned with automatic hyper-parameter search algorithm[10] for each knowledge graph embedding methods. For each method, the model with best validation set result are selected, and test set results with selected models are recorded below (better performances are in bold font).

	Loss Function	Metric[1] @5	Metric @10	Metric @100
TransE	Original	0.000	0.000	0.090
ComplEx		**0.060**	0.075	0.383
CompGCN		0.128	0.162	0.451
KGAT		**0.129**	0.188	**0.771**
TransE	Focal	**0.004**	**0.004**	**0.117**
ComplEx		0.048	**0.094**	**0.447**
CompGCN		**0.132**	**0.192**	**0.519**
KGAT		0.115	**0.239**	0.692

[1] TransE, ComplEx, CompGCN: Hits@X, KGAT: Recall@X

Table 1: Evaluation metrics for models fitting with original loss function and loss function adjusted with focal.

Overall, deep learning models, including CompGCN and KGAT surpass the distance translation method TransE, and the shallow semantic learning method ComplEx by a significant margin. This is not surprising to us because deep learning models are more expressive than shallow embedding models. On the other hand, as we had anticipated, ComplEx performs better than TransE as the latter is known to have a shortcoming of fitting one-to-many relations. Finally, we observe improvements in at least one of the metrics from applying focal loss for each of the models.

As a robustness check of our focal weight approach, we conduct a separate run on the benchmark knowledge graph dataset `amazon-book` with KGAT, using the best hyperparameter set mentioned in the paper[19], to compare the original model and the focal-adjusted model. The results are presented in Table 2.

	Recall @3	Recall @5	Recall @10	Recall @100
Original	0.043	0.061	0.094	0.307
Focal	**0.044**	**0.062**	**0.097**	**0.310**

Table 2: Evaluation metrics for KGAT models fitted with original loss function and focal-adjusted loss function

Summary and Future Extension

We propose an effective and robust approach for alleviating the impacts of imbalanced relation types in the training and evaluation of recommendation systems based on knowledge graph completion methods. The adjusted loss function with focal loss was proven to be able to alleviate the bias towards dominant relations in the knowledge graph across model architectures. The success of our method can be attributed to the fact that, by down-weighting the abundant relations, the models are forced to pay extra attention to the scarce relations. As a result, the trained models are less biased towards abundant relations and more capable of detecting the scarce relations. We believe that this is a valuable property for a model to have when used in a knowledge graph based recommender system, where the target relation is often a minority.

Beyond our achievements, various other factors may also affect the prediction of minority relations, such as possible data leakage among train, valid, and test sets. Substitute and complementary relations among products, or more broadly the diverse specifications of knowledge graph ontology that we did not explore, plus the potential to fusion with knowledge bases [10] like WikiData,[11] NELL.[12] Further investigation of the impacts of all the above factors is worth exploring in the future. Additionally, we could explore different ways of weighting the relations in the knowledge graph, such as explicit weighting, to see if this leads to improved performance. Finally, we could investigate the differences between the errors produced by the original loss function and the focal loss adjusted loss function to see if there are any patterns that could be exploited to improve the performance of our method.

[10] Implemented with `optuna` and `ax` (through `LibKGE`)

[11] https://www.wikidata.org/

[12] http://rtw.ml.cmu.edu/rtw/kbbrowser/

References

[1] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[2] Antoine Bordes et al. "Translating embeddings for modeling multi-relational data". In: *Advances in neural information processing systems* 26 (2013).

[3] Lisa Ehrlinger and Wolfram Wöß. "Towards a definition of knowledge graphs." In: *SEMANTiCS (Posters, Demos, SuCCESS)* 48.1-4 (2016), p. 2.

[4] Julian McAuley Jianmo Ni Jiacheng Li. "Justifying Recommendations Using Distantly-Labeled Reviews and Fined-Grained Aspects". In: *Empirical Methods in Natural Language Processing (EMNLP)* (2019).

[5] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[6] Tsung-Yi Lin et al. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[7] Xuan Lin et al. "KGNN: Knowledge Graph Neural Network for Drug-Drug Interaction Prediction." In: *IJCAI*. Vol. 380. 2020, pp. 2739–2745.

[8] Yankai Lin et al. "Learning entity and relation embeddings for knowledge graph completion". In: *Twenty-ninth AAAI conference on artificial intelligence*. 2015.

[9] Shuwen Liu et al. "Indigo: Gnn-based inductive knowledge graph completion using pair-wise encoding". In: *Advances in Neural Information Processing Systems* 34 (2021).

[10] Tom Mitchell et al. "Never-ending learning". In: *Communications of the ACM* 61.5 (2018), pp. 103–115.

[11] Mengye Ren et al. "Learning to reweight examples for robust deep learning". In: *International conference on machine learning*. PMLR. 2018, pp. 4334–4343.

[12] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. "You can teach an old dog new tricks! on training knowledge graph embeddings". In: *International Conference on Learning Representations*. 2019.

[13] Kaihua Tang, Jianqiang Huang, and Hanwang Zhang. "Long-tailed classification by keeping the good and removing the bad momentum causal effect". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 1513–1524.

[14] Théo Trouillon et al. "Complex embeddings for simple link prediction". In: *International conference on machine learning*. PMLR. 2016, pp. 2071–2080.

[15] Shikhar Vashishth et al. "Composition-based multi-relational graph convolutional networks". In: *arXiv preprint arXiv:1911.03082* (2019).

[16] Petar Veličković et al. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* (2017).

[17] Bin Wang et al. "Inductive learning on commonsense knowledge graph completion". In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–8.

[18] Peifeng Wang et al. "Logic attention based neighborhood aggregation for inductive knowledge graph embedding". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 7152–7159.

[19] Xiang Wang et al. "Kgat: Knowledge graph attention network for recommendation". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 950–958.

[20] Rex Ying et al. "Graph convolutional neural networks for web-scale recommender systems". In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 974–983.

Appendices

Focal Weights Formulation

In binary classification, given a predicted probability for $y_i = 1$, the loss function is $L = \sum_i \big(y_i \ln(p_i) + (1 - y_i)\ln(1 - p_i)\big)$. As in [6], the focal weight for example i is p_i^γ if y_i is 1, and $(1 - p_i)^\gamma$ if y_i is 0. That is, the focal weight is taken to be the predicted mis-classified probability. This is very intuitive - when the model is largely predicting an incorrect outcome, we would like the example weight to increase.

In knowledge graph models we worked with, a pairwise logistic loss function is derived from contrastive learning using negative samples. In general terms, suppose the un-noramlized ("energy") score of a knowledge triplet is given by $\phi(.)$. Let (s, p, o) be a positive example actually observed in the training set and (s', p, o') be a corrupt example negatively sampled from the set of entities and relation. The pairwise logistic loss for this pair is: $-\ln \sigma\big(\phi(s, p, o) - \phi(s', p, o')\big)$. Here σ is the sigmoid activation. Despite not having the interpretation of 1-P(ground truth) in binary cross entropy loss case, we propose to have similar formulation:

$$L^* = \sum_{(s,p,o),(s',p,o')} -\lambda^\gamma \ln \sigma\big(\phi(s,p,o) - \phi(s',p,o')\big)$$
$$\lambda = 1 - \sigma\big(\phi(s,p,o) - \phi(s',p,o')\big)$$

The intuition is that whenever score differential between the true and corrupt triplets, $\phi(s, p, o) - \phi(s', p, o')$ is low, it means that the model is not learning the contrast between true and corrupt interactions well. So we emphasize the example by assigning a (positive) weight loss decreasing in $\phi(s, p, o) - \phi(s', p, o')$. The sigmoid activation bounds the loss to within $[0, 1]$ and provides a convenient transformation: 1-sigmoid activated score. To prevent gradient explosion, we clip λ and restrict it lie in the interval $[\varepsilon, 1 - \varepsilon]$.

Hyperparameter Set

TransE and ComplEx:

```
{
  "num_trials": 200,
  "num_sobol_trials": 15,
  "batch_size": {64, 128, 256, 512},
  "learning_rate": (0.0003, 1.0),
  "lookup_embedder_dim": {64, 128, 256, 512, 1024
      ↪ },
  "lookup_embedder_dropout": (0.2, 0.8),
  "lookup_embedder_regularize_weight": (0.0, 0
      ↪ .1),
  "train_type": "negative_sampling",
  "max_epochs": 40,
  "loss": {"bce", "bce_focal"}
}
```

CompGCN:

```
{
  "score_func" : {"conve", "transe", "distmult"},
  "opn" : {"corr","sub", "mult"},
  "batch" : (128, 600),
  "gamma" : (0, 44.0),
  "lr" : (0.005, 0.01),
  "lbl_smooth" : (0.01, 0.3),
  "gcn_dim" : (10, 200),
  "gcn_layer" : {1, 2},
  "gcn_drop" : (0.0, 0.3),
  "hid_drop" : (0.0, 0.3),
  "hid_drop2" : (0.0, 0.3),
  "feat_drop" : (0.0, 0.3),
  "k_w" : (5, 20),
  "k_h" : (5, 20),
  "num_filt" : (10, 200),
  "ker_sz" : (10, 30),
  "fc_loss_gamma" : {0, 0.9999},
}
```

KGAT:

```
{
  "kg_batch_size" : (1024, 2048),
  "laplacian_type" : {"random-walk",
      ↪ "symmetric"},
  "aggregation_type" : {"bi-interaction", "gcn",
      ↪ "graphsage"},
  "kg_l2loss_lambda" : (1e-5, 1e-2),
  "cf_l2loss_lambda" : (1e-5, 1e-2),
  "lr" : (0.0001, 0.01),
  "use_gamma": {0, 0.9999}
}
```

Knowledge Graph Visualization

Figure 4 shows all edges and nodes connected to the three wig products featured in Figure 1. The products nodes are represented by the product images in the figure. There are 3,667 users (blue) who reviewed these products. Common user nodes that share review history between any two of the products are represented by emojis. There are 195 edges that represent non-interactions, such as price, brand, and category instances.

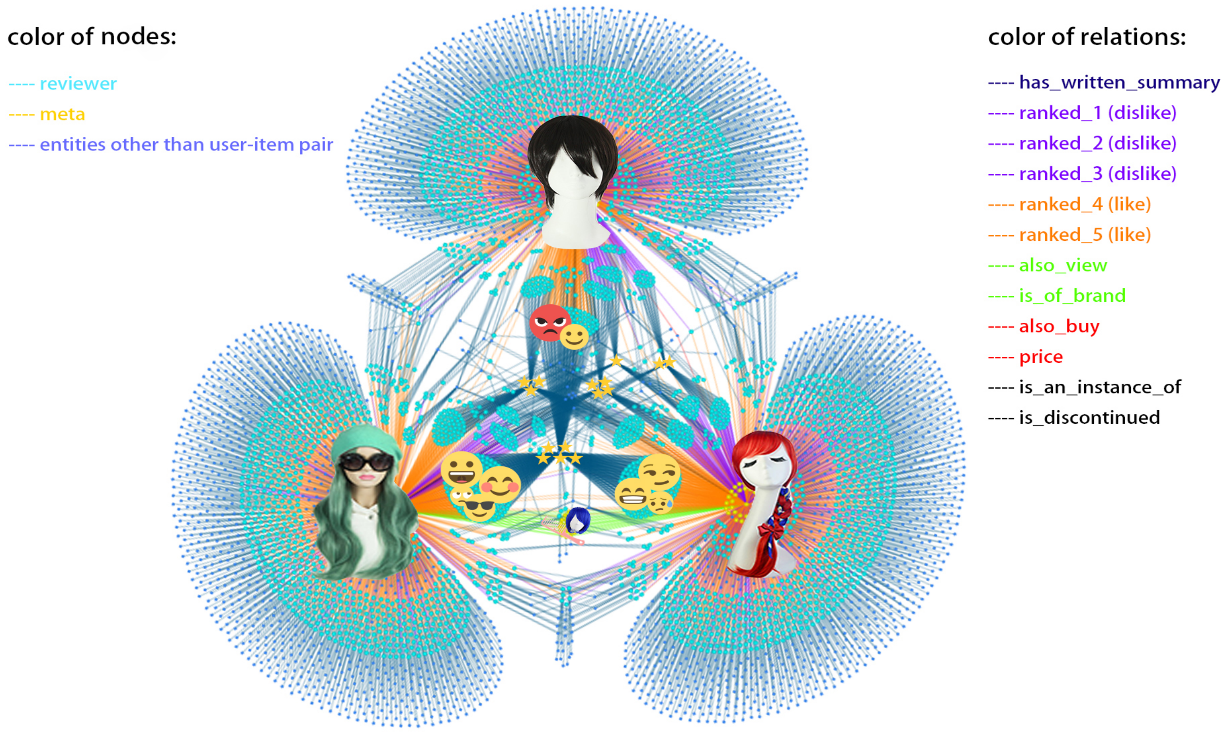


Figure 4: Knowledge Graph of Amazon Beauty Product Review (best viewed in color)