Harsh Rajkumar Vaish
Prof. Yongfeng Zhang
Recommendation based on MovieLens-20M Dataset
March 20, 2020

MovieLens-TranseE

This document details work done on the MovieLense-TransE project over the last semester. Code is accessible here.

Code Structure: There are 4 branches - baselines, TransE, RL-DQN and master. There is a separate data processing done for each of the models as detailed in section Code.

Data Sources:

Data sources can be accessed here.

KB4Rec: There are 3 files in this dataset, however, we just use one: ml2fb.txt. It maps MovieLens 20M to Freebase.

Each line in ml2fb.txt is in the following format:

RS_item_ID[\tab]*FB_item_ID*

where RS_item_ID & FB_item_ID denote item_ID's from ML20M and FreeBase dataset respectively.

FreeBase: We use 1 step graph within our knowledge graph. This has been provided in graph_movie_1step.txt

MovieLens20M(ML20M):

We also used freebase dataset: You can find the data in

We used the MovieLense-20M dataset which can be access through **here**. As of *March 20 2020*, there's a new dataset MovieLens-25M available. This dataset contains 20 million ratings & 465K tags of 27K movies rated by 138K users. The dataset contains 5 csv files:

- **movies.csv** (movieId, title, genre)
  - Errors and inconsistencies may exist in titles.
  - Genres are a pipe-separated list of [Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western, (no genres listed)]
- r**atings.csv** (userId, movieId, rating, timestamp)
  - Ratings are made on a 5-star scale, with 0.5-star increments (0.5-5.0 stars).

    - Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.
  - **tags.csv** (userId, movieId, tag, timestamp)
    - Tags are user-generated metadata about movies. Each tag is typically a single word or short phrase. The meaning, value, and purpose of a particular tag is determined by each user.
  - **links.csv** (movieId, imdbId, tmdbId)
    - movieId, imdbId, tmdbId are identifiers used by https://movielens.org, http://www.imdb.com, and https://www.themoviedb.org respectively.
  - **genome-tags.csv** (tagId, tag)
  - **genome-score.csv** (movieId, tagId, relevance)

Code:

Translational-Embedding Model:

Data Cleaning: Preprocessed data can be accessed **here**.

Relevant code: util.py and class MovieLens -> kg_construction.py

Since the amount of data to be processed was huge, multiprocessing is used to make the code efficient. Relevant functions with self-explanatory comments included in *kg_construction.py* and *util.py*

- Using *graph_movie_1step.txt* we first create *data_processed/data_freebase.txt* in the format :
  FB_item_ID*[\tab]*relation_name*[\tab]*FB_item_ID
  This involves removing links and just retaining the id's present for FB_item_ID.
  We format relation_name as entity_name.relation_name.entity_name
- We map all the entities and relations from this file and store it in entities/<entity_name>.txt and relations/<relation_name>.txt.
  - Each of these files store distinct FB_item_ID separated by newline character.
  - Using KB4Rec/ml2fb.txt we transform FB_item_ID to RS_item_ID.
  - We also create user.watched.movie relation which we populate using ml20m/ratings.csv
  - We again transform all the freebase ids into numerical ids starting with 0
- We only use relations which have occurrence frequency>10000. This can be modified to reduce/ increase density of out embeddings.
- We skip users who have watched less than or equal to 5 movies.
- The final data(with only indexes) that we use can be found in relation_indices/<relation_name>.txt

Training/ Test Data

- We use all relations and 70% of user.watched.movie relation as training_data
- We use remaining 30% of user.watched.movie relation as test_data

Running the code:

- All the data is loaded and stored in edicts defined in class MovieLensDataLoader -> kg_construction.py
- We create embeddings using code in train_embedding.py
- Evaluation of the embeddings is done using code in test_embedding.py

Results

The results are formatted as following:

```
Precision: train 0.45, test 0.10.
Recall: train 0.08, test 0.07.
AUC: train 0.92, test 0.91.
```

Results are stored in results.txt for all the algorithms

Baselines:

We ran multiple baseline models:

- CKE - TensorFlow implementation
  - We use ml20m/ratings.csv
  - Relevant code can be found in CKE/train_cke.py

- BPR - PyTorch implementation
  - We use ml20m/ratings.csv
  - Relevant code can be found in CKE/train_cke.py

- OpenKE - PyTorch implementation
  - We create the following files: run OpenKE/kg_construction_OpenKE.py to generate all these files except for type_constraint.txt
    - **Training**
      - entity2id.txt - *<FB_item_ID> index.* First line contains total number of entities.
      - relation2id.txt - <relation_name> index. First line contains total number of relations.
      - train2id.txt - <entity_1_index entity_2_index relation>. First line contains total number of triples for training.
    - **Testing**
      - test2id.txt - <entity_1_index entity_2_index relation>. First line contains total number of triples for testing.
      - valid2id.txt - <entity_1_index entity_2_index relation>. First line contains total number of triples for validating.

- type_constraint.txt - First line is the total number of relations followed by the type constraint for each relation. This file can be generated using OpenKE/*n-n.py*
  - For instance: relation 1268 has 2 head entities (13005, 12781) and 1 tail entity (12683).

```
1268    2       13005   12781
1268    1       12683
```

  - Run OpenKE/train_transe.py to generate embeddings
  - Run OpenKE/test_transe.py to get the results

Results
The results are formatted as following:

```
Precision: train 0.45, test 0.10.
Recall: train 0.08, test 0.07.
AUC: train 0.92, test 0.91.
```

Results are stored in results.txt for all the algorithms

RL-DQN model:

This was done using PyTorch based OpenAI gym implementation. The same convention as Baselines and Translational-Embedding is followed here as well.