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](https://github.com/harsh2kumar/movielens-transe/tree/master).

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](https://drive.google.com/drive/folders/1EbGfW3f4ly37RgfB5PbZFN1y4FmDNUs7?usp=sharing).

**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](https://grouplens.org/datasets/movielens/)**. 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](https://movielens.org/), [http://www.imdb.com](http://www.imdb.com/), and [https://www.themoviedb.org](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](https://drive.google.com/drive/folders/1Kb7KNyoYYF0QoSxXoEv-JRs8wfv3cClL?usp=sharing).

**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](https://github.com/thunlp/OpenKE)
    - 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.