

Data and Code Instruction

The resources mainly include three folders: data and preprocessing, representation learning and change detection, validation. This document provides a detailed introduction to the experimental steps of the research and the implementation process for tables and figures in the manuscript.

Running Environment:

- Python 3.8
- PyTorch 1.13

Framework:

The framework consists of three parts: data and preprocessing, representation learning and change detection, validation.

1.Data and Preprocessing

Running Code: KG2id.py

Input Data:

Located in *data and preprocessing/input*, including:

- CSV files storing **structured triples**
- TXT files storing **street view imagery(SVI) captions**
- JPG files storing very high resolution(**VHR**) **images**

Output Data:

Results are stored in *data and preprocessing/sample data*, including:

- **Multimodal data** vectors: **emb.npy**
- **Entity** mapping IDs: **entity2id.txt**
- **Relation** mapping IDs: **relation2id.txt**
- Split **training/validation/test** datasets: **train2id.txt, valid2id.txt, text2id.txt**

Description:

It converts knowledge graph from structured triples, text and image formats into entity and relation ID mappings, while splitting data into training/validation/test sets in preparation for subsequent model training and change detection.

2. Representation Learning and Change Detection

This section contains modules (1) and (2), which provide step-by-step instructions for reproducing the results in Table 3 and Figure 8.

(1)Representation learning framework:

Running Code: Train.py

Input Data:

Located in *data and preprocessing/sample data*, including:

- **Multimodal data** vectors: **emb.npy**
- **Entity** mapping IDs: **entity2id.txt**
- **Relation** mapping IDs: **relation2id.txt**
- Split **training/validation/test** datasets: **train2id.txt**, **valid2id.txt**, **test2id.txt**

Output Data:

Results are stored in *representation learning and change detection/outputdata*, including:

- entity2vector.pickle: entity embedding vectors
- relation2vector.pickle: relation embedding vectors

Description:

Train.py defines training parameters and evaluation metrics, conducts training based on knowledge graph data, and validates model accuracy (Table 3). The models compared with our proposed representation learning model in the table are all open-source. For easier reference, we have provided hyperlinks to these models.

It integrates three modules:

- readTrainingData.py: Reads training data
- generatePosAndCorBatch.py: Processes knowledge graph triples

- DCKRL.py: The proposed multimodal fusion representation learning framework based on dual cross-attention mechanism and TransE module

Result Analysis:

By running train.py, we can obtain the accuracy results of representation learning (Table 3).

(2) Change detection:

Running Code: Run emb.py first, then change_calculate.py.

Input Data:

Located in *representation learning and change detection/outputdata*, including:

- entity2vector.pickle: entity embedding vectors
- relation2vector.pickle: relation embedding vectors

Output Data:

Results are stored in *changerate.csv*, containing 102 columns:

- Grid entity similarities: Normalized cosine similarity in the first column are quantified on a 10-point scale (0-9)
- Vector representations: Columns 2 to 101 contain the vector representations of grid area
- Change rates: Change rates are quantified on a 10-point scale (0-9) in column 102, where each value corresponds to a 10% interval. Specifically, 0 represents 0-10% change, 1 represents 10-20%, and so on, with 9 representing the highest change rate interval of 90-100%.

Description:

- **emb.py**: Outputs knowledge graph vector representations to CSV format files for further change rate calculations
- **change_calculate.py**: Calculates normalized cosine similarity and change rate between grid entity vectors from 2017 and 2023, outputting results to changerate.csv

Result Analysis:

Calculates the degree of change for each grid between 2017 and 2023 running **change_calculate.py**. Results can be linked with Qinhuai district grid data in ArcGIS

to produce Figure 8.

3.Validation

This section mainly focuses on two parts: the impact of multimodal data on change detection and the comparison between knowledge graph-based change detection and traditional methods, corresponding to Figure 10 and Table 4 respectively.

Running Code: validate.py

Input Data:

validation/validation.csv, containing:

- **Change detection results** using **different data combinations**

- Expert annotation results: Data columns: 1. Results using **VHR** only; 2. Results using VHR+SVI; 3. Results using VHR+POI; 4. Results using VHR+POI+SVI; 5. Expert annotations (5-point scale); 6. Expert annotations (binary: 1 and -1)

Output Data:

Evaluation metrics including: *Accuracy, Precision, Recall, F1 score*

Description:

Run **validate.py** to validate change detection effects using multiple metrics.

Result Analysis:

Run validate.py to compare results using different data with expert annotations (column 5 in validation.csv) to obtain measurement results for various metrics. Import change detection results based on knowledge graphs with different data combinations into ArcGIS for visualization. Then create bar charts showing the count of each category. With these steps, Figure 10 is completed.

We run validate.py to compare the change detection results from various **traditional change detection models** with **the proposed knowledge graph-based change detection** results against the annotated results in column 6 of validation.csv. All traditional change detection models used for comparison are open-source. For easier reference, we have provided hyperlinks to these models. With these steps, Table 4 is completed.

