Ontology classification for YAGO2geo locations using Knowledge Graph Embeddings PROGRESS REPORT 2

Panagiotis Stasinos

October 2020

Contents

1 Introduction

In a few words for most of the implementation the steps from the paper were followed with only some small changes. The most significant of them is that all the locations were used (no pruning of non-populated was done).

The main idea to evaluate the embeddings was to test if the spatial proximity will be preserved in the dense latent space of the embeddings. That was tested by comparing the closest neighbors in the latent with those in the latitude-longitude space. The best percentage of common neighbors achieved was 40%.

The first idea for evaluation was to make a classification model that will predict the `OS_Category` of a location given. But unfortunately I couldn't make a model that has good performance. Best accuracy achieved was 60% that was about the percentage of the `OS_Category` with the most records.

2 Weighted graph

For the construction of the graph the locations were extracted from `OS_extended.ttl` and `OS_new.ttl` files from `yago2geo_uk\os\` folder.

2.1 Nodes

For every location a node is created (19,468 in total), and all the features of the location are stored there. From the dataset we have the name, ID, `OS_type` and the borders (given as a multipolygon) and from the borders we get the area and the center of each location.

2.2 Edges

Next step in the weighted graph construction are the edges and their weights.

Methods to calculate distance As the paper suggests for the graph to be meaningful the weights are calculated by the geodesic distance between two locations. To estimate such great-circle distance harvesine equation[1] is used. Two different approaches were tried,

[1] harvesine equation

- center distance, where center's latitude and longitude are used in the harvesine equation
- polygon distance, where the closest points between the boarders of the two locations are estimated and the latitude and longitude of these points are used in the equation

Method to find closest To avoid many calculations nodes that have better probability to be closer are used. To achieve that we compile two sorted lists of all nodes, one that is sorted by the latitude of the center point and one by the longitude. By using a sliding window, of constant size, over each of those lists, we calculate only the distances between the current location and the `size_of_window` locations that are under it in the list.

By estimating the edges like this we ensure that the walks will move in a South-North and West-East direction at each step.

One difference with the paper is that no pruning of non-populated locations is been done.

3 Weighted DeepWalk

Main idea of DeepWalk is that each random walk is like a sentence in natural language and can be given as input in a Word2Vec model.

3.1 Random walks

From each node a p k-step random walks are initiated. So the total number of "sentences" generated is `number_of_nodes*number_of_walks`.

To make sure that from each node the next step isn't always the same one, and we end up with the same walks, a custom distribution is applied from the weights of the edges.

To produce such a distribution a new dampened weight is computed

$$w' = max(1.0/ln(w), e)$$

Then we get these new weights and we `l1-normalize` them to get the distribution (divide by the sum). The paper suggests that 5 10-step walk give good results, but other values were tested too.

3.2 Embeddings

Then the corpus of random walks, from the step above, is used to embed all nodes in a latent fixed size dimensional space using skip-gram and CBOW embedding algorithms.

Then all vectors are saved in a csv file, where each line contains the name of the location (unique feature) and its vector.

We expect that the spatial proximity will be preserved in the dense latent space, so we can use cosine similarity in the embedding space like we would use harvesine similarity in the latitude-longitude space. That's the main idea that is used for the evaluation of the embeddings.

4 Cases tested

4.1 In the construction of the graph

4.1.1 distance type

Two methods tested.

- Center distance. The harvesine distance was used between the center points of the locations.
- Polygon distance. The area of the locations is given as multipolygon. So we find the closest points of the two polygons and estimate the geodesic distance between them.

In both cases `OS_topological.ttl` file from `yago2geo_uk\os\` folder was used, and for every subject-object, in touches and within RDFs, distance between them was changed and saved as zero.

4.1.2 sliding window size

values tested [10 ,30, 50 ,70]

Because the sliding window method was applied in both sorted by latitude and longitude lists 2*`size_of_window_neighbors` neighbors are kept for every location.

4.1.3 number of walks and steps

3 different combinations [(5,10), (10.5), (15,3)]

4.2 Embedding part

Two training algorithms tested, skip-gram and CBOW. Skip-gram gave better results and CBOW was close to 0% in spatial proximity. So more parameter tuning was done only in the skip-gram model.

Skip-gram parameters

- embedding vector size, dimensionality of the word vectors. [50, 100, 150, 200] tested, best 150.
- embedding window size, Maximum distance between the current and predicted word within a sentence. Window sizes tested [5, 10, 15, 20] and 20 gave best results (could also be the value of the length of the sentence (`number_of_steps`)).
- embedding min count, Ignores all words with total frequency lower than this. Values tested [5, 10] and 5 was the best.
- embedding noise words, specifies how many noise words should be drawn. [5, 10, 15, 20] tested and 5 gave best results.

5 Evaluation

Two methods were followed for the evaluation of the embeddings, one by testing the spatial proximity, and one by creating a classification model.

5.1 Neighbors

The embeddings must reflect spatial proximity so they can be used as feature vectors in a machine learning model.

So to see if the proximity is preserved the cosine similarity is used to find the 10 closest vectors for each location in the latent space, and compare them with the 10 closest neighbors in the knowledge graph.

The best percentage of common neighbors achieved 40%.

5.2 Classification

The dataset with the best percentage in the spatial proximity was used to produce a feature vector for every location. Then those vectors were used in a classification model trying to predict the `OS_type` of the location given. Best accuracy achieved was 60%.

But because of the dataset is not well balanced (about 11,000 of the 19,000 locations belong to the same `OS_category`, `OS_CivilParishorCommunity`) the max accuracy of the model is about same to the percentage that `OS_CivilParishorCommunity` category has.

5.3 Best Parameters

Best parameters for the graph:

- distance type = polygon distance
- sliding window size = 10
- number of steps and walks = 5, 10

Best embedding algorithm was skip-gram and the best parameters were:

- vector size = 150
- window size = 20
- min count = 5
- noise words = 5