Ontology classification for YAGO2geo locations using Knowledge Graph Embeddings PROGRESS REPORT

Panagiotis Stasinos

July 2020

Contents

1 Introduction **1**

2 Embedding part **2**
2.1 Weighted graph Construction 2
2.2 DeepWalk 3
2.3 Best Parameters 3

3 Classification part **8**

4 Polygon to Polygon distance **11**

1 Introduction

Graph embedding is a way of mapping the nodes or edges of a graph into a fixed length vector that captures the key features while reducing the dimensionality. Graph embeddings can be used in many applications such as KG completion, relation extraction, entity resolution and entity classification. In this project the algorithm suggested from Neural Embeddings for Populated Geonames Locations paper [1] is used on YAGO2geo [2] database. A weighted graph is constructed where each node represents a location and the edges

[1] paper link
[2] YAGO2geo site link

are the distances between adjacent locations. Then based on the neighbors of each location we create a vector of fixed length and store its label (the category in which it belongs).

The project was implemented in Python and the libraries used :

- Keras, used for neural networks
- Scikit-Optimize, used to find best hyperparameters and construct the final model
- Shapely, used to find distances, centers of polygons, surface area of location etc
- Pandas, numpy, Matplotlib etc

Link to the source code of project :
https://github.com/PanagiotisStasinos/Yago2geo/tree/master

2 Embedding part

2.1 Weighted graph Construction

For the project it was used the UK dataset (`yago2geo_uk\os\`). The locations were extracted from `OS_extended.ttl` and `OS_new.ttl` files and for each location a node was made. Then the locations are sorted based on their center's longitude and latitude (the center is estimated from the polygons given).

Then the `OS_topological.nt` file is used and for every couple of locations we store the distance between them as zero.

Then by using a sliding window we find the distances for each location with its closest in the 2 sorted lists (one for longitude and one for latitude). The distance is estimated from the center of one polygon to the center of the other. With this distance type, center to center, most of the testing has been done, but also polygon to polygon distance was tested with window size 11 (simplest case) to see if it has better results.

Parameters

- `window_size`
- `distance_type`
- locations included(more here)

2.2 DeepWalk

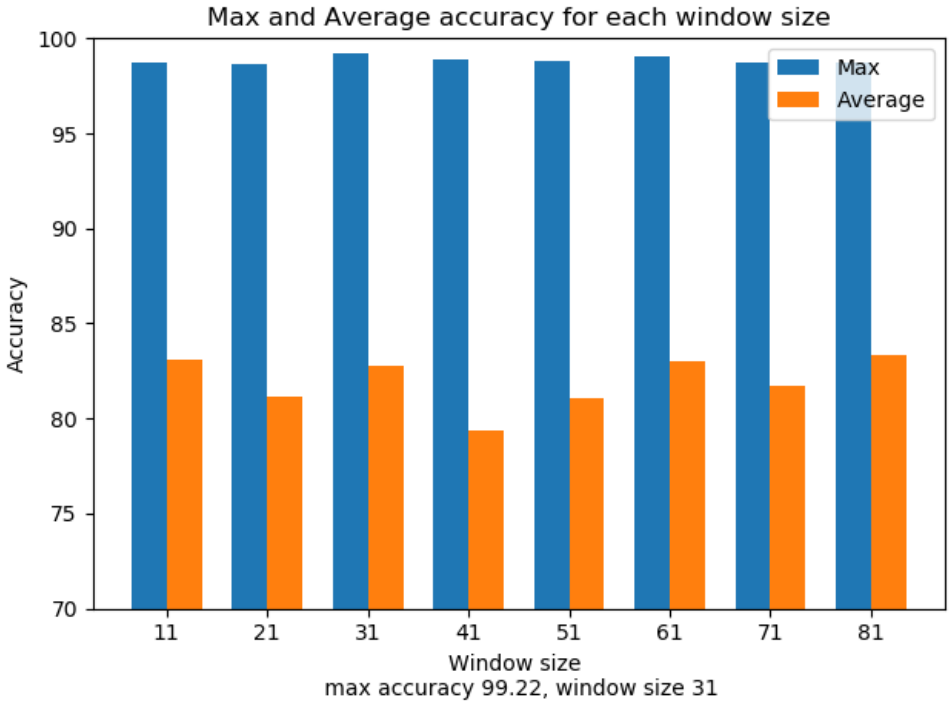
From each node a p k-step random walks are initiated. For each neihgbor we visit, we store the features we want in the current location's vector. The probability to visit a certain location from the current location is inversely proportional to the distance between them. Like so we construct a vector for each location and we also keep its category type to use it as label in the classification model.

Parameters

- Number of steps, Number of walks. Two cases were tested (2,10) and (10,5)
- features used, length of vector (more here)

2.3 Best Parameters

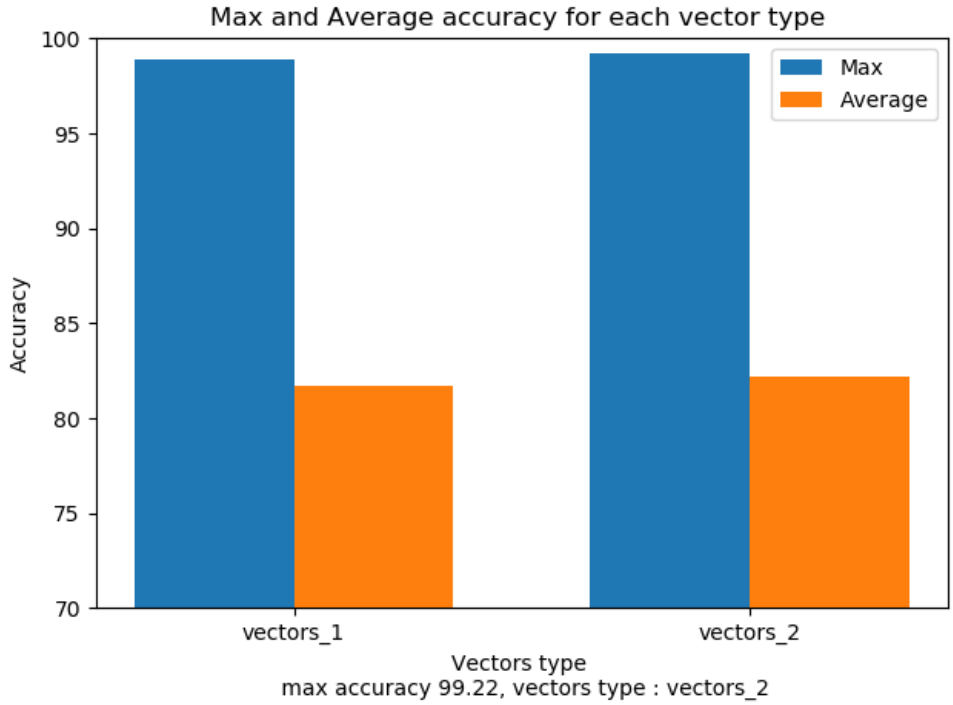
Window size. Sizes tested [11, 21, 31, 41, 51, 61, 71, 81]



Best accuracy achieved with window size 31. We notice that bigger window sizes, such as 61 and 81, achieve good accuracy too, but because to produce the vectors with these window sizes takes a lot of time smaller windows are preferred.

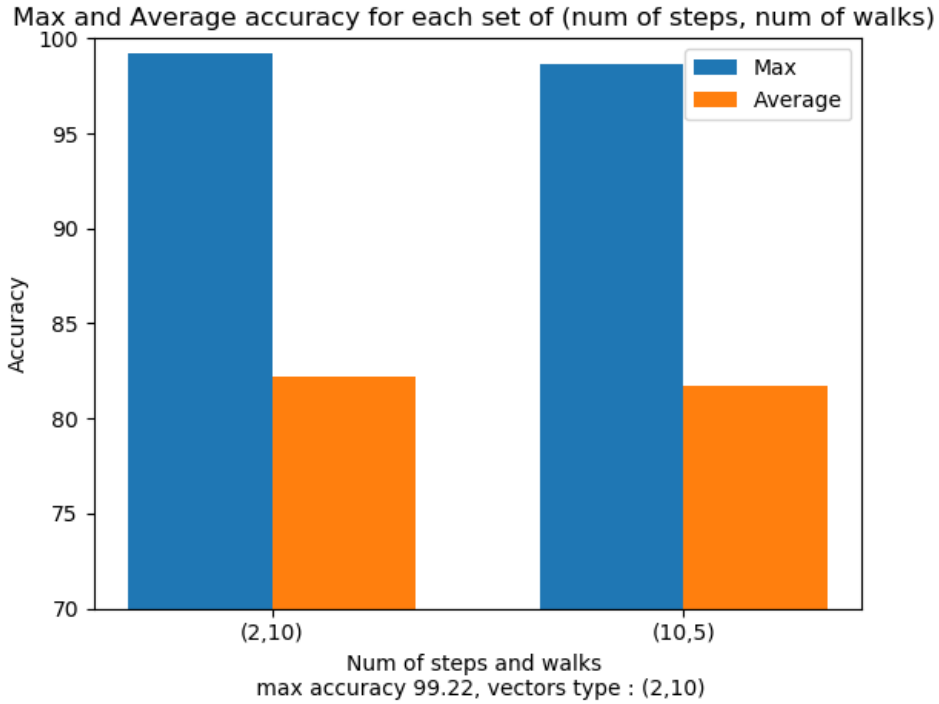
Locations included. To see if we can achieve better accuracy 2 different datasets were tested. In second case (`vectros_2`) the model was trained without including locations that their category only appeared once, because classes with very low frequency prevent the model to achieve high accuracy. In the first case (`vectros_1`) all the locations were used and to overcome this problem weights for each class were applied.

Two cases : `vectors_1` all locations were included and `vectors_2` `OS_GreaterLondonAuthority` and `OS_CCOMMUNITY` categories weren't used because they appeared only once.



Best accuracy achieved with the dataset without the 2 small classes, but there in no significant difference between the 2 cases.

Number of steps and walks. Two different combinations were tested (p,k)=(5,10) and (p,k)=(10,2)



Best accuracy achieved with 10 2-step walks.

Features used. From YAGO2geo dataset we get 4 features for each location :

- ID
- Name
- Polygon, that represents the boarders
- Category, 15 different values

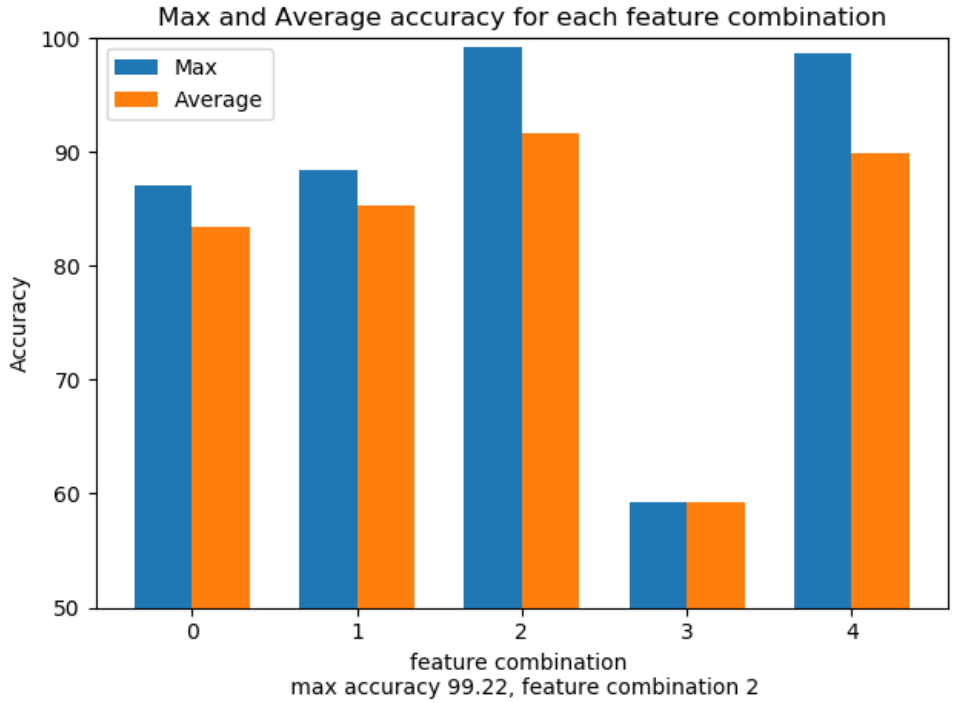
and 2 more features computed from polygon :

- surface area
- center's coordinates, latitude and longitude

To test which combinations of these features give better results 5 cases where tested :

- 0 : surface area
- 1 : surface area, category type

- 2 : surface area, category type, center
- 3 : surface area, category type, center, id
- 4 : surface area, center



From the chart we conclude that combination 2 gives the best results, the surface area, the category and the coordinates of the center are the most important features.

3 Classification part

The classification model predicts the category type of location given but the main idea was to use it to evaluate the vectors and choose the best parameters.

To implement the model keras library was used and scikit-optimize to find best hyperparameters.

The parameters of the vectors used in the final model :

- `distance_type` = `center_distance`
- `locations_used` = `vectors_2`

- `window_size` = 31
- (k, p) = (2,10)
- `features_used` = combination number 2 (and the length of vectors is 80)

Optimization. The code optimize the model was based to this code [3]. The hyperparametrs that were estimated and their values are :

- `learning_rate` = 0.0038113223272439388
- `num_dense_layers` = 4
- `num_input_nodes` = 348
- `num_dense_nodes` = 577
- `activation` = "relu"
- `batch_size` = 83
- `adam_decay` = 0.005689749519548906

Final model.

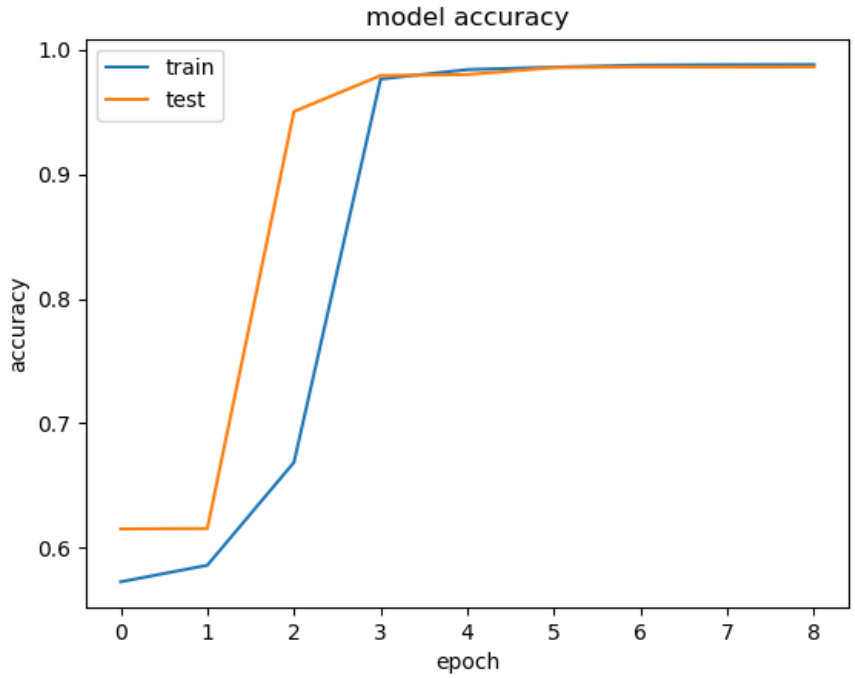
```
learning_rate = 0.0038113223272439388
num_dense_layers = 4
num_input_nodes = 348
num_dense_nodes = 577
activation = 'relu'
batch_size = 83
adam_decay = 0.005689749519548906

path = '../datasets/center_distance/vectors_2/window_size_31
    /2steps_10walks/data2_80.csv'
train_data, train_labels, input_shape = network.read_datasets
    .read_data(path)

callback = [
        tf.keras.callbacks.EarlyStopping(monitor='val_loss',
    patience=2),
        tf.keras.callbacks.EarlyStopping(monitor='
    val_accuracy', patience=2),
        tf.keras.callbacks.EarlyStopping(monitor='loss',
    patience=2),
```

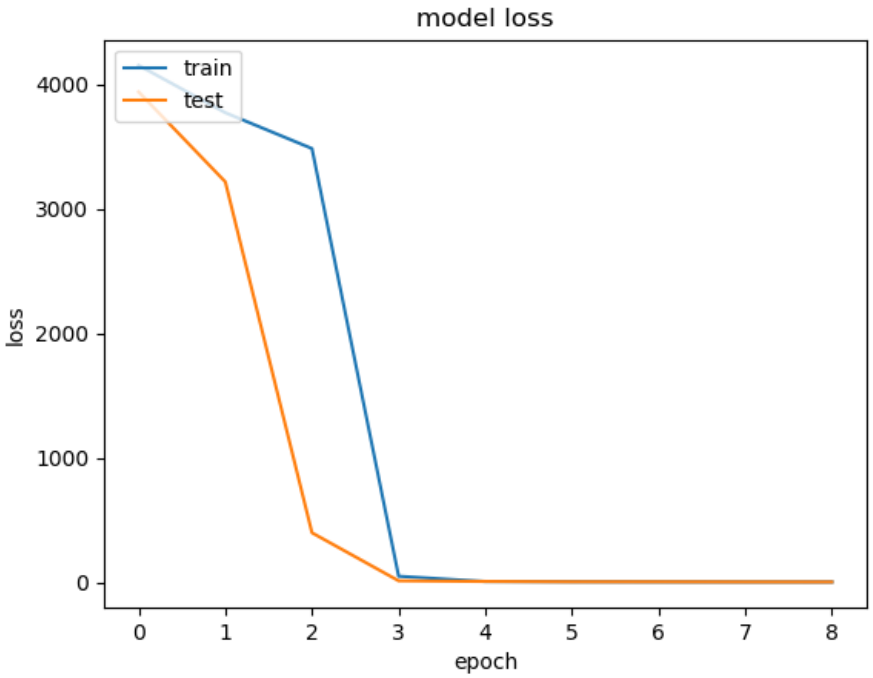
[3] optimization code github link

```
        tf.keras.callbacks.EarlyStopping(monitor='accuracy', patience=2)
    ]

model = Sequential()
input_shape = (input_shape,)
model.add(Dense(num_input_nodes, input_shape=input_shape, activation=activation))

for i in range(num_dense_layers):
    name = 'layer_dense_{0}'.format(i + 1)
    model.add(Dense(num_dense_nodes,
                    activation=activation,
                    name=name))

# add classification layer (15 categories) .
num_output_nodes = 15
model.add(Dense(num_output_nodes, activation='softmax'))

# setup our optimizer and compile
adam = Adam(lr=learning_rate, decay=adam_decay)
model.compile(optimizer=adam, loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

classWeight = network.read_datasets.get_classWeight(train_labels)

history = model.fit(train_data, train_labels,
            batch_size=batch_size,
            epochs=30,
            shuffle=True,
                class_weight=classWeight,
                validation_split=0.3,
                callbacks=callback,
                verbose=2)
```

Listing 1: Python example

Accuracy and Loss plots. Early stop was used in cases of training accuracy, training loss, validation accuracy and validation loss stayed the same or decrease for 2 epochs.


model accuracy
train
test
accuracy
epoch
1.0
0.9
0.8
0.7
0.6
0
1
2
3
4
5
6
7
8


model loss
train
test
loss
epoch
4000
3000
2000
1000
0
0
1
2
3
4
5
6
7
8

4 Polygon to Polygon distance

Distance type Two different ways to estimate the distance between 2 locations were tested, center to center distance and distance between polygons. The second way was tested only for the smallest window size because it takes a lot of time to estimate the distances.