**EXPLAINABLE GRAPH CLASSIFICATION - REPORT**

*Deepti Saravanan*

*School of Computer Science and Engineering,*

*NTU, Singapore*

1. **OBJECTIVE.**

Given a dataset of graphs belonging to two classes, we extract frequent subgraphs whose count across the graphs is greater than the minimum value specified, using gSpan algorithm. We embed these frequent subgraphs into vector space of high dimensions using the concept of subgraph2vec algorithm. We then form 2D-histograms from these embeddings upto n-channels for each subgraph embedding. Then we form image grids corresponding to the original graphs in the dataset by stacking the histograms of their constituent subgraphs respectively, where the pixel values are equal to the normalised sum of the vector values of the subgraphs falling in those particular pixels. These image grids thus formed for each graph is fed into the CNN for training and testing. The classification results are obtained, upon which the DeepLift model can be applied to extract the important features (subgraphs) that resulted in the graph being classified to that particular class.

**SPECIFICATIONS:**

* Ubuntu 16.04.2
* Python 2.7
* Tensorflow 1.1.0
* cuda 9.1
* slurm 17.11.5

**II. PROPOSED METHODOLOGY.**

The objective can be divided into three stages:

1.**gSpan** (to extract the frequent subgraphs).

***REFERENCE.*** *gSpan: Graph-Based Substructure Pattern Mining by X. Yan and J. Han.*

**Steps include:**

● DFS Subscripting

● DFS Code

● DFS Lexicographic Order

● Minimum DFS Code

● DFS Code Tree

**ADVANTAGES.**

* gSpan introduced lexicographic ordering and formation of DFS Tree using DFS Code.
* Combines the growing and checking of frequent subgraphs into one procedure, thus accelerating the mining process.

***NOTE.*** *Please refer to the gSpan PPT.*

***Requirements.***

* numpy 1.11.0

**INPUTS AND OUTPUTS.**

***main2.py*** (runs gspan over training graphs; subgraphs of test graphs are generated and matched against the frequent subgraphs obtained from training data to retain only the frequent subgraphs that are present in it.)

* ***algorithms.py*** gspan stepwise code.

***Input.*** Input dataset of graphs (text file), minimum support value(in the code).

***Output.*** Out.txt - DFS codes of the frequent subgraphs.

Shuffled\_graph\_idxs.npy - numpy array of the ids of graphs after shuffling.

id/train - IDs of training graphs (70%).

id/test - IDs of test graphs (30%).

extensions.npy - DFS codes in numpy array format (used in deeplift model).

support.npy - support values of the frequent subgraphs.

***dfs2g.py*** (converts dfs codes to graph text format similar to the initial dataset).

***Input.*** out.txt (DFS Codes).

***Output.*** textout/nodes - nodes information of each subgraph.

textout/edges - edges information of each subgraph.

***convert.py*** (subgraphs information in .csv format using the nodes and edges information).

***Input.*** textout/nodes and textout/edges.

***Output.*** csvout/\*

2. **Embeddings** (To create embeddings of the frequent subgraphs in high-dimensional vector space.)

***REFERENCE.*** *subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs by A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan.*

To learn Distributed Representations of Rooted Sub-graphs from Large Graphs using graph2vec algorithm (an unsupervised representation learning technique to learn latent representations of rooted subgraphs present in large graphs.)

That is, for a set of graphs G = {g1,g2,...} and a given D, a vocabulary for all the rooted nodes around every node for every graph is to be extracted which includes the neighbours upto the mentioned degree( 0 <= d <= D) such that sgvocab = {sg1,sg2,..}.

***APPROACH.***

● Subgraph2vec considers all the rooted subgraphs (up to a certain degree) of neighbours of r as the context of sg(d) r, for all r in G with root r.

● Skipgram modiﬁcation - considers radial context instead of ﬁxed linear context.

***NOTE.*** *Please refer to the subgraph2vec PPT.*

***Requirements****.*

1. *tensorflow (version == 1.4.0)*
2. *networkx (version <= 2.0)*
3. *scikit-learn (+scipy, +numpy)*

**INPUTS AND OUTPUTS.**

***final.js*** (converts the csv graph files to .gexf extensions for embedding).

***Input.*** csv/\*

***Output.*** results/\* - .gexf files of subgraphs.

***main.py*** (Embedding algorithm)

* ***utils.py*** functions to read the input files and save the final graph embeddings.
* ***skipgram.py*** skipgram model functions.
* ***make\_graph2vec\_corpus.py*** creates the corpus particular to that class.
* ***corpus\_parser.py*** function that scans and reads the corpus created and function that generates batches of files for training the skipgram model.
* ***train\_utils.py*** trains the skipgram model to create the final embeddings.

***Input.*** data/input/test/\* - the .gexf files of subgraphs

***Output.*** embeddings/\* - final .gexf files of embeddings of each subgraph

3. **Image Grids Formation** (To create 2D histograms and stack them up to form image grids for each main graph.)

***REFERENCE.*** *Graph Classification with 2D Convolutional Neural Networks A.J.P. Tixier, G. Nikolentzos, P.Meladianos and M. Vazirgiannis.*

Represents graphs as multi-channel image-like structures that can be classified into the corresponding categories by the vanilla 2D CNNs.

**ADVANTAGES.**

* By converting all graphs in a given dataset to representations of the same dimensionality, and by using a classical 2D CNN architecture for processing those graph representations, this method offers constant time complexity at the instance level, and linear time complexity at the dataset level.
* In a 2D CNN classifier, features are learned directly from the raw data during training to optimize performance, unlike SVMs.

***NOTE.*** *Please refer to the graphs to image grids PPT.*

***Requirements.***

* numpy 1.11.0
* sklearn
* keras 1.2.2

**INPUTS AND OUTPUTS.**

***train\_new.py*** (Concatenates the vector values of the subgraphs present in the original training graph and converts it into a numpy array, using the information from the id folder obtained from the gspan part.)

***Input.*** embeddings/\* - the .gexf embedding files of subgraphs.

***Output.*** datasets/ten/train/\* - tensors of original training graphs (70% of initial dataset) in the form of numpy arrays - input to form histograms of each training graph.

***test\_new.py*** (Concatenates the vector values of the subgraphs present in the original test graph and converts it into a numpy array, using the information from the id folder obtained from the gspan part.)

***Input.*** embeddings/\* - the .gexf embedding files of subgraphs.

***Output.*** datasets/ten/test/\* - tensors of original test graphs (30% of initial dataset) in the form of numpy arrays - input to form histograms of each test graph.

**III. VALIDATION.**

The important features(subgraphs) for each class is obtained using the deeplift model. The support values of these subgraphs are observed and the minimum of all these values picked. Gspan is run using this minimum support value and embeddings and grids obtained subsequently. It is observed that it increases the accuracy to a good extent, thus confirming that the subgraphs identified as important features contribute to the increase of accuracy.

**IV. FUTURE SCOPE.**

The deeplift model can be extended to higher layers to extract higher level features(combination of subgraphs) for better accuracy and validation.