H-Index Prediction

Team BMV : Bertrand PATUREL[*] , Martin BRICENO[*] , Vanessa CHAHWAN[*]
[*]École Polytechnique
bertrand.paturel@polytechnique.edu, jesus-martin.briceno-anicama@polytechnique.edu, vanessa.chahwan@polytechnique.edu

Abstract

The treatment of data structures such as graphs and texts is a long and complex process considering that there are several methods to extract features from them and that it is not easy to know which of these are suitable for a subsequent task such as classification or regression.

In this article we will focus on our methods to extract features made in our H-index prediction process. We will also show the rest of the prediction chain such as regression and post-processing. Finally, this treatment allowed us to be in 2nd place on the public leaderboard with a difference of 0.028 in score from the first place. You can find our model on *GitHub*.

1 Introduction

H-index is an indicator suggested in 2005 by Jorge Hirsch with the objective of characterizing the quality, relevance and quantifying the scientific production of a researcher. This indicator depends on two parameters:

- The number of articles published by the researcher
- The number of citations of each article

A scientist has "index h if h of his or her Np papers have at least h citations each and the other (Np-h) papers have h citations each" [1].

In this challenge we seek to predict the H-index of an author having as information:

- A graph indicating which authors have published a paper together.
- The abstracts of several papers in a database.

Since we cannot process the raw material described above, then our goal is to extract information (i.e. features) from these data structures using Natural Language Processing (NLP), graph mining and Deep Learning techniques in order to generate a model that correctly predicts the H-index according to the mean absolute value metric.

We tested several models such as Graph Neural Networks (GNN) [2], Graph Auto-Encoder (GAE) [3], Bidirectional Encoder Representations from Transformers (BERT) [4] and Multi Layer Perceptron (MLP). The most performing solution was the one in which we, independently from the H-index, extracted features and finally give them to a regressor model. The steps of this model are shown below, Figure 1:

- We created 4 other graphs (weighted co-authorship graph, author similarity graph, paper graph and paper similarity graph) from the original graph and the embedding of the abstracts obtained with Doc2Vec [5].
- We extracted features from these graphs such as: degree, average neighbors, core number, page rank, betweenness centrality, eigenvector centrality, clustering coefficient and the number of papers.
- We concatenated all these new features with the embeddings of the authors, obtained by adding the embedding of the abstracts, and the embeddings of each of the graphs, obtained with Node2Vec.
- We apply a Light Gradient Boosting Model (LGBM) regressor on these features and test our performance by cross-validation using MAE as the loss.
- We take the nearest integer value of the regressor as the predicted value and we bound it.

In the following sections we will first discuss the preprocessing performed on the data to obtain the new graphs as well as the embeddings, then we will indicate some of the models used in addition to our best model to make our prediction. After, we will detail the regularization used to prevent the overfitting phenomenon. Next, we will see the post-processing step where we correct aspects of our prediction such as the rounding to an integer or the suppression of negative values. Finally we will show our results and conclusion indicating some improvements or variants on the different processing steps.

2 Processing and features

In this part we will expose our different authors' representation and papers' embeddings: both authors and papers are represented by vectors.

2.1 Paper representation

Since we have the abstract of the papers, one approach would be to represent them by vectors. Therefore we need to learn

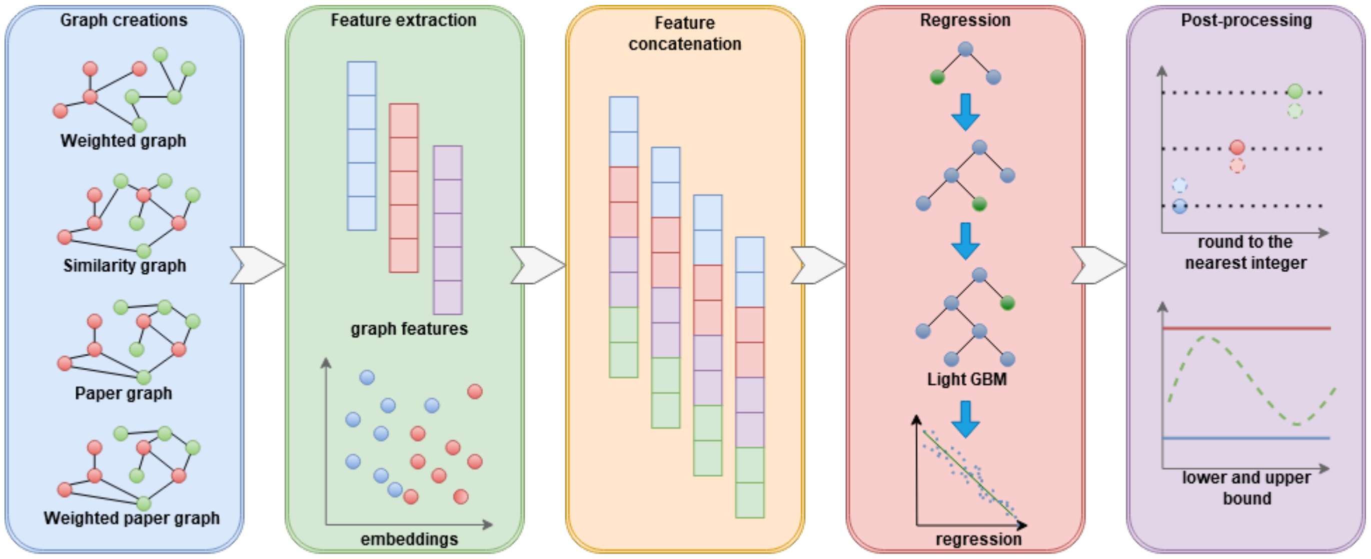

Figure 1: **Main model:** We show the main stages of our main model: First we have the creation of graphs from the authors and papers. Then we perform the extraction of features from these graphs such as core number, page rank, betweenness centrality, etc. We then concatenate these features to train our regression algorithm *Light GBM* which will produce a model that will predict new h-index values. These predicted values will be post-processed with respect to the h-index definition and to reduce the loss.

the embeddings of each abstract. In order to do that, we used Doc2Vec with the Distributed Memory Model of Paragraph Vectors (PV-DM) model and the Distributed Bag of Words version of Paragraph Vector (PV-DBOW). For the PV-DM, the paragraph vectors are obtained by training a neural network on predicting a center word based on an average of both context word vectors and full paragraph's vector. Whereas for the PV-DBOW, the paragraph vectors are obtained by training a neural network on predicting a target word just from the full paragraph vector.

2.2 Author embedding

Using average
A first representation of authors (i.e. vectors) is the mean of their papers' vectors. Yet, the problem is that each paper has the same influence and weight for their author.

Using weighted average
Then a better idea was to add a weight for each paper measuring its number of author. A paper with many authors would have a stronger weight than a paper with less authors. The basic idea is that, a paper with a lot of authors would have more chance to be cited in other papers.

Using sum
The idea is that each author's vector would be the sum of his papers' vectors. A paper's vector with high coefficients could take the lead while not being relevant for its h-index. As mentioned, a problem is that each paper has the same influence. We finally used this paper embedding.

Using weighted sum
Then we add a weight for each paper measuring its number of author. A paper for an author could have as weight the number of its authors. Then, a paper with many authors would have a stronger coefficient than the other.

An author is represented by its most popular paper
Another idea that we implemented is to encode each author's vector as his paper's vector with the biggest number of co-authors.

2.3 Graphs Creation

Since we have a lot information concerning the authors, their papers and the abstracts of their most cited papers, we decided to use these information and create multiple graphs:

- A weighted co-authorship graph *WG*
- An author similarity graph *SG*
- A paper similarity graph *WPG*
- A paper graph *PG*

Weighted Co-authorship Graph
We have for each author the list of the most cited papers, we can therefore reconstruct a new weighted co-authorship graph *WG* where nodes correspond to authors and two nodes are connected by an edges with a weight equal to the number of times they have co-authored a paper.

Author Similarity Graph
We have obtained author embeddings from the papers' abstracts, we can therefore use these embeddings to create an author similarity graph *SG* where nodes correspond to authors and two authors that have co-authored together are connected by an edge with a weight equal to the cosine similarity of the embeddings of the two authors.

Paper Similarity Graph
Since we have the embeddings of each paper, we use these vectors to create a graph where the weight of each edge corresponds to the cosine similarity of the embeddings of the papers forming this edge. Given the huge number of nodes

(papers), we only kept those edges that were numerically significant (i.e. greater than the mean). The intention is that this new graph *WPG* will show the semantic relationship between the papers by the weight of their edges and thus be able to extract features from it as we will see later.

Paper Graph
Following the same idea and the same process of creating the previous graph, we also created this graph *PG*. The objective now is only to know if 2 papers were related or not, so it is not necessary to have a weighted graph.

2.4 Graph Features

From the given co-authorship graph and the other created graphs, we can extract graph based features such as the degree, the average neighbors' degree, the core number, the PageRank, the betweenness centrality, the clustering coefficient and the eigenvector centrality.

Core Number
In order to define the core number of a node, we need to first define what is a k-core. The k-core is the largest subgraph in which every node has a degree of k within the subgraph. Then the core number of a node is the largest integer k of a k-core containing that node. This measure gives us an idea of the importance of an author in a given set, i.e., with respect to the set of its neighbors sharing the same core number.

Page Rank
The PageRank of a node is a ranking of the node based on the structure of the incoming relationships and the importance of the corresponding source nodes. Therefore it measures the importance of the node within the graph. Finally, the PageRank is an important feature for our task because it is correlated with the h-index due to the fact that authors that are linked to many other authors will have a high number of written papers and a high pagerank value and therefore they will eventually have a high h-index.

Eigenvector Centrality
The eigenvector centrality of a node is a measure of its level of influence in the graph. It is relative to the number of connections a node will have to other nodes. Moreover, connections to high-scoring eigenvector centrality nodes contribute more to the score of the node than equal connections to low-scoring eigenvector centrality nodes. Finally, the PageRank is a variant of the eigenvector centrality and therefore it is an appropriate feature for the same reason.

Betweenness Centrality
The betweenness centrality of a node is a centrality measure in a graph based on shortest paths. In other words, if a node lies in many shortest paths between two others nodes, then it will have a high betweenness centrality score.

Clustering Coefficient
The clustering coefficient of a node is a measure of the likelihood that its neighbours are also connected. It can be quantified by counting the number of possible triangles through the node that exist. If there exists a triangle then probably in our task the three authors worked together on the same paper.

2.5 Graph Node Embeddings

In order to create features for the regression task, we decided to use learn node embeddings by using methods such as DeepWalk [6], Node2Vec [7], Graph Autoencoders (GAE) [3] and Graph Neural Networks (GNN) [2]. DeepWalk, Node2Vec and GAE are unsupervised methods whereas GNN is a supervised method.

DeepWalk simulates a series of random walks to learn representations by treating the walks as sentences. Node2Vec simulates a family of biased random walks which efficiently explores diverse neighborhoods. GAE generalizes autoencoders to graphs: they reconstruct the adjacency matrix at the output given the adjacency and node features as the input. Finally, GNN learn features by inspecting neighboring nodes through message passing layers then these features are passed to regression layer that predicts the h-index.

2.6 Papers Features

Since the H-index is linked to the number of written papers for an author, and since we have for each author the list of its most cited papers, we can therefore computer the number of written papers for each author.
We can also compute, for each author, the average of the number of written papers of its co-authors.

2.7 Preprocessing

As we don't have the abstracts of all the papers and therefore an author's papers may all be with no abstract, we may have an author with no representation from the papers' representations. In order to solve this problem, we can fill the embeddings with nan values, and do some imputations such as filling all nan values with zeros or replacing missing values with the mean, or even keeping the nan values to be managed by the regression model itself.

Since the distribution of the h-index is skewed, we will use a tool called TransformedTargetRegressor from the Scikit Learn library. It will apply a non-linear transformation to the targets, increasing the accuracy of our regressor.

3 Main models

After testing all the methods that we have previously presented, we got several models by combining the following blocks:

- **D2V(2):** Two author representations obtained by applying first on the abstracts of all papers *(abstracts.txt)* the Doc2Vec with both PV-DM and PV-DBOW with a dimension of 64, a window size of 5 and ignoring all words with total frequency lower than 2 *(paper_representations.py)*. Then for each author, we sum the representations of all his papers to get finally authors' representations *(author_representations.py)*. After obtaining these papers' and authors' representations, we can then create the graphs that we described before *(create_graphs.py)*.
- **GF:** Graph features *(graph_features.py)* such as:

  - The degree, the average neighbor degree, the core number, the pagerank and the betweenness centrality of the nodes of graph *G*.
  - The degree, the average neighbor degree, the pagerank, the betweenness centrality and the clustering coefficient of the nodes of the graph *WG*.
  - The degree, the average neighbor degree, the pagerank, the clustering coefficient and the eigenvector centrality of the nodes of the graph *SG*.
- **PF:** Papers features *(papers_features.py)* such as:
  - The number of written papers for each author.
  - The average of the number of written papers of an author's co-authors.
- **GAE(SG):** Graph Autoencoder embeddings of dimension 64 with the author representations obtained with PV-DM and the adjacency matrix of the graph *SG* as input.
- **DW(G), DW(WG) and DW(2):** Deepwalk embeddings of dimension 64, obtained with the Skip-gram model with 20 walks of length 15 and with a window size of 5 applied respectively on the graphs *G*, *SG* and both.
- **N2V(2) and N2V(4):** Node2Vec embeddings of dimension 64, obtained with the Skip-gram model with 10000 walks of length 15, with a window size of 10, and with p=1 and q=0.5, for respectively both the graphs *WG* and *SG* and for 4 graphs *WG*, *SG*, *PG* and *WPG (node2vec.py, author_node2vec.py and preprocessing_node2vec.py)*.
- **GNN(SG):** Graph Neural Network with a regression layer with the author representations obtained with PV-DM and the adjacency matrix of the graph *SG* as input.

Morever, we filled nan values with zeroes and used TransformedTargetRegressor.
The models that we obtained by combining these blocks are all presented in the table 1. Following the concatenation of the features of the embedding spaces we implemented a Light Gradient Boosting Model for the regression task.
As we can notice from the Results section (table 1), the model that gave the lowest MAE score was the model: D2V(2) + GF + PF + N2V(4).

4 The impact of tuning

In this section we will show the impact of tuning on our last layer. Our last layer, i.e. our layer that deals with all the embedding vectors, the vectors encoding papers, authors, and our graphs, to make the regression task. Our last layer is a LightGradientBoosting, i.e. an eXtremeGradientBoosting algorithm with special trees as weak classifiers.

4.1 Gridsearch Implementation

Our first idea for tuning consists in searching through a grid of parameters for a regularisation of our weak classifiers. We tuned an Elastic Net regularisation using both Ridge and Lasso regularisation. Our idea was to reduce the overfitting phenomena on both the public and private set. Our algorithm could have learned random noise within our training set. We also tuned other parameters to find the right complexity of our LightGradientBoosting model.

4.2 Hyperopt Implementation

Then we decided to implement tuning within a smarter scheme. Our second idea was not to go through all the possibilities but to search as regards to our decreasing score : the Mean Absolute Error (MAE). Our tuning was much more rapid since all the possibilities were not explored.

For instance, if you want to explore for the Elastic Net within the gridsearch case, and that we wish to test 50 values for both Lasso and Ridge coefficients, the gridsearch will test $50 * 50 = 2500$ models.

4.3 Regularization

For this part let's defined our objective to minimize while tuning our model as:

$$\mathrm{Obj}(\Theta) = L(\Theta) + \Omega(\Theta) \tag{1}$$

where $L(\Theta)$ is our loss (here the MAE) defined below:

$$L(\Theta) = MAE(\Theta) = \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{n} \tag{2}$$

If we defined a tree f_t of Q leaves such as $f_t(x) = w_{q(x)}$ and $w \in R^T$ are the values of each leave and $q : R^d \to \{1, 2, \ldots, Q\}$ indicates whether each leaf x should be activated, then the regularization term noted $\Omega(\Theta)$ on a LightGradientBoosting regressor for each tree is:

$$\Omega(f_t) = \frac{1}{2}\lambda \left(\sum_{j=1}^{Q} w_j^2\right) + \alpha \left(\sum_{i=1}^{Q} |w_i|\right) + \gamma Q \tag{3}$$

where :

- α is the L1 coefficient
- λ is the L2 coefficient
- γ is the regularisation coefficient for the leaf number

If α is large then we might have leaves predicting zeros (more w_j set to 0). If λ is large then we might have leaves with smaller values for its leaves. If γ grows, the number of leaves in the trees will drop.

4.4 Tuning results

For the LGBMRegressor (which is only a special type of xgboost), which corresponds to our last model on Kaggle, the following features were tuned by hyperopt :

- The boosting type used is DART. DART differs from the usual gradient boosting and incorporates some dropouts within the boosting scheme to lower dependencies between weak learners.
- The colsample bytree which is the ratio of variables used to constructed each weak learners was tuned to 0.936.
- The learning rate is 0.088, this rate is used to reduce the number of leaves of each weak learner.

- The max depth of a weak tree is 10.
- The number of estimators is 100000 (weak learners).
- The number of leaves for each tree is maximum 32.
- The coefficient for the Ridge regression is 0.788.
- The coefficient for the Lasso regression is 0.961.
- The subsample training coefficient is 0.904.

5 Post-processing

At this stage we have already obtained the predictions of our model, however we noticed that these predictions are real values and in some cases are negative.

5.1 Nearest integer

Since the H-index is a non-negative integer value, we are going to round this number to the nearest integer and to bound it down in order to reduce the loss.

5.2 Lower and upper bound

We arbitrarily choose to bound it with a value of 1 and not 0 since the authors have at least 1 article in our database.
On the other hand, from the definition of H-index we can deduce that it is upper bounded by the total number of published papers. This is not a parameter that we always know, however, from the file *author_paper.txt* we can deduce that if the number of papers published by an author is less than 10, then this is his true number of papers published in total and that this represents the upper bound of the H-index for those authors. Then if the value of H-index predicted by our model for those authors who satisfies this condition is greater than their number of published papers, we can rewrite their H-index by the author's number of papers to reduce the loss.

6 Results

We show in table 1 our cross-validation scores (CV Score) for each model using the same LGBMRegressor model with the same hyperparameters:

Model	CV Score
GNN(SG)	4.835
D2V(2) + GF + PF	3.247
D2V(2) + GF + PF + GAE(SG)	3.250
D2V(2) + GF + PF + DW(2)	3.239
D2V(2) + GF + PF + N2V(2)	3.191
D2V(2) + GF + PF + N2V(4)	**3.156**
D2V(2) + GF + PF + N2V(4) + DW(G)	3.168
D2V(2) + GF + PF + N2V(4) + DW(WG)	3.164
D2V(2) + GF + PF + N2V(4) + DW(2)	3.162

Table 1: H-index predictions results

Our main method is to add information to our model in order to lower our score.
Moreover, we can note that the results that we obtained for comparison are not computed with the best hyper parameters of LGBMRegressor that we found due to long execution time.

7 Conclusion and improvements

7.1 Conclusion

The performance of our model is due to the quality of the embedding spaces. Our effort to find the best representative vectors of the papers, authors and graphs gave us the ability to perform well during this contest. For instance the Node2Vec embeddings of the created graphs, mixed with other vectors allowed us to lower our Kaggle score to 3.01 which is the second best score of the competition on the public leaderboard. Moreover, thanks to the work done on the embedding spaces we were able to avoid overfitting. The key thing is that, the process to resume a graph or a document into a vector has been done without the information of H-index. It is only in the last layer, once the vectors were produced we have trained a LGBMRegressor to link the inputs to their respective H-index.

7.2 Improvements

Our ideas of improvements could be to stack different models to forecast the H-index once the embedding spaces are obtained. For instance we could add to our model the results of a Graph Neural Network trained to learn the H-index or one of its layers. Moreover, we could train the abstracts' texts with BERT to obtain better embeddings.

References

[1] J. E. Hirsch. An index to quantify an individual's scientific research output. *PNAS*, 102(46):16569–16572, November 2005.

[2] Jure Leskovec Keyulu Xu, Weihua Hu and Stefanie Jegelka. How powerful are graph neural networks? *7th International Conference on Learning Representations*, 2019.

[3] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[4] Kenton Lee Jacob Devlin, Ming-Wei Chang and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. page 1188–1196, 2014.

[6] Rami Al-Rfou Bryan Perozzi and Steven Skiena. Deepwalk: Online learning of social representations. *arXiv preprint arXiv:1403.6652*, 2014.

[7] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *arXiv preprint arXiv:1607.00653*, 2016.

8 Appendix

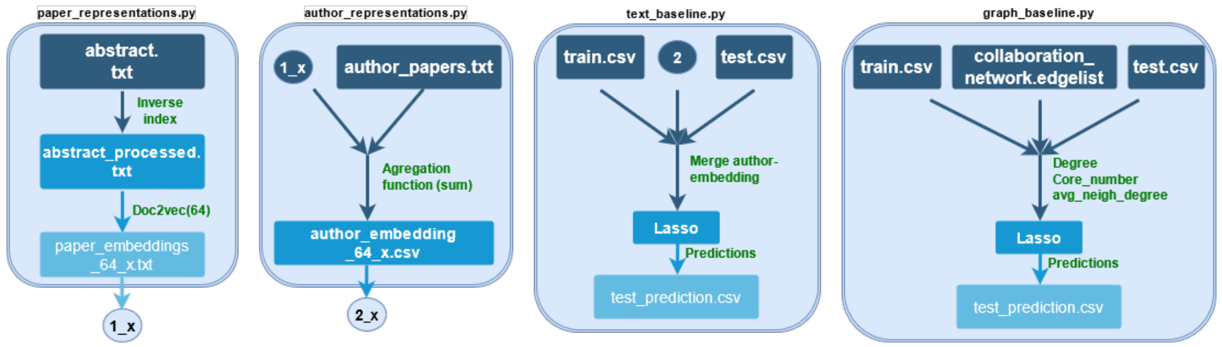

Figure 2: Embedding creation and basic models. ”x” means ”dm” or ”dbow”, so we will obtain 2 files.

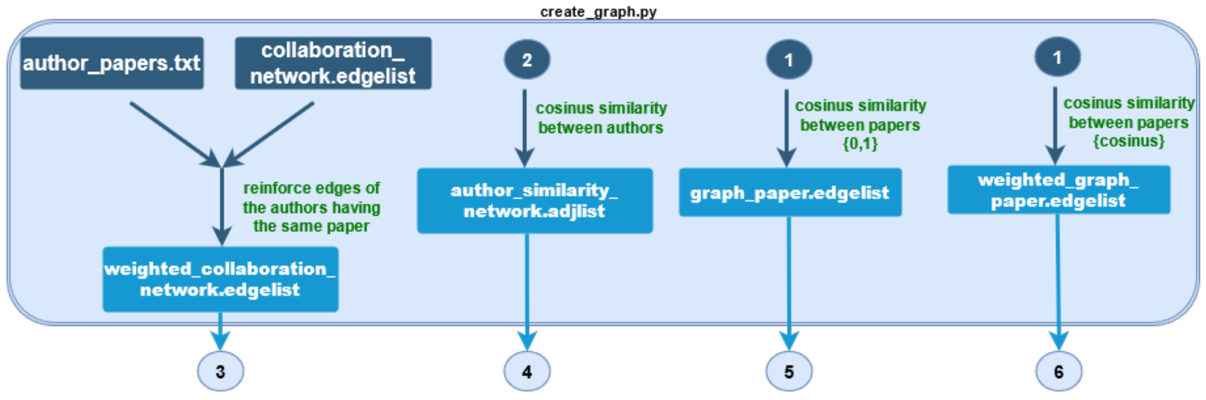
create_graph.py
author_papers.txt
collaboration_network.edgelist
reinforce edges of the authors having the same paper
weighted_collaboration_network.edgelist
2
cosinus similarity between authors
author_similarity_network.adjlist
1
cosinus similarity between papers {0,1}
graph_paper.edgelist
1
cosinus similarity between papers {cosinus}
weighted_graph_paper.edgelist
3
4
5
6

Figure 3: Graphs creation

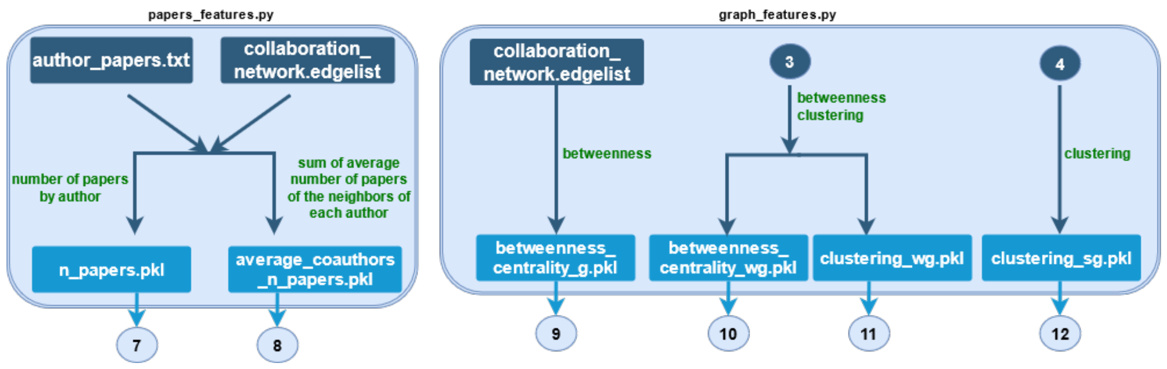
papers_features.py
author_papers.txt
collaboration_network.edgelist
number of papers by author
sum of average number of papers of the neighbors of each author
n_papers.pkl
average_coauthors_n_papers.pkl
7
8
graph_features.py
collaboration_network.edgelist
betweenness
3
betweenness
clustering
4
clustering
betweenness_centrality_g.pkl
betweenness_centrality_wg.pkl
clustering_wg.pkl
clustering_sg.pkl
9
10
11
12

Figure 4: Feature extraction from graphs

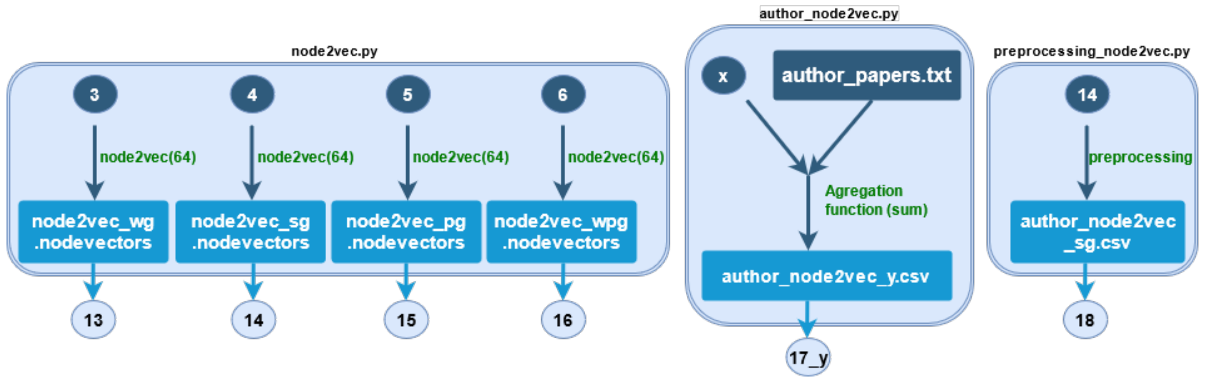

Figure 5: Embedding extraction from graphs. ”x” takes 15 or 16 as values while ”y” takes ”pg” or ”wpg”.