Performance of Whole Graph Embedding Algorithms on Real Networks Database

Lili Kata Szakács
Advisor: Ferenc Béres, András Benczúr

Project Work III.
2022/23 I. semester

1 Graph Embedding Algorithms

Graph embedding algorithms aim to assign low-dimensional Euclidean vectors to graphs such that the representation of structurally similar graphs would also be close to each other in the embedded plane. Moreover, it is a natural expectation that the embedding would be invariant to the permutation of nodes.

Two main approaches in embedding are some type of spectral analysis of the Laplace matrix (or some other special matrix) of the graph or the aggregation and mapping of structural properties via random walks.

Embedding a graph can be useful when it is the input of a machine learning task and usual representations (e.g. adjacency matrix) would not be effective due to size.

1.1 Results from previous semesters

1.1.1 Project I: Runtime of Embedding Methods

Using the results of my first project work, we left out the slowest embedding algorithms in this project: IGE[1], GL2Vec[2], FGSD[3] and GeoScattering[4]. NetLSD[5] and SF [6] were used even though they were quite slow. For the latter, we know it is a cause of the computationally costly problem of finding eigenvalues and eigenvectors.

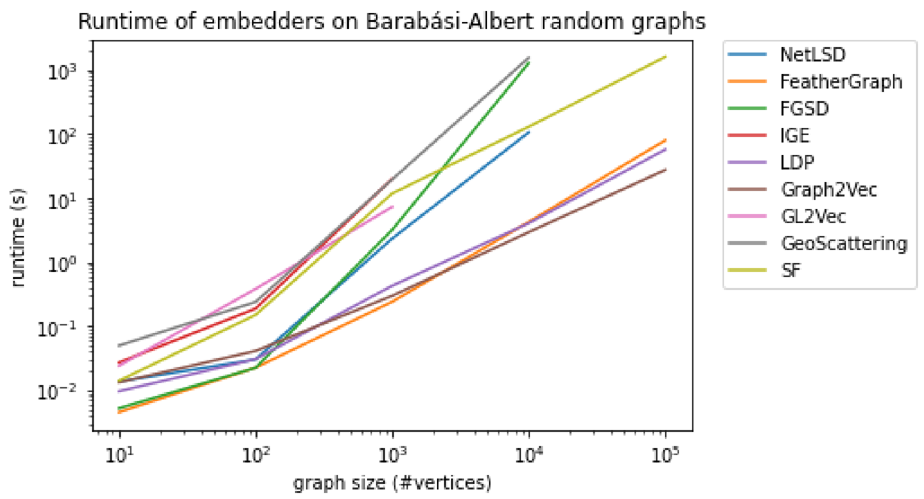


Figure 1: Runtime of Graph Embedding Algorithms – Project I. results

1.1.2 Project II: Performance on Classification of Random Generated Graphs

In the second project, we measured the performance of different classification tasks of randomly generated graphs. Since *FGSD* and *GeoScattering* were crossed out because of runtime issues, we had high hopes for the following well-performing candidates: *LDP*[7] and *FeatherGraph*[8].

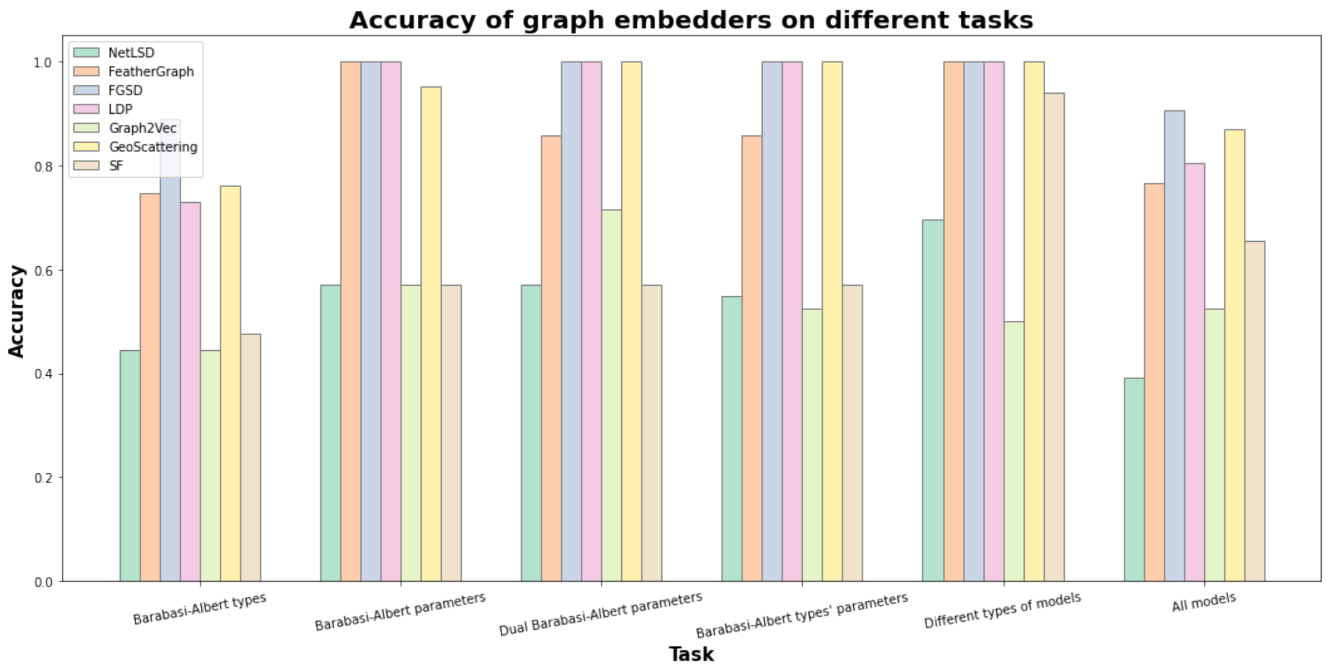


Figure 2: Accuracy of Graph Embedding Algorithms – Project II. results

2 Measurements

Embedding algorithms can be thought of as a preprocessing method on graph-like data before machine learning tasks (e.g. classification) and can be useful in the analytics of real-life networks. It is important that these algorithms keep the diversity of network properties in their output vectors, such that the different types or functions can still be distinguished.

In this semester, I examined the performance of embedding algorithms on a dataset of real networks from six domains in a domain classification problem. I experimented with various parameter setups (A) of five embedding algorithms from the *KarateClub*[9] Python package and two different classification methods to identify the class labels of the input.

2.1 Dataset and Motivation

The dataset of *M. Nagy* and *R. Molontay* [10] consists of 500 graphs, gathered one by one from online databases, whose self-loops were removed and were treated as undirected, unweighted graphs. These networks belong to six different domain, as shown in Table 1, and their sizes cover a wide range from small graphs to networks with thousands of nodes.

Domain	**Description**	**Range of network size**	**Number of networks**
Brain	Human and animal connectomes	50 - 2,995 (avg: 946)	100
Cheminformatics	Protein-protein (enzyme) interaction networks	44 - 125 (avg: 55)	100
Food	What-eats-what, consumer-resource networks	19 - 1,500 (avg: 118)	100
Infrastructural	Transportation (metro, bus, road, airline) and distribution networks (power and water)	39 - 40K (avg: 4,562)	68
Social	Facebook, Twitter and collaboration networks	85 - 34K (avg: 5,183)	118
Web	Pieces of the World Wide Web	146 - 16K (avg: 4,488)	14

Table 1: Real-world network dataset for graph classification[11]

In previous work, *M. Nagy* and *R. Molontay* managed to perform 0.912 accuracy in a graph domain classification task for this dataset with only 8 dimensional vectors as network embeddings. They used eight carefully selected topological features from four main domains: degree distribution, shortest path, centrality and clustering-related metrics. The beauty of this approach is the explainability of the coordinates, but calculating some of them is computationally costly (e.g. the average path length divided by the logarithm of the size or the maximum eigenvector centrality is computable in $O(n^3)$).

They used *Graph2Vec* and *AttentionWalk* as baseline methods and made the classification from different graph representations with *kNN*, *Random Forest* and *Decision Tree*. Their results can be seen in Table 2:

Approach	**kNN**	**Decision tree**	**Random Forest**
Topological features	78.5%	85.2%	89.3%
Selected features	87.0%	85.5%	91.2%
graph2vec ($\delta = 8$)	52.3 %	51.2%	55.8%
graph2vec ($\delta = 200$)	79.2 %	75.4%	82.1%
graph2vec ($\delta = 1100$)	84.2 %	74.1%	84.5%
AttentionWalk ($\delta = 128$)	27.0%	43.0%	54.0%
AttentionWalk ($\delta = 256$)	28.4%	37.9%	54.7%

Table 2: Results of *R. Molontay* and *M. Nagy*

This measurement's main aim was to try out embedding methods other than *graph2vec* and *AttentionWalk*, since there are newer approaches of whole graph embedding.

The question is not only if these accuracies can be outperformed but also the size of the embedded vector – a relatively small representation would be more interesting.

2.2 Karateclub – Embedding functions

The embedding algorithms used were implemented in the Karateclub Python package with customizable parameters. All the parameter setups used can be seen in the Appendix (A). They were chosen by hand in a way that the effect of the parameters would be visible and that the dimension–accuracy trade-off could be seen. (See more details in part 3 Results.)

4 of the tested 5 embedding algorithms use some randomness (mainly for initialisation), the exception is LDP. The other 4 algorithms were tried out with random seeds 1 to 10 for all of their parameters and we calculated the mean performance of those.

These implementations can only handle connected graphs, but not every graph in the database was connected, so I used the largest component (with the most nodes) of them.

The algorithms and their properties are the following:

2.2.1 SF *(Spectral Features)*[6]

SF is a simple baseline algorithm which uses the k smallest nonzero eigenvalues of the normalized Laplacian matrix of a graph as the embedded vector.

Parameters: number of eigenvalues desired (k)
Dimension: k

2.2.2 NetLSD *(Network Laplacian Spectral Descriptor)*[5]

Imagine the heating of graph nodes, the heat diffusion can be described with a differential equation. Its solution for passed time t:

$$\frac{\delta u_t}{\delta t} = -Lu_t \qquad H_t = e^{-tL} = \sum_{j=1}^{n} e^{-t\lambda_j} \phi_j \phi_j^T$$

where L is the Laplace matrix of the graph, λ_k and ϕ_k are eigenvalues and eigenvectors respectively. The trace of the network is $h_t = \sum e^{-t\lambda_j}$, calculating it for different t moments in time gives the embedded vector.

Parameters: beginning and ending of timescale interval ($scale_m in$, $scale_m ax$); number of timescale steps ($steps$); number of eigenvalue approximation steps ($approx$)
Dimension: $steps$

2.2.3 LDP *(Local Degree Profile)*[7]

For each node v, let $DN(v)$ denote the multiset of the degree of all the neighbouring nodes of v. We take five node features, which are $(degree(v), min(DN(v)), max(DN(v)), mean(DN(v)), std(DN(v)))$, then perform a histogram on each node feature.

Parameters: number of bins of the histogram (b)
Dimension: $5b$

2.2.4 Graph2Vec[12]

This model is analogous to *doc2vec*, an NLP algorithm: graphs and the neighbourhoods of nodes can be thought of as documents and the context of words. Here, these neighbourhoods are going to be rooted subtrees and we take the assumption that these trees are forming graphs in a similar manner to texts consisting of words. After the generation of the rooted subtrees via random walks the embedding of the graph is learned with the *Skipgram* model exactly the same way as in doc2vec.

Parameters: dimension of the embedded vector (d); number of learning epochs (e); learning rate (r)
(The other parameters not mentioned were left with the default settings.)
Dimension: d
Graph2Vec is already tried out by R. Molontay and M. Nagy, so the parameters are not fully reviewed.

2.2.5 FeatherGraph[8]

A characteristic function is defined for each vertex based on fixed-length random walks started from them – these functions can have a different shape for vertices with different structural properties. The pooling of the evaluation of these characteristic functions on either fixed or learned points (fixed in this test) calculates the embedded vector. In each evaluation point, we gain a 4-dimensional vector by the four quadrants of the characteristic function.

Parameters: Maximal distance to be considered in the characteristic function, or highest adjacency matrix power (o); number of evaluation points of the characteristic function (e); maximal evaluation point value (t); type of pooling (*min, max* or *mean*)
Dimension: $4 \cdot o \cdot e$

2.3 Classification methods

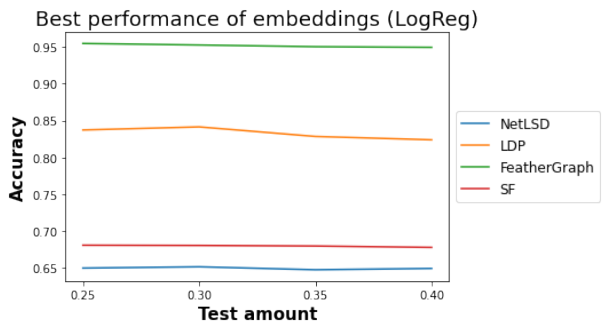
To classify the embedded vectors I used two different methods from *scikit-learn* [13] Python-package: Logistic Regression and Random Forest. In every classification measurement, 10 train-test splits were performed for each embedded vector set (the embedding of all networks with the exact same algorithm and parameters) and the performances of the classifier for these inputs were mean-pooled; this resulted in 10 mean-performances per embedding parameter setup (except LDP, where it is only one), one for each random seed (where applicable).

- **Logistic Regression:** The classifier was used with the default settings except for the solver, which was *Newton CG (Newton conjugate gradient method)* optimizer. Four ratios of the test dataset were tried: $0.25, 0.30, 0.35, 0.40$.
- **Random Forest:** There were two different measurements with this classifier:
  - **Varying test ratio:** Like for Logistic Regression, four ratios of the test dataset were tried: 0.25, 0.30, 0.35, 0.40. The number of trees (e) was set to 15 with a maximal tree depth (d) of 7, the other settings of the classifier were the default.
  - **Parameter optimisation:** The ratio of the test dataset was set to 0.3 for these measurements, and the best setting of the two main parameters (the number of trees (e) and the maximal tree depth (d)) were searched in the set: $(e, d) \in \{5, 9, 13, 17, 21\} \times \{5, 7, 9, 11\}$

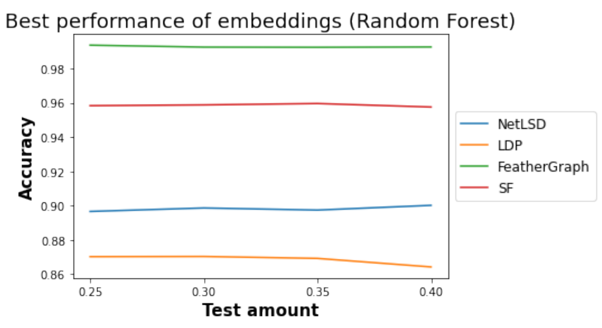
3 Results

3.1 Accuracy of Embedding Methods

In *Figure 3* the accuracy of the 5 tested embedding algorithms can be seen as a function of the test amount of the dataset with both Logistic Regression and Random Forest (with the number of trees set to 15 and maximal tree depth set to 7). The best performance here means the highest accuracy for the given test amount with some parameter setup of the embedder, so it might be that the points of a curve correspond to different parameters of the same embedder.



(a) Logistic Regression Classifier



(b) Random Forest Classifier

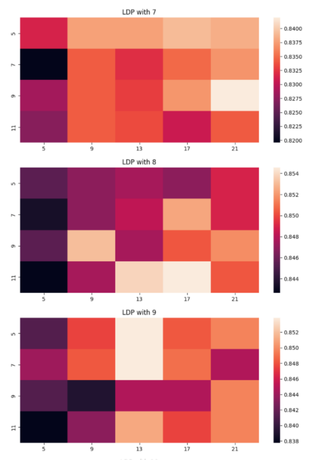
Figure 3: Accuracy of different embedding methods as a function of test amount

It can be seen that the modification of the test ratio will not make a significant effect, telling us that only 25% of the data (125 graphs) being used as the training dataset yields the same performance as 40% (200 graphs) for both classifiers.

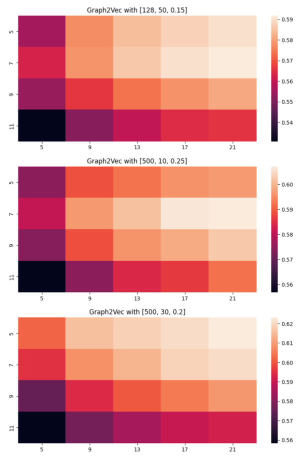
The most powerful embedder was *Feather Graph* for both classifiers with accuracy around 0.95 and 0.98. All of the embedders had a better performance with the Random Forest classifier, except for *LDP*, which fell back to the fourth place from the second.

3.1.1 Random Forest Parameter-testing

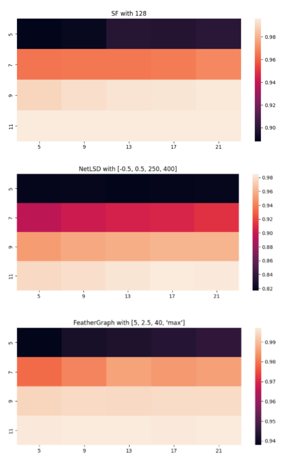
In the following figures (4), some samples can be seen of the heatmaps of the accuracies with the *Random Forest Classifier* with test ratio of 0.3 and $(e, d) \in \{5, 9, 13, 17, 21\} \times \{5, 7, 9, 11\}$. Only *LDP* and *Graph2Vec* were interesting, the former does not seem compatible with this classifier and the latter has its best performance with shallow but many trees. On the contrary, all the other embedding algorithms performed better with deeper trees disregarding their number.



(a) LDP



(b) Graph2Vec

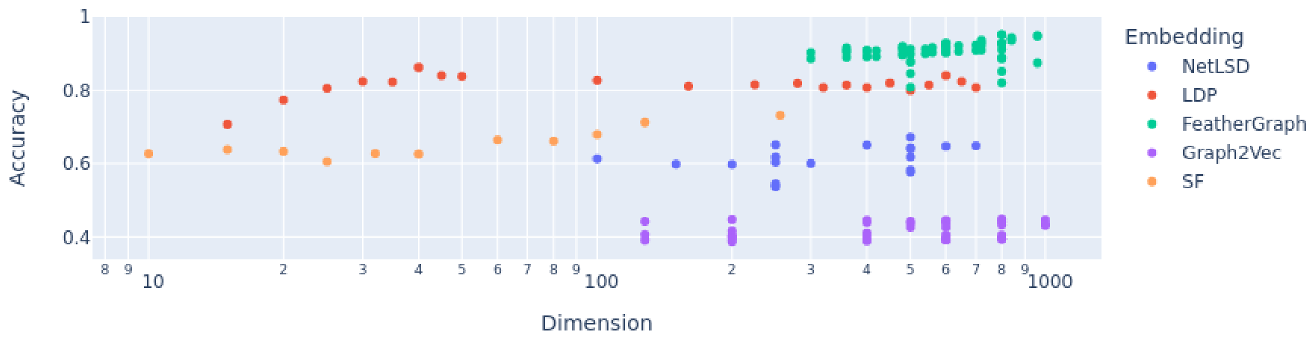


(c) SF, NetLSD and FeatherGraph

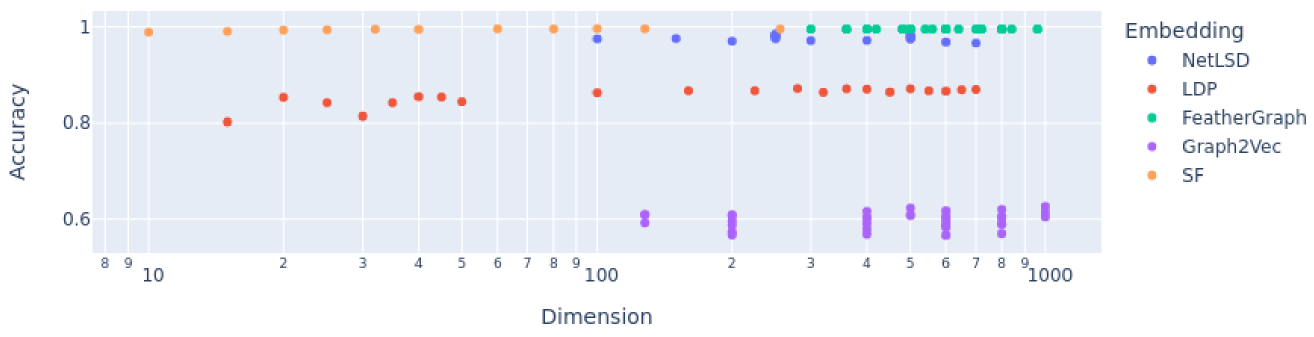
Figure 4: Accuracy of embedding methods with different parameters of Random Forest Classifier
(horizontal axis: number of trees; vertical axis: max. depth of trees; the lighter the color the better the performance)

3.2 Dimension – Accuracy trade-off

In Figure 5 the Accuracy as a funcion of dimension on a log-linear scale. For smaller dimensions, LDP or SF have the best performance for Logistic Regression and Random Forest respectively, while for higher dimensions FeatherGraph is the best option for both classifiers.



(a) Logistic Regression Classifier



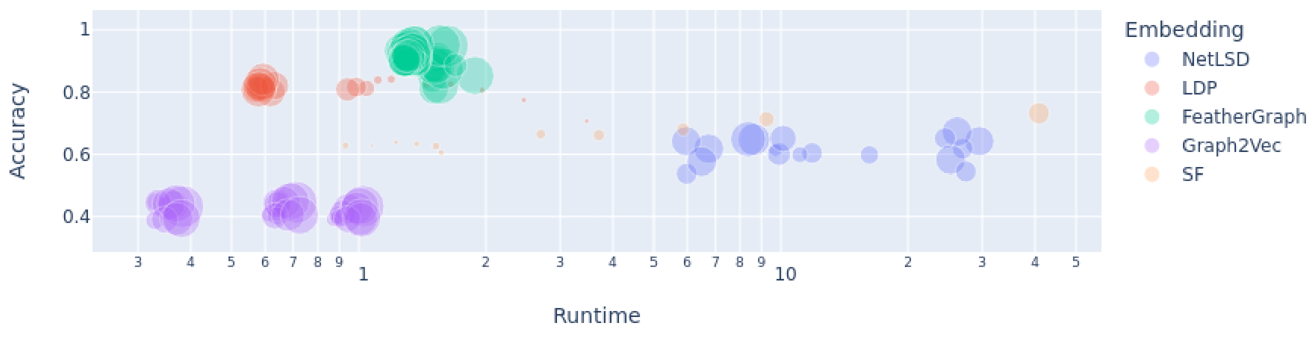
(b) Random Forest Classifier

Figure 5: Dimension – Accuracy trade-off of embedding algorithms with different classifiers

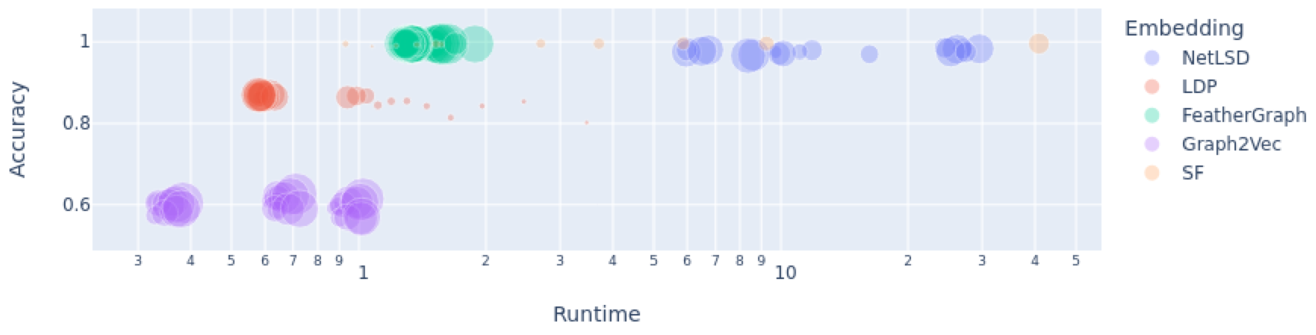
3.3 Runtime – Accuracy trade-off

In Figure 6 the Accuracy as a function of the runtime of the embedding is shown on a log-linear scale, where the bubble size is proportional to the dimension of the embedded vector. We can see that *Graph2Vec* is the quickest algorithm in general, however, it has the worst performance with these parameters. *LDP* is the second and *FeatherGraph* is the third regarding the runtime and the latter has a very promising accuracy as well. The runtime of *NetLSD* and *SF* is dependent on their parameters, but they are slower than the other three algorithms.

The overall best performance, accuracy of 0.996266 was gained by *SF* with parameter 100, resulting in 100-dimensional embedded vectors, which could be classified with Random Forest with tree number of 9 and max. depth of 11. The average computing time of a graph was 5.87s here, which is much slower than 1.28s, the average computing time of the slightly less accurate (0.995367) *FeatherGraph* with parameters 3 (highest power), 2.5 (maximal evaluation point value), 50 (number of evaluation points) and *min* (pooling), resulting in 600-dimensional embedded vectors classified with Random Forest with tree number of 9 and max. depth of 9.



(a) Logistic Regression Classifier



(b) Random Forest Classifier

Figure 6: Runtime – Accuracy trade-off of embedding algorithms with different classifiers

3.4 Effect of Parameters

In this section, some properties are detailed about the embedding algorithms (illustrated in the figures 5 and 6).

3.4.1 SF *(Spectral Features)*[6]

The runtime of *SF* quickly becomes very slow with the increase of dimension (because of the complexity of the eigenvalue search) but the performance just slightly gets better, there is no use in going over $150 - 160$ eigenvalues since *FeatherGraph* will outperform *SF* with quicker runtime.

3.4.2 NetLSD *(Network Laplacian Spectral Descriptor)*[5]

NetLSD gets slower for a higher number of eigenvalue approximation steps, while the performance increases slightly – it is not worth no use *NetLSD* since *FeatherGraph* will outperform it with quicker runtime in the same dimension range.

3.4.3 LDP *(Local Degree Profile)*[7]

LDP has a peak accuracy around dimension 40, for higher dimensions it stays nearly the same. There's no big difference in runtime either.

3.4.4 Graph2Vec[12]

The runtime of *Graph2Vec* is surprisingly quick, it slows down with the increasing number of training epochs, and the runtime seems to be a linear function of the number of epochs. However, it is not resulting in better performance, the highest accuracies for the same dimensions (of the tested parameter setups) belong to 10 epochs with a learning rate of 0.25 or 30 epochs with a learning rate of 0.2.

3.4.5 FeatherGraph[8]

The runtime of *FeatherGraph* stays almost the same for every tested parameter-setups and the importance of parameters forming the same dimension is not clear (whether higher matrix powers or more evaluation points are more useful). However, it can be seen that the worst pooling method is *mean*, it is always worse than *max* or *min*, but the relation of these two is also not obvious.

4 Summary of the project

During the three semesters, I examined whole graph embedding algorithms, and I could get a picture of their usefulness with real-life networks. Since networks can have thousands of nodes, in the first semester, I was experimenting with their runtime depending on the graph size. In the second semester, I was testing the performance of the embedding algorithms on the classification of randomly generated graphs to model their performance on some real-life networks. In the third semester, I gathered my previous experience with embedding algorithms and tried them out on a database of six network domains. The last task was very well solved with the help of embedding algorithms, choosing the best one depends on our main goal: it could be *LDP* for quick runtime, *SF* for small dimension, or *FeatherGraph* for (almost) the highest performance in a reasonable time.

Acknowledgements

I would like to especially thank Ferenc Béres for being a wonderful and motivating advisor, always helping with the next step and from whom I could learn a lot about data science and programming. I am also thankful to András Benczúr for setting the guidelines of the project, and Roland Molontay and Marcell Nagy for contacting us with the research idea.

References

[1] A. Galland and M. Lelarge, "Invariant embedding for graph classification," in *ICML 2019 Workshop on Learning and Reasoning with Graph-Structured Data*, (Long Beach, United States), June 2019.

[2] H. Chen and H. Koga, "Gl2vec: Graph embedding enriched by line graphs with edge features," in *Neural Information Processing - 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12-15, 2019, Proceedings, Part III* (T. Gedeon, K. W. Wong, and M. Lee, eds.), vol. 11955 of *Lecture Notes in Computer Science*, pp. 3–14, Springer, 2019.

[3] S. Verma and Z.-L. Zhang, "Hunt for the unique, stable, sparse and fast feature learning on graphs," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[4] F. Gao, G. Wolf, and M. Hirn, "Geometric scattering for graph data analysis," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 2122–2131, PMLR, 09–15 Jun 2019.

[5] A. Tsitsulin, D. Mottin, P. Karras, A. M. Bronstein, and E. Müller, "Netlsd: Hearing the shape of a graph," *CoRR*, vol. abs/1805.10712, 2018.

[6] N. de Lara and E. Pineau, "A simple baseline algorithm for graph classification," *CoRR*, vol. abs/1810.09155, 2018.

[7] C. Cai and Y. Wang, "A simple yet effective baseline for non-attribute graph classification," *CoRR*, vol. abs/1811.03508, 2018.

[8] B. Rozemberczki and R. Sarkar, "Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models," 2020.

[9] B. Rozemberczki, O. Kiss, and R. Sarkar, "An API oriented open-source python framework for unsupervised learning on graphs," *CoRR*, vol. abs/2003.04819, 2020.

[10] M. Nagy and R. Molontay, "Network classification-based structural analysis of real networks and their model-generated counterparts," *Network Science*, pp. 1–24, 2022.

[11] "Structural analysis of real networks and their model-generated counterparts – supplementary data." https://github.com/marcessz/Complex-Networks.

[12] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," *CoRR*, vol. abs/1707.05005, 2017.

[13] "scikit-learn python package." https://scikit-learn.org/stable/index.html.

A Parameter Setups of Embeddings

Name of Embedding	**Parameter description**	**Parameter setups**
NetLSD	• beginning and ending of timescale interval • number of timescale steps • number of eigenvalue approximation steps	[-0.2,0.2,250,200], [-0.2,0.2,500,400], [-0.2,0.2,250,400], [-0.2,0.2,500,200], [-0.5,0.5,250,200], [-0.5,0.5,500,400], [-0.5,0.5,250,400], [-0.5,0.5,500,200], [-0.1,0.1,250,200], [-0.1,0.1,500,400], [-0.1,0.1,250,400], [-0.1,0.1,500,200], [-0.5,0.5,100,200], [-0.5,0.5,150,200], [-0.5,0.5,200,200],[-0.5,0.5,300,200], [-0.5,0.5,400,200], [-0.5,0.5,600,200], [-0.5,0.5,700,200]
LDP	• number of bins of the histogram	3, 4, 5, 6, 7, 8, 9, 10, 20, 32, 45, 56, 64, 72, 80, 90, 100, 110, 120, 130, 140
FeatherGraph	• maximal distance to be considered in the characteristic function, or highest adjacency matrix power • maximal evaluation point value • number of evaluation points of the characteristic function • type of pooling	[5,2.5,25,”mean”], [5,2.5,25,”min”], [5,2.5,25,”max”], [5,2.5,40,”mean”], [5,2.5,40,”min”], [5,2.5,40,”max”], [5,1.5,25,”mean”], [5,1.5,25,”min”], [5,1.5,25,”max”], [5,1.5,40,”mean”], [5,1.5,40,”min”], [5,1.5,40,”max”], [6,4,40,”mean”], [6,4,40,”min”], [6,4,40,”max”], [5,2.5,15,”min”], [5,2.5,15,”max”],[5,2.5,20,”min”],[5,2.5,20,”max”],[5,2.5,30,”min”], [5,2.5,30,”max”],[5,2.5,35,”min”],[5,2.5,35,”max”],[5,2.5,40,”min”], [5,2.5,40,”max”],[6,3,15,”min”],[6,3,15,”max”],[6,3,20,”min”], [6,3,20,”max”], [6,3,25,”min”], [6,3,25,”max”],[6,3,30,”min”], [6,3,30,”max”], [6,3,35,”min”], [6,3,35,”max”],[4,2.5,25,”min”], [4,2.5,25,”max”], [4,2.5,30,”min”], [4,2.5,30,”max”],[4,2.5,35,”min”], [4,2.5,35,”max”], [4,2.5,40,”min”], [4,2.5,40,”max”],[4,2.5,45,”min”], [4,2.5,45,”max”], [4,2.5,50,”min”], [4,2.5,50,”max”],[3,2.5,30,”min”], [3,2.5,30,”max”], [3,2.5,35,”min”], [3,2.5,35,”max”],[3,2.5,40,”min”], [3,2.5,40,”max”], [3,2.5,45,”min”], [3,2.5,45,”max”],[3,2.5,50,”min”], [3,2.5,50,”max”]
Graph2Vec	• dimension of the embedded vector • number of learning epochs • learning rate	[128,10,0.25], [128,30,0.2], [128,50,0.15], [200,10,0.25],[200,30,0.2], [200,50,0.15],[200,10,0.025],[200,30,0.02], [200,50,0.015], [400,10,0.25], [400,30,0.2],[400,50,0.15],[400,10,0.025], [400,30,0.02],[400,50,0.015], [500,10,0.25], [500,30,0.2], [500,50,0.15], [600,10,0.25],[600,30,0.2], [600,50,0.15],[600,10,0.025],[600,30,0.02], [600,50,0.015], [800,10,0.25], [800,30,0.2],[800,50,0.15],[800,10,0.025], [800,30,0.02],[800,50,0.015], [1000,10,0.25], [1000,30,0.2], [1000,50,0.15]
SF	• number of eigenvalues desired	10, 15, 20, 25, 32, 40, 60, 80, 100, 128, 256

Table 3: Parameter setups tried out in the experiment