3-12-2020

Graph Classification with Kernels, Embeddings and Convolutional Neural Networks

Monica Golahalli Seenappa
San Jose State University

Katerina Potika
San Jose State University, katerina.potika@sjsu.edu

Petros Potikas
National Technical University of Athens

Recommended Citation

Monica Golahalli Seenappa, Katerina Potika, and Petros Potikas. "Graph Classification with Kernels, Embeddings and Convolutional Neural Networks" *2019 First International Conference on Graph Computing (GC)* (2020). https://doi.org/10.1109/GC46384.2019.00021

Graph Classification with kernels and embeddings.

Monica Golahalli Seenappa and Katerina Potika
Department of Computer Science,
San Jose State University,
San Jose, USA
Email: monica.golahalliseenappa, katerina.potika@sjsu.edu@sjsu.edu

Petros Potikas
School of Electrical and Computer Engineering,
National Technical University,
Athens, Greece
Email: ppotik@cs.ntua.gr

***Abstract*—In the Graph classification problem, given is a family of graphs and a group of different categories, and we aim to classify all the graphs (of the family) into the given categories. Earlier approaches, such as graph kernels and graph embedding techniques have focused on extracting certain features by processing the entire graph. However, real world graphs are complex and noisy and these traditional approaches are computationally intensive. With the introduction of the deep learning framework, there have been numerous attempts to create more efficient classification approaches. For the experiments, we use eight publicly available real world datasets, ranging from biological to social networks. For these datasets we provide results using a graph kernel algorithm and a spectral decomposition of Laplacian graph approach.**

Keywords - Graph Classification, Graph Kernels, Spectral decomposition, graph Laplacian

I. Introduction

Graphs can be used to represent most real-world data. Objects can be denoted as nodes of the graph and edges can be used to represent relationship between them. Graphs are used almost in every field. In social networks, graphs are used to provide online recommendations, implement newsfeed and calculate page rank [16]. In the field of neuroscience, neurons are denoted by nodes and connections between them as edges. These graphs are then used to analyse the functionality of brain networks [2]. In chemical engineering, covalent structures are represented as graphs [8]. Hydrocarbon structure, protein structure are represented in the form of graphs in bioinformatics field [14]. There are many more applications in other fields. These prove the importance of working with graphs.

Graph mining involves various tasks such as node classification, graph classification, link prediction, graph embedding, community detection. Since the introduction of machine learning approaches, there have been many attempts to discover useful information present within a graph. For applying such algorithms to graph domain, there should exist meaningful ways to compute similarity measures between graphs. Graph problems are not easy to solve. For example, the problem of finding maximum number of common subgraphs is computationally intractable. But graph similarity can be computed in various ways and the similarity measures need not be exact [7]. Approximate similarity measures are sufficient to work on graph related tasks. Even though there is significant progress in the field of graph mining, extracting graph features that truly represent the underlying graph structure still remains a challenge.

Problem Statement Graph classification is the problem of determining the category or target label of the graph. If we have a dataset consisting of many input graphs, the problem is to classify each of the graphs to their correct category or target label. For example, in the case of chemical compounds, nodes represent the atoms and edges represent bonds between the atoms. The classification problem might be to determine if the chemical compound is toxic or non-toxic by looking at its structure. The model would be trained with known examples of toxic and non-toxic compounds. When the model encounters an unknown or new sample, it should predict whether it is toxic or non-toxic.

Real world graphs are large and complex. They are known to contain lot of noise elements as well. These noise elements do not add any valuable information. It is crucial to eliminate them, else they might introduce wrong insights. The classifier model should be capable of handling large graphs as well has eliminate insights obtained from noise elements. The model should be robust, efficient to compute and not consume too much space.

Given a dataset of input graphs $G = \{G_1, G_2, G_3, ..., G_N\}$, and their corresponding labels, the task is to build a model that learns from these graphs and predicts the label of new, unseen graphs. Graph features are computed and compared to make prediction for these new graphs. A popular approach is the usage of graph kernels, which focuses on calculating occurrences of different patterns in the input graphs. These include counting shortest-paths, performing random walks on the graphs, etc. Graphs which share lot of features are considered as similar and are placed in the same category [15].

Following are few of the challenges encountered during the dataset processing: (i) If the dataset is partially labeled, learning from the input graphs might lead to inaccurate information, (ii) If dataset is collected from multiple sources, then aggregation can cause information inconsistency, (iii) If the dataset is collected using some hardware instruments, care must be taken to ensure these instruments are not faulty [11], (iV) Dataset collected might be huge. Processing it efficiently by eliminating noisy features might be difficult.

Applications of the Graph Classification problem. Some of the applications of the graph classification problem are in:

Bioinformatics and Chemoinformatics, for predicting the function of a protein structure, predicting if the cells are cancerous or not, predicting if a protein is enzyme or not, checking the toxicity of a chemical compound. Social Network analysis, where the users of social networking sites such as Facebook or LinkedIn are represented as nodes and the interaction between them is captured using edges. Such networks help in providing recommendations for a page or user account to follow [16]. In this paper, we have performed experiments focusing on improving the graph spectrum algorithm as well as kernel graph convolutional neural networks. We compare our results with a baseline algorithm using Weisfeiler-Lehman kernel as well as related previous work.

II. Related Work

Graphs have been of great interest for a long time. Most earlier approaches dealt with identifying if two graphs are identical or not. This problem is hard to solve and until recently it was not known to be either tractable or intractable. In 2016, the author of [1], showed that graph isomorphism can be solved in $(exp((logn)^{O(1)}))$ time. That is, we can compute if graphs are identical or not in quasipolynomial time. In our problem, we are not interested in knowing if two graphs have same structure or not. We want to explore if two graphs are similar. This paves path for finding a more faster, efficient approach for our classification.

There are many approaches proposed for the task. Initial approaches would make lot of assumptions about the dataset. Most of them lacked a proper embedding technique. They processed only few nodes which they assumed to be important and also had certain assumptions about the graph data like its labeled or unlabeled, weighted or unweighted. Embedding techniques should be good enough to capture the relationship between nodes and retain the structure of the graph [11]. There are several graph embedding techniques. Using these embeddings, the graphs are represented in the form of a vector or group of vectors. Working on vectors are convenient and easier than processing the entire graph. Also, most programming languages support several packages to transform vectors. Approaches like DeepWalk and Node2Vec [3] perform random walks on the graph to capture information about their neighborhood. The embeddings should capture meaningful information from the graphs such as the interaction between subgraphs and neighborhood information for a node.

In [9], the authors define the normalized laplacian of a graph inorder to extract features of the graph. This method is further explained and extended in section III-B.

Other techniques focus on developing a greedy algorithm for the comparison between two graphs. To compare two graphs G and G', all we had to do is search each subgraph from G in G'. If all the subgraphs are present, we would declare they are similar else they are not similar. With the advancements in machine learning, many attempts were made to incorporate them to the field of graphs. The most famous among these approaches is the use of graph kernel [12]. The kernel approach computes a similarity matrix internally and passes this to a classification algorithm. There are several kernels proposed over the past few years. We have wide range of kernels ranging from Random walks, Shortest-path to Weisfeiler-Lehman kernels [18].

Until recently, graph kernels dominated the graph classification. All graph kernels are developed with the same generic idea. They are represented in the form of a matrix which can then be passed onto a kernel-based classifier. The challenge is to develop a kernel function which can be computed relatively faster. The similarity function need to be symmetric and positive semidefinite.

Random walks kernels are one of the oldest graph kernels proposed. The basic idea is to count the common walks in the graphs and compare them [5]. Product is computed between the two graphs called as direct product graph. But this method is too slow and its complexity amounts to $O(n^6)$. The walks may iterate over the same nodes again and leads to tottering effect. Many approaches were later proposed to improve the random-walk kernel. Notable among them is the cyclic-pattern kernel [4]. In this method, a graph is decomposed into many cyclic patterns. We compare the two graphs by comparing the number of cyclic patterns which appear in both the graphs. Computation power involved is low, but it does not work well for all the graphs. It works good only in the presence of simple cycles.

Instead of focusing on walks, focus shifted to paths [17]. Label enrichment techniques were developed to improve runtime for especially graphs will simple labels [10]. Also, optimizations were performed using Singluar Value Decomposition (SVD) to generate lower rank matrices. Linear-algebra concepts were applied along with Kronecker product to reduce complexity to $O(n^3)$. Another major graph kernel developed was the shortest-path kernel. Computing all-path is not tractable, but shortest path between all pair of vertices can be computed in $O(n^3)$. For a given graph, its all-pair shortest path matrix is computed. Another improvement made was to compute only k-shortest paths instead. Though the runtime was improved, it was still not fast enough.

The Weisfeiler-Lehman (WL) kernel by [13], outperformed all the graph kernels developed till then on most standard datasets. Specifically, the subtree variant, compared each label of the graph by using a compressed form. Computation was low since the labels were compressed and hashing was done. Desiging the kernel function dictates how fast the method will perform. It's crucial to develop a function which can computed easily. Many other kernels where developed later like the optimal assignment kernel and graphlet kernels.

III. Methodology

Given a collection of graphs $G = \{G_1, G_2, G_3, ..., G_N\}$, where each $G_i = (V_i, E_i)$ has V_i vertices and E_i edges, and their target labels, the graph classification problem aims to classify unknown graphs into appropriate categories. We begin each of our approach by building a model trained on the input dataset. The model should capture the relationship between the structure of a graph and its target label. When the model

is given an unlabeled graph as input, it should determine the correct category of the graph.

A. Weisfeiler-Lehman Subtree Kernel

Graph kernels are one of the most important approaches used for graph classification. There are several kernels available, but we will be using the WL Subtree kernel for our experiments. Graph kernels make use of the kernel trick to reduce dimensionality. In each step of the algorithm, labels of the node are renamed with a set of labels formed by combining the immediate neighbors. The labels are renamed to a compressed version. The steps are repeated until the two graph's labels vary.

Graphs kernels are a supervised approach to perform classification. Typically, a kernel matrix is computed upon applying a graph kernel. This matrix is passed to a kernel-based machine algorithm like Support Vector Machines (SVM) to perform classification. The WL Subtree kernel is based on WL test for isomorphism between two graphs.

The WL Subtree kernel is an extension of the idea of the 1 dimensional WL test for graph isomporhism. We start with all the input graphs in the dataset. Algorithm 1 is used to compute the kernel.

Algorithm 1 One iteration of the Weisfeiler-Lehman subtree kernel computation on N graphs

1: Multiset-label determination
- Assign a multiset-label $M_i(v)$ to each node v in G which consists of the multiset $\{l_{i-1}(u)|u \in N(v)\}$.

2: Sorting each multiset
- Sort elements in $M_i(v)$ in ascending order and concatenate them into a string $s_i(v)$.
- Add $l_{i-1}(v)$ as a prefix to $s_i(v)$.

3: Label compression
- Map each string $s_i(v)$ to a compressed label, using a hash function $f : \Sigma^* \rightarrow \Sigma$ such that $f(s_i(v)) = f(s_i(w))$ if and only if $s_i(v) = s_i(w)$.

4: Relabeling
- Set $l_i(v) := f(s_i(v))$ for all nodes in G.

In this case, the algorithm runs in $O(hm)$ time. In the first step, we compute the multiset label l_i for all the N graphs within the dataset. All the graphs are processed simultaneously and operations are performed in parallel in all the h iterations. We obtain the neighborhood set for a given node and concatenate all their names into a single string in the second step. The neighbors of a node are sorted before adding to the multiset using radix sort. Let f denote the function that represents the mapping of neighborhood strings to a compressed label. Functio f can also be implemented using a perfect hash function. The time complexity would be linear and is equal to $O(Nn+Nm) = O(Nm)$. This denotes the sum of the length of the string an the current alphabet. In the third and fourth step, we are compressing the label and renaming it.

With this, we complete the first iteration in our h iterations. At the end of each iteration, we would be computing a new feature vector for both the graphs. We compare the original graph with the new graph. We count the labels newly formed. We initially set a threshold for the labels. If the labels vary more than the given threshold, then the algorithm terminates and we say that the graphs are not identical. If not, we continue our iterations till we reach h iterations.

B. Graph Embedding using Laplacian Decomposition

Since a graph is non-linear, we need to extract features from the graph that captures information within it.

There are mainly two kind of graph embeddings. One focuses on embedding the entire graph and the other embeds nodes. We will be working with the entire graph embedding to perform classifications. We perform experiments using the spectral features of a graph as descibed in [9]. We derive features from the graph spectrum and pass it as input to a classifier. We have experimented with various classifiers ranging from Support Vector Machines to Multi-layer Perceptrons.

Assume we have a set of undirected and unlabeled collection of graphs $G = (V, E)$, given as a boolean adjacency matrix $A \in \{0, 1\}$ which indicates 1 if there exists an edge between two nodes or 0 otherwise. Similarly a degree matrix D is constructed which contains degrees for each node. We assume the graph is connected. If it is not, then we extract the largest connected component from the graph.

In [9], the authors define the normalized laplacian of a graph by $L = I - D^{-1/2}AD^{-1/2}$, where A is the adjacency matrix and D is the degree matrix. The pseudocode for the model is given in Algorithm 2. There are three steps involved to obtain the spectral features.

Algorithm 2 Spectral decomposition of graph Laplacian

1: For graph $G = (V, E)$ with V vertices and E edges, derive the following:
- A boolean adjacency matrix $A \in \{0, 1\}^{|V|\times|V|}$.
- A degree matrix $D = diag(A1)$ of node degrees.

2: Derive normalized laplacian for the graph
- If G is not connected, then extract the largest connected component.
- Compute laplacian of G as: $L = I - D^{-1/2}AD^{-1/2}$

3: Derive spectral features of the graph
- Perform eigenvector decomposition on the graph and obtain k smallest positive eigenvalues of L.
- If graph has less than k nodes, pad zeroes to the right end of the vector.

4: Provide the spectral features as input to the classifier.

The Laplacian matrix computed might be huge for large graphs. Instead of considering the entire L matrix, we can focus only on the relevant elements which give us information about the graph. For this, we perform eigen decomposition of the matrix. The L matrix is now split into eigenvectors and eigenvalues. We chose the eigenvector which is the largest and

its corresponding eigenvalue. The spectral features are defined by the eigenvlaues. These form the basis for comparison between two graphs. If the graphs are similar, then their corresponding eigenvectors are similar. They might just be a permutation of one another.

The spectral features are crucial for detecting similarity between graphs. From the eigenvalue decomposition performed earlier, we will choose only the k smallest eigenvalues. These eigenvalues need to be positive as well. If there are less than k eigenvalues, we append required number of zeroes to the vector. Finally, we sort the values. This vector represents the information in the graph. This is provided as input to a chosen classifier.

The method is fast and comparable with benchmark algorithms. Since the eigenvalues of the laplacian matrix lies between 0 and 2, preprocessing of the graph wouldn't take much time and therefore the model is fast.

IV. Experimental Evaluation

Datasets

We will be working with eight publicly available real-world datasets as summarized in Table I. Two of them are social networks with movie collaborations (IMDB-BINARY and IMDB-MULTI) and the rest are from Biology.

The above benchmark datasets for graph kernels are collected from [6], where n is the number of nodes, m is the number of edges, and N is the number of graphs.

TABLE I: Dataset statistics and properties.

Dataset	Graphs	Avg. Nodes	Graph labels
NCI-1	4110	29.8	2
MUTAG	188	17.9	2
PTC	344	25.5	2
ENZYMES	600	32.6	6
PROTEINS	1113	39.1	2
IMDB-BINARY	1000	19.77	2
IMDB-MULTI	1500	13	3
DD	1178	241	2

Results for the Weisfeiler-Lehman Subtree Graph Kernel

The graph kernel approach is used as a baseline for our experiments. There are many graph kernels available: Random Walk, Shortest-Path, Weisfeiler-Lehman, Optimal Assignment, Weighted Decomposition and many more. Among these kernels, Weisfeiler-Lehman subtree kernel provides competitive results. It's accuracy levels are comparable to benchmark models and its runtime is faster for smaller datasets. For the above reasons, we have used Weisfeiler-Lehman (WL) subtree kernel as our baseline algorithm.

For implementing this kernel, we used the popular Python package *"graphkernels"*.

Table II provides a summary of accuracies achieved with the WL subtree kernel. These results will be used as a baseline for our graph embedding approach.

Results for the Graph Embedding using Laplacian Decomposition

TABLE II: Experimental accuracy with Weisfeiler-Lehman subtree Kernel.

Dataset	NCI-1	MUT.	PTC	ENZ.
Accuracy	80.13	82.05	56.97	52.22
Dataset	PROT.	IMDB-BIN.	IMDB-M.	DD
Accuracy	72.92	68.6	48.13	71.3

We have used the eight datasets described previously for our experiments. For building the classifiers, we make use of the machine learning library *scikit-learn* in Python. The classifiers used for obtaining the results are: AdaBoost (AB) Classifier, ultilayer Perceptron (MLP), k-Nearest Neighbors (k-NN) Classifier, Gaussian Naive Bayes (GNB) Classifier, Decision Tree (DT) Classifier,Random Forest (RF) classifier and Support Vector Machine (SVM) Classifier.

Hyperparameters were tuned to obtain better results with each classifier. The embedding dimension is set to the average number of nodes for each dataset. To compute accuracy of the model, k-fold cross validation is used. We have set k=10 for our experiments. This means the dataset is divided into ten folds. Nine of the folds act as training set, while the remining one is the test set. This is repeated for all the 10 folds and the accuracy of the model is the average value across all the folds. The results for each of the classifier is tabulated in Table III.

We can notice that Random Forest classifier gives very good results. AdaBoost classifier was first constructed with decision trees. But upon replacing it with Random Forest as weak learner, we get better results. For getting the right parameters, we focus on Random Forest classifier and vary the depth, number of observations and estimators. We can choose the exact value of each hyperparameter and pass this to our AdaBoost classifier.

By performing the above experiments, we set the following hyperparamters for the Random forest classifier within AdaBoost: Number of estimators is 500, number of leaf samples is 1 and maximum depth is 50. We notice that the AdaBoost classifier outperforms other classifiers on six datasets.

V. Conclusions and Future Work

We have performed extensive experiments on the graph decomposition method using the graph laplacian. For the graph decomposition method, we use several supervised learning algorithms available in the *scikit-learn* package. Upon comparing all the two methods, we conclude that the graph embedding using laplacian decomposition performs well on five out of eight datasets, see Table IV. We obtained this result when we used the AdaBoost classfier.

For future work, we can try to reduce the execution time by introducing parallelism in the code. Also, we could perform patch normalization using different combination of graph kernels and try to increase the performance of the model.

TABLE III: Experimental accuracy of different models.

Model	NCI1	MT	PTC	EZ	PF	BIN	MUL	DD	Time
AB	**75.29**	75.29	**75.72**	**75.97**	**74.7**	**75.88**	**74.36**	75.29	325.31
MLP	67.29	85.66	55.49	31.83	71.88	67.2	45	72.4	36.5
K-NN	67.05	84.17	55.17	32.5	69.53	67.8	40.06	71.29	1.49
GNB	60.21	83.59	60.19	22	68.29	56.8	40.6	75.72	**0.34**
DT	68.12	86.72	59.90	33	66.03	69.4	47.2	68.49	5.04
RF	75.23	**88.39**	62.79	43.67	73.59	72.6	48.33	75.37	217.97
SVM	62.48	84.2	59.93	26	72.41	62.5	45.2	**75.97**	20.95
LR	62.6	85.75	58.12	26.33	71.16	61.4	44.2	73.93	6.41

TABLE IV: Comparison of accuracies from the methods: WL subtree kernel, Graph embedding using spectral decomposition.

Dataset	WL subtree kernel	Graph embedding
NCI-1	**80.13**	75.29
MT	82.05	75.29
PTC	56.97	**75.72**
EZ	52.22	**75.97**
PF	72.92	**74.7**
BIN	68.6	**75.88**
MUL	48.13	**74.36**
DD	71.3	75.29

REFERENCES

[1] Babai, L.: Graph isomorphism in quasipolynomial time (2015). https://doi.org/abs/1512.03547

[2] Garcia, J.O., Ashourvan, A., Muldoon, S., Vettel, J.M., Bassett, D.S.: Applications of community detection techniques to brain graphs: Algorithmic considerations and implications for neural function. Proceedings of the IEEE **106**(5), 846–867 (2018)

[3] Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016. pp. 855–864 (2016)

[4] Horvath, T., Gartner, T., Wrobel, S.: Cyclic pattern kernels for predictive graph mining. In: Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD). pp. 158–167 (2004)

[5] Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernels between labeled graphs. In: Proceedings of the International Conference on Machine Learning. pp. 321–328 (2003)

[6] Kersting, K., Kriege, N.M., Morris, C., Mutzel, P., Neumann, M.: Benchmark data sets for graph kernels (2016), http://graphkernels.cs.tu-dortmund.de

[7] Kriege, N.M., Mutzel, P.: Subgraph matching kernels for attributed graphs. In: Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1 (2012)

[8] Kulkarni, S.J.: Graph theory: Applications to chemical engineering and chemistry. Galore International Journal of Applied Sciences and Humanities **1**(2) (2017)

[9] de Lara, N., Pineau, E.: A simple baseline algorithm for graph classification. CoRR **abs/1810.09155** (2018)

[10] Mahe, P., Ueda, N., Akutsu, T., Perret, J., Vert, J.: Extensions of marginalized graph kernels. In: Proceedings of the Twenty-First International Conference on Machine Learning. pp. 552–559 (2004)

[11] Neumann, M., Garnett, R., Bauckhage, C., Kersting, K.: Propagation kernels: efficient graph kernels from propagated information. Machine Learning **102**(2), 209–245 (2016)

[12] Nikolentzos, G., Meladianos, P., Tixier, A.J., Skianis, K., Vazirgiannis, M.: Kernel graph convolutional neural networks. In: Artificial Neural Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I. pp. 22–32 (2018)

[13] Nikolentzos, G., Meladianos, P., Vazirgiannis, M.: Matching node embeddings for graph similarity pp. 2429–2435 (2017)

[14] Pavlopoulos, G.A., Secrier, M., Moschopoulos, C.N., Soldatos, T.G., Kossida, S., Aerts, J., Schneider, R., Bagos, P.G.: Using graph theory to analyze biological networks. BioData Mining **4**, 10 (2011)

[15] Rogers, D., Hahn, M.: Extended-connectivity fingerprints. Journal of Chemical Information and Modeling **50**(5), 742–754 (2010)

[16] Tang, L., Liu, H.: Graph mining applications to social network analysis. In: Managing and Mining Graph Data, pp. 487–513 (2010)

[17] Vishwanathan, S., Schraudolph, N., Kondor, R., Borgwardt, K.: Graph kernels. Journal of Machine Learning Research **11**(5), 1201–1242 (2010)

[18] Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of AI (IAAI-18), and the 8th AAAI Symposium on Educational Advances in AI (EAAI-18). pp. 4438–4445 (2018)