Machine Learning Methods with Graph-Level Features and Application Use Cases

Koffka Khan
Department of Computing and Information Technology, Faculty of Science and Agriculture, The University of the West Indies, St. Augustine Campus, TRINIDAD AND TOBAGO
Email address: koffka.khan@gmail.com

Abstract— *In this paper we talk about graph-level characteristics and graph kernels, which will enable us to forecast the behaviour of entire graphs. Therefore, the objective is to have a single feature that describes the overall graph's structure. We discussed graph kernels, specifically the graphlet kernel and the WL, which stands for the Weisfeiler-Lehman graph kernel. Its runtime scales only linearly in the number of edges of the graphs and the length of the Weisfeiler-Lehman graph sequence. Finally, we give some WL use cases from the literature.*

Keywords— *Graph: kernel: feature: graphlets: Weisfeiler-Lehman: runtime: sequence.*

I. Introduction

There are properties at the node and edge levels that can be used to forecast a graph [20]. However, in this paper will now talk about graph-level characteristics and graph kernels [11], which will enable us to forecast the behaviour of entire graphs [5]. Therefore, the objective is to have a single feature that describes the overall graph's structure. For instance, if you have a graph like the one in Figure 1, you may think about how you would describe its structure in words.

-

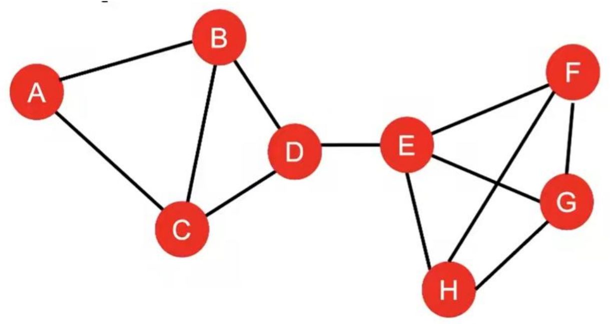


Fig. 1. Connected graph with eight nodes (A to F).

It appears to have two loosely connected portions, a lot of edges connecting the nodes in each part, and just one edge between nodes D and E connecting the two sections.

How do we develop the feature descriptor that will enable us to describe the structure, then? And to accomplish this, we'll employ kernel-level techniques. Additionally, traditional machine learning in graph-level prediction uses kernel approaches extensively. Instead of creating a feature vector, the goal is to create a kernel. Let's briefly introduce and explain what a kernel is. Therefore, a kernel between graphs G and G' returns a real number and assesses the similarity between these two graphs or, more generally, the similarity between other data points.

Thus, a kernel matrix [12] is a matrix that merely calculates the degree of similarity between all sets of data points or graphs. Additionally, the kernel matrix must be positive semi-definite for a kernel to be considered genuine. That is to say, it must have positive eigenvalues [7], for instance, and as a result, it must be a symmetric matrix. The existence of a feature representation, such that the kernel between two graphs is just a feature representation of the first graph dot product with the feature representation of the second graph, is another crucial characteristic of kernels.

The value of the kernel is just a dot product of this vector representation of the two graphs, with the feature representations of G and G’ being vectors. And what's sometimes good about kernels is that we can compute the kernel's value without even having to explicitly generate this feature representation. Once the kernel has been established, predictions can then be made using pre-built machine learning models such kernel support vector machines.

Once the kernel has been established, predictions can then be made using pre-built machine learning models such kernel support vector machines. In this paper, we'll talk about several graph kernels that let's quantify how similar two graphs are to one another. We'll talk in particular about the graphlet kernel and the Weisfeiler-Lehman kernel [14]. Other kernels are also suggested in the literature, for example the random-walk kernel and the shortest-path kernel, and numerous others. Additionally, these kernels often offer very competitive performance in tasks at the graph level.

This paper consists of five sections. In Section II we introduce graph kernels and graphlet kernels. In Section III we discuss the Weisfeiler-Lehman graph kernel, while some use cases of this graph kernel is given in Section IV. Finally, the conclusion in given in Section V.

II. Graph Kernels

What are the kernels' central concept? The main objective of kernels is to specify the feature vector of a given graph G. The concept is that we should consider this feature vector to be a form of bag-of-words representation of a graph. What then is a bag of words? When we have text documents, one method to represent them is to just think of them as a collection of words. Basically, we would say, we keep track of the frequency of each term throughout the document.

We can consider of words as being arranged alphabetically, and at position i of this representation of a bag of words [18], we will get the frequency, or the number of times the word, i appears in the document. In our situation, this holds true, and a simplistic application of this concept to graphs would be to treat the nodes as words. We would obtain the same feature vector or representation for two very distinct graphs, which is a concern because graphs might have very diverse structures but the same number of nodes. Since this network has four nodes (see Figure 2) and this graph also has four nodes (see Figure 3), their representations would be the same if we were to think of nodes as words.

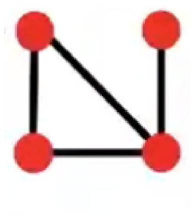


Fig. 2. Graph with 4 nodes.

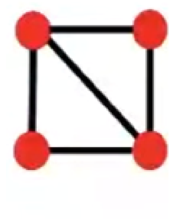


Fig. 3. Graph with 4 nodes, but different from Figure 2.

As a result, we require a different candidate for the word in this particular bag-of-words form. We could have something we could refer to as a degree kernel that would allow us to say, "how are we going to describe a graph," for instance, to be a little more expressive. It will be represented as a bag of node degrees. From Figure A, "We have one node of degree 1, three nodes of degree 2, and 0 nodes of degree 3," is what we say [1, 3, 0]. In a similar vein from Figure B, we may inquire as to the number of nodes that we have that are of various degrees. 0 nodes of degree 1, 2 nodes of degree 2, and 2 nodes of degree 3 are present, [0, 2, 2]. As a result we would now be able to distinguish between these various graphs by obtaining various feature representations for each of them.

Now, the graphlets kernel and the Weisfeiler-Lehman kernel both make use of the concept of a "bag-of-*" representation of a graph, where the star is more complex than node degree. Let's start by discussing the graphlets kernel. The rationale behind this is that by writing 1, we represented the graph as a count of the various graphlets it included. We want to emphasise an essential point: A graphlet's definition for a graphlet kernel differs slightly from that of a graphlet's definition for node-level characteristics. Furthermore, graphlets in the node-level characteristics do not need to be connected and are not rooted, which are two significant differences.

Therefore, graphlets in the kernel for graphlets are not rooted and are not required to be connected [8]. For instance, if you have a list of graphlets in which we are interested from 1 to n. Let's assume these graphlets are k-dimensional, then let's say k = 3, then there are four different graphlets. On three nodes, there are four distinct graphs, each with two directed, fully connected edges, one edge, and no edges. This is how graphlets are defined in the graph kernel. For instance, there exist 11 distinct graphlets for k = 4, ranging from a fully linked graph to a graph with only four nodes and no connections. Thus, given a graph, we can just count the number of distinct graphlets that appear in the graph to describe it.

For instance, we define the graphlet count vector f [24], given a graph and the graphlet list, as the straightforward number of instances of a given graphlet that appear in our graph of interest. For example, if the graph G is the one we are interested in, then no triplets of unconnected nodes exist in this graph, nor do there exist any isolated nodes on any of the six edges that make up the graph. There are also one triangle, three different parts of land 2, and no isolated nodes on any of the six edges that make up the graph, see Figure 4.

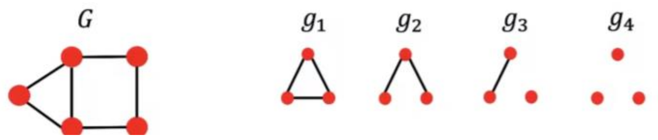


Fig. 4. For k =3 there are 4 graphlets.

The graphlet feature vector in this instance, then, would be as follows: 1, 3, 6, and 0 would be available.

Now that we have two graphs, we can define the graphlet kernel as the dot product of the graphlet count vectors of the first and second graphs. Although this is an excellent idea, there is a small issue. The issue is that graphlets in distinct graphs may have extremely varied row counts due to the possibility that graphs G1 and G2 have different sizes. Therefore, normalising this feature vector representation for the graph is a popular solution.

This implies that the graphlet vector representation for a given graph is just the canonical division of the total number of graphlets contained in the graph by the count of individual graphlets.

The graphlet kernel is defined as the dot product between these feature vector representations of graphs, which basically normalise for the size and density of the underlying graph. These feature vector representations of graphs capture the frequency or the proportion our given graphlet, in a graph. The graphlet graph kernel has a significant restriction. And another drawback is the high cost of counting graphlets. It takes time or n raised to the power k to enumerate all of the k-size graphlets in a graph with n nodes. As a result, counting graphlets of size k is exponential in terms of graphlet size but polynomial in terms of the number of nodes in the graph.

And in the worst-case scenario, this is unavoidable due to the fact that determining if a sub-graph is isomorphic to another graph is NP-hard [1]. There are also quicker algorithms. There is a much faster approach to count the graphlets of size k if the graphs node degree is constrained by

d. However, the problem still exists that it takes a lot of time and money to count these discrete objects in a graph. Therefore, we are only able to count graphlets with a small number of nodes. The exponential complexity then takes control, and we are unable to count graphlets that are greater than that.

III. WEISFEILER-LEHMAN GRAPH KERNEL

The question is, how can we create a graph kernel that is more effective? And the Weisfeiler-Lehman graph kernel succeeds in accomplishing this. The concept is to use a neighbourhood structure to repeatedly enhance the vocabulary of nodes, hence the objective is to build an effective graph feature descriptor of G. And since node degrees are one hot-one-hop neighbourhood information to multi-hop neighbourhood information, we generalise a version of node degrees. The Weisfeiler-Lehman graph isomorphism test [4], commonly known as colour refinement, is the algorithm that does this.

The concept is that we will assign an initial colour so that this is an initial colour to each node, to a graph G that has a set of nodes V. Then, in order to create new colours, we will iteratively aggregate or hash colours from our neighbours. The new colour for a given node v will therefore be composed of its own colour's hashed value from the previous time step plus a concatenation of colours from the node v's interesting neighbours, where hash is essentially a hash function that converts distinct inputs into different colours. And after k iterations of this colour refining, v summarises the graph's structure at the level of K-hop neighbourhood.

Let's explain and give you an example. In this case, I have two graphs that are quite similar structurally but differ only somewhat. This edge and this edge, respectively, are different. The diagonal edge and the triangle closing edge are both different. Therefore, we will start by giving nodes their basic colours. As a result, each node receives a colour of 1, which is the same for all nodes, see Figure 5.

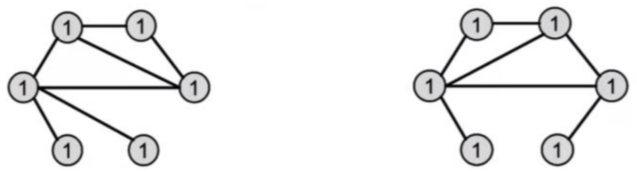


Fig. 5. Initial assignment of colors: Color refinement of 2 graphs.

Now we are going to aggregate neighbouring colours, see Figure 6.

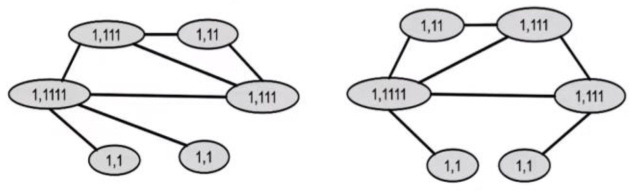
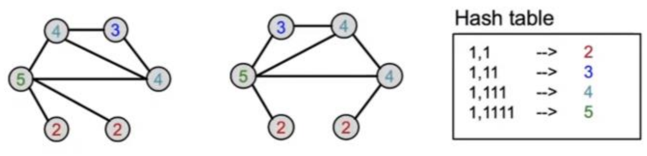


Fig. 6. Aggregate neighboring colors: Color refinement of 2 graphs.

In contrast to this specific node up to the top left of Figure 6, which collects colours from its neighbours, one and one and this particular node aggregates colours from its neighbours, 1, 1, 1, and adds it to it- to itself. The exact same procedure also applies to the second graph. After collecting the colours, we should go ahead and hash them. Therefore, we use a hash function, which creates new colours by combining the colours of a node's neighbours and its own colour. Let's assume that the hash function here produces the following results for the first combination: 1, 2, 3, 4, and 5, respectively. So, based on these more revised colours, are we now colouring the graphs?



1,1	-->	2
1,11	-->	3
1,111	-->	4
1,1111	-->	5

Fig. 7. Aggregate Colors: Color refinement of 2 graphs.

Based on the hash values [23] of the aggregated colours from the first step, this is the colouring of the first graph (see Figure 7, left graph), and this is the colouring of the second graph (see Figure 7, right graph). We now take these two graphs and use the same colour aggregating method once more. As an illustration, this node, which has the colour 4, aggregates the colours of its neighbours, including the 3, 4, and 5, see Figure 8. As a result, we have 3, 4, and 5 here, whereas, for instance, this node here with the colour 2 aggregates from its neighbour, who is coloured 5, so it receives 2, 5. The same thing occurs for this graph as well.

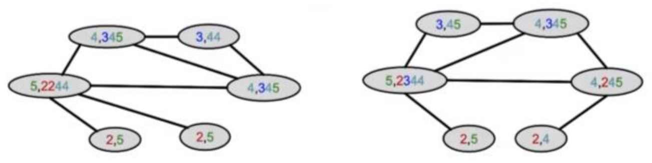


Fig. 8. Aggregated hash colors: Color refinement of 2 graphs.

Now, once more, we take these aggregated colours and hash them. And let's imagine that the new colours that our hash function assigns to these previously aggregated colours from past timesteps are different from one another. The colours on this original, aggregated coloured graph can now be renamed based on the hash value, see Figures 9 and 10. And if we continued iterating, we would continue to polish the colours of the graph and arrive at more and more refined results.

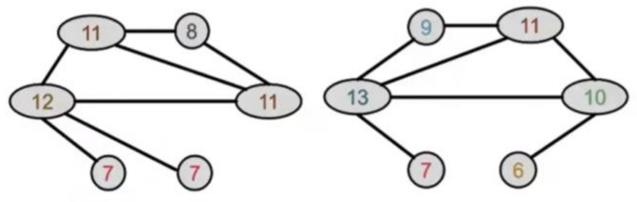
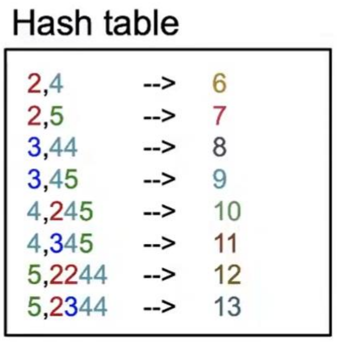


Fig. 9. Aggregated hash colors: Color refinement of 2 graphs.



Hash table

2,4	-->	6
2,5	-->	7
3,44	-->	8
3,45	-->	9
4,245	-->	10
4,345	-->	11
5,2244	-->	12
5,2344	-->	13

Fig. 10. Aggregated hash values.

The Weisfeiler-Lehman kernel, then, counts the number of nodes with a certain colour after we have ran this colour refining for a specified number of steps, let's say k iterations. We run this three times in our scenario, giving us 13 different colours, see Figures 11, 12, 13 and 14. And so, the number of nodes with a specific colour serves as the feature description for a given graph. In the initial iteration, each of the six nodes had the same colour, which was applied uniformly throughout. So, there are six occurrences of colour 1 in total.

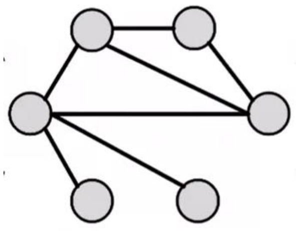
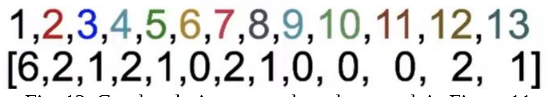


Fig. 11. Graph coloring, example 1.



1,2,3,4,5,6,7,8,9,10,11,12,13
[6,2,1,2,1,0,2,1,0, 0, 0, 2, 1]

Fig. 12. Graph coloring counts based on graph in Figure 11.

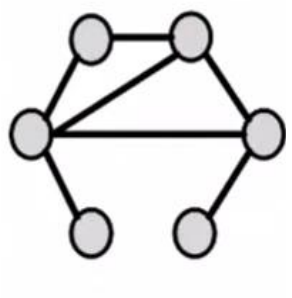
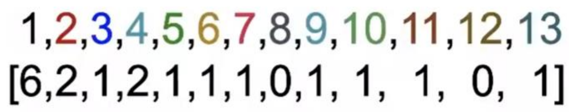


Fig. 13. Graph coloring, example 2.



1,2,3,4,5,6,7,8,9,10,11,12,13
[6,2,1,2,1,1,1,0,1, 1, 1, 0, 1]

Fig. 14. Graph coloring counts based on graph in Figure 13.

Then, once we iterated, aggregated, and refined the colours, there were two nodes for colour 2, one for colour 3, two for colour 4, and so forth. Following is an explanation of the characteristic in terms of colour counts for the first graph's various colours and the second graph's various colours. The Weisfeiler-Lehman graph kernel would then take the dot product between these two feature descriptors and return a value now that we know the feature descriptions. For instance, in our situation, the dot product of the feature descriptors is the Weisfeiler-Lehman kernel similarity between the two graphs.

When we compute the dot product of these two feature descriptors, we obtain a result of 49. Since the time complexity of this colour refining at each step is linear in the size of the graph, the WL kernel is both widely used and very powerful, giving strong performance. It is also computationally efficient. It has a linear number of edges since each node just has to collect colours from nearby nodes in order to produce new colours. This is done by applying a basic hash function to each collection of colours to create new colours.

Many hues that appeared in two graphs, need to be tracked when calculating the kernel value. Therefore, the maximum number of colours in the network will be equal to the number of nodes. Therefore, this time it won't be too big. Additionally, since it only involves a sweep over the nodes, counting the colours again takes linear time. Therefore, the complexity of computing the Weisfeiler-Lehman graph kernel between two graphs is just linear in their combined number of edges. This indicates that it is really quick and that it actually functions quite well in practice.

In order to summarise the graph level properties that we have explored, let's start with the idea of graph kernels. A graph kernel can be thought of as a bag of graphlets or a bag of colours. When we represent the graph as as a bag of graphlets, this is a very expensive representation because counting the graphlets takes exponentially more time as the size of the graph increases.

The Weisfeiler-Lehman kernel is based on this case-step colour refining process, which enriches and generates new node colours from the colours of the node's close neighbours. And as more iterations of this colour refinement are carried out, the node begins to collect colour data from remoter regions of the network. So, in this case, the graph is represented by a palette. This is effective in terms of computation. In addition to being closely related to graph neural networks, which we will cover later in this course, the time is linear in the size of the graph. Weisfeiler-Lehman is therefore a really good technique to gauge similarity between graphs, and in many instances, it is quite difficult to surpass.

IV. Weisfeiler-Lehman Graph Kernel Applications And Use Cases

We now discuss some recent applications and use cases for the Weisfeiler-Lehman kernel. Weisfeiler-Lehman graph kernels are still among the most popular graph kernels after more than ten years because of their exceptional predictability and temporal complexity. Researchers in [16] offer a modification of Weisfeiler-Lehman graph kernels that, in contrast to equality, considers a more subtle and natural degree of similarity across Weisfeiler-Lehman labels in order to get over this constraint. They demonstrate how the Wasserstein distance between certain vectors encoding Weisfeiler-Lehman labels can be used to determine the proposed similarity in an efficient manner. This and other facts lead to the obvious choice of using the Wasserstein k-means method to divide the vertices. They empirically show that our generalisation significantly outperforms this and other state-of-the-art graph kernels in terms of predictive performance on datasets that contain structurally more complex graphs beyond the typically considered molecular graphs, including the Weisfeiler-Lehman subtree kernel, one of the most well-known Weisfeiler-Lehman graph kernels.

The decoupling of kernel building and learning is one of a general kernel's key disadvantages. Common kernels for molecular graphs, like the WL subtree, which are based on the substructures of the molecules, treat all possible substructures as having the same relevance, which may not be practical in real-world applications. Researchers in [13] suggest a method in this research to learn the weights of subtree patterns within the framework of WWL kernels [21], the cutting-edge approach for a graph classification challenge. They provide an effective learning algorithm and derive a generalisation gap bound to illustrate its convergence in order to solve the computational problem on big scale data sets. They also show the efficiency of their suggested strategy for figuring out the weights of subtree patterns through trials on artificial and actual data sets.

The majority of widely used graph kernels define graph similarity in terms of shared substructures and are based on the idea of Haussler's R-convolution kernel [9]. In their study [17], researchers investigate graph filtrations to enrich these similarity measures: They can think about a graph at various granularities by using meaningful orders on the set of edges, which enable them to build a series of nested graphs. Tracking graph features during the duration of such graph resolutions is a crucial idea in their methodology. This makes it possible to compare features according to when and how long they appear in sequences, rather than just comparing their frequencies in graphs. The researchers suggest a family of graph kernels that take these feature existence intervals into account. Although any graph characteristic can be used with their method to produce efficient kernels, they focus especially on Weisfeiler-Lehman vertex labels. They demonstrate that, in terms of determining graph isomorphism, applying Weisfeiler-Lehman labels over specific filtrations strictly increases expressive power over the standard Weisfeiler-Lehman technique. Due to their near resemblance to the Weisfeiler-Lehman approach, this result actually directly produces more potent graph kernels based on such features and has implications for graph neural networks. In terms of predictive performance on several real-world benchmark datasets, they empirically validate the expressive capability of their graph kernels and demonstrate notable gains over state-of-the-art graph kernels.

The following [19] distinguishes a recent Wasserstein Weisfeiler-Lehman (WWL) Graph Kernel: displaying the distribution of a graph's Weisfeiler-Lehman (WL) embedded node vectors in a histogram that permits the calculation of the Wasserstein distance between two graphs. It has been demonstrated to generate classification results with higher accuracy than other graph kernels that do not use the Wasserstein distance and such distribution. The Isolation Graph Kernel (IGK), a substitute that assesses the similarity of two attributed graphs, is introduced in this study [22]. In two ways, IGK stands out from other graph kernels. First of all, it is the first graph kernel to use a distributional kernel within the kernel mean embedding architecture. By doing this, the computationally expensive Wasserstein distance is avoided. Second, it is the first graph kernel to take into account the distribution of attributed nodes in a collection of graphs while disregarding the edges. [22] demonstrate how important it is to extract this distributional data from an isolation kernel feature map in order to create a productive and successful graph kernel. They demonstrate that when utilised in the context of SVM classification, IGK performs orders of magnitude better than WWL in terms of classification accuracy and runs orders of magnitude faster in big datasets.

Methods for detecting DDoS attacks [3] are crucial for maintaining computer network security. The current flow-based DDoS attack detection techniques, however, have a non-negligible time delay and are not universal for different DDoS attack types occurring at different rates. A quick all-packets-based DDoS attack detection method (FAPDD) is suggested to close this research gap [10]. As opposed to flow-based detections, the FAPDD first creates a new time series network graph model to more efficiently simplify the processing of network traffic management. Additionally, the directed Weisfeiler-Lehman graph kernel is being developed for the first time in order to assess the divergence between the current network graph and the normalised network graphs. Different types and rates of DDoS attacks can be specifically detected thanks to the new graph model and kernel measuring approach to assess network changes. The dynamic threshold and freezing method are designed to display changes in regular traffic and stop attack traffic from contaminating the network. To assess the usefulness of the suggested method, the overall time efficiency, and the effectiveness of the detection, many genuine DDoS attack datasets are used. The FAPDD can more effectively meet real-time needs and provide good detection results in various DDoS attack types with various attack rates when compared to other approaches.

The development of kernels for structured data, such as graphs, is an important area of research since the effectiveness of kernel algorithms depends on the kernel it utilises. The optimal assignment kernel framework, which is a lesser-known graph framework than its counterpart, the R-convolution kernels, was used to create two graph kernels. The proposed kernels are known as "optimal node assignment kernels" [2] because in their work, the bijection associated with the optimal assignment framework is established between sets that comprise of the graph's nodes (ONA). On the basis of the labels produced by the Weisfeiler-Lehman (WL) test for graph isomorphism, the nodes of the provided data are categorised into neighbourhood sets in ONA, and a matrix representation is defined for them. The neighbourhood set domain is then described by a kernel in terms of such matrices, and an ONA kernel is defined using an aggregate measure of those kernel values. The proposed kernels' viability is mathematically demonstrated [15]. Also covered is the kernel calculation using the hierarchical framework linked to the optimal assignment framework. Graph classification datasets [6] were used to analyse the performance of the suggested ONA kernels, and it was discovered that they performed better than other cutting-edge graph kernels.

V. Conclusion

In this paper we talk about graph-level characteristics and graph kernels, which will enable us to forecast the behaviour of entire graphs. Therefore, the objective is to have a single feature that describes the overall graph's structure. We discussed graph kernels, specifically the graphlet kernel and the WL, which stands for the Weisfeiler-Lehman graph kernel. Its runtime scales only linearly in the number of edges of the graphs and the length of the Weisfeiler-Lehman graph sequence.

References

[1] Bittel, Lennart, and Martin Kliesch. "Training variational quantum algorithms is np-hard." Physical Review Letters 127.12 (2021): 120502.

[2] Doan, Khoa D., et al. "Interpretable graph similarity computation via differentiable optimal alignment of node embeddings." Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. 2021.

[3] Eliyan, Lubna Fayez, and Roberto Di Pietro. "DoS and DDoS attacks in Software Defined Networks: A survey of existing solutions and research challenges." Future Generation Computer Systems 122 (2021): 149-171.

[4] Huang, Ningyuan Teresa, and Soledad Villar. "A Short Tutorial on The Weisfeiler-Lehman Test and Its Variants." ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2021.

[5] Ju, Wei, et al. "GHNN: Graph Harmonic Neural Networks for semi-supervised graph-level classification." Neural Networks 151 (2022): 70-79.

[6] Le, Tuan, et al. "Parameterized hypercomplex graph neural networks for graph classification." International Conference on Artificial Neural Networks. Springer, Cham, 2021.

[7] Li, Shuchao, Wanting Sun, and Yuantian Yu. "Adjacency eigenvalues of graphs without short odd cycles." Discrete Mathematics 345.1 (2022): 112633.

[8] Liang, Fangzhou, and Yueming Lu. "Rooted Subtrees Recursive Neural Networks on Graphs." 2021 IEEE Sixth International Conference on Data Science in Cyberspace (DSC). IEEE, 2021.

[9] Liu, Kai, Lulu Wang, and Yi Zhang. "An Aligned Subgraph Kernel Based on Discrete-Time Quantum Walk." Asian Conference on Machine Learning. PMLR, 2021.

[10] Liu, Xinqian, et al. "A fast all-packets-based DDoS attack detection approach based on network graph and graph kernel." Journal of Network and Computer Applications 185 (2021): 103079.

[11] Long, Qingqing, et al. "Theoretically improving graph neural networks via anonymous walk graph kernels." Proceedings of the Web Conference 2021. 2021.

[12] Mei, Song, Theodor Misiakiewicz, and Andrea Montanari. "Generalization error of random feature and kernel methods: hypercontractivity and kernel matrix concentration." Applied and Computational Harmonic Analysis 59 (2022): 3-84.

[13] Nguyen, Dai Hai, Canh Hao Nguyen, and Hiroshi Mamitsuka. "Learning subtree pattern importance for Weisfeiler-Lehman based graph kernels." Machine Learning 110.7 (2021): 1585-1607.

[14] Park, Sun Woo, et al. "The PWLR graph representation: A Persistent Weisfeiler-Lehman scheme with Random Walks for graph classification." Topological, Algebraic and Geometric Learning Workshops 2022. PMLR, 2022.

[15] Salim, Asif, S. S. Shiju, and S. Sumitra. "Graph kernels based on optimal node assignment." Knowledge-Based Systems 244 (2022): 108519.

[16] Schulz, Till Hendrik, et al. "A generalized weisfeiler-lehman graph kernel." Machine Learning (2022): 1-29.

[17] Schulz, Till, Pascal Welke, and Stefan Wrobel. "Graph filtration kernels." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 36. No. 8. 2022.

[18] Sonawane, Sheetal S., Parikshit N. Mahalle, and Archana S. Ghotkar. "Text Analytics Using Graph Theory." Information Retrieval and Natural Language Processing. Springer, Singapore, 2022. 117-134.

[19] Togninalli, Matteo, et al. "Wasserstein weisfeiler-lehman graph kernels." Advances in Neural Information Processing Systems 32 (2019).

[20] Weis, James W., and Joseph M. Jacobson. "Learning on knowledge graph dynamics provides an early warning of impactful research." Nature Biotechnology 39.10 (2021): 1300-1307.

[21] Wijesinghe, Asiri, Qing Wang, and Stephen Gould. "A Regularized Wasserstein Framework for Graph Kernels." 2021 IEEE International Conference on Data Mining (ICDM). IEEE, 2021.

[22] Xu, Bi-Cun, Kai Ming Ting, and Yuan Jiang. "Isolation graph kernel." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 35. No. 12. 2021.

[23] Xu, Zhi, et al. "Structure-Preserving Hashing for Tree-Structured Data." Signal, Image and Video Processing (2022): 1-9.

[24] Zhou, Jingbo, et al. "Competitive Relationship Prediction for Points of Interest: A Neural Graphlet Based Approach." IEEE Transactions on Knowledge and Data Engineering (2021).