TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Chapter One

Context and related works

In this chapter, we introduce graph networks, how they arise from real world problems. Then we present the graph classification problem and the applications in which it can be used. After that, we proceed to state-of-the-art methods and its limitations, and we finish be stating our contribution.

1.1 Graph structures and its presence in real world

Formally, a graph of size v is a pair $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{u_1, ..., u_v\}$ is a set of graph nodes, and $\mathcal{E} \in \mathcal{V} \times \mathcal{V}$ is the set of edges between these nodes, i.e. $(u_i, u_j) \in \mathcal{E}$ means that the graph has an edge between node u_i and node u_j. Graph structures are used to represent a set of objects and the interactions/relations that link between different pairs of these objects. While the nature of these objects and their interactions vary depending on the application, they are still represented the same way where objects are reduced to nodes, and a relation between two objects is reduced to an edge between the corresponding two nodes. For example, Fig. 1.1 shows how we can represent chemical reactions mechanisms by graph structures.

The need to graph structures and its analysis can be traced back to 1679, when G.W. Leibniz wrote to C. Huygens about the limitations of traditional analysis methods of geometric figures

Network	Nodes	Node features	Edges	Edge features
Transportation System	cities	registered cars	Routes	Length, cost
Banking Network	Account holders	account status	Transactions	Transaction value
Social Network	users	name, country	Interactions	type (like, comment)

Table 1.1 Some real world graphs

and said that "we need yet another kind of analysis, geometric or linear, which deals directly with position, as algebra deals with magnitude", (P. Patel and C. Patel, 2013). Then it was the invent of graph structures, which provided a pictorial form to represent data instead of using a lot of words to do so. With graph structures, these data become easier to be processed and analyzed, and that helps solving many problems that have been unsolved before (P. Patel and C. Patel, 2013).

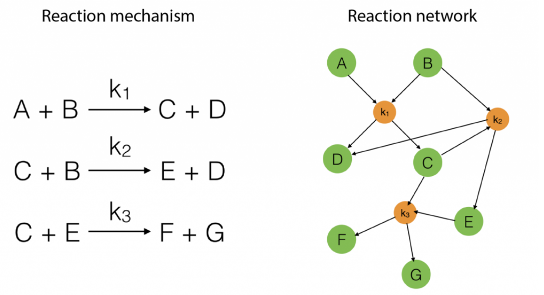
Reaction mechanism
A + B $\xrightarrow{k_1}$ C + D
C + B $\xrightarrow{k_2}$ E + D
C + E $\xrightarrow{k_3}$ F + G
Reaction network
A
B
k1
k2
D
C
k3
E
F
G

Figure 1.1 Graph structures in representing chemical reactions mechanisms

To better understand how graphs are built in real life, we show in Table.1.1 some examples of these graphs in different domains.

1.2 Graph classification problem

Let's consider the case when we have a set $\mathcal{D}$ of all valid graphs $\mathcal{G}$ that can occur in some real-world domain, a fixed set of classes/categories $\beta = \{\beta_1, \dots, \beta_l\}$ of finite size l, and a mapping function $f : \mathcal{D} \mapsto \beta$ which maps each graph $\mathcal{G}$ in $\mathcal{D}$ to the class $\beta_{\mathcal{G}}$ it belongs to. Graph classification is the problem of estimating the mapping function f in the case where it is only known on a subset $\mathcal{X} \subset \mathcal{D}$. Formally, we have a dataset $(\mathcal{X} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}, \mathcal{Y} = \{y_1, \ldots, y_n\})$ of size n such that $\mathcal{X} \in \mathcal{D}^n$ and $\mathcal{Y} \in \beta^n$, where for each graph $\mathcal{G}_i \in \mathcal{X}$ we have that $y_i = f(\mathcal{G}_i)$ is the class of $\mathcal{G}_i$, and that is why it is called labeled dataset. The classification task is to have a **predictive model** which can predict well, based on some-predefined metric, the class for any new graph $\mathcal{G}$ in $\mathcal{D}$ not only in $\mathcal{X}$. This prediction functionality of the model is gained using the dataset $(\mathcal{X}, \mathcal{Y})$ to optimize the parameters of the model that is believed to govern the behavior of the mapping function f on $\mathcal{D}$. This optimization completed in this paradigm is called the learning algorithm.

In some applications each node/edge in the graph dataset has its own features vector that can be used along with the graph structure to classify different graphs into different classes, while in other applications all we have is the graph structure or it happens that some nodes/edges have their features but others don't. In this project, we consider the one case of graph classification with the absence of any node/edge features. In practice, this problem is drastically being addressed in many fields, as in:

- ***Marketing analytics:*** advertisers and marketers are interested in detecting the influential people communities in Social Networks in the sense that addressing their products' advertisements to such groups would be a more rewarding investment. This can be approached with graph classification applied on these networks (Hill, Provost, Volinsky, et al., 2006).

- ***Banking security:*** graph classification is used to catch unusual patterns of fraudulent transactions (Lebichot et al., 2016).

- ***Biology and genomics:*** graphs are based on proteins such that nodes correspond to amino acids which compound the protein and a pair of amino acids are linked by an edge if they are less than 6 Angstroms apart. The task is to detect whether a protein is an enzyme or not (Nikolentzos, Meladianos, and Vazirgiannis, 2017), to mention a few.

1.3 State-of-the-art methods for graph classification

We present here the traditional algorithms deployed in graph classification problem, we simultaneously state the limitations of each of them. In general, these algorithms can be traced back into four main categories: Set based, frequent sub-graph based, kernel based, and graph neural networks based algorithms.

Set based algorithms: this type of algorithms is particularly applied on graphs whose nodes/edges are supplied with features or attributes, so each graph is reduced to a set of nodes, edges or both. Then a distance function of interest between the graphs is computed based on the similarity between pairs of edges/nodes in the corresponding sets. The drawback of this method is that it does not take the structure (topology) of the graph itself into consideration. For example, if we just compare how much the edges of one graph are similar to the ones of another, we can have two graphs with the same set of edge features, which will lead to maximum similarity, but we in reality ignore other important information that can make these graphs completely different such that how many connected communities of nodes these edges form in each graph, how many circles of nodes these edges promote in each graph, etc. On the other hand, a strength point of these algorithms compared to others is the low computations cost that is usually linear or quadratic in the number of nodes and edges (Shervashidze et al., 2009).

Frequent sub-graph based algorithms: these algorithms can be applied in two stages, the graph dataset (all the graphs we have in an application) is first analyzed to pick the frequent sub-graphs that occur in different graphs. Then, another analysis is done to choose the most discriminative sub-graphs out of the ones resulted from the first stage. The disadvantage of using this method is the computational cost that grows exponentially with the graph size (Shervashidze et al., 2009).

Graph kernels based algorithms: it is a middle ground between both previous methodologies, where the graph structure (topology) is well considered, and in most cases, these algorithms are designed in a way that the computational time is a polynomial function of the graph size (Shervashidze et al., 2009). However, some effective and competitive kernels still require exponential time, and this is in short the problem we approached in this work using random features to approximate these kernels or to compete them in notably lower computational time.

Graph neural networks (GNNs) based algorithms: GNNs revolutionized learning on graph structures, as it computes a representation vector (embedding vector) for every node in the graph, where this vector is recursively computed by aggregating the representation vectors of neighbor nodes. The goal of this aggregation technique is that nodes that are neighbors (or even close) to each other in the graph are more likely to have a similar representations (with respect to some similarity function) and vise versa. On the graph level, a representation vector is computed by aggregating its nodes representation vectors, and then this vector is used as the features vector which can be fed to a typical deep neural network to learn the classification task. Traditional GNNs such as graph convolutional networks (GCNs) and GraphSAGE failed to provide high performance classifying graphs whose node/edges don't include any original features vectors, and that even applies on graphs with simple topology (Xu et al., 2018). However, another GNN structure was developed to overcome this weakness point, and it is referred to by Graph Isomorphism Network (GIN). Regarding the computational time, it is mainly a matter of the layers number in the network, since this parameter in reality represents how far from a node we want to go in order to compute its representation vector.

1.4 Context and our contribution

K-graphlet kernel is one of the aforementioned graph kernels which has proven to be competitive in graph classification. Theoretically and empirically, it was shown that a desired performance or a required amount of information to be preserved from the original graph can be reached with sufficiently large k. However, the computational cost ecomes prohibitive as (k: the graphlet size) and/or (v: the size of the graph to be sampled) become too large. Thus it cannot be applied on large-scale graph datasets. The advent of Optical Processing Units (OPUs) opened a new horizon solving this problem, since it can apply enormous number of *Random Projections* in light speed.

In this work, we did the sufficient mathematical analysis to prove that OPUs' light-speed random feature projections compete the k-graphlet kernel with respect to *Maximum Mean Discrepancy (MMD)* Euclidean metric. Moreover, we empirically tested this hypothesis and made sure that the the theoretical MMD error is aligned with the empirical one with respect to the parameters introduced in the problem (sampling technique, number of sampled sub-graphs, number of random features, etc).

Chapter Two

Preliminaries and necessary background

In this chapter we present the necessary background that the reader should have in order to proceed through the next two chapters, which are the core of this work. Here, we show the analysis done separately in graph kernels and in random features. However, we show our analysis combining these different notions in one algorithm in the next chapter.

2.1 Kernel trick and random features

2.1.1 Kernel trick

Kernel trick promotes the use of positive semi-definite kernels $\mathcal{K}(x, x')$ in learning models as a similarity function between data points (x, x'). Where having $\mathcal{D}$ as the set of all possible data points, a function $\mathcal{K} : \mathcal{D} \times \mathcal{D} \mapsto \mathbb{R}$ is said to be a positive semi-definite kernel if:

- $\forall (x, x') \in \mathcal{D} \times \mathcal{D}, \mathcal{K}(x, x') = \mathcal{K}(x', x)$
- $\forall n \in \mathbb{N}, \forall \alpha_1, \ldots, \alpha_n \in \mathbb{R}, \forall x_1, \ldots, x_n \in \mathcal{D}, \sum_{i,j}^{n} \alpha_i \alpha_j \mathcal{K}(x_i, x_j) \geq 0$

We show now how kernels can be incorporated within learning models and how that is useful to learn more complex functions. To do that let's consider the specific case when $\mathcal{D} = \mathbb{R}^d; d \in \mathbb{N}$, and let $(\mathbf{X}, \mathbf{Y})$ be a labeled dataset of size n with $\mathbf{X} \in \mathbb{R}^{n \times d}$ and for simplicity $\mathbf{Y} \in \{-1, 1\}^n$, which is referred to as 2-classes dataset. Many of the learning models designed to solve this problem, like support vector machine and perceptron binary classifier, rely on the inner product as a measure of similarity between data points, as their output has the following formula (Kim and Scott, 2009):

$$\hat{y}(\mathbf{x}) = sign\{\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i^T \mathbf{x}\}; \quad \forall i, (\mathbf{x}_i, y_i) \in (\mathbf{X}, \mathbf{Y}), \alpha_i \in \mathbb{R} \tag{2.1}$$

The values $[\alpha_i]_{i=1}^n$ in Eq. 2.1 are optimized based on the dataset $(\mathbf{X}, \mathbf{Y})$ by the learning algorithm. The intuition behind Eq. 2.1 is that the output class for every new data point x is expected to be the same class of nearby points in the input set $\mathcal{D}$. This is achieved by introducing the inner product $\mathbf{x}_i^T\mathbf{x}$ to control how much the class y_i contributes in the output $\hat{y}(\mathbf{x})$. The parameters $[\alpha_i]_{i=1}^n$ controls how strongly can the train data point x_i affect other neighbor points, and they mainly depend on how both classes are distributed in the input set $\mathcal{D}$, and on how much the dataset $(\mathbf{X}, \mathbf{Y})$ is noisy.

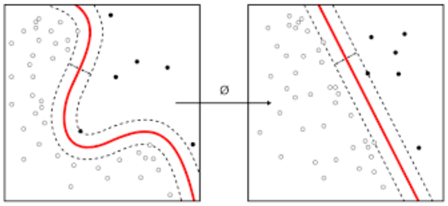

Figure 2.1 The left figure shows a case where the input data in their original space are not separable by a linear boundary. The right figure shows the same data transformed to a new space using a lifting function ø, and we can see that different classes are now separable using linear boundary.

However, with Eq. 2.1 can be rewritten as $\hat{y}(\mathbf{x}) = sign\{\mathbf{x}^T(\mathbf{X}^T\ diag([\alpha]_{i=1}^n)\ \mathbf{Y})\}$, where $diag([\alpha]_{i=1}^n)$ is the diagonal square matrix whose eigenvalues are $[\alpha]_{i=1}^n$, and $\mathbf{X}$ are the data points in the dataset. To get the decision boundary of such models we solve $\mathbf{x}^T\mathbf{q} = 0$, where $\mathbf{q} = (\mathbf{X}^T\ diag([\alpha]_{i=1}^n)\ \mathbf{Y}) \in \mathbb{R}^d$, so what we get is that it is an equation of a hyper-plane in the input space $\mathbb{R}^d$, and we refer to this as a linear decision boundary. The question here is what if the the two classes are not separable by a hyper-plane as shown in Fig. 2.1. One common solution to this problem is to map the data points from $\mathbb{R}^d$ to another space $\mathbb{R}^m$ through a proper mapping function ø such that classes become separable with a linear decision boundary in $\mathbb{R}^m$.Then we can apply the same learning models specified in Eq. 2.1 but on the transformed data. Let's consider for example the dataset shown in Fig. 2.2, we can use the following mapping function ø: $(x_1, x_2) \mapsto (\sqrt{2}x_1x_2, x_1^2, x_2^2)$ to move from $\mathbb{R}^2$ on the left where data are not linearly separable to $\mathbb{R}^3$ on the right where they are.

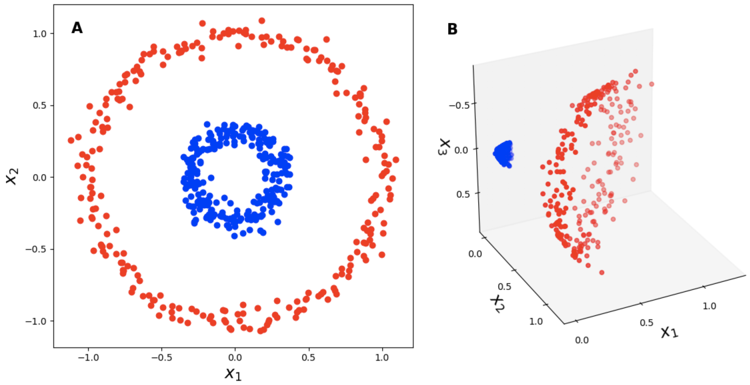

Figure 2.2 Using the mapping function ø: $(x_1, x_2) \mapsto (\sqrt{2}x_1x_2, x_1^2, x_2^2)$ to map the data on the left in $\mathbb{R}^2$ to $\mathbb{R}^3$ where they are linearly separable

Now the role of kernels can tackled here, but first we state the key theorem that justifies what follows.

Theorem 1 (Mercer theorem)**.** *To every positive semi-definite kernel* $\mathcal{K} : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$*, there exists a Hilbert space* $\mathbb{H}$ *and a feature map* $\phi : \mathbb{R}^d \mapsto \mathbb{H}$ *such that for all* $x, x' \in \mathbb{R}^d$ *we have:*

$$\mathcal{K}(\mathbf{x}, \mathbf{x}') =< \phi(\mathbf{x}), \phi(\mathbf{x}') >_{\mathbb{H}} \tag{2.2}$$

where $< \phi(\mathbf{x}), \phi(\mathbf{x}') >_{\mathbb{H}}$ *is the inner product defined in* $\mathbb{H}$*.*

This theorem state that replacing the inner product $\mathbf{x}_i^T\mathbf{x}$ in Eq. 2.1 by a positive semi-definite kernel $\mathcal{K}(\mathbf{x}, \mathbf{x}')$ is equivalent to implicitly map the data from the original input space $\mathcal{D}$ to another feature space $\mathbb{H}$ and then apply the classical inner product learning models. This is important because using such kernels, it is not important to know the implicit mapping function ϕ nor the new feature space $\mathbb{H}$. Instead, it is sufficient to evaluate the kernel for pairs of data points in the original input space $\mathcal{D}$. Indeed, that has two main advantages:

- Kernels allow us to transform data to a new Hilbert space of very high or even infinite dimensionality, which can make the learning model able to represent more complex functions. This is not possible with the explicit mapping.
- Kernels are computationally cheaper, since they save the time required to compute the explicit co-ordinates of the data in the new feature space by directly calculate the inner product between the transformed data.

To better illustrate these benefits, we take the Gaussian kernel as an example, where:

$$\mathcal{K}_G(\mathbf{x}, \mathbf{x'}) = exp(-\frac{\|\mathbf{x} - \mathbf{x'}\|^2}{2\sigma^2}) \tag{2.3}$$

where σ is called the bandwidth parameter of the kernel. The lifting function ϕ_G of this kernel is located in a Hilbert space of infinite dimension, but the kernel can be easily evaluated for any pair $(\mathbf{x}, \mathbf{x'}) \in \mathcal{D} = \mathbb{R}^d$.

Despite kernel methods, also referred to as kernel machines, have the previous advantages, they still have two drawbacks:

- Since for most kernels we have access only to evaluate the kernel on pairs of data points, then for a dataset $(\mathbf{X}, \mathbf{X})$ of size n, we need $O(n^2)$ memory entries to compute what is called Gram matrix. Where the $(i, j)_{th}$ entry equals the kernel between points $(\mathbf{x_i}, \mathbf{x_j}) \in \mathbf{X}$.
- Even that most kernels are designed so that they can be evaluated in polynomial time in the dimensionality of the input space $\mathcal{D}$, still some of the efficient kernels needs exponential time in the same dimensionality (Shervashidze et al., 2009).

To overcome these disadvantages, random feature projections discussed in the next section is a technique developed to approximate these kernels with less computational time and less memory storage.

2.1.2 Random features

Random features is a method developed to approximate kernel machines whose computational time scales exponentially in the input dimensionality, and of course the goal is to reduce this time. The idea is that instead of considering the true lifting function ϕ in Eq. 2.2, we explicitly map the data points to an Euclidean space of low dimensionality. This mapping is done using an appropriate randomized feature map $\varphi : \mathcal{D} \rightarrow \mathbb{R}^m$. Then we approximate the kernel of two data points x, x' by the inner product of their random features:

$$\mathcal{K}(x, x') =< \phi(x), \phi(x') >\approx \varphi(x)^* \varphi(x') \tag{2.4}$$

Considering this approximation, we can transform the input with φ and then apply a fast linear learning method as in Eq. 2.1 to have a similar learning power as the original non-linear kernel machine. In what follows, random Fourier features method to construct the random mapping function φ is presented.

Theorem 2 (Bochner's theorem)**.** *A continuous and shift-invariant kernel* $\mathcal{K}(x, x') = \mathcal{K}(x - x')$ *on* $\mathbb{R}^d$ *is positive definite if and only if* $\mathcal{K}(\delta)$ *is the Fourier transform of a non-negative measure.*

Direct consequence, we can easily scale a shift-invariant kernel so that its Fourier transform $p(w)$ is a correct probability distribution, since it is non-negative measure and integral-bounded function, and we write:

$$\mathcal{K}(x - x') = \int_{\mathbb{R}^d} p(w) e^{jw^T(x-x')} dw = E_w[e^{jw^Tx} e^{jw^Tx'^*}] \tag{2.5}$$

Both $p(w)$ and $\mathcal{K}(\delta)$ are real-valued functions, thus from Eq. 2.5 one can prove that:

$$\mathcal{K}(x - x') = \int_{\mathbb{R}^d} p(w) cos(w^T(x - x'))dw = E_w[\xi_w(x)\xi_w(x')] \tag{2.6}$$

where $\varphi_w(x) = \sqrt{2} cos(w^T x + b)$ such that w is drawn from $p(w)$ and b is drawn uniformly from $[0, 2\pi]$.

As a result, $\xi_w(x)\xi_w(x')$ is an unbiased estimate of $\mathcal{K}(x, x')$. Moreover, we can achieve lower variance estimation to the expectation (Eq. 2.6) by averaging m instances of the estimator with different random frequencies w, *i.e.* the low-variance estimator can be written as: $\varphi(x)^T\varphi(x') = \frac{1}{m}\Sigma_{j=1}^{m}\xi_{w_j}(x)\xi_{w_j}(x')$ where frequencies w_j are identically and independently drawn from $p(w)$. This estimator and based on Hoeffding's inequality guarantees exponentially fast convergence in m between $\varphi(x)^T\varphi(x')$ and the kernel true value:

$$Pr(|\varphi(x)^T\varphi(x') - \mathcal{K}(x, x')| \geq \epsilon) \leq 2e^{\frac{-m\epsilon^2}{4}}; \quad \forall\epsilon > 0 \tag{2.7}$$

To better illustrate that, we consider the Gaussian kernel in Eq. 2.3 as an example. This kernel is a function of the difference between both input points $\mathcal{K}_G(x, x') = f(x - x')$ and it is a positive semi-definite kernel. If we multiply this kernel with the constant $\frac{1}{2\pi}^{d}$ where d is the dimensionality of the input data, then its Fourier transformation is a correct probability distribution:

$$p(w) = FT\big(\mathcal{K}_G(\delta)\big)(w) = (\frac{\sigma^2}{2\pi})^{\frac{d}{2}}e^{-\frac{\sigma^2\|w\|^2}{2}}$$

Where $\delta \in \mathbb{R}^d$ and $w \in \mathbb{R}^d$.

2.2 Graph kernels

We first introduce necessary notations related to graph definition and graph kernels.

$F = (\mathcal{V}_F, \mathcal{E}_F)$ is said to be a subgraph (graphlet) of $\mathcal{G}$ and we write $F \sqsubseteq \mathcal{G}$, if and only if there exist an injective function $\mathcal{M} : \mathcal{V}_F \rightarrow \mathcal{V}$ such that $(u, u') \in \mathcal{E}_F \Leftrightarrow (\mathcal{M}(u), \mathcal{M}(u')) \in \mathcal{E}$.

Any edge (u_i, u_i) is called a self loop. In a general graph two vertices u_i and u_j may be connected by more than one edge. A simple graph is a graph with no self loops or multiple edges. Here we always consider simple graphs.

A simple graph can equivalently be represented by an adjacency matrix $\mathbf{A}$ of size $v \times v$. The $(i, j) - th$ entry of $\mathbf{A}$ is 1 if an edge (u_i, u_j) exists and zero otherwise.

Two graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ are isomorphic and we write $\mathcal{G}' \cong \mathcal{G}$ if there exists a bijective function $\mathcal{M} : \mathcal{V} \rightarrow \mathcal{V}'$ such that $(u_i, u_j) \in \mathcal{E}$ iff $(\mathcal{M}(u_i), \mathcal{M}(u_j)) \in \mathcal{E}'$.

2.2.1 Convolutional graph kernels

Traditional kernel machines approach problems with vector-valued input data, where they compare different data points ($x, x' \in \mathcal{R}^d$) using the difference between correspondent pairs of vector entries. Based on that, these kernels are imperfect to be used with graphs, since the graph structure is permutation invariant, i.e. isomorphic graphs have different adjacency matrices but they represent the same structure. So in this case distance-kernels between graph representation vectors (adjacency matrices for example) are not a good choice. As a result it is necessary to measure distance between graphs in ways that are permutation invariant as well. Here the concept of isomorphism is critical in learning algorithms on graphs, not only because there is no known polynomial-time algorithm for testing graph isomorphism (except for graphs with specific structures), but isomorphism is also too strict for learning in a similar way to learning with equality operator (Kriege, Johansson, and Morris, 2020).

Most of graph kernels developed for graph learning problems are convolution kernels, where given two graphs, the trick is to divide each into smaller subgraphs and then to pairwise compute the kernel between the resulted subgraphs.

Definition 1 (Convolution Kernel)**.** *let $\mathcal{R} = \mathcal{R}_1 \times ... \times \mathcal{R}_d$ denote a space of components such that a composite object $X \in \mathcal{X}$ decomposes into elements of $\mathcal{R}$. Let $R : \mathcal{R} \rightarrow \mathcal{X}$ denote the mapping from components to objects, such that $R(x) = X$ iff the components $x \in \mathcal{R}$ make up the object $X \in \mathcal{X}$, and let $R^{-1}(X) = \{x \in \mathcal{R} : R(x) = X\}$. then, the R-convolution kernel is:*

$$K_{CV}(X, Y) = \sum_{x \in R^{-1}(X)} \sum_{y \in R^{-1}(Y)} \underbrace{\prod_{i=1}^{d} k_i(x_i, y_i)}_{k(x,y)} \tag{2.8}$$

with k_i is a kernel on $\mathcal{R}$ for $i \in \{1, ..., d\}$.

Applying this definition on graphs, $R^{-1}(\mathcal{G} = (\mathcal{V}, \mathcal{E}))$ includes all the components in graph $\mathcal{G}$ that we want to compare with the components $R^{-1}(\mathcal{G}' = (\mathcal{V}', \mathcal{E}'))$ in graph $\mathcal{G}'$. One example of these kernels is the node label kernel, where for two graphs $\mathcal{G}, \mathcal{G}'$, the mapping function R maps the features $x_u \in \mathcal{R}$ of each node $u \in \mathcal{V} \cup \mathcal{V}'$ to the graph that u is a member of. Another example that is mainly related to our work is the k-graphlet kernel, where R here maps the subgraphs of size k to the graph in which it occur. The advantage of using convolution kernel framework with graphs is that kernels are permutation invariant on the graphs level as long as they are permutation invariant on the components level. As a drawback, the sum in Eq. 2.8 iterates over every possible pair of components. As a result, when we choose our components to be more specific such that the kernel value is high between a component and itself while it is low between two different components, each graph becomes drastically similar to itself but distant from any other graph. Thus, a set of weights is usually added so this problem is resolved.

2.2.2 Graphlet Kernel

In general, we have two sets that can be referred to as size-k graphlets. The first set $\mathcal{H} = \{H(1), ..., H(N_k)\}$ contains all graphs of size k and treat isomorphic graphs as different graphs, thus we have here $N_k = 2^{k(k-1)/2}$ different graphlets. The other set $\mathcal{H} = \{H(1), ..., H(\bar{N_k})\}$ treat all isomorphic graphs of size k as one graph, so we have here $\bar{N_k} < N_k$ but it is still exponential in k. For either set, we define for a graph G the vector $f_G \in \mathcal{R}^{N_k}$, the i-th entry equals the normalized-number of occurrences of $graphlet(i)$ in G $(\#(graphlet(i) \sqsubseteq G))$. f_G is usually referred to by the k-spectrum of G, and this vector is the key idea behind graphlet kernel.

Definition 2 (Graphlet Kernel). *Given two graphs $\mathcal{G}$ and $\mathcal{G}'$ of size $v_{\mathcal{G}}, v_{\mathcal{G}'} \geq k$, the graphlet kernel $\mathcal{K}_g$ is defined as (Shervashidze et al., 2009):*

$$\mathcal{K}_g(G, H) = f_{\mathcal{G}}^T f_{\mathcal{G}'}. \tag{2.9}$$

Which naturally involves an associated Euclidean metric $d_{\mathcal{K}}(\mathcal{G}, \mathcal{G}') = \|f_{\mathcal{G}} - f_{\mathcal{G}'}\|_2$. The drawback of this kernel is that computing the k-spectrum vector costs huge computational time, where regardless the cost of isomorphism test if it is required, there are $O\binom{v}{k}$ subgraphs of size k in a graph $\mathcal{G}'$ of size v. As a result, there is a trade off between a more accurate representation of the graph (larger value of k) and the computational cost. However, some techniques are used in order to resolve this limitation as sampling from graph technique (section 2.2.3).

2.2.3 Graph sampling to approximate k-graphlet spectrum

The problem of graph sampling arises when we deal with a large-scale graph and the task is to pick a small-size sample subgraphs that would be similar to the original graph with respect to some important properties.

Sampling from graph techniques are used to resolve the processing cost limitation of graphlet kernel, and it can be done by directly sample s sugraphs $\{F_1, ..., F_s\}$ of size k, and then estimate the k-spectrum vector empirically: $f_{\mathcal{G}}(i) = \frac{1}{s}\Sigma_{j=1}^{s}\mathbb{1}[f_j = H_i]$

Uniform sampling Random walk sampling

to be continued . . .

REFERENCES

Hill, Shawndra, Foster Provost, Chris Volinsky, et al. (2006). “Network-based marketing: Identifying likely adopters via consumer networks”. In: *Statistical Science* 21.2, pp. 256–276.

Kim, JooSeuk and Clayton Scott (2009). “L2 Kernel Classification”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.10, pp. 1822–1831.

Kriege, Nils M, Fredrik D Johansson, and Christopher Morris (2020). “A survey on graph kernels”. In: *Applied Network Science* 5.1, pp. 1–42.

Lebichot, Bertrand et al. (2016). “A graph-based, semi-supervised, credit card fraud detection system”. In: *International Workshop on Complex Networks and their Applications*. Springer, pp. 721–733.

Nikolentzos, Giannis, Polykarpos Meladianos, and Michalis Vazirgiannis (2017). “Matching node embeddings for graph similarity”. In: *Thirty-First AAAI Conference on Artificial Intelligence*.

Patel, Pranav and Chirag Patel (2013). “Various graphs and their applications in real world”. In: *International Journal of Engineering Research and Technology* 2.12, pp. 1499–1504.

Shervashidze, Nino et al. (2009). “Efficient graphlet kernels for large graph comparison”. In: *Artificial Intelligence and Statistics*, pp. 488–495.

Xu, Keyulu et al. (2018). “How powerful are graph neural networks?” In: *arXiv preprint arXiv:1810.00826*.