Deep Graph Kernels

Pinar Yanardag
Department of Computer Science
Purdue University
West Lafayette, IN, 47906, USA
ypinar@purdue.edu

S.V.N. Vishwanathan
Department of Computer Science
University of California
Santa Cruz, CA, 95064, USA
vishy@ucsc.edu

ABSTRACT

In this paper, we present Deep Graph Kernels, a unified framework to learn latent representations of sub-structures for graphs, inspired by latest advancements in language modeling and deep learning. Our framework leverages the dependency information between sub-structures by *learning* their latent representations. We demonstrate instances of our framework on three popular graph kernels, namely Graphlet kernels, Weisfeiler-Lehman subtree kernels, and Shortest-Path graph kernels. Our experiments on several benchmark datasets show that Deep Graph Kernels achieve significant improvements in classification accuracy over state-of-the-art graph kernels.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - Data Mining; I.2.6 [**Artificial Intelligence**]: Learning; I.5.1 [**Pattern Recognition**]: Model - Statistical

Keywords

R-convolution kernels, graph kernels, deep learning, structured data, string kernels, social networks, bioinformatics

1. INTRODUCTION

In domains such as social networks, bioinformatics, chemoinformatics and robotics, we are often interested in computing similarities between *structured objects*. Graphs, including sequences and trees as special cases, offer a natural way to represent structured-data. To illustrate one example where graph similarity can be useful, consider the problem of identifying a sub-community (also referred as *subreddits*) on Reddit[1]. To tackle this problem, one can represent an online discussion thread as a graph where nodes represent users, and edges represent whether two users interact, for instance, by responding to each other's comments (see Figure 1[2]). Then, the task is to predict which sub-community a discussion thread belongs to based on its communication graph. Similarly, in bioinformatics, one might be interested in the problem of identifying whether a given protein is an enzyme or not. In this case, the secondary structure of a protein is represented as a graph where nodes correspond to atoms and edges represent the chemical bonds between atoms. If the graph structure of the protein is similar to known enzymes, one can conclude that the given graph is also an enzyme [33]. Therefore, computing semantically meaningful similarities between graphs is an important problem in various domains.

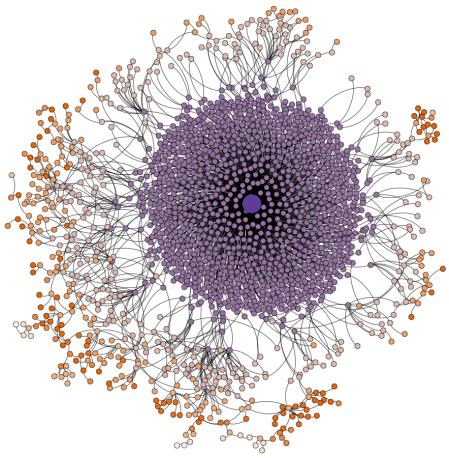


Figure 1: A graph of a random post on http://reddit.com/r/askreddit.

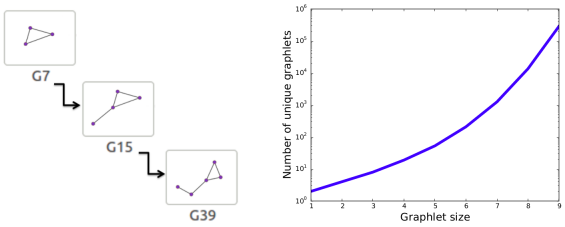
One of the increasingly popular approaches to measure the similarity between structured objects is to use kernel methods. Roughly speaking, kernel methods measure the similarity between two objects with a kernel function which corresponds to an inner product in reproducing kernel Hilbert space (RKHS) [26]. The challenge for kernel methods is then to find a suitable kernel function that captures the semantics of the structure while being computationally tractable. R-convolution [11] is a general framework for handling discrete objects where the key idea is to recursively decompose structured objects into "atomic" sub-structures and define valid local kernels between them. In the case of graphs, given a graph $\mathcal{G}$, let $\phi(\mathcal{G})$ denote a vector which contains counts of atomic sub-structures, and $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ denote a dot product in a RKHS $\mathcal{H}$. Then, the kernel between two graphs $\mathcal{G}$ and $\mathcal{G}'$ is given by

$$\mathcal{K}(\mathcal{G}, \mathcal{G}') = \langle \phi(\mathcal{G}), \phi(\mathcal{G}') \rangle_{\mathcal{H}}. \quad (1)$$

[1]Reddit is a popular content-aggregation website: http://reddit.com.
[2]Image is created with Gephi: http://gephi.github.io.

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.

ACM 978-1-4503-3664-2/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2783258.2783417.



(a) Dependency of sub-structures (b) Exponential growth

Figure 2: Dependency schema of a set of graphlets of size $k \in \{3, 4, 5\}$ where G_{39} can be derived from G_{15} (similarly, G_{15} can be derived from G_7) by adding a new node and an edge (a). Exponential growth of feature space in graphlets up to size $k = 9$ (b).

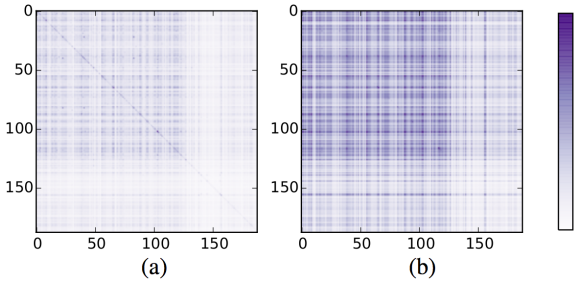


Figure 3: Kernel matrix K for Weisfeiler-Lehman subtree kernel on the MUTAG dataset [7] (a), and its deep variant obtained from our framework (b). Entry K_{ij} encodes the similarity between graph $\mathcal{G}_i$ and graph $\mathcal{G}_j$ and the color map encodes the degree of the similarity (darker color indicates higher similarity).

However, this representation does not take a number of important observations into account. First, sub-structures that are used to compute the kernel matrix are not independent. Let us consider an example on *graphlets*, a popular sub-structure type that is used for decomposing graphs [23, 28], which are defined as induced, non-isomorphic sub-graphs of size k (see Figure 4). Graphlets exhibit a strong dependence relationship, that is, size $k+1$ graphlets can be derived from size k graphlets by addition of nodes or edges (similarly, size k graphlets can be recovered by deletion of nodes or edges from size $k+1$ graphlets). For instance, G_{39} can be derived from G_{15} by adding a new node and an edge (see Figure 2 (a)). Second, the dimension of the feature space often grows exponentially. Figure 2 (b) illustrates the growth of the number of unique features in graphlets as graphlet size k increases. Consequently, as the number of features grows, we encounter the sparsity problem: only a few sub-structures will be common across graphs. This leads to *diagonal dominance*, that is, a given graph is similar to itself but not to any other graph in the dataset. Figure 3 (a) illustrates such a kernel matrix[3] where diagonal dominance is visibly observable using Weisfeiler-Lehman subtree kernel [27]. Ideally, we would like to have a kernel matrix where all entries belonging to a class are similar to each other, and dissimilar to everything else (see Figure 3 (b)). To alleviate this problem, consider an alternative kernel between two graphs $\mathcal{G}$ and $\mathcal{G}'$ such that

$$\mathcal{K}\left(\mathcal{G}, \mathcal{G}'\right) = \phi\left(\mathcal{G}\right)^T \mathcal{M} \phi\left(\mathcal{G}'\right) \tag{2}$$

where $\mathcal{M}$ represents a $|\mathcal{V}| \times |\mathcal{V}|$ positive semi-definite matrix that encodes the relationship between sub-structures and $\mathcal{V}$ represents the vocabulary of sub-structures obtained from the training data. Therefore, one can design an $\mathcal{M}$ matrix that respects the similarity of the sub-structure space. In cases where there is a strong mathematical relationship between sub-structures, such as *edit-distance*, one can design an $\mathcal{M}$ matrix that respects the geometry of the space. In cases where a clear mathematical relationship between sub-structures might not exist, one can *learn* the geometry of the space directly from data. In this paper, we propose recipes for designing such $\mathcal{M}$ matrices for graph kernels. For our first recipe, we exploit an edit-distance relationship between sub-structures and directly compute an $\mathcal{M}$ matrix. In our second recipe, we propose a framework that computes an $\mathcal{M}$ matrix by *learning* latent representations of sub-structures. Our contributions are as follows:

- We propose a general framework that *learns* hidden representations of sub-structures used in graph kernels, inspired by latest advancements in natural language processing and deep learning,
- We demonstrate our framework on three popular graph kernels, namely Graphlet kernels, Weisfeiler-Lehman subtree kernels, and Shortest-Path kernels and achieve significant improvements on several benchmark datasets,
- We discuss the connection of our framework to R-convolution kernels and apply our framework to derive deep variants of *string kernels*,
- We introduce several new large graph kernel datasets in social network domain associated with novel tasks such as *community prediction*.

The rest of this paper is as follows. In Section 2, we review three popular families of graph kernels for which our framework is applicable. In Section 3.1, we design an $\mathcal{M}$ matrix for graphlet kernels by exploiting edit-distance relationship between sub-structures. In Section 3.2, we introduce deep graph kernel framework and discuss the connection of our framework to R-convolution kernels. In Section 4, we discuss related work. In Section 5, we compare the classification performance of deep graph kernels to their base variants as well as to other state-of-the-art graph kernels. We report results on classification accuracy on graph benchmark datasets and discuss the run-time cost of our framework. Section 6 concludes the paper.

2. GRAPH KERNELS

We first introduce basic concepts and notation that will be used throughout the paper. Then, we discuss three major graph kernel families, namely, graph kernels based on limited-sized subgraphs [13, 28], graph kernels based on subtree patterns [24, 27], and graph kernels based on walks [14, 33] and paths [3]. We briefly discuss each of the above kernels, and recap how they can be viewed as instances of the more general R-convolution framework [11].

[3] In both cases we observe a block matrix since the first 125 entries of the matrix correspond to graphs which are labeled as $+1$ and the remaining entries correspond to graphs which are labeled as -1.

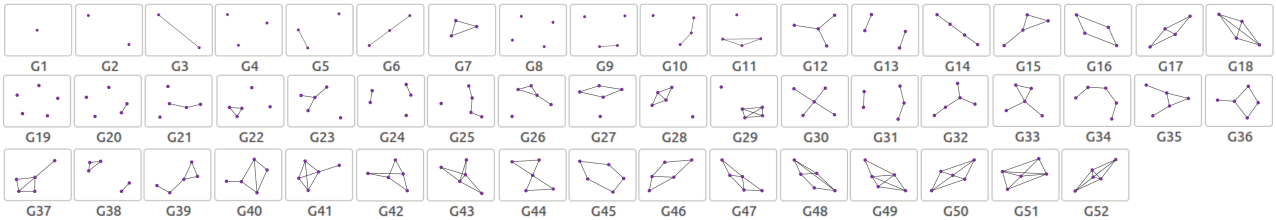


Figure 4: Connected, non-isomorphic induced sub-graphs of size $k \leq 5$.

2.1 Notation

Let $\mathcal{G} = (V, E)$ represent a graph where V is a set of *vertices* and $E \subseteq (V \times V)$ is a set of *edges*. Let $\mathbb{G}$ represent a set of n graphs where $\mathbb{G} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n\}$ and let $\mathcal{Y}$ represent a set of labels associated with each graph in $\mathbb{G}$ where $\mathcal{Y} = \{y_{\mathcal{G}_1}, y_{\mathcal{G}_2}, \ldots, y_{\mathcal{G}_n}\}$. Given $\mathcal{G} = (V, E)$ and $H = (V_H, E_H)$, H is a *sub-graph* of $\mathcal{G}$ if and only if there is an injective mapping $\alpha : V_H \rightarrow V$ such that $(v, w) \in E_H$ if and only if $(\alpha(v), \alpha(w)) \in E$. A graph $\mathcal{G}$ is called a *labeled graph* if there is a function $l : V \rightarrow \Sigma$ that assigns labels from an alphabet Σ to vertices in the graph. A graph $\mathcal{G}$ is called an *unlabeled graph* if individual vertices have no distinct identifications other than their inter-connectivity. Throughout the paper, we will refer $\mathcal{K}(\mathcal{G}, \mathcal{G}')$ as a kernel function that measures the similarity between graphs $\mathcal{G}$ and $\mathcal{G}'$.

Graph classification task considers the problem of classifying graphs into two or more categories. Given a set of graphs $\mathbb{G}$ and a set of class labels $\mathcal{Y}$, the task in graph classification is then to learn a model that maps graphs in $\mathbb{G}$ to the label set $\mathcal{Y}$. A popular approach is to first use a graph kernel to compute a kernel matrix K of size $n \times n$ where K_{ij} represents the similarity between $\mathcal{G}_i$ and $\mathcal{G}_j$, and then to plug the computed kernel matrix into a kernelized learning algorithm such as Support Vector Machines (SVM) [12] to perform classification.

2.2 Graph kernels based on subgraphs

A *graphlet* G is an induced and non-isomorphic sub-graph of size-k (see Figure 4) [23]. Let $\mathcal{V}_k = \{G_1, G_2, \ldots, G_{n_k}\}$ be the set of size-k graphlets where n_k denotes the number of unique graphlets of size k. Given two *unlabeled* graphs $\mathcal{G}$ and $\mathcal{G}'$, the graphlet kernel is defined as follows [28]:

$$\mathcal{K}_{GK}(\mathcal{G}, \mathcal{G}') = \left\langle \mathbf{f}^{\mathcal{G}}, \mathbf{f}^{\mathcal{G}'} \right\rangle, \tag{3}$$

where $\mathbf{f}^{\mathcal{G}}$ and $\mathbf{f}^{\mathcal{G}'}$ are vectors of normalized counts, that is, the i^{th} component of $\mathbf{f}^{\mathcal{G}}$ (resp. $\mathbf{f}^{\mathcal{G}'}$) denotes the frequency of graphlet G_i occurring as a sub-graph of $\mathcal{G}$ (resp. $\mathcal{G}'$). Furthermore, $\langle \cdot, \cdot \rangle$ denotes the Euclidean dot product.

2.3 Graph kernels based on subtree patterns

The second family of graph kernels decomposes a graph into its subtree patterns. The Weisfeiler-Lehman subtree kernel [27] belongs to this family. The key idea here is to iterate over each vertex of a *labeled* graph and its neighbors in order to create a *multiset label*. The multiset at every iteration consists of the label of the vertex and the sorted labels of its neighbors. The resultant multiset is given a new label, which is then used for the next iteration. When comparing graphs, we simply count the co-occurrences of labels in both graphs. This procedure is inspired by the Weisfeiler-Lehman test of graph isomorphism, and is equivalent to comparing the number of shared subtrees between two graphs. Formally, given $\mathcal{G}$ and $\mathcal{G}'$, the Weisfeiler-Lehman subtree kernel is defined as:

$$\mathcal{K}_{WL}(\mathcal{G}, \mathcal{G}') = \left\langle \mathbf{l}^{\mathcal{G}}, \mathbf{l}^{\mathcal{G}'} \right\rangle. \tag{4}$$

As before, $\langle \cdot, \cdot \rangle$ denotes the Euclidean dot product. If we assume that we perform h iterations of relabeling, then $\mathbf{l}^{\mathcal{G}}$ consists of h blocks. The i^{th} component in the j^{th} block of $\mathbf{l}^{\mathcal{G}}$ contains the frequency with which the i^{th} label was assigned to a node in the j^{th} iteration.

2.4 Graph kernels based on random-walks

The third family of graph kernels decomposes a graph into random-walks [14, 33] or paths [3] and counts the co-occurrence of random-walks or paths in two graphs. Let $\mathbb{P}_{\mathcal{G}}$ represent the set of all shortest-paths in graph $\mathcal{G}$, and $p_i \in \mathbb{P}_{\mathcal{G}}$ denote a triplet (l_s^i, l_e^i, n_k) where n_k is the length of the path and l_s^i and l_e^i are the labels of the starting and ending vertices, respectively. The shortest-path kernel between *labeled* graphs $\mathcal{G}$ and $\mathcal{G}'$ is defined as [3]:

$$\mathcal{K}_{SP}(\mathcal{G}, \mathcal{G}') = \left\langle \mathbf{p}^{\mathcal{G}}, \mathbf{p}^{\mathcal{G}'} \right\rangle, \tag{5}$$

where the i^{th} component of $\mathbf{p}^{\mathcal{G}}$ contains the frequency of the i^{th} triplet occurring in graph $\mathcal{G}$. The vector $\mathbf{p}^{\mathcal{G}'}$ is defined analogously for $\mathcal{G}'$.

2.5 A Unified View using R-convolution

One can show that all graph kernels summarized above as well as other somewhat sophisticated variants are all instances of the R-convolution framework. In a nutshell, the recipe for defining graph kernels is as follows: First, recursively decompose a graph into its subgraphs. For instance, the graphlet kernel decomposes a graph into graphlets, Weisfeiler-Lehman kernel decomposes a graph into subtrees, and Shortest-Path kernel decomposes a graph into shortest-paths. In next step, the decomposed sub-structures are represented as a vector of frequencies where each item of the vector represents how many times a given sub-structure occurs in the graph. Finally, the Euclidean space or some other domain-specific RKHS is used to define the dot product between the vectors of frequencies.

3. METHODOLOGY

In this section, we first discuss how to compute an $\mathcal{M}$ matrix by using the *edit-distance* relationship between sub-structures. Then, we discuss how to compute an $\mathcal{M}$ matrix by *learning* the similarity between sub-structures inspired by latest advancements in language modeling and deep learning.

3.1 Sub-structure similarity via edit-distance

When sub-structures exhibit a clear mathematical relationship, one can exploit the underlying similarities between sub-structures to compute an $\mathcal{M}$ matrix. For instance, in graphlet kernels, one

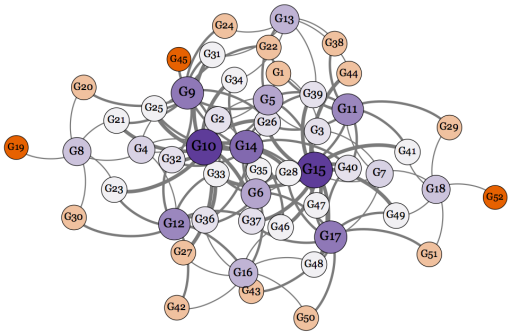


Figure 5: Undirected edit-distance graph $\mathfrak{G}$ for graphlets of size $k \leq 5$ (size and colors of the nodes are based on degree).

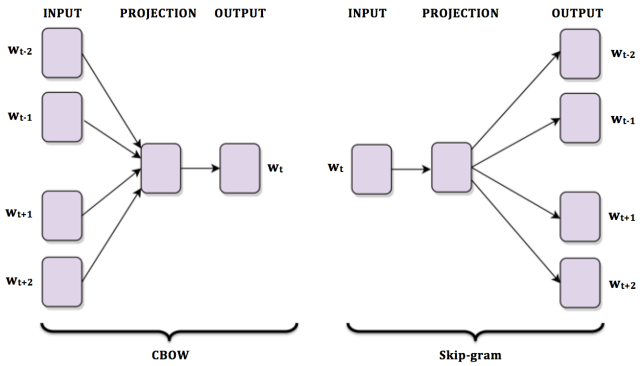


Figure 6: Architecture for the CBOW and Skip-gram method [20]. w_t is the current word, while w_{t+j} are the surrounding words where c is the size of the context, and $-c \leq j \leq c$. Here, CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

can derive an *edit-distance* relationship to encode how similar one graphlet to is another (see Figure 2 (a)). Given a graphlet G_i of size k, and a graphlet G_j of size $k+1$, let us build an undirected edit-distance graph $\mathfrak{G}$ by adding an undirected edge from G_i to G_j if and only if G_i can be obtained from G_j by deleting a node of G_j (or vice versa, if G_j can be obtained from G_i by adding a node to G_i). Given such an undirected graph $\mathfrak{G}$, one can simply compute the shortest-path distance between G_i and G_j in order to compute their *edit-distance* (see Figure 5). While this approach enables us to directly compute an $\mathcal{M}$ matrix, the cost of computing the shortest-path distances on $\mathfrak{G}$ becomes prohibitively expensive as a function of graphlet size k (see Figure 2 (b)). For instance, in order to compute an $\mathcal{M}$ matrix at level $k = 9$, one needs to compute all pairwise shortest-path distances of an undirected graph having $288,267$ nodes. On the other hand, one can observe that while the number of unique graphlets grow exponentially, only a few of them will be observed in a given graph. Therefore, instead of computing a complete $\mathcal{M}$ matrix of size $|\mathcal{V}| \times |\mathcal{V}|$, one can design an $\mathcal{M}$ matrix of size $|\mathcal{V}'| \times |\mathcal{V}'|$ with $|\mathcal{V}'| \ll |\mathcal{V}|$ by only taking the *observed* sub-structures into account. In next section, we discuss an approach that utilizes this observation.

3.2 Sub-structure similarity via learning

Our second approach is to *learn* the latent representations of sub-structures by using recently introduced language modeling and deep learning techniques. The learned representations are then utilized to compute an $\mathcal{M}$ matrix that respects the similarity between sub-structures. Next, we review the related background in language modeling, and then transform the ideas to learn representations of sub-structures.

3.2.1 Neural language models

Traditional language models estimate the likelihood of a sequence of words appearing in a corpus. Given a sequence of training words $\{w_1, w_2, \ldots, w_T\}$, n-gram based language models aims to maximize the following probability

$$\Pr(w_t | w_1, \ldots, w_{t-1}). \tag{6}$$

In other words, they estimate the likelihood of observing w_t given n previous words observed so far.

Recent work in language modeling focused on distributed vector representation of words, also referred as *word embeddings*. These neural language models improve classic n-gram language models by using continuous vector representations for words. Unlike traditional n-gram models, neural language models take advantage of the notion of *context* where a context is defined as a fixed number of preceding words. In practice, the objective of word embedding models is to maximize the following log-likelihood

$$\sum_{t=1}^{T} \log \Pr(w_t | w_{t-n+1}, \ldots, w_{t-1}), \tag{7}$$

where $w_{t-n+1}, \ldots, w_{t-1}$ are the context of w_t.

Several methods are proposed to approximate Equation 7. Next, we discuss two such methods that we utilize in our framework, namely continuous bag-of-words (CBOW) and Skip-gram models [20].

3.2.2 Continuous bag-of-words

CBOW model predicts the current word given the surrounding words within a given window. The model architecture is similar to feedforward neural network language model [2] where the non-linear hidden layer is removed and the projection layer is shared for all words (see Figure 6). Formally, CBOW model aims to maximize the following log-likelihood,

$$\sum_{t=1}^{T} \log \Pr(w_t | w_{t-c}, \ldots, w_{t+c}), \tag{8}$$

where c is the length of the context. The probability $\Pr(w_t | w_{t-c}, \ldots, w_{t+c})$ is computed using the softmax, defined as

$$\frac{\exp(\bar{\mathbf{v}}^\top \mathbf{v}'_{w_t})}{\sum_{w=1}^{\mathcal{V}} \exp(\bar{\mathbf{v}}^\top \mathbf{v}'_{w})}. \tag{9}$$

Here, $\mathbf{v}_w$ corresponds to the input vector representation of w and $\mathbf{v}'_{w_t}$ corresponds to the output vector representation of w_t. The averaged vector representation from the context is computed as

$$\bar{\mathbf{v}} = \frac{1}{2c} \sum_{-c \leq j \leq c, j \neq 0} \mathbf{v}_{w_{t+j}}. \tag{10}$$

3.2.3 Skip-gram model

The Skip-gram model maximizes co-occurrence probability among the words that appear within a given window. In other words, instead of predicting the current word based on surrounding words,

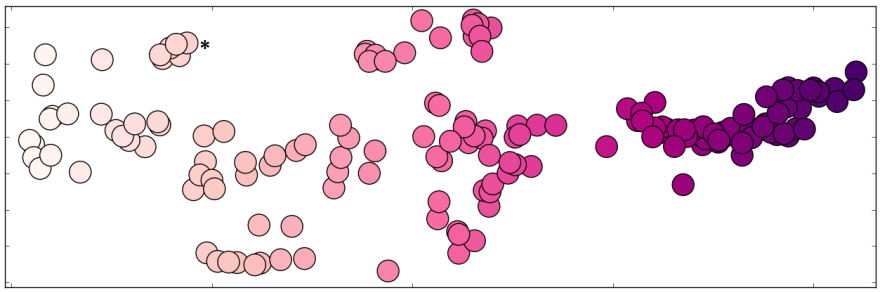


Figure 7: The learned shortest-path sub-structures on ENZYMES dataset [4] in $\mathbb{R}^2$. Each node corresponds to a different shortest-path sub-structure and colored by the value of their first dimension (labels on nodes are omitted for clarity). For instance, the shortest path sub-structures embedded in the cluster marked with * are $2 \cdot 2_{23}, 2 \cdot 2_{24}, 2 \cdot 2_{25}, 2 \cdot 2_{26}, 2 \cdot 2_{27}, 2 \cdot 2_{28}$ where first two characters represent the start and end point of the shortest-path and sub-script represents the length of the path). Note that the shortest-paths having the same start and endpoints with similar shortest-path lengths are close to each other in the latent space.

the main objective of the Skip-gram is to predict the surrounding words given the current word (see Figure 6). More precisely, the objective of the Skip-gram model is to maximize the following log-likelihood,

$$\sum_{t=1}^{T} \log \Pr(w_{t-c}, \ldots, w_{t+c} | w_t). \tag{11}$$

where the probability $\Pr(w_{t-c}, \ldots, w_{t+c}|w_t)$ is computed as

$$\prod_{-c \leq j \leq c, j \neq 0} \Pr(w_{t+j}|w_t). \tag{12}$$

Here, the contextual words and the current word are assumed to be independent. Furthermore, $\Pr(w_{t+j}|w_t)$ is defined as

$$\frac{\exp(\mathbf{v}_{w_t}^{\top} \mathbf{v}'_{w_{t+j}})}{\sum_{w=1}^{\mathcal{V}} \exp(\mathbf{v}_{w_t}^{\top} \mathbf{v}'_{w})} \tag{13}$$

where $\mathbf{v}_w$ and $\mathbf{v'}_w$ are the input and output vectors of word w.

Hierarchical softmax and negative sampling are two efficient algorithms that are used in training the Skip-gram and CBOW models. Hierarchical softmax uses a binary Huffman tree to factorize expensive partition function of the Skip-gram model. An alternative to the Hierarchical softmax is negative sampling, which selects the contexts at random instead of considering all words in the vocabulary. In other words, if a word w appears in the context of another word w', then the vector representation of the word w is closer to the vector representation of word w' comparing to any other randomly chosen words. In practice, one should try both the Skip-gram and CBOW models with Hierarchical softmax and negative sampling algorithms in order to decide which pair is more suitable to the application in hand.

After the training converges, similar words are mapped to similar positions in the vector space. Moreover, the learned word vectors are empirically shown to preserve semantics. For instance, word vectors can be used to answer analogy questions using simple vector algebra where the result of a vector calculation $\mathbf{v}(\textbf{“Madrid”}) - \mathbf{v}(\textbf{“Spain”}) + \mathbf{v}(\textbf{“France”})$ is closer to $\mathbf{v}(\textbf{“Paris”})$ than any other word vector [20].

These properties make word vectors attractive for our task since the order independence assumption provides a flexible notion of ‘nearness’ for sub-structures. A key intuition we utilize in our framework is to view *sub-structures* in graph kernels as *words* that are generated from a special language. In other words, different sub-structures compose graphs in a similar way that different words form sentences when used together. With this analogy in mind, one can utilize word embedding models to unveil dimensions of similarity between sub-structures. The main expectation here is that similar sub-structures will be close to each other in the d-dimensional latent space. Figure 7 illustrates shortest-path sub-structures in $\mathbb{R}^2$ learned by our framework. Note that similar sub-structures are close together in latent space.

3.2.4 *Deep Graph Kernels*

Our framework takes a list of graphs $\mathbb{G}$ and decomposes each graph into its sub-structures. The list of decomposed sub-structures for each graph is then treated as a sentence that is generated from a vocabulary $\mathcal{V}$ where vocabulary $\mathcal{V}$ simply corresponds to the unique set of *observed* sub-structures in the training data. However, unlike words in a traditional text corpora, sub-structures do not have a linear co-occurrence relationship. Therefore, one needs to build a corpus where the co-occurrence relationship is meaningful. Next, we discuss how to generate corpora where co-occurrence relationship is meaningful on three major graph kernel families.

- **Corpus generation for graphlet kernels:** Exhaustive enumeration of all graphlets in a graph $\mathcal{G}$ is prohibitively expensive for even moderate sized graphs [23]. Several sampling heuristics are proposed for sampling sub-graphs efficiently, such as random sampling scheme of [28]. In practice, the random sampling of graphlets of size k in a graph $\mathcal{G}$ involves placing a randomly generated window of size $k \times k$ on the adjacency matrix of $\mathcal{G}$ and collecting the observed graphlet in that window. This procedure is repeated n times where n being the number of graphlets we would like to sample. However, since this is a random sampling scheme, it does not preserve any notion of co-occurrence relationship which is a desired property for our framework. Therefore, we modify the random sampling scheme to partially preserve the co-occurrence between graphlets by using the notion of *neighborhoods*. That is, whenever we randomly sample a graphlet G, we also sample its immediate neighbors. The graphlet and its neighbors are then interpreted as *co-occurred* by our method. Therefore, graphlets which have similar neighborhoods will acquire similar representations. While in this paper we utilized only the immediate neighbors of a graphlet, one can extend the co-occurrence relationship to consider neighborhoods of distance ≥ 1.

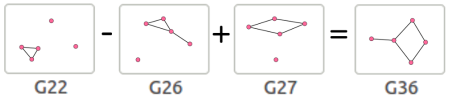


Figure 8: An example of sub-structure regularity in graphlet sub-structure space.

In order to verify this intuition, we explored the idea of having a similar effect of linguistic regularities in language models to graphs. Word embeddings are known to successfully answer queries such as *"What is the word that is similar to small in the same sense as biggest is similar to big?"* where answer is correctly recovered as *smallest* [21]. Therefore, we investigate whether a meaningful response with a similar analogy can be recovered, such that *"What is the graphlet that is similar to a square (G_{27}) in the same sense as triangle with a tail (G_{26}) is similar to triangle (G_{22})?"*. We used the multiplicative combination objective proposed by [18]:

$$\underset{b^* \in \mathcal{V}}{\operatorname{argmax}} \frac{\cos(b^*, b)\cos(b^*, a^*)}{\cos(b^*, a) + \epsilon} \tag{14}$$

where a and b are sub-structures from a vocabulary $\mathcal{V}$, cos is the cosine similarity between sub-structure vectors and $\epsilon = 0.001$ is used to prevent division by zero. This objective function amplifies the differences between small quantities and reduces the differences between larger ones. We used two positive (G_{22}, G_{27}) and one negative example (G_{26}), and recovered G_{36} as the top sub-structure by this arithmetic operation on embedded vectors (see Figure 8).

- **Corpus generation for Shortest-Path graph kernels:** Shortest-path graph kernel compares the sorted endpoints and the length of shortest-paths that are common between two graphs. Similar to graphlet kernel, one needs to find a meaningful co-occurrence relationship between shortest-path sub-structures. One can show that all sub-paths of a shortest-path are also shortest-paths with the same source [8]. In other words, whenever we observe a shortest-path sub-structure p of length l, we must also observe all of its sub-paths of length $< l$ as well. Inspired by this property, whenever we generate a shortest-path sub-structure, we also collect all possible shortest-path sub-structures that share the same source node, and treat them as *co-occurred*. Therefore, shortest-path sub-structures which have similar labels will acquire similar representations (see Figure 7).

- **Corpus generation for Weisfeiler-Lehman kernels:** The Weisfeiler-Lehman subtree kernel iterates over each vertex and its neighbors in order to create a multiset label. The resultant multiset is given a new label, which is then used for the next iteration. Therefore, multiset labels that belong a given iteration h can be treated as *co-occurred* in order to partially preserve a notion of similarity.

After generating a corpus where a co-occurrence relationship is partially preserved, we simply build the model by using CBOW or Skip-gram algorithms and train them with Hierarchical softmax or negative sampling[4]. Let s represent an arbitrary sub-structure from a vocabulary $\mathcal{V}$, and Φ_s represent learned vector representation of s using our framework. Given the vector representations of sub-structures, we compute a diagonal $\mathcal{M}$ matrix such that each entry on the diagonal, $\mathcal{M}_{ii}$ computed as $\langle \Phi_i, \Phi_i \rangle$ where Φ_i corresponds to learned d-dimensional hidden of sub-sequence i and $\mathcal{M}_{ij} = 0$ where $i \neq j$ and $1 \leq i \leq |\mathcal{V}|$ (resp. j). After computing the $\mathcal{M}$ matrix, we simply plug it into Equation 2 in order to compute the kernel between each sub-structure.

[4]We used Gensim library [25] for all algorithms.

3.2.5 *Other Deep Kernels*

In a similar fashion, we can plug other graph kernels into our framework such as random-walk kernels [10], labeled version of graphlet kernel [28], subtree kernels [6, 24], cyclic pattern kernels [13] and p-step random-walk kernel [30]. Moreover, our framework is applicable to any R-convolution kernel where there is a notion of dependency between sub-structures, such as *string kernels*. String kernels are other popular instances of R-convolution kernels where we are interested in computing a kernel between two *sequences*. Given an input sequence $\mathcal{S}$ over an alphabet $\mathcal{V}$ and a number $k \geq 1$, $k-$spectrum of the sequence $\mathcal{S}$ is defined as the set of all k-length contiguous sub-sequences $\mathcal{S}$ contains [17]. The feature vector $\phi(\mathcal{S})$ is then simply constructed as a frequency vector over sub-sequences in its k-spectrum and the kernel between two sequences are computed via Equation 1. Similar to graph kernels, the co-occurrence relationship between sub-sequences are not taken into account. Similar to graph kernels, we treat all length k sub-sequences of a string as co-occurred and learn the hidden representation of each spectrum using our framework. In case of string kernels, we compute $\mathcal{M}$ matrix such that each entry $\mathcal{M}_{ij}$ computed as $\langle \Phi_i, \Phi_j \rangle$ where Φ_i corresponds to learned d-dimensional vector of sub-sequence i (resp. Φ_j).

4. RELATED WORK

The closest work to our paper is the recently proposed model, DeepWalk by [22]. DeepWalk learns social representations of vertices of graphs by modeling short random-walks. We distance ourselves from DeepWalk in several aspects. First, instead of learning similarities between *nodes* we are interested in learning similarities between *structured objects*, such as graphs and strings. In other words, DeepWalk operates on a *single* graph, while we are interested in the relationship between *multiple* graphs. Moreover, instead of using random-walks, our framework can be configured to work with any type of sub-structures, including graphlets, shortest-paths, sub-trees and strings.

Many different graph kernels focusing on different types of sub-graphs have been defined in the past which can be categorized into three major families: graph kernels based on limited-sized sub-graphs [13], [28], graph kernels based on subtree patterns [24], [27] and graph kernels based on walks [14] and paths [3]. Our framework is *complementary* to existing graph and string kernels where the sub-structures have a similarity relationship between them.

5. EXPERIMENTS

The aim of our experiments is threefold. First, we want to show that using an $\mathcal{M}$ matrix that infers the relationship between sub-structures improves the classification accuracy. Second, we want to show that our framework is *robust* to random noise. Third, we want to show that the deep kernels are comparable to or outperform state-of-the-art graph kernels in terms of classification accuracy, while remaining competitive in terms of computational requirements. Next, we discuss bioinformatics and social network datasets that we use in our experiments.

Table 1: Properties of the Bioinformatics and Social network datasets used in graph kernel experiments.

Dataset	Size	Classes	Avg.nodes	Labels
MUTAG	188	2	17.9	7
PTC	344	2	25.5	19
ENZYMES	600	6	32.6	3
PROTEINS	1113	2	39.1	3
NCI1	4110	2	29.8	37
NCI109	4127	2	29.6	38
COLLAB	5000	3	74.49	-
IMDB-BINARY	1000	2	19.77	-
IMDB-MULTI	1500	3	13	-
REDDIT-BINARY	2000	2	429.61	-
REDDIT-MULTI-5K	5000	2	508.5	-
REDDIT-MULTI-12K	11929	11	391.4	-

5.1 Datasets

In order to test the efficacy of our model, we applied our framework to real-world datasets from bioinformatics and social networks (see Table 1 for summary statistics of these datasets).

5.1.1 *Bioinformatics datasets*

We applied our framework to benchmark graph kernel datasets, namely, MUTAG, PTC, ENZYMES, PROTEINS and NCI1, NCI109. MUTAG is a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds [7] with 7 discrete labels. PTC [31] is a dataset of 344 chemical compounds that reports the carcinogenicity for male and female rats and it has 19 discrete labels. NCI1 and NCI109 [34] datasets (4100 and 4127 nodes, respectively), made publicly available by the National Cancer Institute (NCI) are two subsets of balanced datasets of chemical compounds screened for ability to suppress or inhibit the growth of a panel of human tumor cell lines, having 37 and 38 discrete labels respectively. ENZYMES is a balanced dataset of 600 protein tertiary structures obtained from [4] and has 3 discrete labels. PROTEINS is a dataset obtained from [4] where nodes are secondary structure elements (SSEs) and there is an edge between two nodes if they are neighbors in the amino-acid sequence or in 3D space. It has 3 discrete labels, representing *helix, sheet or turn*.

5.1.2 *Social network datasets*

In order to test the efficacy of our framework on social network domain, we derive several unlabeled graph datasets with different tasks as follows.

- **Reddit datasets:** *REDDIT-BINARY* is a balanced dataset where each graph corresponds to an online discussion thread where nodes correspond to users, and there is an edge between two nodes if at least one of them responded to another's comment. We crawled top submissions from four popular subreddits, namely, *IAmA, AskReddit, TrollXChromosomes, atheism*. *IAmA* and *AskReddit* are two question/answer-based subreddits and *TrollXChromosomes* and *atheism* are two discussion-based subreddits. The task is then to identify whether a given graph belongs to a *question/answer*-based community or a *discussion*-based community. *REDDIT-MULTI-5K* is a balanced dataset from five different subreddits, namely, *worldnews*, *videos*, *AdviceAnimals*, *aww* and *mildlyinteresting* where we simply label each graph with their correspondent subreddit. *REDDIT-MULTI-12K* is a larger variant of *REDDIT-MULTI-5K*, consists of 11 different subreddits, namely, *AskReddit, AdviceAnimals, atheism, aww, IAmA, mildlyinteresting, Showerthoughts, videos, todayilearned, worldnews, TrollXChromosomes*. The task in both datasets is to predict which subreddit a given discussion graph belongs to.

- **Scientific collaboration dataset:** *COLLAB* is a scientific-collaboration dataset, derived from 3 public collaboration datasets [16], namely, *High Energy Physics*, *Condensed Matter Physics* and *Astro Physics*. Following the approach of [29], we generated ego-networks of different researchers from each field, and labeled each graph as the field of the researcher. The task is then to determine whether the ego-collaboration graph of a researcher belongs to High Energy, Condensed Matter or Astro Physics field.

- **Movie collaboration datasets:** *IMDB-BINARY* is a movie-collaboration dataset where we collected *actor/actress* and *genre* information of different movies on IMDB. For each graph, nodes represent actors/actresses and there is an edge between them if they appear in the same movie. We generated collaboration graphs on *Action* and *Romance* genres and derived ego-networks for each actor/actress. Note that a movie can belong to both genres at the same time, therefore we discarded movies from Romance genre if the movie is already included to the Action genre. Similar to COLLAB dataset, we simply labeled each ego-network with the genre graph it belongs to. The task is then simply to identify which genre an ego-network graph belongs to. *IMDB-MULTI* is multi-class version of *IMDB-BINARY* and contains a balanced set of ego-networks derived from *Comedy, Romance* and *Sci-Fi* genres.

5.2 Experimental setup

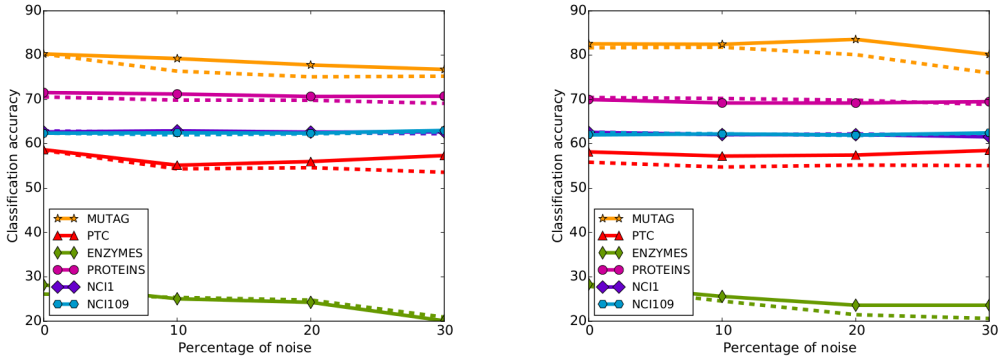
We compare our framework against representative instances of major families of graph kernels in the literature. Other than base kernels of our framework, namely, Weisfeiler-Lehman subtree kernel [28], graphlet kernel [28], and shortest-path kernel [3], we also compare our kernels with the random walk kernel [10], the subtree kernel [24], and p-step random-walk kernel [30]. The Random-Walk, p-step Random-Walk and Ramon-Gärtner kernels are written in Matlab and were obtained from the authors of [28]. All other kernels were coded in Python[5]. In order to ensure a fair comparison, all experiments are performed on the same hardware.

All kernels are normalized to have a unit length in the feature space. Moreover, we use 10-fold cross validation with a binary C-SVM [5] to test classification performance. The C value for each fold is independently tuned using training data from that fold. In order to exclude random effects of the fold assignments, this experiment is repeated 10 times, and average prediction accuracies with their standard deviations are reported.

5.3 Parameter selection

We chose parameters for the various kernels as follows: the window size and dimension for deep graph kernels is chosen from $\{2, 5, 10, 25, 50\}$, the decay factor for random-walk kernels is chosen from $\{10^{-6}, 10^{-5}, \ldots, 10^{-1}\}$, the p value in the p-step random-walk kernel is chosen from $\{1, 2, \ldots, 10\}$ and the height parameter in Ramon-Gärtner subtree kernel is chosen from $\{1, 2, 3\}$. For each kernel, we report the results for the parameter which gave the best classification accuracy. For Weisfeiler-Lehman subtree kernel, we experimented with the height parameter $h = 2$ due to exponentially increasing feature space of the original kernel. For the

[5]Our code and datasets are available at http://web.ics.purdue.edu/~ypinar/kdd



(a) Edit-distance graphlet kernels

(b) Deep graphlet kernels

Figure 9: Classification accuracy (y-axis) vs. edge noise (x-axis) on different datasets using Graphlet kernel vs. Edit-distance graphlet kernel (a) and Graphlet kernel vs. Deep graphlet kernel (b). Dashed lines represent original graphlet kernel while non-dashed lines represents its corresponding variant derived from our framework.

graphlet kernel, we set the size of the graphlets k to be 7 since it exhibits the sparsity problem that we are interested in. We used Nauty [19] to get canonically-labeled isomorphic representations of each graphlet which are then used to construct the feature representation.

5.4 Results

In this section, we apply our framework to several benchmark datasets and compare the classification accuracy of our kernels against their base variants.

Graphlet Kernels under noise We have two variants of graphlet kernels, namely, Edit-distance Graphlet Kernel (EGK) introduced in Section 3.1 and Deep Graphlet Kernel (DGK) introduced in Section 3.2. Since graphlet kernels do not exploit label information on the vertices and only compare graphs based on their structural similarity, an interesting experiment is to see how our kernels behave under random noise on the edges. Therefore, we derive noisy variants of the datasets by randomly flipping $10\%, 20\%$ and 30% of the edges. Figure 9 (a) shows the comparison between original graphlet kernel and EGK where 0% represents the classification accuracy on the original dataset without noise. As can be seen from the figure, EGK outperforms the base kernel in MUTAG, PTC, PROTEINS, NCI1, NCI109, but outperformed by the original kernel in ENZYMES dataset. We believe this is due to the fact that EGK only uses a mathematical relationship between sub-structures rather than *learning* a sophisticated relationship. Therefore, we applied our deep kernel framework on graphlet kernels (see Figure 9) (b). As can be seen from the figure, learning latent representations of the graphlets outperforms its base variant significantly in *all* datasets except PROTEINS.

Graphlet Kernels on social network datasets Next, we test the efficacy of our framework on several social network datasets using graphlet kernels. As can be seen from Table 2, deep graphlet kernels are able to outperform its base variant in all cases.

Deep Graph Kernels on bioinformatics datasets Table 3 shows the classification accuracy between graph kernels and their deep variants using Graphlet kernel, Weisfeiler-Lehman kernel, and Shortest-Path kernel. As can be seen from the table, our method is comparable or outperforms the base variants in all datasets.

Table 2: Comparison of classification accuracy ($\pm$ standard deviation) of the Graphlet Kernel (GK) and Deep Graphlet Kernel (DGK) on social network datasets.

Dataset	GK	DGK
COLLAB	72.84 ± 0.28	73.09 ± 0.25
IMDB-BINARY	65.87 ± 0.98	66.96 ± 0.56
IMDB-MULTI	43.89 ± 0.38	44.55 ± 0.52
REDDIT-BINARY	77.34 ± 0.18	78.04 ± 0.39
REDDIT-MULTI-5K	41.01 ± 0.17	41.27 ± 0.18
REDDIT-MULTI-12K	31.82 ± 0.08	32.22 ± 0.10

Comparison against other kernels Table 4 shows the classification accuracy of Ramon & Gärtner, p-random-walk and random-walk graph kernels where the first column is constructed by picking the best result of Deep Graph Kernels from Table 3. As can be seen from Table 4, Deep Graph Kernels are able to outperform other graph kernels.

5.5 Computational Cost

For Edit-distance Graphlet Kernel, computing an $\mathcal{M}$ matrix involves a one-time computation of the undirected graph between 1253 nodes for $k = 7$ which empirically takes 7 minutes. After that, one needs to compute all-pairs-shortest-path distances on the obtained undirected graph which empirically takes 8 seconds. For deep graph kernels, the overhead of computing an $\mathcal{M}$ matrix involves learning latent representations of the observed sub-structures. The runtime averaged out of all datasets for learning the latent representations is 21.5 seconds for deep graphlet kernel, 4.5 seconds for deep shortest-path graph kernel and 1.75 seconds for deep Weisfeiler-Lehman graph kernel. All runtime experiments use a fixed window size and dimension at 25 and this process is repeated 10 times to eliminate random effects.

5.6 Deep String Kernels

As a proof-of-concept, we derive a deep variant of k-spectrum string kernel and perform experiments on benchmark bioinformatics datasets.

Table 3: Comparison of classification accuracy ($\pm$ standard deviation) of the graphlet kernel (GK), shortest-path kernel (SP), Weisfeiler-Lehman kernel (WL) to their deep variants on bioinformatics datasets.

Dataset	GK	Deep GK	SP	Deep SP	WL	Deep WL
MUTAG	81.66 ± 2.11	82.66 ± 1.45	85.22 ± 2.43	87.44 ± 2.72	80.72 ± 3.00	82.94 ± 2.68
PTC	57.26 ± 1.41	57.32 ± 1.13	58.24 ± 2.44	60.08 ± 2.55	56.97 ± 2.01	59.17 ± 1.56
ENZYMES	26.61 ± 0.99	27.08 ± 0.79	40.10 ± 1.50	41.65 ± 1.57	53.15 ± 1.14	53.43 ± 0.91
PROTEINS	71.67 ± 0.55	71.68 ± 0.50	75.07 ± 0.54	75.68 ± 0.54	72.92 ± 0.56	73.30 ± 0.82
NCI1	62.28 ± 0.29	62.48 ± 0.25	73.00 ± 0.24	73.55 ± 0.51	80.13 ± 0.50	80.31 ± 0.46
NCI109	62.60 ± 0.19	62.69 ± 0.23	73.00 ± 0.21	73.26 ± 0.26	80.22 ± 0.34	80.32 ± 0.33

Table 4: Comparison of classification accuracy ($\pm$ standard deviation) of Ramon & Gärtner, p-random-walk and random-walk graph kernels. $>$ 72H indicates that the computation did not finish after 72 hours.

Dataset	Deep Graph Kernels	Ramon&Gärtner	p-random-walk	Random-walk
MUTAG	87.44 ± 2.72	84.88 ± 1.86	80.05 ± 1.64	83.72 ± 1.50
PTC	60.08 ± 2.55	58.47 ± 0.90	59.38 ± 1.66	57.85 ± 1.30
ENZYMES	53.43 ± 0.91	16.96 ± 1.46	30.01 ± 1.01	24.16 ± 1.64
PROTEINS	75.68 ± 0.54	70.73 ± 0.35	71.16 ± 0.35	74.22 ± 0.42
NCI1	80.31 ± 0.46	56.61 ± 0.53	$>$ 72h	$>$ 72h
NCI109	80.32 ± 0.33	54.62 ± 0.23	$>$ 72h	$>$ 72h

5.6.1 *Datasets*

In order to test the efficacy of our model, we applied our method to benchmark datasets in string kernels. SCOP (Structural Classification of Proteins) is a manually-curated database that groups proteins together based on their 3-D structures [1]. The task is then to classify protein sequences into 7 distinct super-families. SCOP database has a 4-level structure-based hierarchy of classes where protein sequences are classified into one of the classes, namely, *class*, *fold*, *super-family* and *family*. Similar to the setting in [32], we tackled family and super-family classification problems where a family contains proteins with clear evolutionary relationship, and super-family contains the same evolutionary origin without being detectable at the level of sequences [15]. In *family-classification* problem, we considered NAD(P)-binding Rossmann-fold and (trans)-glycosidases domains where our main task is to classify proteins in a super-family into their families. NAD(P)-Rossmann dataset has 246 sequences having an average length of 218 with 6 classes where (trans)-glycosidases has 95 sequences having an average length of 375 with 2 classes.

In *super-family* classification problem, we used Triose Phosphate Isomerase (TIM) beta/alpha-barrel protein fold where we classify each protein to one of the 7 distinct super-families. TIM beta/alpha dataset has 330 sequences having an average length of 332 with 7 classes. In order to derive the labels of each sequence, we used Astral SCOPe 2.04 genetic domain sequence database [9], based on PDB SEQRES records, with less than 95% identity to each other.

Moreover, we derived a new dataset for string kernels using the transcripts of `TED.com` talks. We collected the transcript of 385 talks having an average length of 9425 from three categories, namely, **T**echnology, **E**ntertainment, **D**esign. The task is then to predict which of the three category a talk belongs to.

5.6.2 *Results*

The comparison between original k-spectrum string kernel with $k = 3$ and our method can be seen from Table 5. As shown in Table 5, deep variant of k-spectrum string kernel is able to outperform the base k-spectrum string kernel in all datasets.

Table 5: Classification accuracy and error reduction for string kernel experiments where numbers next to the accuracy results represents the standard deviation.

Dataset	K-Spectrum	Deep Spectrum
TIM beta/alpha	67.60 ± 1.13	69.03 ± 1.03
(trans)glycosidases	93.88 ± 2.17	95.33 ± 1.02
NAD(P)-Rossmann	69.87 ± 0.78	75.54 ± 0.85
TED	74.31 ± 0.88	77.39 ± 0.97

6. CONCLUSION

We presented a novel framework for graph kernels inspired by latest advancements in natural language processing and deep learning. We applied our framework to three popular graph kernels, namely, graphlet kernel, shortest-path kernel, and Weisfeiler-Lehman subtree kernels. We introduced several large graph kernel datasets in social network domain, and showed that our framework outperforms its base variants in terms of classification accuracy while introducing a negligible overhead.

Moreover, while we mainly restricted ourselves to graph kernels in this paper, we discussed that our framework is rather general, and lends itself to many extensions. For instance, it can be plugged directly into any R-convolution kernel as long as there is a dependency between sub-structures. We demonstrated one such extension on string kernels and achieved significant improvements in classification accuracy.

An interesting extension of our framework would be applying it to attributed graphs with continuous values. Since a certain divergence between attribute values needs to be tolerated, learning hidden representations of the sub-structures would help to obtain a better classification accuracy.

7. ACKNOWLEDGMENTS

We thank to anonymous KDD reviewers for their constructive comments. We also thank to Hyokun Yun, Jiasen Yang, Joon Hee Choi, Marjan Momtazpour, Parameswaran Raman, Sebastian Moreno, Shihao Ji for reviewing our paper. This work is supported by the National Science Foundation under grant No. #1219015.

References

[1] A. Andreeva, D. Howorth, S. E. Brenner, T. J. Hubbard, C. Chothia, and A. G. Murzin. Scop database in 2004: refinements integrate structure and sequence family data. *Nucleic acids research*, 32(suppl 1):D226–D229, 2004.

[2] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155, 2003.

[3] K. M. Borgwardt and H.-P. Kriegel. Shortest-path kernels on graphs. In *Proc. Intl. Conf. Data Mining*, pages 74–81, 2005.

[4] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. In *Proceedings of Intelligent Systems in Molecular Biology (ISMB)*, Detroit, USA, 2005.

[5] C. Chang and C. Lin. *LIBSVM: a library for support vector machines*, 2001.

[6] F. Costa and K. De Grave. Fast neighborhood subgraph pairwise distance kernel. In *Proceedings of the 26th International Conference on Machine Learning*, pages 255–262, 2010.

[7] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *J Med Chem*, 34:786–797, 1991.

[8] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt. Scalable kernels for graphs with continuous attributes. In *Advances in Neural Information Processing Systems*, pages 216–224, 2013.

[9] N. K. Fox, S. E. Brenner, and J.-M. Chandonia. Scope: Structural classification of proteins-extended, integrating scop and astral data and classification of new structures. *Nucleic acids research*, 42(D1): D304–D309, 2014.

[10] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In B. Schölkopf and M. K. Warmuth, editors, *Proc. Annual Conf. Computational Learning Theory*, pages 129–143. Springer, 2003.

[11] D. Haussler. Convolution kernels on discrete structures. Technical Report UCS-CRL-99-10, UC Santa Cruz, 1999.

[12] T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. Technical Report 156, Max-Planck-Institut für biologische Kybernetik, 2006. To appear in the Annals of Statistics.

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

[14] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Proc. Intl. Conf. Machine Learning*, pages 321–328, San Francisco, CA, 2003. Morgan Kaufmann.

[15] T. Kin, T. Kato, and K. Tsuda. Protein classification via kernel matrix completion. In B. Schölkopf, K. Tsuda, and J.-P. Vert, editors, *Kernel Methods in Computational Biology*, pages 261–274, Cambridge, MA; USA, 2004. MIT Press.

[16] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 177–187. ACM, 2005.

[17] C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 564–575, Singapore, 2002. World Scientific Publishing.

[18] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In M. Welling, Z. Ghahramani, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2177–2185, 2014.

[19] B. D. McKay. Nauty user's guide (version 2.4). *Computer Science Dept., Australian National University*, 2007.

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[21] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, 2013.

[22] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. *arXiv preprint arXiv:1403.6652*, 2014.

[23] N. Przulj. Biological network comparison using graphlet degree distribution. In *2006 European Conference on Computational Biology (ECCB)*, September 2006.

[24] J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. Technical report, First International Workshop on Mining Graphs, Trees and Sequences (held with ECML/PKDD'03), 2003.

[25] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

[26] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[27] N. Shervashidze and K. Borgwardt. Fast subtree kernels on graphs. In *Neural Information Processing Systems*, 2010.

[28] N. Shervashidze, S. V. N. Vishwanathan, T. Petri, K. Mehlhorn, and K. Borgwardt. Efficient graphlet kernels for large graph comparison. In M. Welling and D. van Dyk, editors, *Proc. Intl. Conference on Artificial Intelligence and Statistics*. Society for Artificial Intelligence and Statistics, 2009.

[29] A. Shrivastava and P. Li. A new space for comparing graphs. In *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*, pages 62–71. IEEE, 2014.

[30] A. J. Smola and R. Kondor. Kernels and regularization on graphs. In B. Schölkopf and M. K. Warmuth, editors, *Proc. Annual Conf. Computational Learning Theory*, Lecture Notes in Comput. Sci., pages 144–158, Heidelberg, Germany, 2003. Springer-Verlag.

[31] H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma. Statistical evaluation of the predictive toxicology challenge 2000-2001. *Bioinformatics*, 19(10):1183–1193, July 2003.

[32] S. V. N. Vishwanathan, K. M. Borgwardt, O. Guttman, and A. J. Smola. Kernel extrapolation. *Neurocomputing*, 69(7-9):721–729, 2006.

[33] S. V. N. Vishwanathan, N. N. Schraudolph, I. R. Kondor, and K. M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 2010. In press.

[34] N. Wale, I. A. Watson, and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems*, 14(3):347–375, 2008.