TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter One

Theoretical analysis

Before proceeding to the analysis of used methods, we first introduce the necessary notations related to Graph definition and Graph kernels. A graph by definition is a pair $G = (V, E)$, where V is the set of the graph nodes (vertices) $V = v_1, ..., v_n$, and $E \in V \times V$ is the set of edges between these nodes, i.e. $(v_i, v_j) \in E$ means that the graph has an edge from node v_i to node v_j (or vice versa since we consider undirected graphs in this work).

$H = (V_H, E_H)$ is said to be a subgraph (graphlet) of G ($H \sqsubseteq$) if and only if exist an injective function $\mathcal{M} : V_H \rightarrow V$ such that $(v, w) \in E_H \Leftrightarrow (\mathcal{M}(v), \mathcal{M}(w)) \in E$.

Any edge (v_i, v_i) is called a self loop. In a general graph two vertices v_i and v_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 A of size $n \times n$. The $(i, j) - th$ entry of A is 1 if an edge (v_i, v_j) exists and zero otherwise.

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

1.1 Kernels and Random Features

The kernel trick is a way to generate features for algorithms that depend only on the inner product between pairs of input points. The mathematical basis behind it is that any positive definite function $k(x, y)$ with $x, y \in \mathcal{R}^d$ defines an inner product and a lifting function ϕ so that the inner product between lifted data points equals its kernel $<\phi(x), \phi(y)>= k(x, y)$. The convenience one gets is that there is no need to access the lifting function ϕ, instead, the used algorithm accesses the data only through evaluations of $k(x, y)$.

1.1.1 Graphlet Kernel

Referring by $\mathcal{G} = \{graphlet(1), ..., graphlet(N_k)\}$ to the set of k-nodes graphlets, and considering a graph G of size n we can define a vector f_G of length N_k whose i-th component equals the normalized-number of occurrences of $graphlet(i)$ in G $(\#(graphlet(i) \sqsubseteq G.$ What should be noticed based on this definition is that no two different graphlets in $\mathcal{G}$ are isomorphic. f_G is referred to by k-spectrum of G, and this vector is the key idea behind graphlet kernel.

Definition 1 (Graphlet Kernel) *Given two graphs G and G′ of size $n, n' \geq k$, the graphlet kernel k_g is defined as (Shervashidze et al., 2009):*

$$k_g(G, G') = f_G^T f'_G. \tag{1.1}$$

The drawback of this kernel is that computing the k-spectrum vector costs a lot of computational time, since there are $\binom{n}{k}$ size-k subgraphs in a graph G of size n ($O(n^k)$ processing steps required), thus there is a trade off between a more accurate representation of the graph (large value of k) and the computational cost. However, some techniques are used in order to resolve this limitation as Sampling From Graph technique (section 1.1.2).

1.1.2 Sampling From Graph

The problem of Graph Sampling arises when we deal with a large-scale graph and the task is the pick a small-size sample subgraph/s 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 deployed in two different manners:

1. directly sample m sugraphs $\{H_1, ..., H_m\}$ of size k, and then estimate the k-spectrum vector empirically: $f_G(i) = \frac{1}{m}\Sigma_{j=1}^{m} \mathbb{1}[H_j = graphlet(i)]$
2. sample m subgraphs $\{H_1, ..., H_m\}$ of size n' such that $n \gg n' > k$, then estimate f_G as follows: $f_G = \frac{1}{m}\Sigma_{j=1}^{m} f_{H_j}$

The important thing here is whether a sufficiently large number of random samples will lead to an empirical k-spectrum vector close to the actual vector. The number of samples needed to achieve a given confidence with a small probability of error is called the sample complexity.

Theorem 1 *Let f be a probability distribution on the finite set $\mathcal{G} = g_1, ..., g_a$, $X = X_{j}{}_{j=1}^{m}$ be a set of independent identically distributed (iid) random variables X_j drawn from f, and $\hat{f}(g_i) = \frac{1}{m}\Sigma_{j=1}^{m} \mathbb{1}(X_j = g_i)$. Then for a given $\epsilon > 0$ and $\delta > 0$ we have (Shervashidze et al., 2009):*

$$m = \left\lceil \frac{2(log(2)a + log(\frac{1}{\delta}))}{\epsilon^2} \right\rceil \tag{1.2}$$

samples suffice to ensure that $P\{\left\|f - \hat{f}\right\|_1 \geq \epsilon\} \leq \delta$

Therefore this theorem gives a lower bound on the number of samples considered respecting the first manner using Graph Sampling to approximate the Graphlet Kernel in order to ensure some certaintyϵ with prabability $1 - \delta$.

1.1.3 Random Features

Kernel machines are of interest as they approximate any function arbitrarily well with sufficiently large training data set. On the other hand, the methods that operate on the kernel matrix (Gram matrix) require a lot of time in order to compute this matrix and to train the machine; for example, a dataset with half a million training examples might take days to train on modern workstations (Rahimi and Recht, 2008). Unlike kernel machines, linear support vector machines and regularized regression run much faster especially with low-dimensionality training data. One way to combine the advantages of the linear and nonlinear vector machines is to convert the training and evaluation of any kernel machine into the corresponding operations of a linear machine by mapping data into a relatively low-dimensional randomized feature space. Instead of considering the implicit lifting function which corresponds to the kernel, it was proposed to explicitly map the data to a low-dimensional Euclidean inner product space using a randomized feature map $z : \mathcal{R}^d \to \mathcal{R}^D$ so that the inner product between a pair of transformed points approximates their kernel:

$$k(x, y) =< \phi(x), \phi(y) >\approx z(x)^T z(y) \tag{1.3}$$

Considering this approximation, we can simply transform the input with z and then apply a fast linear learning method to approximate the answer of the real kernel.
In what follows, two methods to construct the random feature map function z are to be presented.

Random Fourier Features

The following Theorem represents the key idea behind this mapping technique

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

What that means is that when a shift-invariant kernel k is properly scaled, its Fourier transform $p(w)$ is a proper probability distribution, we can write:

$$k(x-y) = \int_{\mathcal{R}^d} p(w) e^{jw'(x-y)} dw = E_w[e^{jw'x} e^{jw'y^*}] \tag{1.4}$$

But both $p(w)$ and $k(\delta)$ are real-valued functions, thus from Eq. 1.4 we can prove that:

$$k(x-y) = \int_{\mathcal{R}^d} p(w) cos(w'(x-y)) dw = E_w[z_w(x) z_w(y)] \tag{1.5}$$

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

As a straight result, $z_w(x) z_w(y)$ is an unbiased estimate of k(x,y). We can achieve lower variance estimation to the expectation (1.5) by averaging D instances of the estimator with different random frequencies w. i.e. the low-variance estimator can be written as: $z(x)'z(y) = \frac{1}{D} \Sigma_{j=1}^{D} z_w(x) z_w(y)$. this estimator and based on Hoeffding's inequality guarantees exponentially fast convergence in D between $z(x)'z(y)$ and the kernel true value:

$$Pr(|z(x)'z(y) - k(x,y)| \geq \epsilon) \leq 2e^{\frac{-D\epsilon^2}{4}} \tag{1.6}$$

REFERENCES

Rahimi, Ali and Benjamin Recht (2008). “Random features for large-scale kernel machines”. In: *Advances in neural information processing systems*, pp. 1177–1184.

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