Graph Kernels

Viveka Kulharia,Arnab Ghosh,Professor Harish Karnick

Computer Science And Engineering ,IIT Kanpur

Abstract

In the real world similarity between graphs is an important measure needed for several real life problems like classification of Proteins and Enzymes and even in the Social Network analysis whereby 2 social graphs might have to be clustered by similar properties.

In the quest to determine the similarity between pairs of graphs in 2005 [1] Borgwardt et al looked at the similarity between the graphs in terms of shortest path kernels which was an improvement over the random walk kernels over product graphs which had its limitations on tractability but in 2010 [2] SVN Vishwanathan et al came up with a novel method to compute random walk kernels upto any length and that became much faster in terms of computation than the Shortest Path Kernel .

We looked at the 2010 paper and implemented the methods for computing the random walk kernel that were used in the paper and also implemented the shortest path kernel as shown by the 2005 paper and compared the running times and the kernels obtained from them ,

Keywords: Kernels, Graph

1. The Different Kernels

1.1. Shortest Path Kernels

The essential first step in the shortest-path kernel is to transform the original graphs into shortest-paths graphs. A shortest-paths graph S contains the same set of nodes as the input graph I. Unlike in the input graph, there exists an edge between all nodes in S which are connected by a walk in I. Every edge in S between nodes v_i and v_j is labeled by the shortest distance between these two nodes.

Any algorithm which solves the all-pairs-shortest-paths problem can be applied to determine all shortest distance edge labels in S for example in our implementation we have used the Floyd Warshall Algorithm to get the all-pairs shortest paths between the nodes of the graph.

1.1.1. Shortest Path Graph Kernel

Let G_1 and G_2 be two graphs that are Floyd transformed into S_1 and S_2 . We can then define our shortest path graph kernel on $S_1 = (V_1, E_1)$ and $S_2 = (V_2, E_2)$ as

$k_{shortestpaths}(S_1, S_2) = \sum_{e_1 \in E_1} \sum_{e_2 \in E_2} k^{(1)}_{walk}(e_1, e_2)$
where $k^{(1)}_{walk}$ is a positive definite on edge walks of length 1.

1.1.2. The shortest path graph kernel is positive definite

We first have to choose a psd Kernel on nodes and a psd Kernel on edges. Then we have to define a kernel on pairs of walks of length 1 as the product of kernels on nodes and edges encountered along the walk .

Since Kernels are closed under product then it is also closed under Tensor Product.As a tensor product of node and edge kernels the kernel is positive semidefinite .

In our implementation we have used the kernels on the edges and the nodes to be constant for simplicity although there is an option to change.

1.2. Random Walk Kernels

Kernels between two graphs G and G' is define as:

$$k(G, G') = \sum_{k=0}^{\infty} \mu(k) q_x^T W_x^k p_x \tag{1}$$

where,

$$W_x = \phi(X) \otimes \phi(X') \tag{2}$$

is obtained by Kronecker product. Here, $\phi(X)$ is a feature matrix of graph G and it can be considered as normalized form of adjacency matrix of graph G. So, W_x^k is the probability of simultaneous random walk of length k on graphs G and G'

$$q_x = q \otimes q' \tag{3}$$

where q and q' is a vector of stopping probability on each of the nodes of G and G' resp.

$$p_x = p \otimes p' \tag{4}$$

where p and p' is a vector of starting probability on each of the nodes of G and G' resp.

$\mu(k)$ is appropriately chosen so that 1 converges. Here, $q_x^T W_x^k p_x$ is expected similarity of simultaneous k random walks on G and G' i.e. the probability of simultaneously stopping after a random walk of length k on graph G and G'.
Here is the most important lemma of the paper:
Lemma 12: If $A \in X^{n \times m}$, $B \in R^m \times p$, and $C \in X^{p \times q}$, then

$$vec(\phi(A)B\phi(C)) = (\phi(C)^T \otimes \phi(A))vec(B) \in R^{nq \times \mathbf{1}} \tag{5}$$

LHS requires $O(n^3 d)$ computations (working in feature space)
while RHS takes $O(n^4)$ kernel computations. As the number of labels is generally less than number of nodes in a graph, so LHS is faster.

The proof of the kernel 1 being p.s.d requires that the coefficients $\mu(k)$ are such that it converges. The proof heavily uses lemma 12. We can write the terms of the kernel $q_x^T W_x^k p_x$ as $p_k(G^T)p_k(G')$. Each of these terms are p.s.d. kernels and non-negative linear combination over p.s.d. kernels which has pointwise limit is also p.s.d. kernel.

1.2.1. Notations

n is the number of nodes in a graph, m is the number of graphs (so kernel matrix has m^2 elements), and d is the number of labels in the graph. Sparsity means there are $O(n)$ edges in the graph.

1.2.2. Kernel is generalized

Equation 1 is a generalized kernel as by setting different values of $\mu(k)$ we can obtain specific kernels used in past:

1. When $\mu(k) = \lambda^k$ then we can get the kernel given by Kashima et al.(2004):

$$\sum_{k=0}^{\infty} \lambda^k q_x^T W_x^k p_x = q_x^T (1 - \lambda W_x)^{-1} p_x \tag{6}$$

which is just a sum of infinite geometric progression. This is the kernel we worked on.

2. When $\mu(k) = \lambda_k$ and $p_i = q_i = 1/n$ with linear feature map, then kernel given by Gartner et al.(2003) can be recovered:

$$k(G, G') = \frac{1}{n^2 n'^2} \sum_{i=1}^{n} \sum_{j=1}^{n'} \sum_{k=0}^{\infty} \lambda_k [\tilde{A}_x^k]_{ij} \tag{7}$$

3. In the previous Kernel by Gartner et al. (2003), we can replace $\lambda_k = \frac{\lambda^k}{k!}$ to get exponential kernel defined as:

$$k(G, G') = \frac{1}{n^2 n'^2} \sum_{i=1}^{n} \sum_{j=1}^{n'} [e^{\lambda \tilde{A}_x}]_{ij} = \mathbf{e}^T e^{\lambda \tilde{A}_x} \mathbf{e} \tag{8}$$

where $\mathbf{e}$ is a column vector of ones.

1.2.3. Computing Kernel Efficiently

Here, efficient computations need to be employed as for kernel 6, time taken for inverse is of the order of $O(n^6)$. There are many methods for efficient computation and all of them use lemma 12 except the last two: Spectral decomposition method and Nearest Kronecker Product Approximation.

1. Sylvester Equation Methods:
   The following generalized form of Sylvester equation can be solved to obtain M

$$M = \sum_{i=1}^{d} \lambda ({}^iA') M ({}^iA^T) + M_0 \tag{9}$$

   Using lemma 12 (5) we can obtain:

$$(I - \lambda \sum_{i=1}^{d} ({}^iA) \otimes ({}^iA')) vec(M) = vec(m_0) = p_x \tag{10}$$

   Hence, the given M, the kernel can be obtained by:

$$q_x^T vec(M) = q_x^T (I - \lambda W_x)^{-1} p_x \tag{11}$$

   Given M, the graph kernel can be obtained in $O(n^2)$ time, but for $d = 1$, computing M takes $O(n^3)$. The problem is that it can only

be solved for $d \leq 2$ using matlab's dlyap which also doesn't handle sparsity. Hence, for labeled graphs (i.e. $d \geq 2$), Nearest Kronecker Product Approximation is used.

2. Conjugate Gradient Methods:
   Here, the following equation is solved for x;

$$(I - \lambda W_x)x = p_x \tag{12}$$

   but, if computing Mv takes $O(m)$ time and effective rank of M is r then for solving $Mv = u$, CG solver takes $O(r)$ iterations and hence $O(rm)$ time. So, here we need to optimize the multiplication time. If W is $n^2 \times n^2$) then then computing Wy will takes $O(m^4)$ time. So, we again use lemma 12:

$$W_x y = (\phi(X) \otimes \phi(X'))vec(Y) = vec(\phi(X')Y\phi(X)^T) \tag{13}$$

   which can be computed in $O(dn^3)$ time. For sparse graphs it will only take $O(n^2)$ time.

3. Fixed Point Iterations:
   Another way to solve 12 is to expand it as:

$$x_{t+1} = p_x + \lambda W_x x_t \tag{14}$$

   where $x_0 = p_x$. Its guaranteed to converge in k iterations if all eigenvalues of λW_x lie inside unit disk. Its ensured if $\lambda \leq$ inverse of largest eigen value of W_x. It also involves lemma 12 as everything that speeds up $W_x \times x_t$ is helpful. So, worst case time complexity goes to $O(dn^3)$. Here,

$$k = \bigcirc(\frac{\ln \epsilon}{ln\lambda + ln|\xi_1|}) \tag{15}$$

4. Spectral Decomposition Method:
   This method works for all $\mu(k)$ and not just geometric as seen above. It involves spectral decomposition of W_x.

$$k(G, G') = \sum_{k=0}^{\infty} \mu(k) q_x^T (P_x D_x P_x^{-1})^k p_x = q_x^T P_x (\sum_{k=0}^{\infty} \mu(k) D_x^k) P_x^{-1} p_x \tag{16}$$

   The good thing is that D is diagonal, so to invert D, invert each entry and to take its exponential, exponentiate each entry. But computing

spectral decomposition of $n^2 \times n^2$ matrix takes $O(n^6)$ time. So, intead we do spectral decomposition of matrices whose product is W_x and by property of Kronecker product, it is same as above:

$$k(G, G') = (q_i^T P_i \otimes q_j^T P_j)(\sum_{k=0}^{\infty} \mu(k)(D_i \otimes D_j)^k)(P_i^{-1} p_i \otimes P_j^{-1} p_j) \quad (17)$$

Here, left and right side in RHS can be computed using $O(n^2)$, middle summation in $O(pn^2)$ where $O(p)$ is the time taken for computing power series and precomputation of spectral decomposition of each of the m graphs take $O(mn^3)$ time, so $O(mn^2(n + mp))$ overall. In practice, $p = 1$ (as power series of known limit). But, it doesnt work for labeled graph so, Nearest Kronecker Product Approximation is used for this solver as well.

5. Nearest Kronecker Product Approximation:
   Here S and T are computed such that:

$$W_x \approx S \otimes T \quad (18)$$

which can be obtained by solving:

$$min_{S,T} norm||W_x - S \otimes T||_F \quad (19)$$

which is a frobenius norm. For this, instead of W_X, its permuted version: $\hat{W}_x$ is used. It is solved iteratively. If W_x is the sum of d kronecker products, then $O(dn^2)$ per iteration, if each graph is sparse then $O(n^2)$ time for each iteration is taken. The number of iterations required are:

$$k' = \bigcirc(\frac{\ln n}{ln|\xi_1| - ln|\xi_2|}) \quad (20)$$

2. Experiments

2.1. The different datasets that were available :

Dataset	avg. nodes / graph	avg. edges / graph
MUTAG	17.72	38.76
PTC	26.70	52.06
E (Enzyme)	32.63	124.27
PROTEIN	38.57	143.75

As Professor Harish Karnick had suggested we checked out the social media graphs but they were typically of the order of hundreds of thousands of nodes and hence our implementation whose best performance is in the $O(n^3)$ couldn't deal with it.

2.2. Time Statistics

Dataset	Sylvestre Equations	Conjugate Gradient	Shortest Path	Product Graph
MUTAG	7.7279s	5.6300 s	5.7280 s	17.3661 s
PTC	15.2994 s	9.2300 s	22.5813 s	28.9442 s
E	19.0723 s	10.8200 s	27.1933 s	100.49 s
PROTEINS	67.0005	36.1900 s	166.949 s	498.35 s

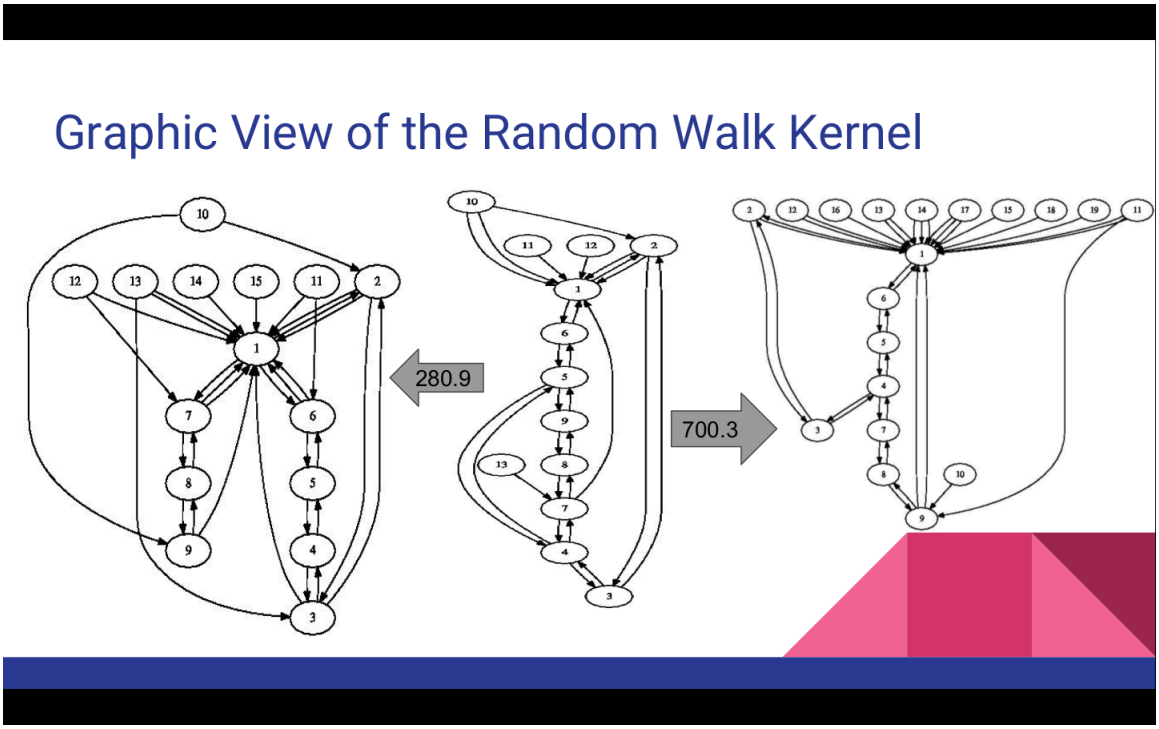

Figure 1: Graph 1

2.3. An example of graph and similarity

As 1 shows we are considering the 2 graphs with which it has the most (except itself) and the least kernel value respectively among all the graphs in the MUTAG set of graphs. The value in Gray arrow represents the kernel value assigned by calculating the Random Walk Kernel between the 2 graphs.

3. Conclusion

This project allowed us to look into the intriguing field of Graph Kernels and since the problem of Graph Isomorphism is NP complete hence it was an interesting project to realize that 2 graphs can be compared by taking simultaneous random walks on the 2 graphs and even the fact that the walk on a product graph represents a simultaneous walk on the 2 graphs .

We also realized via Shortest Path Kernels that substructures in graphs play a critical role in determining its intrinsic character.

Though we had some ideas about building our own Kernels using approximation algorithms such as the Steiner Vertices but we couldn't prove the positive semi-definiteness of the resulting kernel.

4. References

[1] H.-P. K. Karsten M. Borgwardt, Shortest path kernels, ICDM (2005).

[2] K. M. B. SVN Vishwanathan, Nicol N. Schraudolph, Graph kernels, Journal of Machine Learning Research (2010).