

# Seminar Report

## Semi-supervised Learning using Graph Kernels

Sandeepkumar B. Satpal  
Roll No. 05329004  
sandeep@it.iitb.ac.in

Guide: Prof. Sunita Sarawagi  
KReSIT, IIT Bombay

March 29, 2006

### Abstract

Traditional classifiers need labeled data to build a model. However, obtaining labeled instances are often expensive, time consuming, and difficult while unlabeled instances are obtained relatively easy. But, there must be a way to use these unlabeled instances. Semi-supervised learning addresses this problem by using large amount of unlabeled data, together with labeled data to build a classifier. There are many semi-supervised learning methods which includes *EM with generative mixture model, self-training, co-training, transductive support vector machine, and graph based methods*. In *graph based methods*, instances are represented as a node in the graph and edges are represented as a similarity ( dissimilarity ) measure between two nodes. This report provides an overview of few *graph based methods*, which uses *kernel* function to determine the label of all unlabeled instances.

## 1 Introduction

Semi-supervised learning methods uses both labeled and unlabeled data for training. Unlabeled data used to modify the hypothesis obtained from labeled data alone. Semi-supervised learning can be either *transductive* or *inductive*. *Transductive* learning only works on labeled and unlabeled training data, and cannot handle unseen data. While inductive learning handle unseen data also.

Kernel function is a function which returns the similarity measure between two instances given as a input. Suppose there is a real-valued function  $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  with the property that there exists a map  $\phi : \mathbb{R}^d \rightarrow \mathbb{H}$  into a dot product “feature” space  $\mathbb{H}$  such that for all  $x_i, x_j \in \mathbb{R}^d$ , we have  $\phi(x_i)\phi(x_j) = k(x_i, x_j)$ .  $k(x_i, x_j)$  is called kernel function which can be viewed as nonlinear similarity measure. Matrix with each element  $k(x_i, x_j)$  at  $K(i, j)$  is called kernel matrix. Size of the matrix is equal to the total number of instances on which matrix is defined. Kernel matrix must be positive definite. [1] discuss some other properties of kernel matrix and proof of its positive definiteness. Kernel

representation offer an alternative solution by providing the data into a high dimensional feature space to increase the computational power of the linear learning machines. By replacing the inner product with an appropriately chosen *kernel function*, one can implicitly perform a non-linear mapping to a high dimensional feature space without increasing the number of tunable parameters.

Graph kernels are the special type of kernels where we used the structured information to find the similarity between two graphs or two nodes in the same graph. Problem of finding whether two graphs are isomorphic is hard. Hence Kernel functions on graph tries to approximate the difference and find out the similarity between two graphs. Most method based on the idea of objects decomposed into substructures, finding the similarity between those substructures, and finally combining those similarity to find out the similarity between objects.

Rest of the report is organized as follows; Section 2 provides various definitions and notations which I will be using in this report. Section 3 discuss the requirements of graph kernels. Section 4 explains two variants of graph kernel used for semi-supervised learning. Section 5 explains different types of graph kernel. [2] suggest graph kernel based on count, [2] and [3] explains graph kernel based on random walk, [4] explains graph kernel using subtree pattern, finally [5] discuss graph kernel based on shortest path. All four methods also explains advantages, problems and solutions. Section 6 discuss one application of web graph which applies semi-supervised learning to assign label to unlabeled nodes using graph kernel. Finally, the report concludes with the summary and pointers to reference.

## 2 Definitions and Notations

A graph  $G$  is described by a finite set of Vertices  $V$ , and finite set of Edges  $E$ . It is commonly denoted by  $G(V, E)$ .  $V$  is the set of all vertices:  $V = \{v_i\}_{i=1}^n$ .  $E$  is the set of all edges. For undirected graph, each edge is a set containing two vertices. For directed graph, each edge is a tuple containing initial and final vertices:  $E \subseteq VXV$ . The neighborhood of a vertex  $v$  in a directed graph  $G(V, E)$  is given by,  $\delta^+(u) = \{v : (u, v) \in E\}$  and  $\delta^-(u) = \{v : (v, u) \in E\}$ .  $|\delta^+(u)|$  is called 0 of a vertex  $u$  and  $|\delta^-(u)|$  is called 1 of a vertex  $u$ .

In labeled directed graph, each node and edge has some label associated with it :  $\text{label}(v) \in L$  where  $v \in V$  and  $\text{label}(e) = (\text{label}(u), \text{label}(v)) = \text{label}(u, v)$  where  $(u, v) \in E$  and  $u \in V, v \in V$ .  $L$  is a set of all labels. Direct product of two labeled graphs  $G_1(V_1, E_1)$  and  $G_2(V_2, E_2)$  is denoted by  $G_1XG_2$ . Vertex and edge of a direct product graphs are respectively define by,

$$V(G_1XG_2) = \{(v_1, v_2) \in V_1XV_2 : (\text{label}(v_1) = \text{label}(v_2))\}$$

$$E(G_1XG_2) = \{((u_1, u_2), (v_1, v_2)) \in V^2(G_1, G_2) : (u_1, v_1) \in E_1 \wedge (u_2, v_2) \in E_2 \wedge (\text{label}(u_1, v_1) = \text{label}(u_2, v_2))\}$$

A walk  $w$  of a graph  $G(V, E)$  is a sequence of vertices  $w = v_1v_2 \dots v_n$ , where  $v_i \in V$ , and  $(v_i, v_{i+1}) \in E$ . The length of a walk is equal to the number of edges in the sequence. A path is a walk in which  $v_i \neq v_j \Leftrightarrow i \neq j$ . A cycle is a path with  $(v_{n-1}, v_n) \in E$ . For a graph  $G(V, E)$ , node  $r \in V$  is a subtree pattern

rooted at  $r$ . If  $t_1, t_2, \dots, t_n$  are subtree pattern rooted at  $r_1, r_2, \dots, r_n$  respectively then  $r, t_1, t_2, \dots, t_n$  is a subtree pattern rooted at  $r$  iff  $(r, r_1), (r, r_2), \dots, (r, r_n) \in E$

Two graphs are said to be isomorphic, if there is an edge preserving bijection between all vertices in one graph and all vertices in other graph.

A graph is said to be homomorphic to another graph if there is an edge preserving surjection between all vertices in one graph and all vertices in other graph.

### 3 Requirements of Graph kernel

1. The graph kernel must be positive definite.
2. Its computation should be possible in polynomial time.
3. It should be applicable to all kinds of graphs, not just small subset of graphs.

### 4 Variants of graph based semi-supervised learning

One variant of graph based semi-supervised learning defines a graph where the nodes are either labeled or unlabeled and edges represent similarity between two nodes. In this case, each node is one training example. Let  $N_L$  be the set of labeled nodes and  $N_{UL}$  be the set of unlabeled nodes. Hence  $N_L = \{(n_i, y_i): \text{label}(n_i) = y_i\}$  and  $N_{UL} = \{n_i\}$ . Other variant defines a set of graphs where each graph is either labeled or unlabeled. Here, each graph represents one training examples. Let  $G_L$  be the set of labeled graphs and  $G_{UL}$  be the set of unlabeled graphs. Hence  $G_L = \{(g_i, y_i): \text{label}(G_i) = y_i\}$  and  $G_{UL} = \{G_i\}$ .

### 5 Types of Graph Kernel

In this section, we deal with the later variant where kernel function takes two graphs as input and output the similarity between them. Let  $K_{n*n}$  be the kernel matrix, where  $n$  is the total number of graphs (labeled and unlabeled) and each term  $K_{ij}$  is given by  $K(G_i, G_j)$ .

#### 5.1 Graph kernel based on count

Let  $X_G$  represent a vector for graph  $G$ . Let kernel function  $K(G_i, G_j)$  be the inner product of two vectors  $X_{G_i}$  and  $X_{G_j}$ .

$$K(G_i, G_j) = X_{G_i} X_{G_j}^T \tag{1}$$

The most simplest way of defining a vector  $X_G$  is to define each element of a vector using number of times a particular vertex label appears in graph.

$$X_G = \left( \frac{\#label(l_1, G)}{|V|}, \frac{\#label(l_2, G)}{|V|}, \dots, \frac{\#label(l_m, G)}{|V|} \right) \tag{2}$$

where  $\#label(l_i, G)$  is the number of times vertex label  $l_i$  appears in the graph, and  $m$  is the total number of labels.

$$K(G_i, G_j) = X_{G_i} X_{G_j}^T \quad (3)$$

$$= \frac{1}{|V_i||V_j|} \sum_{v_1 \in V_i} \sum_{v_2 \in V_j} k(v_1, v_2) \quad (4)$$

$$k(v_1, v_2) = I(v_1, v_2) \quad (5)$$

where  $I$  is an indicator function that returns 1 when the label of two nodes  $v_1$  and  $v_2$  are equal and 0 otherwise. In this case the graph is decomposed into substructures which is simply a node.

### Advantages

1. Very simple to calculate.
2. Complexity is  $O(nm)$  where  $n, m$  are the number of nodes in graphs  $G_1$  and  $G_2$  respectively.

### Disadvantages

1. It does not consider the connectivity of graph, hence does not incorporate any local information around nodes  $v_1$  and  $v_2$ .
2. It does not perform well for complex graph.

## 5.2 Graph kernel based on random walk

Instead of just using Indicator function for nodes  $v_1$  and  $v_2$ , we can redefine  $k(v_1, v_2)$  so as to take higher score when not only the labels of  $v_1$  and  $v_2$  are equal but also the labels of edges adjacent to these nodes are equal and the further labels of nodes and edges are identical and so on. The new definition of  $k(v_1, v_2)$  is given as :

$$\begin{aligned} k(v_1, v_2) &= pk_0(v_1, v_2) \\ &+ p(1-p) \sum_{\substack{e_1 \in A(v_1) \\ e_2 \in A(v_2)}} k_1(v_1, v_2, e_1, e_2) \\ &+ p(1-p)^2 \sum_{\substack{e_1 \in A(v_1) \\ e_2 \in A(v_2)}} * \sum_{\substack{e'_1 \in A(\delta(v_1, e_1)) \\ e'_2 \in A(\delta(v_2, e_2))}} k_2(v_1, v_2, e_1, e_2, e'_1, e'_2) \\ &+ \dots \end{aligned} \quad (6)$$

where,

$$k_0(v_1, v_2) = I(v_1, v_2) \quad (7)$$

$$k_1(v_1, v_2, e_1, e_2) = k_0(v_1, v_2) * \frac{I(e_1, e_2) * I(\delta(v_1, e_1), \delta(v_2, e_2))}{|A(v_1)||A(v_2)|} \quad (8)$$

$$\dots \quad (9)$$

For details formulae please refer [2].  $A(v)$  is a set of edges adjacent to  $v$  and  $\delta(v, e)$  is a transition function that returns the vertex at other side of  $e$  adjacent to  $v$ .

For every vertex in one graph it calculates the similarity between every other vertex in other graph and each such computation takes lots of time if the graph is dense. Basically, this method does not consider random walk but all possible walks of particular length configured by user.

If the graphs are similar then there is high probability that the two random walks from two different graphs matches. The new method on random walk suggested by [3]. In this method, the random walk  $h = (h_1, h_2, \dots, h_l)$  is generated as follows.  $p_s(h_1)$  be the initial probability of selecting first node for the random walk  $h$ ,  $p_t(h_i|h_{i-1})$  be the transition probability, and the random walk may end at node  $i - 1$  with probability  $p_q(h_{i-1})$ .

Hence

$$\sum_{j=1}^{|G|} p_t(j|i) + p_q(i) = 1 \quad \forall i \quad (10)$$

The probability of a random walk for a given graph is given by

$$p(h|G) = p_s(h_1) * \prod_{i=2}^l p_t(h_i|h_{i-1})p_q(h_l) \quad (11)$$

Let  $h$  and  $h'$  be the random walk of two graphs  $G_1$  and  $G_2$  respectively. The joint kernel on the random walk is given by

$$K_r(z, z') = \begin{cases} 0 & (l \neq l') \\ K_v(l(h_1), l(h'_1)) \prod_{i=2}^l K_e(l(h_{i-1}, h_i), l(h'_{i-1}, h'_i)) * K_v(l(h_l), l(h'_l)) & (l = l) \end{cases}$$

where  $z = (G, h)$ , and  $l(h_i)$  is label associated with node  $h_i$ .  $K_v$  is the indicator function on node and  $K_e$  is the indicator function on edge.

The kernel function on two graphs  $G_1$  and  $G_2$  is given by

$$K(G_1, G_2) = \sum_{l=1}^{\infty} \sum_h \sum_{h'} p(h|G)p(h'|G')K_r(z, z') \quad (12)$$

For detail expansions of the formulae. please refer [3]

### Advantages

1. This method consider the connectivity of graph and incorporate the local information around nodes.

### Problems

1. Since every longer walk contains same small walk most of the time, small identical substructures in input graph can lead to high similarity scores. This problem of generating high score due to visiting smaller substructures large number of times is called *Tottering*.

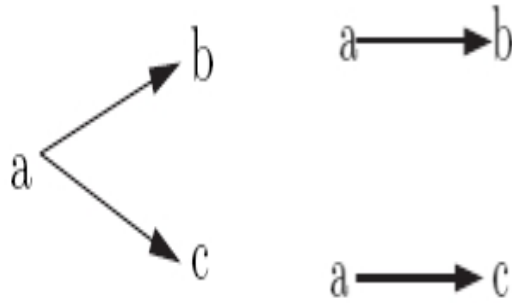


Figure 1: Directed Graph

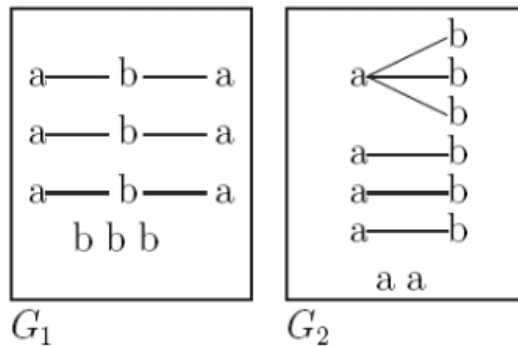


Figure 2: Undirected graphs

- When we map two different graphs to feature space using random walk then they may map to the same point in feature space.

Consider directed and undirected graphs shown in Figure 1 and Figure 2 respectively. In both the figures, two graphs are different but they map to the same point.

### 5.3 Graph kernel based on subtree pattern

Above problem arise because we do not consider the subtree structure in random walk. [4] discuss graph kernel which uses subtree pattern. Each element of a feature vector of graph is the number of times a particular subtree pattern appears in the graph. Kernel function on graphs  $G_1$  and  $G_2$  based on subtree pattern is given as

$$K(G_1, G_2) = \sum_{r \in V_1} \sum_{s \in V_2} K_{r,s,h} \tag{13}$$

where  $h$  is the height of a tree. If  $h = 1$ , then  $K_{r,s,h} = 1$  if  $\text{label}(r) = \text{label}(s)$  and 0 otherwise. For  $h > 1$ ,  $K_{r,s,h}$  can be computed as follows:

$$M_{r,s} = \{R \subseteq \delta^+(r) * \delta^+(s) \mid (\forall(a,b)(c,d) \in R : a = c \Leftrightarrow b = d) \wedge (\forall(a,b) \in R : \text{label}(a) = \text{label}(b))\}$$

$$K_{r,s,h} = \lambda_r \lambda_s \sum_{R \in M_{rs}} * \prod_{(r',s') \in R} K_{r',s',h-1} \quad (14)$$

### Problems

1. It also suffers from *Tottering*, since it visits nodes at higher level most of the times
2. The number of nodes to consider grows exponentially with the height of the subtree.

### 5.4 Graph kernel based on Shortest path

Graph kernels based on walks, subtree have been proposed. As a general problem these kernels are either computationally expensive or limited in their expressiveness. To overcome this limitations, [5] suggested graph kernel based on shortest path. In this method, first transform the original graph  $G$  into shortest-path graph  $S$ .  $S$  contains same number of nodes as in  $G$ . Unlike  $G$ ,  $S$  is a complete graph where each edge is a set of labeled nodes which represents the shortest distance between them.

Let  $S_1(V_1, E_1)$ ,  $S_2(V_2, E_2)$  be the shortest path graph of  $G_1$ ,  $G_2$  respectively. Shortest path graph kernel is defined as,

$$K(S_1, S_2) = \sum_{e_1 \in E_1} \sum_{e_2 \in E_2} k_w^1(e_1, e_2) \quad (15)$$

where  $k_w^1$  is a positive definite kernel on edge walk of length 1.

### Advantages

1. The shortest path graph kernel avoids *Tottering*
2. Shortest path graph can be obtained in  $O(n^3)$  time and shortest path graph kernel can be computed in  $O(n^2 * m^2)$  or  $O(n^4)$  if  $n > m$  as there are  $n^2$  and  $m^2$  edges in  $S_1$  and  $S_2$  respectively, while total runtime complexity of random walk graph kernel is  $O(n^6)$ .

### Improvements in shortest path graph kernel

The computational cost can be reduced further by storing some extra information with each edge like number of nodes in that set. The kernel function  $k_w^1$  can be 0 when number of nodes in two sets are not equal.

## 6 Semi-supervised learning on Web-graph

In web graph, each node represents a document and link between two nodes is the hyperlink which connects two nodes. Each node is either labeled or unlabeled. Our task is to provide label to all unlabeled nodes. Web graph has following properties

1. If there are many common words in the two documents or
2. If two documents are cited by same document

Then there is a high probability that they belongs to the same category.

Based on these two properties, two different kernel function exist. One incorporates the link structure and other incorporates common words in the documents. We can combine these two kernel in various ways and get another kernel. For different types of combinations please refer [7] which has performed various experiments using different combination with different weightage to each kernel.

### Kernels for Text

Each document can be represented as a vector in feature vector space where each element in the vector is the frequency of a particular word appear in document. The whole graph can be represented as a matrix  $D_{nm}$ , where each column represent a document and each row represent a term.  $n$  and  $m$  are the total number of words and documents respectively. Each entry  $D(i, j)$  is the number of times  $i^{th}$  word occur in  $j^{th}$  document. Matrix  $D$  is also called as *term by document* matrix. We can define *document by document* matrix to be  $G = D'D$  and *term by term* matrix to be  $T = DD'$ .

The kernel function  $K_w$  (based on text ) on two documents ( or two nodes in the web graph ) is given as,

$$K_w(d_1, d_2) = \langle d_1, d_2 \rangle \quad (16)$$

$$= d_1' d_2 \quad (17)$$

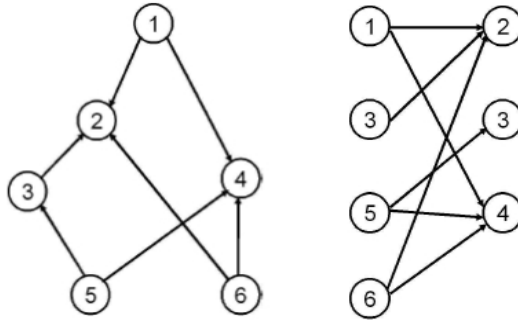


Figure 3: Hub-Authority model

### Co-citation Kernel

A graph  $G$  can be represented as a bi-partite graph  $G = (H, A, L)$  as shown in Figure 3. It is a special type of graph that consist of two sets of vertices called Hub and Authority denoted by  $H$  and  $A$  respectively and a set of edges denoted by  $L \in HXA$ . This bi-partite graph is called Hub-Authority web model where *authoritative pages* are those which are relevant to some topic and *hub pages* are those which links to relevant pages. Note that same document may appear in both the sets. In this case  $H = \{1, 3, 5, 6\}$  and  $A = \{2, 3, 4\}$ .

Each document from the authority set can be represented as a vector  $d$  where element  $i$  in the vector is either 1 or 0 depending on whether document  $i$  from hub links to this document or not. The kernel function  $K_a$  ( based on links ) on two documents is given as,

$$K_a(d_1, d_2) = \langle d_1, d_2 \rangle \quad (18)$$

$$= d'_1 d_2 \quad (19)$$

Where  $K_a(d_1, d_2)$  gives the number of documents points to both  $d_1$  and  $d_2$ .

One more property on web graph states that if two document link to same document then there is some probability that they belongs to same category.

To exploit this property, we can have one more kernel where each document from hub can be represented as a vector where each element  $i$  in the vector is either 1 or 0 depending on whether this document links to document  $i$  from authority set or not.

These kernel functions are combined with different weightage to get another kernel, which is used to find out the label of unlabeled node in graph.

The problem with this approach is that the co-citation kernel matrix only represents local relationships among nodes and ignores the global structures of graph. [6] exploits the global structure of graph to label the unlabeled node using Regularization Framework instead of using Kernel functions.

## 7 Summary and Conclusion

This report explains four different methods to find the similarity between two graphs. Each methods perform well for specific kind of graphs. If the graphs are disconnected or sparse then all methods perform well but kernel based on count works very fast. However for dense graph, counting method does not perform well, since it does not consider the local connectivity of graphs. If the number of times labels appear in the graphs are small, then random walk perform well, because there is very less probability that two graphs with the same random walk appear differently. However, random walk method fails if same labels appears large number of times. Also, random walk suffer with *tottering*. Subtree pattern overcomes former problem by finding similarity between subtrees of two graphs. But, this method is expensive, since its complexity grows exponential to the height of tree. Also, it fails to solve the problem of *tottering*. To overcome above limitations, kernel based on shortest path is introduced. Finding shortest path is polynomial time algorithm. This method uses kernel on walks of length one on the transformed graph, which too runs in polynomial time. Hence, it is efficient and does not suffer from tottering.

It also discusses one application on web graph where kernel functions are used to find the category of documents. There are various applications on semi-supervised learning where we used graph kernels. [8] discusses application on protein function prediction using graph kernels.

## References

- [1] Cristianini and Shawe-Taylor. An Introduction to Support Vector Machines.
- [2] Hisashi Kashima and Akihiro Inokuchi: Kernels for Graph Classification, In Proc. 1st ICDM Workshop on Active Mining (AM-2002), Maebashi, Japan, 2002.
- [3] Hisashi Kashima, Koji Tsuda, Akihiro Inokuchi: Marginalized Kernels Between Labeled Graphs. ICML 2003: 321-328
- [4] Jan Ramon and Thomas Gartner. Expressivity and Efficiency of Graph Kernels.
- [5] Karsten M. Borgwardt, Hans-Peter Kriegel: Shortest-Path Kernels on Graphs. ICDM 2005
- [6] Dengyong Zhou, Bernhard Scholkopf, Thomas Hofmann: Semi-supervised Learning on Directed Graphs. NIPS 2004. 2003
- [7] Thorsten Joachims, Nello Cristianini, John Shawe-Taylor: Composite Kernels for Hypertext Categorisation. ICML 2001: 250-257
- [8] Protein Function Prediction via Graph Kernels. ISMB 2005