UNIVERSITY OF CALIFORNIA
Santa Barbara

Representation Learning on Unstructured Data

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Fangqiu Han

Committee in Charge:

Professor Xifeng Yan, Chair

Professor Subhash Suri

Professor Linda Petzold

Sep 2016

The Dissertation of
Fangqiu Han is approved:

Professor Subhash Suri

Professor Linda Petzold

Professor Xifeng Yan, Committee Chairperson

Sep 2016

Representation Learning on Unstructured Data

by

Fangqiu Han

To my incredible parents who always encouraged me to pursue everything I want, who always told me there is a home behind me in my dark time, who always showed their endless love they have for me.

Acknowledgements

My deepest gratitude goes to my Ph.D. advisor, Professor Xifeng Yan. Without his guidance, nothing in this thesis can be achieved. I have benefitted tremendously by his vision in our field and his thinking on specific projects, His guiding transferred me from a student to a researcher, from a people who can learn, to a people to can think, analysis and solve real problem. Professor Yan also guides me beyond research. Like said in the old Chinese saying: "a teacher for one day equals a father for the whole life". I have gradually learned to be thorough and meticulous about not only every detail in my research but every part of my life.

I also owe many thanks to all my collaborators and lab-mates. Their knowledge and experiences greatly inspired me. I truly enjoy the moment when ideas come out from our discussions. I will never forget the time we spent together.

Curriculum Vitæ

Fangqiu Han

Education

09/2011 - 09/2016 **Ph.D.** Computer Science
Department of Computer Science
University of California, Santa Barbara
Research Areas: Data Mining, Machine Learning
Advisor: Dr. Xifeng Yan

09/2007 - 07/2011 **B.S.** Mathematics and Applied Mathematics
Shing-Tung Yau Honors Class
Chu Kochen Honors College
Zhejiang University

Work Experiment

06/2015 - 09/2015 **Software Engineer Intern at Facebook**
Project: **Dynamic Feature in News Feed Ranking**
Developed a dynamic feature to improve the Feed Ranker

06/2014 - 09/2014 **Research Intern at IBM T.J.Watson**
Project: **Distributed Expertise Representation**
Proposed a distributed representation for expertise modeling

02/2013 - 04/2013 **Visiting Scholar at BBN Technologies**
Project: **Modeling Network Structure**
Proposed a hierarchy network model for collaborative networks

07/2010 - 08/2010 **Research Intern at Microsoft Research Asia**
Project: **Scheduling and Data Allocation In Cloud Computing**
Proposed resource allocation algorithms in cloud operating system

Publication

[1] Yang Li, Chi Wang, Fangqiu Han, Jiawei Han, Dan Roth, and Xifeng Yan. Mining evidences for named entity disambiguation. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1070–1078, New York, NY, USA, 2013. ACM.

[2] Yang Li, Pegah Kamousi, Fangqiu Han, Shengqi Yang, Xifeng Yan, and Subhash Suri. Memory efficient minimum substring partitioning. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 169–180. VLDB Endowment, 2013.

[3] Fangqiu Han, Zhiyi Tan, and Yang Yang. On the optimality of list scheduling for online uniform machines scheduling. *Optimization Letters*, 6(7):1551–1571, 2012.

[4] Fangqiu Han, Subhash Suri, and Xifeng Yan. Observability of lattice graphs. *Algorithmica*, 76(2):474–489, 2016.

[5] Fangqiu Han, Shulong Tan, Huan Sun, Mudhakar Srivatsa, Deng Cai, and Xifeng Yan. Distributed representations of expertise. 2016.

[6] Yu Su, Fangqiu Han, Richard E Harang, and Xifeng Yan. A fast kernel for attributed graphs. 2016.

[7] Shengqi Yang, Fangqiu Han, Yinghui Wu, and Xifeng Yan. Fast top-k search in knowledge graphs. 2016.

[8] Honglei Liu, Fangqiu Han, Hongjun Zhou, Xifeng Yan, and Kenneth S Kosik. Fast motif discovery in short sequences. 2016.

Abstract

Representation Learning on Unstructured Data

Fangqiu Han

Representation learning, which transfers real world data such as graphs, images and texts, into representations that can be effectively processed by machine learning algorithms, has became a new focus in machine learning community. Traditional machine learning algorithms usually focus on modeling hand-crafted feature representations manually extracted from the raw data and performance of the model highly depends on the quality of the data representation. However, feature engineering is laborious, hardly accurate, and less generalizable. Thus the weakness of many current learning algorithms is not how well they can model the data, but how good their input data representation are.

In this thesis, we adopt learning algorithms both on representing and modeling the graph data in two different applications. In the first work, We first developed representation on nodes, and later apply a well-known VG kernel on this representation. In the second work, we show the power of representation captured by applying jointly optimization on the nodes representations and the model. The results of both work show significant improvement over traditional machine learning methods.

Contents

List of Figures

Chapter 1

Introduction

The performance of machine learning algorithms usually highly relies on the quality of their input feature representation. For a long time, these feature representations, which utilize understanding or prior knowledge of input data, are usually manually constructed by researchers and engineers. Taken an image classification task in which we aim to detect whether there is a car in the given image as an example, a good feature can be whether there exists any wheel in the image as the existence of wheels will greatly help to infer the existence of a car. These representations, though may offer great help in learning tasks, suffer from several weakness. First, manually constructing feature representations is laborious and requires rich knowledge and expertise of the task. Only experts are able to identify important features for a specific task and even for an expert, discovering impor-

tant task-related features can be costly. Second, these representations are lack of generalization as different tasks may need completely different features. Last, the quality of a manually constructing feature representation is constrained but the experts who created it. This limits the performance of the algorithm applied on it, especially when the difficulty of the task is beyond the experts‘ knowledge.

In this thesis, we adopt representation learning, which transfers real world data structures such as graphs, images and texts into representations that can be effectively processed by machine learning algorithms. To be specific, we developed two representations on graph data for two different tasks. Our first task is a classic graph classification problem in which we predicts labels for attributed graphs. While traditional methods directly apply classifiers on the graph, we first construct a novel real value representation for each node in the graph which encodes its attributes as well as its neighborhood information by propagating categorical attributes along edges. Two nodes with similar attributes and neighbors will have similar vector representations; computing the similarity of two graphs therefore resorts to matching the vectors of their nodes. We then adapt well known Vocabulary-Guided pyramid matching (VG) kernel on the representation in order on the representation. In the second work we aimed to learn representations on the nodes together with the modeling algorithm on a social graph called collaborative network. In collaborative networks, tasks are routed among a network

of experts until they are resolved, and our goal is to predict whether an expert can solve a task based on his task-solving history. An expert has to meet two constraints in order to solve a task: (1) Topic Match: the specialized areas of the expert shall match the topic of the task. (2) Difficulty Level Match: the difficulty level of the task should match the proficiency of the expert. Our idea is to represent the expertise of an expert by utilizing real valued vector in which each dimension represents the expertise level of an expert in a specific domain, i.e., the difficulty level of tasks in the domain that this expert can solve. By learning this expertise representation together with tasks representation, we achieve better performance in prediction whether an expect can solve a task compared to using standard classifiers.

It is worth to mention that the ideas behind these two representation tasks are fundamentally different. In the first task, feature learning and modeling are considered as two different tasks. We first apply unsupervised feather learning and then apply modeling algorithms on top of the feature representations. In the second work, we jointly optimize the feature learning and modeling as one task. The first approach is usually more efficient while the second representation could benefit from the modeling algorithm of the task.

The dissertation is organized as follows: We introduce a graph classification method by applying well known VG kernel on a new node representation in Chap-

ter 2. In Chapter 3, we propose a joint representation and modeling learning algorithm for expertise learning in a collaborative network. We conclude our work with Chapter 4.

Chapter 2

A Fast Kernel for Attributed Graphs

2.1 Introduction

Large graph databases are increasingly popular in many domains such as chemoinformatics [8], bioinformatics [10] and the web [6]. These graph datasets are characterized by their rich attribute information. For example, in chemoinformatics, molecules are often modeled as graphs, with atoms being nodes and covalent bonds being edges. Rich attributes are associated with both nodes and edges: categorical attributes on nodes like element types, numerical attributes on nodes like their partial charges and on edges like the spatial distance between

elements. Many interesting questions arise with these graph datasets, e.g., how to predict the mutagenicity of a chemical compound by comparing its graph representation with other chemical compounds having known functionality?

Graph kernels have been successfully applied to various graph problems [10, 36]. A graph kernel is basically a function measuring the similarity of two graphs. A significant advantage of the kernel method is that it can decouple data representation from the learning machines: As long as a graph kernel is provided, readily-available learning machines like SVM [11] or the kernel PCA [40] become directly applicable.

The large scale and heterogeneous attributes of modern graph data call for graph kernels which are (1) efficient to compute and (2) capable of handling the rich attribute information on nodes and edges. More specifically, we argue that a *linear-time* graph kernel that can handle *both* categorical and numerical attributes is desired, while being linear-time means the runtime scales linearly with respect to the graph size $n + m$, where n is the number of nodes and m the number of edges. Few graph kernels proposed so far achieve the two goals simultaneously. Some graph kernels [49, 27, 14] achieve linear-time computation. However, they are restricted to graphs with only categorical attributes since their efficiency mainly comes from the sparseness of the feature space resulted from the mutually orthogonal categorical attributes. A few recent graph kernels [20] try to

speed up computation on graphs with numerical attributes. Unfortunately, they are not linear-time kernels.

In this work, we propose a linear-time kernel for graphs with both categorical and numerical attributes. The proposed kernel, which we denote as the descriptor matching (DM) kernel, is based on a simple idea: Map each graph into a set of vectors (descriptors), and then apply a set-of-vectors matching kernel to measure graph similarity. We first propose a propagation based algorithm to generate feature vectors on nodes in linear time. By propagating categorical attributes along edges, we are able to generate a real vector for each node which encodes its attributes as well as its neighborhood information. Two nodes with similar attributes and neighbors will have similar vector representations; computing the similarity of two graphs therefore resorts to matching the vectors of their nodes. We then adapt the well-known Vocabulary-Guided pyramid matching (VG) kernel [23] to identify an approximately optimal matching between two vector sets, which is also done in linear time. We rigorously prove the linear scalability of the DM kernel. The most related work is the propagation kernel [43], which also propagate attribute information. It is initially proposed as a linear-time kernel for graphs with categorical attributes, and is recently extended to handle numerical attributes [42]. We will discuss about the differences later in the paper and also empirically compare with it.

We empirically experiment on both synthetic datasets and real-world datasets from chemo- and bioinformatics, and compare DM with several state-of-the-art graph kernels. Experiments on synthetic datasets confirm the linear scalability of the DM kernel. On real-world datasets, DM shows competitive performance in both classification accuracy and efficiency. Particularly, the experiment results demonstrate that DM can well exploit additional numerical attributes to improve classification accuracy, as opposed to when only using categorical attributes. Another salient characteristic of DM shown by the experiments is that its classification performance is very stable across different tasks. Even when its accuracy is not the best on a dataset, the difference to the best is usually small.

2.2 Related Work

We summarize existing graph kernels with an emphasis on computational complexity. The computation of a graph kernel is often done in two steps: (1) decomposing each graph into a set of features, and (2) comparing feature sets. In order to achieve overall linear scalability, a graph kernel has to be linearly scalable in both steps.

For the first step, many graph kernels choose to exhaustively enumerate a certain type of features in a graph, such as random walks [22, 31], paths [1], shortest paths [9, 20], subtrees [37], etc. Although algorithms have been proposed

to reduce the effect of combinatorial explosion, due to their exhaustive nature, these kernels are still inefficient and hard to be applied to large graphs with hundreds or more nodes. A few recently proposed graph kernels achieve linear scalability in the first step by limiting the size of their feature space [49, 27, 14]. Our kernel follows the same strategy: A graph is decomposed into a set of vectors on nodes. The idea of propagating categorical attributes to get local feature vectors is also employed by some other kernels [43, 51, 42], where a random walk based propagation scheme is used. However, as we show in Appendix A, the random walk based propagation process, if run for enough iterations, will end up with feature vectors irrelevant to the initial labeling of the nodes and their neighbors. Our propagation scheme, as we will present soon, generate feature vectors well encoding the labeling and neighborhood information.

The linear scalability in the second step is harder to achieve. Comparing all possible feature pairs in two sets results in a quadratic time complexity. Linear-time comparison becomes possible when graphs have only categorical attributes, which yields a sparse discrete feature space [49, 27, 14]. Take [49] for example. It decomposes a graph into a set of size-limited subtrees, hashes each subtree into a string, and then counts common strings via string equality check. However, for graphs with numerical features this strategy fails, as we have to take the similarity of the continuous features into account, other than merely making a

binary decision of whether two features are the same. [43, 42] try to tackle this problem via locality sensitive hashing, which is basically putting feature vectors into some uniform bins and then count. We employ a different approach. From a geometric point of view, our method identifies where the feature vectors really reside in the feature space and divide the space into non-uniform bins based on the real data distribution.

Our kernel seeks for a one-to-one matching between two sets of features, for which an efficient solution exists. The graph kernels proposed in [21] and [51] try to find an *optimal* one-to-one matching for their specific type of features. Unfortunately, they are not efficient and are not positive semi-definite kernels [56]. Our kernel efficiently identifies an approximately optimal correspondence between two feature sets by employing an existing set-of-vectors matching kernel, the VG kernel [23], whose computational complexity is linear with respect to the set size with mild adaptation.

2.3 Preliminaries

Following convention, we define an undirected graph G as a 4-tuple $(V, E, \mathcal{L}_c, \mathcal{L}_n)$, where V is the set of nodes, E the set of edges, and $\mathcal{L}_c$ and $\mathcal{L}_n$ the labeling function for categorical and numerical attributes, respectively. $\mathcal{L}_c : V \rightarrow \Sigma$, where $\Sigma = l_1, \ldots, l_L$ is the alphabet of categorical attributes. The labeling function for

numerical attributes $\mathcal{L}_n : V \to \mathbb{R}^K$ assigns K numerical attributes to each node. For simplicity, we will work on a graph dataset with N graphs, and each graph has n nodes and m edges. We define graph size as $n + m$, and call a graph kernel a *linear-time* kernel if its runtime complexity is linear to graph size. $\mathcal{N}(v)$ is the neighborhood of node v, which is the set of nodes directly connected to v.

Throughout the paper, we will use the term *set* to denote a multiset which allows duplicate elements. Given two sets $\mathbf{X}$ and $\mathbf{Y}$ where $n_1 = |\mathbf{X}|$, $n_2 = |\mathbf{Y}|$, and $n_1 \leq n_2$, a one-to-one correspondence or a matching $\mathcal{M}(\mathbf{X}, \mathbf{Y}; \pi) = \{(\mathbf{x}_i, \mathbf{y}_{\pi_i}) | 1 \leq i \leq n_1\}$ matches every element in $\mathbf{X}$ to some unique element in $\mathbf{Y}$. $\pi = [\pi_1, \ldots, \pi_{n_1}], 1 \leq \pi_i \leq n_2$ is a permutation of indices where π_i specifies a match $(\mathbf{x}_i, \mathbf{y}_{\pi_i})$, for $1 \leq i \leq n_1$. We will use the terms one-to-one correspondence and matching interchangeably in the rest of the paper. We will also use the terms attribute and label interchangeably. We follow the kernel foundation in [46].

Definition 2.3.1 (Gram Matrix)**.** *Let $\mathcal{X}$ be a nonempty set. Given a function $k : \mathcal{X}^2 \to \mathbb{R}$ and elements $x_1, \ldots, x_m \in \mathcal{X}$, the $m \times m$ matrix K with elements $K_{ij} = k(x_i, x_j)$ is called the* gram matrix *(or* kernel matrix*) of k with respect to $x_1, \ldots, x_m$. A gram matrix is p.s.d. if it is a positive semi-definite matrix.*

Definition 2.3.2 ((Valid) Kernel)**.** *Let $\mathcal{X}$ be a nonempty set. A function k on $\mathcal{X} \times \mathcal{X}$ which for all $m \in \mathbb{N}$ and all $x_1, \ldots, x_m \in \mathcal{X}$ gives rise to a p.s.d. gram*

matrix is called a valid kernel*, or a p.s.d. kernel. We will simply refer to it as a* kernel.

It is easy to construct new kernels from existing ones. Given two kernels k_1 and k_2, and $\alpha_1, \alpha_2 \geq 0$, $\alpha_1 k_1 + \alpha_2 k_2$ is still a kernel, and the pointwise product $k_1 k_2$ defined as $(k_1 k_2)(x_1, x_2) = k_1(x_1, x_2) k_2(x_1, x_2)$ is also a kernel [46].

2.4 Descriptor Matching Kernel

2.4.1 Local Descriptor.

We first introduce a concept, *local descriptor*. A local descriptor (or simply descriptor) is a fixed-length real-valued vector associated with a node. It encodes the labeling information of the node, as well as the topological and labeling information in its neighborhood, thus serving as the *identity* of the node: *Similar nodes should have similar descriptors.* Descriptor similarity is defined based on their Euclidean distance, while node similarity is defined in a recursive manner: Two nodes are more similar if their attributes and neighborhood are more similar. With this property, it becomes meaningful to measure graph similarity by matching their node descriptors. A *descriptor generator* f is a function mapping a node v to a descriptor $f(v) \in \mathbb{R}^D$, where $D = \|f(v)\|$. $\mathcal{F}(G) = \{f(v) | v \in V(G)\}$ is the descriptor set of a graph G.

Now we define our descriptor generator. The basic idea is to capture the labeling *and* neighborhood information about a node by propagating categorical attributes. The outcome of the propagation process is a series of feature vectors for each node. The continuous features are the key for incorporating numerical attributes. Since the features are continuous, numerical attributes can be directly appended to the feature vectors. The idea is that the numerical attributes of a node, such as the partial charge value of an atom in a molecule, are a direct part of the node's identity. Other linear-time graph kernels like [49, 27, 14] are hard to incorporate numerical attributes because their features are discrete.

For better presentation, we first leave out numerical attributes. Because of the recursive nature of the node similarity definition, it is natural to generate descriptors via an iterative process in which nodes exchange information with their neighborhood. We therefore define the Stochastic Cascade (SC) descriptor generator. The SC descriptor of a node v, $f_{sc}(v) = (A_1(v), \ldots, A_L(v))$, is a vector of length L, with the ith component $A_i(v)$ indicating the *strength of association* between the categorical attribute l_i and the node. Intuitively, the more nodes with attribute l_i there are in $\mathcal{N}(v)$, the stronger the association will be. Let $\eta \in [0, 1]$ be a scalar, h be the number of iterations, we model this intuition via the following iterative process, which generates a sequence of descriptors $f_{sc}^{(r)}(v) = (A_1^{(r)}(v), \ldots, A_L^{(r)}(v)), 0 \leq r \leq h$ for v:

(1) Initialization:

$$A_i^{(0)}(v) = \begin{cases} 1 & \text{if } \mathcal{L}_c(v) = l_i, \\ 0 & \text{otherwise;} \end{cases}$$

(2) Updating:

$$A_i^{(r+1)}(v) = \begin{cases} 1 & \text{if } A_i^{(r)}(v) = 1, \\ 1 - \prod_{u \in \mathcal{N}(v)} (1 - \eta A_i^{(r)}(u)) & \text{otherwise,} \end{cases}$$

$$for\ i = 1, \ldots, L,\ 0 \leq r < h.$$

To understand the above process, let's focus on the attribute l_1. In the beginning of iteration r, the strength of association $A_1^{(r)}(v)$ is regarded as the probability of v propagating l_1 to all of its neighbors. Here η is a decay factor, or can be thought as the loss ratio of propagation. Initially, $A_1^{(0)}(v)$ is set to 1 if l_1 is v's categorical attribute, and otherwise 0. In each iteration k, $A_1^{(r+1)}(v)$ is updated to the probability of the node receiving *at least one* l_1 from its neighborhood: the probability of the neighboring node u not propagating l_1 to v in iteration r is $1 - \eta A_1^{(r)}(u)$, so the probability of v not receiving any l_1 from $\mathcal{N}(v)$ is $\prod_{u \in \mathcal{N}(v)} (1 - \eta A_i^{(r)}(u))$, therefore we end up with the above updating rule.

A competitor of our SC descriptor generator is a descriptor generator based on a random walk on graphs, which, although termed differently, has been exploited in some way in [43, 51, 42]. But we prove in Appendix A that it does not have the descriptor property, i.e., similar nodes should have similar descriptors. Two nodes

in a graph, as long as they have the same degree, will always end up with the same descriptors irrelevant to the initial labeling of the nodes and their neighborhoods[1].

Theorem 2.4.1. *The SC descriptors for N graphs can be computed in time $O(NLhm)$.*

Proof. In each iteration, each categorical attribute in Σ will be propagated for at most $2m$ times, and each propagation will incur an $O(1)$ number of operations, so the overall runtime complexity of computing SC descriptors for N graphs and h iterations is $O(NhmL)$. □

Numerical attributes are directly appended to the descriptors defined above. We normalize each numerical attributes to $[0, 1]$.

2.4.2 Descriptor Matching Kernel.

Definition 2.4.1 (Descriptor Matching Kernel)**.** *Given a base kernel k defined on sets of vectors, if we denote the set of SC descriptors of graph G in the rth iteration as $\mathcal{F}^{(r)}(G)$, the* descriptor matching kernel k_{dm} *on two graphs G_1 and G_2 is defined as:*

$$k_{dm}^{(h)}(G_1, G_2) = \sum_{r=0}^{h} k(\mathcal{F}^{(r)}(G_1), \mathcal{F}^{(r)}(G_2)).$$

[1] [42] suggested that, pragmatically, random walk based propagation can stop early without getting into the stationary states. We apply this strategy in evaluation.

Theorem 2.4.2. *For any $h \in \mathbb{N}$, $k_{dm}^{(h)}$ is positive semi-definite (p.s.d.) if k is p.s.d.*

Proof. This follows directly from the fact that p.s.d. kernels are closed under addition. □

The next step is to find a base kernel defined for two sets of vectors. There are three requirements for the base kernel: (1) Its computation must be efficient. More specifically, its time complexity should be linear with respect to graph size. (2) It should measure the similarity of two sets of vectors in an intuitive manner. (3) It is able to handle high-dimensional vectors. Putting all these requirements together, we choose the VG kernel [23] from computer vision. It identifies a one-to-one correspondence between two sets of vectors via non-uniform quantification, which makes it suitable for high-dimensional vectors since it can locate where the vectors really reside in the high-dimensional space and divide the space accordingly. Although the original VG kernel did not claim to be linearly scalable, we show next that with mild modification, its time complexity becomes linear with respect to graph size.

2.4.3 VG Kernel.

Given a descriptor generator f and two graphs G_1 and G_2, we now discuss how to define a kernel k to efficiently measure the similarity of their corresponding

descriptor sets $\mathcal{F}(G_1)$ and $\mathcal{F}(G_2)$. Suppose $\mathcal{F}(G_1) = \{\mathbf{x}_1, \ldots, \mathbf{x}_{n_1}\}$, $\mathcal{F}(G_2) = \{\mathbf{y}_1, \ldots, \mathbf{y}_{n_2}\}$, $n_1 \leq n_2$ and $\mathcal{M}(\mathcal{F}(G_1), \mathcal{F}(G_2); \pi)$ is a matching from $\mathcal{F}(G_1)$ to $\mathcal{F}(G_2)$, a set-of-vectors matching kernel k is defined as follow:

$$k(\mathcal{F}(G_1), \mathcal{F}(G_2)) = \sum_{i=1}^{n_1} w(\|\mathbf{x}_i - \mathbf{y}_{\pi_i}\|_2),$$

where $w(\cdot)$ is a weighting function. Note that under this definition k is not necessarily p.s.d.

Now the problem boils down to finding an appropriate matching. The most intuitive way is to find the optimal matching that maximizes $k(\mathcal{F}(G_1), \mathcal{F}(G_2))$, which can be formulated as the classic maximum weighted bipartite matching problem and solved by prominent algorithms such as the Hungarian algorithm [21, 51]. However, it is not favorable for two reasons: (1) The computational complexity is rather high (cubic), and (2) it results in a kernel which is not p.s.d. [56]. Another solution is discretization [43]. The idea is to map a vector into a 1-d histogram, and efficiently match vectors based on whether they fall into the same bin. It scales linearly, but the main problems are: (1) Bins are unweighted, or in other word, w is a constant function; (2) bins are orthogonal, so vectors in different bins are never matched. Nevertheless, the linear computational complexity is appealing. We choose the Vocabulary-Guided (VG) pyramid matching kernel, which is based on a somewhat similar idea, but in a more sophisticated manner. It aims to efficiently find an *approximately optimal* matching, and elegantly solves

both of the problems via replacing the 1-d histogram by a data-dependent multi-resolution histogram with non-uniformly shaped bins. We next reformulate it in a way suitable for our descriptor sets, and adapt it to ensure its linear scalability.

Pyramid construction. Suppose $\mathcal{G}$ is a set of N graphs and $\mathcal{F}(\mathcal{G}) = \{f(v)|v \in G, G \in \mathcal{G}\}$. The VG kernel starts off by partitioning the descriptor space into a pyramid of non-uniformly shaped regions/bins, which is built by performing hierarchical clustering on $\mathcal{F}(\mathcal{G})$. The pyramid structure is controlled by two hyper-parameters, the number of levels t, and the branching factor b. The jth bin at the ith level is denoted as $B_j^{(i)} = (\mathbf{X}_j^{(i)}, c_j^{(i)}, s_j^{(i)})$, where $c_j^{(i)}$ is its center, $s_j^{(i)}$ its diameter with $s_j^{(i)} = \max\{\|\mathbf{x_1} - \mathbf{x_2}\| \,|\, \mathbf{x_1}, \mathbf{x_2} \in \mathbf{X}_j^{(i)}\}$, and $\mathbf{X}_j^{(i)} \subseteq \mathcal{F}(\mathcal{G})$ the set of descriptors in the bin. Then the pyramid is denoted as $\{B_j^{(i)}\}_{0 \le i \le t-1, 1 \le j \le b^i}$, and is constructed as in Algorithm 1.

Lines 4, 9, and 10 compute bin diameters, i.e., the maximum distance between any two descriptors in the bin. The original VG kernel will compute the distance between each pair of descriptors and find the maximum, which results in a quadratic time complexity. We approximate it by two upper bounds, the doubled maximum distance from any descriptor in the bin to the center of the bin, and the diameter of the parent bin, as shown at line 9 and 10, respectively. This grants us linear scalability. Bin diameters are critical and will be used to compute bin weights. Later in §2.5 we empirically demonstrate that the DM kernel built on

Algorithm 1 Pyramid construction

Initialization:

$\mathbf{X}_1^{(0)} \leftarrow \mathcal{F}(\mathcal{G})$

$c_1^{(0)} \leftarrow 2 \times \frac{1}{|\mathcal{F}(\mathcal{G})|} \sum_{\mathbf{x} \in \mathcal{F}(\mathcal{G})} \mathbf{x}$

$s_1^{(0)} \leftarrow \max_{\mathbf{x} \in \mathcal{F}(\mathcal{G})} \|\mathbf{x} - c_1^{(0)}\|$

for $i = 0$ **to** $t - 2$ **do**

  for $j = 1$ **to** b^i **do**

    run k-means clustering to partition $B_j^{(i)}$ into b child bins $\{B_k^{(i+1)}\}_{(j-1)b+1 \leq k \leq jb}$

    for $k = (j-1)b + 1$ **to** jb **do**

      $s_k^{(i+1)} \leftarrow 2 \times \max_{\mathbf{x} \in \mathbf{X}_k^{(i+1)}} \|\mathbf{x} - c_k^{(i+1)}\|$

      $s_k^{(i+1)} \leftarrow \min(s_k^{(i+1)}, s_j^{(i)})$

the approximated VG kernel achieves promising performance in both efficiency and accuracy.

Multi-resolution histogram construction. Given a graph G and its descriptor set $\mathcal{F}(G)$, a multi-resolution histogram is constructed according to the pyramid structure. The multi-resolution histogram is defined as $\Psi(G) = [H^{(0)}(G), \ldots, H^{(t-1)}(G)]$ where $H^{(i)}(G) = [H_1^{(i)}, \ldots, H_{b^i}^{(i)}]$ is a 1-d histogram with b^i bins at the ith level, $0 \leq i \leq t-1$. Algorithm 2 shows how to construct $\Psi(G)$ by walking each descriptor through the pyramid and identifying its bin memberships along the way, where $p = (p_0, \ldots, p_{t-1})$ is a vector with p_i being the index of the bin where the descriptor is located at level i, $0 \leq i \leq t-1, 1 \leq p_i \leq b^i$.

Algorithm 2 Multi-resolution histogram construction

1: **for** $\mathbf{x} \in \mathcal{F}(G)$ **do**
2: $\quad p_0 \leftarrow 1$
3: $\quad H_1^{(0)} \leftarrow H_1^{(0)} + 1$
4: $\quad$ **for** $i = 1$ **to** $t-1$ **do**
5: $\quad\quad p_i \leftarrow \text{argmin}_j \|c_j^{(i)} - \mathbf{x}\|, (p_{i-1}-1)b + 1 \leq j \leq p_{i-1}b$
6: $\quad\quad H_{p_i}^{(i)} \leftarrow H_{p_i}^{(i)} + 1$

Matching multi-resolution histograms. The matching process goes from the finest level ($i = t-1$) to the coarsest level ($i = 0$). In this way, we will first consider matching the closest descriptors (at level $t-1$), and as we climb to the

higher levels in the pyramid, increasingly further descriptors are allowed to be matched. Given two multi-resolution histograms $\Psi(G_1)$ and $\Psi(G_2)$, the number of matches found in $B_j^{(i)}$ is derived via bin intersection:

$$\mathcal{I}_j^{(i)} = min(H_j^{(i)}(G_1), H_j^{(i)}(G_2)).$$

The number of *new* matches found in a bin is computed by subtracting the number of matches found in all its child bins from $\mathcal{I}_j^{(i)}$, which is the *true* number of descriptors matched in this bin:

$$\mathcal{J}_j^{(i)} = \begin{cases} \mathcal{I}_j^{(i)}, & i = t-1; \\ \mathcal{I}_j^{(i)} - \sum_{k=(j-1)b+1}^{jb} \mathcal{I}_k^{(i+1)}, & 0 \leq i \leq t-2. \end{cases}$$

We give an example in Appendix B to illustrate the above process. The VG kernel is defined as follow, where $w_{ij} = \frac{1}{1+s_j^{(i)}}$ is the weight of $B_j^{(i)}$ measuring how much a match found in the bin contributes to the overall similarity:

Definition 2.4.2 (VG Kernel)**.** *Given two descriptor sets $\mathcal{F}(G_1)$ and $\mathcal{F}(G_2)$, and the corresponding multi-resolution histograms $\Psi(G_1)$ and $\Psi(G_2)$, the* VG kernel *k_{vg} is defined as:*

$$k_{vg}(\mathcal{F}(G_1), \mathcal{F}(G_2)) = \sum_{i=0}^{t-1} \sum_{j=1}^{b^i} w_{ij} \mathcal{J}_j^{(i)}. \tag{2.1}$$

Theorem 2.4.3. *k_{vg} is p.s.d.*

Proof. We re-write Eq. (2.1) as $k_{vg}(\mathcal{F}(G_1), \mathcal{F}(G_2)) = \sum_{i=0}^{t-1} \sum_{j=1}^{b^i} (w_{ij} - p_{ij}) \mathcal{I}_j^{(i)}$, where p_{ij} is the weight associated with the parent bin of $B_j^{(i)}$, and that for $B_1^{(0)}$ is

set to 0. Since the bin intersection $\mathcal{I}$ is a p.s.d. kernel [44], and since p.s.d. kernels are closed under addition and scaling by a positive scalar, k_{vg} is a valid kernel as long as $w_{ij} >= p_{ij}$ for all bins. This is guaranteed by (1) w_{ij} is a monotonic decreasing function with respect to $s_j^{(i)}$, and (2) $s_j^{(i)}$ is not bigger than the diameter of its parent bin, which is guaranteed by the line 10 of Algorithm 1. □

Theorem 2.4.4. *Given N graphs and their corresponding descriptor sets, suppose the maximum number of iterations for k-means clustering is H, the N-by-N kernel matrix of k_{vg} can be computed in $O(N(Hb+N)tn)$.*

Proof. Let us examine the time complexity of each step.

First, the pyramid can be built in $O(HNntb)$. On one hand, the hierarchical clustering can be performed in $O(HNntb)$. It takes at most $O(Hb)$ operations to determine the bin membership for each descriptor at each level, and there are in total Nn descriptors. On the other hand, determining all of the bin diameters can be done in $O(Nnt)$, because at each level, each descriptor will be accessed exactly once. So the pyramid construction takes $O(HNntb)$ time.

Second, the N multi-resolution histograms can be constructed in $O(Nntb)$. It can be seen from that, for each of the n descriptor, it takes b comparisons to determine its bin membership at each level.

Finally, matching all pairs of multi-resolution histograms takes $O(N^2nt)$ time. Matching two multi-resolution histograms can be done in $O(nt)$ time via a sparse

representation of the multi-resolution histograms which only stores non-empty entries, and there are N^2 pairs to match. For implementation details, see [23].

Therefore, the overall time complexity is $O(HNntb+Nntb+N^2nt) = O(N(Hb+N)tn)$. □

Theorem 2.4.4 asserts the linear scalability of the VG kernel, which paves the way to the proof of the linear scalability of the DM kernel.

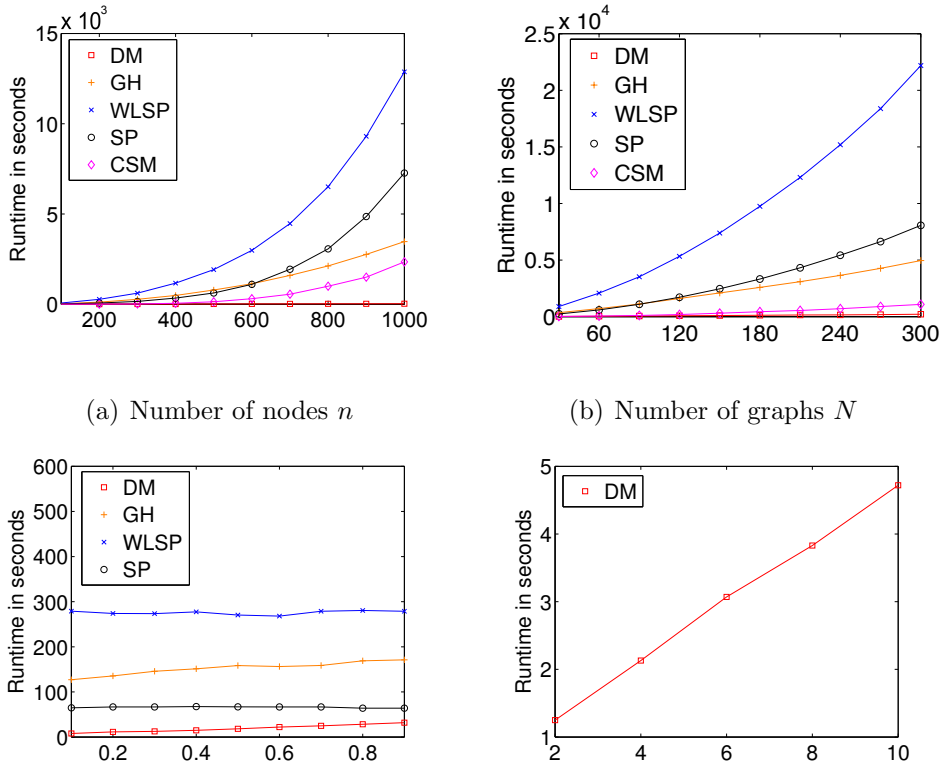
Theorem 2.4.5. *With k_{vg} as the base kernel, k_{dm} on a pair of graphs can be computed in a linear time with respect to graph size.*

Proof. For N graphs, directly following Theorem 2.4.1 and Theorem 2.4.4, k_{dm} can be computed in $O(NLhm + N(Hb + N)htn) = O(Nh(Lm + Htbn) + N^2htn)$ time, where h is the number of iterations, and the amortized cost for a pair of graphs is $O(\frac{h}{N}(Lm + Htbn) + htn) = O(\frac{1}{N}Lhm + (\frac{1}{N}Hb + 1)htn)$, which is linear with respect to graph size. □

2.5 Evaluation

Table 2.1: Statistics of the benchmark datasets

Dataset	MUTAG	ENZYMES	D&D	FR	FM	MR	MM	COX-2	BZR	DHFR	ER
# graphs	188	600	1178	344	351	336	349	303	306	393	446
# positive	125	-	691	121	143	152	129	148	157	126	181
# categorical attributes	8	3	82	20	19	19	21	7	8	7	10
Avg. # nodes	26.03	32.63	284	25.56	26.08	25.05	25.25	41.56	35.04	41.58	41.96
Max. # nodes	28	126	5748	109	109	109	109	56	57	71	93
Avg. # edges	27.89	62.14	716	25.96	26.53	25.4	25.62	43.8	37.5	43.71	43.96



(a) Number of nodes n

(b) Number of graphs N

(c) Graph density σ

(d) Number of iterations h

Figure 2.1: Runtime behavior on synthetic datasets.

We compare DM with state-of-the-art graph kernels: the propagation kernel (PK) [43, 42], the Weisfeiler-Lehman subtree (WL) kernel [49], the Weisfeiler-Lehman shortest-path (WLSP) kernel [50], the shortest-path (SP) kernel [9], the connected subgraph matching (CSM) kernel [32], and the GraphHopper (GH) kernel [20]. DM, PK, WL, SP, WLSP and GH are implemented in Matlab, VG is implemented in C++, and CSM is implemented in Java. DM, PK and WL are linearly scalable while others are not. WL can only be applied on graphs

with categorical attributes, while DM and PK can handle numerical attributes as well. In DM, we directly append numerical attributes to descriptors, while in PK, numerical attribtues are also propagated.

2.5.1 Runtime Analysis on Synthetic Datasets.

In this experiment, we test graph kernels on randomly generated graphs with both categorical and numerical attributes. The main goal is to confirm the linear scalability of the DM kernel. The results of WL and PK are similar to DM and are omitted.

Experiment setup. We randomly generate undirected graphs based on two parameters: the number of graphs N, and the number of nodes n. The default values are $N = 10$ and $n = 200$. Average node degree is set to 5 so that graph size increases linearly with respect to n. n nodes are first generated, then edges are randomly inserted until a certain number is reached. We additionally experiment on graphs with varying density $\sigma = \frac{2m}{n(n-1)}$, and also evaluate DM with varying number of iterations h. For DM, the default value of h is 10, and t and b are set to 4 and 10, respectively. For WLSP, the number of iterations h is set to 3. For CSM, the maximum size of subgraphs k is set to 5. When evaluating one parameter, all other parameters are fixed to the default values. Node categorical and numerical

attributes are randomly generated. The total CPU time to compute the N-by-N kernel matrix is reported.

Result analysis. The results are presented in Figure 2.1. Figure 2.1(a) shows the runtime behavior with respect to n. DM scales linearly with a small increase rate, while the runtime of other kernels increases at least quadratically. Figure 2.1(b) gives the runtime results with varying N. DM scales nearly linearly, which shows that the linear term with respect to N in the overall time complexity is dominating when N is moderate. In Figure 2.1(c), we show how the graph density, namely the number of edges m when n is fixed, affects the runtime of DM. As expected, the runtime of DM increases linearly. The result of CSM is not reported because its extremely high runtime when graphs are dense. We argue that, on real-world graphs, especially when graph size is large, m can rarely get up to $O(n^2)$, therefore a runtime complexity in $O(m + n)$ usually scales more elegantly than $O(n^2)$. Finally, Figure 2.1(d) shows that DM also scales linearly with respect to h.

2.5.2 Classification Performance on Real-world Datasets.

We experiment with 11 well-accepted benchmark datasets from chemo- and bioinformatics. MUTAG [17] is a set of 188 chemical compounds labeled according to whether or not they have a mutagenic effect on a bacterium. EN-

ZYMES [10] comprises of 600 enzymes represented by their secondary structure elements (SSEs), and the task is to classify each enzyme into one of the 6 EC top level classes. D&D is a datasets consisting of 1178 proteins where amino acids are modeled as nodes. The graphs are therefore much larger. The task is to predict whether a protein is an enzyme. The PTC [26] dataset contains chemical compounds labeled according to their carcinogenicity to rodents. Four datasets, mice (MM), female mice (FM), male rats (MR), and female rats (FR), are developed according to their effect on different rodents. We acquired the dataset from ChemDB [13]. We obtained four more chemical compound datasets from [54]: COX-2 is a dataset of 467 cyclooxygenase-2 inhibitors, BZR a dataset of 405 ligands for the benzodiazepine receptor, DHFR a dataset of 756 inhibitors of dihydrofolate reductase, and ER a dataset of 1,009 estrogen receptors. The task is to predict a chemical compound as active or inactive in a certain reaction.

All datasets have categorical attributes on nodes. MUTAG, MM, FM, MR and FR have a node numerical attribute, the partial charge of atoms. COX-2, BZR, DHFR and ER come with the 3D coordinates of atoms, based on which we compute the spatial distance between atoms, and use it as a numerical attribute on edges. We choose the 3D-length of the SSEs as a node numerical attribute for ENZYMES. The dataset statistics are reported in Table 2.1.

Evaluation scheme. We perform 10-fold nested cross-validation of C-Support Vector Machine provided by LIBSVM [11]. In each fold, all hyper-parameters are optimized by an extra 9-fold cross-validation on the training data only. The whole process is repeated for 10 times, and the mean and standard deviation of the classification accuracy over the 10 runs are reported. The reported runtime is obtained by running each kernel with the hyper-parameters most frequently selected by the model selection process. The initialization time for each kernel is included. The "one-against-one" strategy is adopted for the multi-class classification on ENZYMES. See Appendix C for the detailed configuration.

Graphs with only categorical attributes. We first experiment on graphs with only categorical attributes. The results are shown in Table 3. A method is bold-faced in the table if it achieves the highest accuracy, or is not significantly worse than the highest according to the student t test at $p = 0.05$. The results show that our DM kernel achieves comparable accuracy with other kernels. It is in top 3 on all the datasets except COX-2, and achieves the highest accuracy on MUTAG and D&D.

In terms of efficiency, among the linear-time kernels, DM is comparable with WL while in general slower than PK. For the other kernels, GH and CSM are less efficient than DM. Because WLSP and SP utilize the hash-and-check-equality strategy (cf. §3.6), they are quite efficient on datasets with small graphs like

MUTAG, COX-2, BZR, DHFR, and ER. However, these non-linear-time kernels are hard to scale to larger graphs, such as those in D&D. As a result, WLSP and CSM cannot finish within 2 days on D&D, SP takes over 4 hours, and GH takes 3 days. The reason why DM takes more time on D&D than WL is that the runtime of DM grows linearly with respect to L, the number of categorical attributes, while the time complexity of WL is not dependent on L, and $L = 82$ on D&D. Nevertheless, we can safely draw the conclusion that DM can scale to large graphs with a moderate number of categorical attributes, which is the common case in many applications.

Graphs with numerical attributes. We now test on graphs with additional numerical attributes, which are the main targets of this work. WL is not applicable in this case. Table 2.3 shows the experiment results. In terms of classification accuracy, DM is among the best on 9 out of the 10 datasets, and achieves the highest accuracy on 6 of them. The only kernel which is comparable in terms of overall classification performance is WLSP. PK, SP and GH are in general less competitive, while CSM can compete on several datasets. In terms of efficiency, since the incorporation of numerical attributes fails the hash-and-equality-check strategy, SP and WLSP become much slower. We compute the average ratio between the runtime of each method and the runtime of DM over all the datasets. DM is 29 times faster than SP, 18 times faster than GH, 80 times faster than

WLSP, and 53 times faster than CSM. The other linear-time kernel, PK, is more efficient than DM because of its simplicity. However, it is less competitive in accuracy, being among the best in only two datasets. If we consider all the 17 datasets in both Table 2.2 and Table 2.3, DM is significantly better than PK on 11 datasets, while PK wins on 2 datasets (under student t test at $p = 0.05$).

Comparing the results on the 6 common datasets (MUTAG, ENZYMES, COX-2, BZR, DBFR, and ER) in Table 2.2 and Table 2.3 shows each kernel's capability of exploiting numerical attributes. Particularly, we see that the accuracy of DM increases on each dataset after incorporating numerical attributes. On the contrary, on 5 of the 6 datasets, the accuracy of PK actually decreases[2]. This may imply that directly appending the numerical attributes to the SC descriptors is an effective way of exploiting numerical attributes. A more in-depth analysis is of interest for future study.

Another salient characteristic of DM regarding accuracy is that, while the accuracy of other kernels vary from dataset to dataset, DM consistently gives good accuracy. Even on COX-2 and BZR in Table 2.2 where DM seems to fall short, the difference between the accuracy of DM and the highest is small (3.3% at most).

[2]On COX-2, BZR, DHFR and ER, we run two experiments for PK: One uses the the inverse of the 3D-distance as edge weight; the other uses the 3D coordinates as node attributes. PK performed much worse in the former. We report the latter.

2.6 Conclusions

We introduced a linear-time graph kernel which can handle graphs with both categorical and numerical attributes. From experiments on both synthetic and real-world datasets, the proposed kernel showed promising performance in accuracy and efficiency. The proposed kernel is a good alternative to existing kernels for tasks involving small graphs. Moreover, it is among the first kernels applicable to large graphs with rich attributes.

Table 2.2: Experiment results on graphs with only categorical attributes.

Method		MUTAG	ENZYMES	D&D	COX-2	BZR	DHFR	ER
DM	accuracy	**87.89±1.88**	59.48±0.89	**79.69±0.64**	73.97±1.80	75.80±1.10	**80.54±0.94**	**83.61±1.17**
	runtime	4"	27"	1h10'	29"	31"	27"	1'2"
PK	accuracy	84.22±1.47	46.43±1.26	**79.27±0.33**	75.33±2.34	**76.60±1.77**	**80.51±1.66**	81.91±0.78
	runtime	0.2"	2.9"	6'2"	1.5"	0.6"	4.2"	0.5"
WL	accuracy	**86.61±1.40**	53.22±1.30	79.01±0.43	**76.13±1.74**	**78.17±1.60**	**81.03±0.82**	82.52±0.86
	runtime	7"	28"	8'47"	17"	15"	22"	1'10"
SP	accuracy	85.94±1.94	43.20±1.21	78.26±0.76	73.97±2.33	72.83±1.87	75.18±0.97	76.93±1.22
	runtime	0.4"	3"	4h27'	1.4"	1"	2"	2"
GH	accuracy	82.89±1.69	37.98±1.57	75.80±0.46	71.90±2.15	72.93±1.46	74.00±1.40	78.36±1.02
	runtime	37"	12'11"	3d20h	4'24"	3'33"	6'50"	9'
WLSP	accuracy	85.72±1.96	**60.92±0.90**	-	72.47±1.35	**77.17±1.51**	78.95±1.29	**83.80±0.91**
	runtime	3"	1'26"	> 2 days	8"	7"	12"	14"
CSM	accuracy	85.61±1.95	58.68±1.03	-	**77.27±0.68**	71.43±1.91	78.87±0.82	78.80±0.92
	runtime	6'5"	8h24'	> 2 days	5'54"	22'45"	24'31"	4h9'

Table 2.3: Experiment results on graphs with numerical attributes.

Method		MUTAG	ENZYMES	COX-2	BZR	DHFR
DM	accuracy	**90.09±1.87**	**70.37±1.57**	**76.17±2.01**	**78.83±1.31**	**80.92±0.94**
	runtime	11"	44"	19"	52"	32"
PK	accuracy	83.56±1.15	55.38±1.21	74.80±2.55	72.00±2.41	79.67±1.23
	runtime	0.2"	3.3"	0.6"	0.9"	3.4"
SP	accuracy	87.11±1.73	**70.90±0.83**	72.03±1.17	74.60±2.35	77.28±1.14
	runtime	2'25"	19'20"	6'25"	4'15"	8'40"
GH	accuracy	85.78±2.50	62.33±1.07	71.27±2.87	73.10±1.76	74.08±1.21
	runtime	29"	9'	4'44"	3'42"	7'14"
WLSP	accuracy	**89.06±1.98**	**71.38±0.36**	**74.87±2.74**	**77.70±1.84**	78.54±1.07
	runtime	7'30"	32'6"	13'	18'55"	43'
CSM	accuracy	**90.61±2.39**	68.91±0.92	**75.03±1.63**	74.37±2.20	**79.72±1.66**
	runtime	6'25"	1'45"	6'41"	19'16"	35'35"

Method		ER	FR	FM	MR	MM
DM	accuracy	**83.77±1.17**	**66.83±1.26**	**61.94±1.34**	**60.79±1.59**	65.09±1.74
	runtime	1'10"	5"	13"	5"	6"
PK	accuracy	78.57±0.93	65.49±1.54	**61.03±2.61**	58.71±2.10	**66.76±1.36**
	runtime	3.3"	0.5"	1.1"	0.7"	0.3"
SP	accuracy	80.89±1.08	64.66±1.21	59.85±1.32	**60.44±1.38**	65.24±1.04
	runtime	13'	5'18"	5'5"	5'	4'40"
GH	accuracy	78.48±0.84	63.57±1.70	59.68±1.71	58.62±1.29	60.27±1.54
	runtime	9'39"	2'52"	2'40"	2'42"	2'33"
WLSP	accuracy	**83.73±0.97**	**66.09±2.19**	**62.62±2.00**	**59.88±1.53**	**67.03±1.41**
	runtime	49'50"	14'20"	13'46"	13'38"	12'47"
CSM	accuracy	80.16±0.79	**66.49±1.49**	60.71±1.77	58.24±2.37	**65.94±2.45**
	runtime	41'	1'53"	2'50"	13'57"	14'20"

Chapter 3

Distributed Representations of Expertise

3.1 Introduction

Collaborative platforms, such as crowdsourcing service providers, community question answering forums, and customer service centers, are becoming more and more prevalent. Once managed effectively, the rich online human resources have shown great potential to solve problems more economically, efficiently, and reliably [58, 30, 28]. In order to effectively manage and utilize expert resources, an essential problem is how to correctly understand/represent human expertise and identify right experts for a certain task [25, 62]. In this paper, we take collaborative

networks as an example to derive expertise representation so that multiple experts can be compared in the same framework.

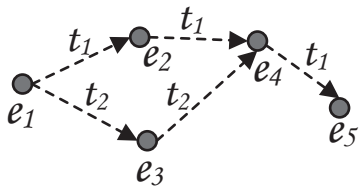


Figure 3.1: A sample collaborative network. Tasks are routed among experts in a collaborative network until they are resolved.

In collaborative networks, tasks are routed among a network of experts until they are resolved. Fig. 3.1 shows a sample collaborative network. Task t_1 starts at expert e_1 and is resolved by expert e_5; task t_2 starts at expert e_1 and is resolved by expert e_4. The sequences $e_1 \rightarrow e_2 \rightarrow e_4 \rightarrow e_5$ and $e_1 \rightarrow e_3 \rightarrow e_4$ are called *routing sequences* of task t_1 and t_2 respectively. One fundamental problem in ticket routing is how to represent experts' knowledge and employ it to estimate the probability of solving a task. Once this problem is solved, the final resolver to a given task can be found quickly.

An expert has to meet two constraints in order to solve a task: (1) Topic Match: the specialized areas of the expert shall match the topic of the task. For example, a programmer can possibly solve a programming problem while he is less likely to solve a physics problem; (2) Difficulty Level Match: the difficulty level of the

task should match the proficiency of the expert. A programmer might be capable of implementing a *Binary Search* algorithm while he is unable to solve the *Eight Queen Puzzle*. These two constraints can be formalized as: (1) Specialization, i.e., the field an expert is specialized in; (2) Proficiency level: to what degree an expert is specialized in that field [3].

Previous studies [3, 4] assumed that a list of possible specialized areas for each expert is given. In real collaborative networks, manually creating these specialized areas is laborious and hardly accurate. An intuitive solution is to use topic modeling such as Latent Dirichlet Allocation (LDA) [7] to automatically learn human expertise from the previously solved tasks. This solution has two main problems: (1) Tasks that an expert has failed to solve cannot be properly modeled to specify what the expert cannot do. (2) Topic models essentially capture the topic distribution of historical tasks. They do not directly measure proficiency level and its difference among experts.

To overcome the aforementioned issues and learn better expertise representations, we propose two expertise models in below:

(Model A) It assumes each expert has one or several specialized functional areas in a collaborative network. A task falling to one of the functional areas will be solved by the expert; otherwise it will be transferred to another expert. Based on this assumption, we define an expertise space in which all experts' expertise

and all tasks will be embedded as numerical vectors. Tasks close to one of the expertise of an expert will be resolved by the expert whereas those far from his/her expertise will not. In this model, we combine the two aspects, specialized area and proficiency level of human expertise. The specialized area of an expert is characterized as a ball centered at his expertise vector and the radius of the ball signifies the range of the expert's duty. The ball is named *functional area* of the expert. This model is referred to as *Functional Area Expertise* (FAE).

(Model B) In some collaborative networks, there is no clear division of experts' responsibility. Experts solve tasks just based on their true capability. In this case, experts could deal with tasks in all difficulty levels below his capacity of solving tasks. Therefore, instead of unifying specialized areas and proficiency levels as in the FAE, our second model learns a vector representation of expertise and characterizes the two aspects separately: dimensions of the expertise vector encode specialized areas and the value in each dimension signifies the proficiency level of the expert on the corresponding area. Our intuitions in this formulation are as follows: (1) If an expert can solve a task, his proficiency level should be greater than or equal to the task difficulty; (2) If an expert cannot solve a task, there must be some dimensions in his expertise where their values are smaller than those required by the task. In this way, the specialized areas together with their

proficiency levels can be modeled naturally. We refer to this model as *All-Round Expertise* (ARE).

FAE and ARE represent two different strategies of assigning task to experts. FAE is going to reserve the capacity of highly skilled experts for difficult tasks, while ARE tries to shorten task processing time as much as possible. We provide a comparison between FAE and ARE w.r.t several properties of collaborative networks, according to which one can decide the model that shall be used. Experimental results on real collaborative networks show that expertise learnt from our models better predict ticket solving than topic model based approaches and other methods in expertise modeling.

In comparison with previous studies on expertise modeling, to the best of our knowledge, we make the first attempt to consider all the factors together:

1. Not only do we utilize tasks solved by each expert but also those unsolved by him, which shall better characterize human expertise.

2. Not only do we characterize the specialized areas of an expert, but also the proficiency level he has in each area.

The rest of the paper is organized as follows. In Section 3.2, we briefly describe our problem setting. In Sections 3.3 and 3.4, the two proposed expertise models

FAE and ARE will be introduced. Section 3.5 presents our experimental results. Related work is reviewed in Section 3.6, followed by the conclusion in Section 3.8.

3.2 Preliminaries

In this section, data characteristics based on which we conduct our expertise models, will be firstly introduced. Then we will depict the big picture of distributed representations of expertise. At last, two expertise necessary properties which are based on our observations will be introduced.

In this section, we introduce the notations and discuss the two aspects of expertise, i.e., specialized areas and proficiency level, that shall be captured by an expertise representation.

3.2.1 Task Routing and Resolution Records.

In a collaborative network, a set of experts work cooperatively to solve tasks. Here we use $\mathcal{E} = \{E_i\}$ to represent the set of experts. $\mathcal{V} = \{v_1, v_2, ..., v_k, ..., v_V\}$ is the set of words used to describe the tasks. Let $\mathcal{T} = \{t_j\}$ be a set of tasks resolved in the collaborative network, where each t_j is a bag-of-word vector with each dimension recording the word frequency in the task description.

Apart from the textual description, each task is also associated with a routing sequence starting from an initial expert to the final resolver of the task. Table 3.1 shows one example problem ticket in an IT service department. The ticket with ID 599 is a problem related to *operating system*, specifically, *the low percentage of the available file system space*. It was assigned to expert IN039, then routed through expert SAV59, and got resolved by expert SAV4F.

Table 3.1: The Lifetime of A Task.

ID	Entry	Time	Expert
599	New ticket: the available space on the var file system is low	9/14/06 5:57:16	IN039
599	...(operations by IN039)...	...	IN039
599	Ticket 599 transferred to SAV59	...	IN039
599	...(operations by SAV59)...	...	SAV59
599	Ticket 599 transferred to SAV4F	...	SAV59
599	...(operations by SAV4F)...	...	SAV4F
599	Problem resolved: free up disk space in the file system	9/14/06 9:57:31	SAV4F

In this paper, we propose to learn distributed representations of expertise for all experts based on their task solving history. For the example above, historical data which we can utilize is experts IN039 and SAV59 did not solve task 599 but SAV4F solved it.

During task routing, an essential problem is to understand a certain expert's knowledge and estimate whether he could solve the current task or not. In this paper, we propose to automatically learn the expertise of all the experts based on the task routing and resolution records. Specifically, as shown in Table 3.1, the fact that expert IN039 and SAV59 did not solve task 599 but SAV4F solved it, will be leveraged to infer their expertise.

HuanWhen a task is solved by an expert, it satisfies that (1) the topic of the task matches the specialized topic(s) of the expert. For example, an expert specialized in computer science tasks probably could not handle problems in Chemistry. (2) Furthermore, the difficulty level of the task matches the expert's ability in dealing with tasks from the same topic.

3.2.2 Expertise Representation.

How to model the capability or knowledge of an expert is critical in collaborative platforms [41**?**]. One intuitive idea is using collaborative tags to describe knowledge areas which the expert is good at [29].

[3] gives a formal definition of expertise, which consists of two aspects: (1) Specialized areas of an expert; (2) Proficiency level of an expert in each specialized area. We formalize a distributed representation of expertise as follows:

$$< level(area_1), level(area_2), ..., level(area_d) >,$$

where $level(area_i)$ is the proficiency level of the expert in $area_i$ and d is the number of all possible specialized areas. Previous studies assume that all these specialized areas are pre-specified [3, 4]. In real collaborative networks, manually creating these specialized areas is often laborious and hardly accurate. Specialization could be strongly correlated with each other, as well as hierarchical, i.e., under one general topic, there could be fine-grained subtopics. Therefore automatic learning expert knowledge, based on the tasks an expert has solved and failed, will be more beneficial to efficient task resolution in collaborative networks.

Previous studies try to use topic modeling to automatically learn expertise areas [41]. The textual descriptions of all the tasks solved by an expert can be combined together as a document to represent his knowledge. LDA can take such documents for all the experts and output the topic distribution for each expert as his expertise. Unfortunately, two main disadvantages lie in this intuitive solution: (1) An expert's unsolved tasks tell what he cannot solve, and therefore should be used together with those solved tasks to characterize his expertise. Topic modeling models like LDA cannot directly utilize this kind of information. Developing more advanced and complicated topic models is non-trivial; (2) The learnt topic distribution does not signify the proficiency level of an expert. According to topic modeling, higher values in the topic distribution for an expert merely mean more

tasks have been solved in the corresponding topics. They do not necessarily denote higher proficiency level of the expert.

For example, if an expert solved task t multiple times, words in task t's description will be counted multiple times, and as a result the topics which this task t belongs to will be emphasized. However, in this case, the expert's proficiency levels in dealing with tasks from such topics is not necessarily high, since t might be an easy task. Otherwise, if the expert can solve "more difficult" tasks from a specific topic but only solved a small number of tasks belonging to topic, the corresponding topic probability in his topic distribution will not be emphasized, even with a small value.

Here we introduce a novel distributed representation of expertise by embedding expertise and task in the same d dimensional space, referred to as the expertise space. We use e_i^k to represent the kth expertise of expert E_i. Each task t_j will be embedded in the expertise space by a transformation matrix W:

$$\tilde{t_j} = Wt_j, \tag{3.1}$$

where $\tilde{t_j}$ is the representation of the ith task t_j in the expertise space. Based on all the solved and unsolved tasks of each expert, we will learn their expertise representation e_i together with the transformation matrix W.

3.3 Modeling Functional Area Expertise

In this section, we introduce our first expertise model called the Functional Area Expertise (FAE) model. In this model we assume each expert in the collaborative network may have one or more specialized functional areas and they could only solve tasks which belong to one of his functional areas. This assumption holds for many real collaborative networks in which explicit functional areas represent responsibilities.

Based on this assumption, we define a d dimensional Functional Area Expertise (FAE) space. Each expert will be assigned to one or several $d \times 1$ vectors, each represents the center of one functional area in expertise space and a corresponding radius parameter r which models the proficiency level in this area. In this new space, as shown in Fig. 3.2, tasks located within one of the functional areas of an expert E will be resolved by the expert and tasks located outside all the functional areas of the expert will not be resolved by E. An expert with larger radius parameter is likely to solve more tasks.

The functional area expertise model can be learned based on historical data. Intuitively, the learnt expertise should be close to his solved tasks but far from those tasks he cannot solve. We design the following objective function based on

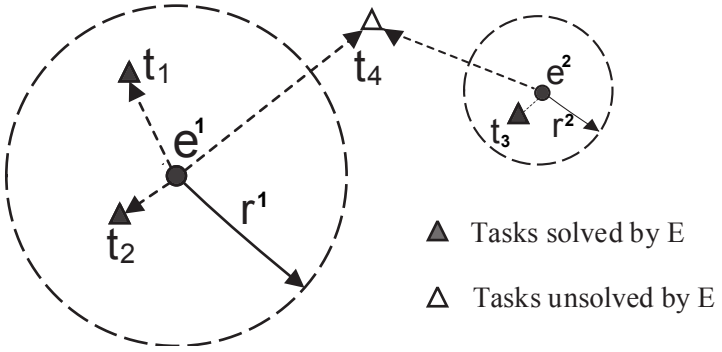


Figure 3.2: An intuitive example for Functional Area Expertise when $d = 2$. Tasks t_1 and t_2 are in the first functional area, t_3 is in the second functional area of E and can be solved by E. Tasks t_4 are out of both functional areas of expert E and thus cannot be solved by E.

this intuition.

$$\begin{aligned}\underset{W,e,r_e}{\operatorname{argmin}} & \sum_{E \text{ solved } t} \min_k f_{r_e^k}(||e^k - Wt||_2) \\ & - \sum_{E \text{ unsolved } t} \sum_k f_{r_e^k}(||e^k - Wt||_2) + \alpha||W||_1 \\ & + \beta \sum_{e, 1 \le k_1, k_2 \le k} \max(r_e^{k_1} + r_e^{k_2} - ||e^{k_1} - e^{k_2}||_2, 0),\end{aligned} \tag{3.2}$$

where r_e^k is the kth radius parameter for expert E, f is a monotonic increasing function referred later as the radius function, which will be described in detail later. The first term in the objective minimizes the distance between a solved task and solver's closest expertise center while the second term maximizes the distance between an unsolved task to expert's all expertise centers. The third term is a L_1 regularization term used to reduce model complexity and avoid overfitting. Intuitively different functional areas represent different expertise areas and thus

should not overlap with each other. The last term in the objective gives penalty to functional area overlapping: it returns 0 when the distance between any two expertise centers is greater than the sum of two corresponding radius parameters and returns a positive penalty otherwise.

Simply setting function f as the identical function leads to a trivial solution. For example if expert E fails to solve task t, the objective function returns negative infinity when we embed all expert centers of E to origin and ticket t to infinity. To avoid this, we set f to be the shifted sigmoid function defined as follows:

$$f_r(x) = \frac{1}{1 + \exp -(x - r)},$$

where r is the radius parameter.

We applied well-established L-BFGS (Limited-memory BFGS) algorithm [34] to optimize objective function Eq. (3.2). L-BFGS is an optimization algorithm in the family of quasi-Newton methods. Instead of storing a dense approximation of the inverse Hessian matrix, L-BFGS algorithm stores only a few vectors that represent the approximation implicitly. This makes it particularly well suited for optimization problems with a large number of variables. Based on learnt W, r_e, and e, the probability of an expert E solving a task t can be predicted by:

$$1 - \min_k f_{r_e^k}(||e^k - Wt||_2).$$

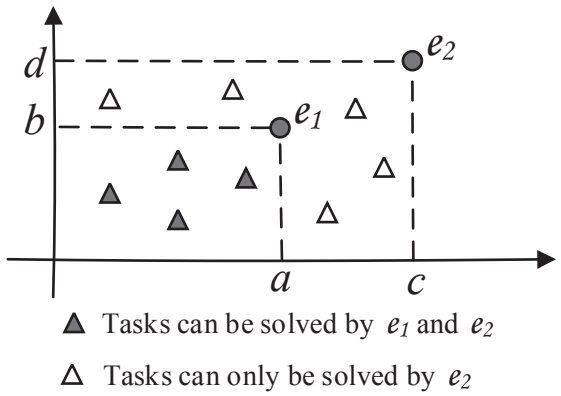


Figure 3.3: An example for the All-Round Expertise model. Expert E_1 can only solve the tasks covered by the small rectangular. But E_2 can solve all the tasks including those E_1 can solve.

3.4 Modeling All-Round Expertise

In some collaborative networks, the assumption above for the functional area formulation may not hold. Instead, an expert can solve tasks whose difficulty level is equal to or below his/her proficiency level. In this section, we propose an All-Round Expertise (ARE) model to handle this scenario. ARE is based on the following intuition: (1) If an expert E can solve a task t, its expertise should be greater than or equal to t in all dimensions. (2) If the expert cannot solve the task, its expertise should be smaller than the task at least in some dimensions. An example with $d = 2$ is shown in Fig. 3.3. e_1, (a, b), and e_2, (c, d), are expertise

of two experts. Tasks lying inside the smaller rectangular can be solved by expert E_1. All tasks lying inside the bigger rectangular can be solved by expert E_2. In this model, dimensions are considered as areas and the value on each dimension is the proficiency level of an expert in that area. Therefore, we no longer need to assign several expertise vectors to one expert.

3.4.1 ARE for Solved Tasks.

Let $e = (e_{(1)}, e_{(2)}, \ldots, e_{(d)})$ be the expertise of expert E and $\tilde{t} = (\tilde{t}_{(1)}, \tilde{t}_{(2)}, \ldots, \tilde{t}_{(d)})$ be the vector representation of task t in the expertise space. We define a new operation $\dot{-}$ in the expertise space as follow:

$$e \dot{-} \tilde{t} = \sum_{1 \leq l \leq d} (\min(0, e_{(l)} - \tilde{t}_{(l)})).$$

Note that $e \dot{-} \tilde{t}$ is always smaller than or equal to 0 and $e \dot{-} \tilde{t} = 0$ if and only if $e_{(l)} \geq \tilde{t}_{(l)}$, $\forall k$. We define the objective function of ARE for solved tasks is as follows:

$$\underset{W,e}{\operatorname{argmax}} \sum_{e \,\text{solved}\, t} (e \dot{-} \tilde{t}). \tag{3.3}$$

We denote dimension k as a **strong dimension** for expert E with respect to task t if $e_{(l)} \geq \tilde{t}_{(l)}$, and as a **weak dimension** with respect to t if $e_{(l)} < \tilde{t}_{(l)}$. When E solved t, the objective function in Eq. 3.3 penalizes all weak dimensions and do not care how 'strong' the strong dimensions are. If all the dimensions are strong,

we have $e \dot{-} \tilde{t} = 0$, which maximizes $e \dot{-} \tilde{t}$. The motivation is, if the expert can solve the task, this expert shall have knowledge deep enough on all the areas required by t.

Only learning experts' expertise from the solved tasks will not work well. For example in order to tell one task t_j is more difficult than another task t_i, we need information like some expert E solved t_i but not t_j. This can also be illustrated by a trivial solution of the above model which embeds all the tasks to 0 and all expertise to some positive number. This solution clearly maximizes the above objective but fails to learn the difficult level for each task. Base on this intuition, we shall include unsolved tasks in learning expertise.

3.4.2 ARE for Unsolved Tasks.

For unsolved tasks, we believe there must be some weak dimensions for expert E. For the optimization purpose, we set a margin parameter $\alpha \geq 0$ to measure how weak these dimensions are. To be specific, if expert E fails to solve task t, the sum of differences between expertise e and t over all weak dimensions should be smaller than a negative threshold, which is set to be $-\alpha$. Fig. 3.4 shows an example of this margin with $d = 2$. If an expert E could solve task t, the learnt expertise e should be located in $area\,1$, while if e fails to solve t, the learnt expertise representation should be located in $area\,2$. We use following formula to

measure the penalty for unsolved tasks:

$$\min(-\alpha - (e \dot{-} \tilde{t}), 0),$$

where $e \dot{-} \tilde{t}$ can be considered as the sum of differences between e and t over all weak dimensions. If the sum is larger than $-\alpha$, the formula equals to $-\alpha - (e \dot{-} \tilde{t})$, which is negative. Otherwise, the formula returns 0.

Margin α distinguishes between solved tickets and unsolved tickets of an expert in the expertise space. Setting margin α to 0 leads to a trivial solution: the objective function returns a global minimum value 0 for both solved and unsolved tasks when mapping all expertise and ticket vectors to the origin. We will describe in detail how to select α later in the experiment section.

We define the objective function of ARE for unsolved tasks as follows:

$$\underset{W,e}{\operatorname{argmax}} \sum_{e \text{ unsolved } t} (\min(-\alpha - (e \dot{-} \tilde{t}), 0)). \tag{3.4}$$

When E fails to solve t, this objective function penalizes all dimensions when $e \dot{-} \tilde{t}$ is above the margin.

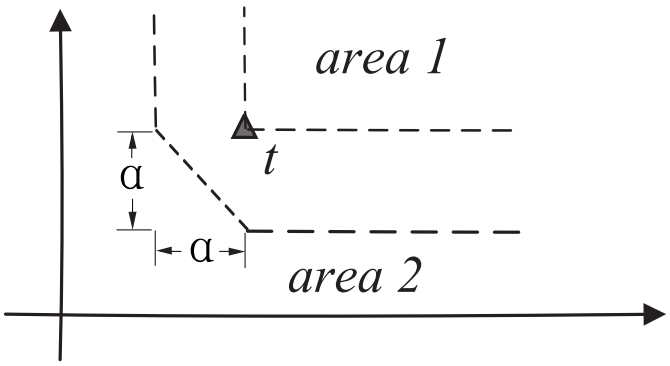


Figure 3.4: An example of applying margin α in the All-Round Expertise model with $d = 2$. The expertise of all experts who are capable of solving t is expected to be located in *area* 1, while the expertise of all experts who cannot solve t is expected to be located in *area* 2.

3.4.3 Objective Function of ARE.

We now combine penalties from both solved and unsolved tasks to the objective function:

$$\begin{aligned} \underset{W,e}{\operatorname{argmin}} \quad & - \sum_{E \text{ solved } t} (e \dot{-} \tilde{t}) \\ & - \sum_{E \text{ unsolved } t} (\min(-\alpha - (e \dot{-} \tilde{t}), 0)) \\ & + \beta ||W||_1. \end{aligned} \tag{3.5}$$

Similar to the FAE model, we add an additional L_1 regularization term and apply L-BFGS algorithm to optimize ARE. Denote the objective function in Eq. 3.5 as $\mathcal{H}(W, e)$. The derivative of $\mathcal{H}$ with respect to $\{W, e\}$ is given by:

$$
\begin{aligned}
\frac{\partial \mathcal{H}}{\partial W_{ij}} &= \sum_{E \text{ solved } t} t_{(j)} * I(e_{(i)} < \tilde{t}_{(i)}) \\
&\quad - \sum_{E \text{ unsolved } t} t_{(j)} * I(E \doteq \tilde{t} > -\alpha) \\
&\quad + \beta * sgn(w_{ij}), \\
\frac{\partial \mathcal{H}}{\partial e_{(i)}} &= \sum_{E \text{ solved } t} -1 * I(e_{(i)} < \tilde{t}_{(i)}) \\
&\quad + \sum_{E \text{ unsolved } t} 1 * I(e \doteq \tilde{t} > -\alpha),
\end{aligned}
\tag{3.6}
$$

where $sgn(x)$ is the sign function and $I(x)$ is the indicator function which returns 1 if statement x is true and returns 0 otherwise. Based on W and e learnt from ARE, whether an expert can or cannot solve a task can be predicted by $e - \tilde{t}$. The decision boundary is determined using 5-fold cross validation on training. The scale of e's each dimension indicates how good the expert is in the corresponding area. Similarly, the scale of t's each dimension indicates how difficult the task is in that dimension.

3.5 Experiments

In this section, we evaluate the proposed two expertise models, FAE and ARE, on real-life datasets. We test their performance in predicting whether an expert E can solve a task t.

3.5.1 Baselines.

The performance of FAE and ARE is compared with standard classifiers and other popular methods in expertise modeling.

(1) **Logistic Regression (LR).** We train a logistic regression model [5] for each expert. This model takes a bag-of-word vector as input and outputs the probability that this expert solves the task. This probability, denoted as $P_i(t)$, is defined as follows:

$$P_i(t) = \frac{1}{1 + exp(-(W_i * t + b_i))},$$

where W_i is the weight vector associated with expert E_i which has the same size as the bag-of-word vector of tasks. W_i can be viewed as an expertise vector learnt for each expert; $W_i * t$ computes the dot product similarity between E_i and t, which is used to predict task solving. The parameters in the model are learned by minimizing the square-loss error, based on the $< expert, task >$ pairs in the training dataset.

$$\underset{W,b}{\text{argmin}} = \sum_{<e_i,t>} (P_i(t) - y_{<E_i,t>})^2,$$

where y is 1 if E_i solved t and is 0 otherwise.

(2) **SVM.** We build an SVM classifier for each expert and use it to predict whether he/she can solve the tasks in the testing set. We tried different kernels in-

cluding linear, polynomial, quadratic and multilayer perceptron; their best results are reported.

(3) **Query Likelihood Language Model (QLL).** In QLL [41], each document is represented as a multinomial distribution over words. The maximum likelihood estimate of this distribution is the frequency of each word in the document divided by the total number of words in the document. We apply Dirichlet smoothing to the distribution as most of the words in the vocabulary do not show together in each individual document. The likelihood of a query task t generated from a document d under a language model with Dirichlet smoothing is defined as:

$$p(t|d) = \prod_{w \in t} \frac{N_d}{N_d + \mu} p(w|d) + \frac{\mu}{N_d + \mu} p(w),$$

$$p(w|d) = \frac{N_d(w)}{N_d},$$

where N_d is the number of words in d, $N_d(w)$ is the number of word w in d, μ is a smoothing parameter, and $p(w)$ is the probability of the word w in the entire corpus. For each expert E we construct two document collections, C_e^+, which is a collection of all his/her solved tasks, and C_e^-, which is a collection of all his/her unsolved tasks. Intuitively, for an expert E, if a new task is close to one task in the solved task collection, then this new task is highly likely to be solved by E. Similar argument works for the unsolved task collection. Thus we define the

likelihood of a query task t obtained from a document collection C as:

$$p(t|C) = \max_{d \in C} p(t|d).$$

For expert E and a new task t, we predict E solve t if $p(t|C_e^+) > p(t|C_e^-)$ and E cannot solve t otherwise.

(4) **Topic Modeling.** Topic modeling can be used to learn expertise. Specifically, we first create two documents for each expert by merging task descriptions in C_e^+ and C_e^-, and denote them as d_e^+ and d_e^- respectively. Latent Dirichlet Allocation (LDA) [7] is then used to train topic models. For each expert, two topic distributions θ_e^+ and θ_e^- are learned corresponding to d_e^+ and d_e^-. θ_e^+ conveys what expert E can do and θ_e^- tells what he/she cannot do. The likelihood of a task t generated from a topic distribution is defined as:

$$p(t|\theta_e) = \prod_{w \in t} \sum_{z \in Z} \theta_e^z \times p(w|z),$$

where z is one of Z topics and $p(w|z)$ is the word probability under topic z which is output by the topic model. For expert E and a new task t, we predict E solve t if $p(t|\theta_e^+) > p(t|\theta_e^-)$ and e cannot solve t, otherwise. Advanced topic models, such as [45], are not chosen as baselines because they do not fit our problem setting. For example, there is no multiple authors for a task as a scientific paper does. Since each task description in our dataset typically contain a few words, directly

applying complex topic models will not work well due to sparse word co-occurrence [59].

3.5.2 Datasets.

We use real-world problem ticket data collected from a problem ticketing system in an IT service department throughout 2006. Two datasets in different problem categories are explored: Windows and AIX, which contain problem tickets occurring in the Windows and AIX operating systems. The details of both datasets are shown in Table 3.2. The data is quite sparse: Only a few tasks were recorded for each expert and there are a few words in each task description.

Table 3.2: Two Datasets on Task Resolution

Datasets	# of tasks	# of experts
WIN	32349	1278
AIX	10519	1988

We apply well-established L-BFGS algorithm to optimize the objective function in FAE, ARE, and LR. All parameters are initialized randomly from $[-1, 1]$ and updated iteratively using the second order gradients estimated by L-BGFS. We conduct a $80\% - 20\%$ random split on the dataset and generate training and testing data. 5-fold cross validation is used on the training dataset for hyper parameters selection on all the methods including FAE and ARE, e.g., α, β in Eq.

3.2 and β in Eq. 3.5. The margin parameter α in ARE should not effect the expertise learning: if W and e are local minimal for Eq. 3.5 with α then $2W$ and $2e$ are local minimal for Eq. 3.5 with 2α. We use the 5-fold cross validation to determine the decision boundary for both FAE and ARE. The whole process is repeated for 5 times; the mean and standard deviation of the classification accuracy over the 5 runs are reported. The bag-of-word representation uses the top $2,000$ most frequent words in all the tickets as used in [53]. The dimensionality of expertise is set at 20 for both FAE and ARE models. In the FAE model, each expert is assumed to have 2 specialized areas (k is set at 2 empirically).

3.5.3 Accuracy.

Table 3.3 summarizes the performance of all the methods. As shown, the proposed two models work significantly better than all the baselines in terms of prediction accuracy. As the ticket dataset is sparse, LR, SVM and QLL cannot learn very good classifiers as they learn a classifier for each individual expert. Our models learn expertise for all the experts together and embed all the tasks in the same expertise space. The learnt transformation matrix W is shared by all the tasks and the sparsity issue is overcome in this way. The performance of our models is better than these three baselines. Not surprisingly, the topic model method, LDA, works badly in this case. The reason is topic models highly rely on

word co-occur information. Tasks in our data are very short (the average number of words in a task is around 5) and only a few short documents are associated with each expert. Therefore, topic modeling cannot learn proper topic distributions. ARE outperforms FAE significantly in the AIX dataset indicating that the WIN and AIX collaborative networks might adopt different task assignment strategies.

From the results we can see that whether a task can be solved cannot be accurately predicted using word level model like QLL. In fact, sice each task contains only a few words (average number of words in a task is round 5) and we have only limit number of tasks solving history for most experts E, many new tasks have zero word overlapping with the collections of both solved and unsolved tasks. This limits the prediction capacity of any word level models. The basic LR model lack the capacity of learning competency of an expert in multi-topics. In addition, as we only have limit training examples for each people, the classic LR and SVM model will likely suffer from overfitting. Thus it is not surprising that our proposed models outperform LR and SVM in all datasets.

3.5.4 Efficiency.

The time complexity of the FAE and ARE models is determined by the number of gradient update iterations multiplied by the complexity of computing the objective derivatives in each iteration. As shown in Eq. 3.6, the complexity of

Table 3.3: Accuracy on task resolving prediction

Models	WIN(%)	AIX(%)
FAE	85.09 ± 0.49	85.38 ± 1.06
ARE	86.52 ± 0.62	88.18 ± 0.63
LR	81.44 ± 0.65	79.48 ± 0.78
SVM	80.14 ± 0.40	79.17 ± 0.59
QLL	78.44 ± 0.47	67.55 ± 3.92
LDA	65.60 ± 0.67	74.20 ± 0.49

the derivatives computation is $O((d_t + N) * d)$, where d_t is the dimensionality of ticket bag-of-word representation, d is the dimensionality of the expertise space, and N is the number of ticket-expert pairs. We did not show the derivative of the objective function in FAE due to the space limit. However, it is easy to see that the complexity is the same as ARE. Note that both models may take a long time to converge. Here we set a limit on iterations and stop the model training after 50 iterations. Figure 3.5 shows how we empirically select this number by plotting the classification accuracy versus the number of training iterations for both models on the AIX dataset.

Table 3.4 summarizes the running time for each model. We list the training time for all the methods. These algorithms are implemented using MATLAB on a 8-core 3.40GHz Intel CPU with 16G memory. The testing time for all the methods is shorter than 1 second, which means after expertise learning all the methods can do prediction in a timely manner.

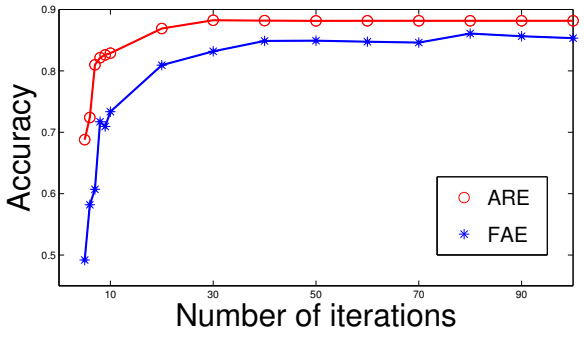


Figure 3.5: Classification accuracy versus the number of iterations for FAE and ARE on the AIX dataset. Both models almost converge after 50 iterations.

Table 3.4: Efficiency on model learning

Models	WIN	AIX
FAE	$5'21''$	$3'38''$
ARE	$5'20''$	$3'02''$
LR	$27''$	$28''$
SVM	$3'23''$	$7'9''$
QLL	$1'3''$	$2'31''$
LDA	$11'59''$	$18'23''$

3.6 Related Work

In this section, we first introduce the background of collaborative platforms and then review previous methods on expertise modeling and representation.

Collaborative Platforms. Recent years have witnessed blossoms of various collaborative platforms, such as community question answering forums Quora[1] and Stack Overflow[2]. Collaborative network is a typical example of collaborative platforms, where a task is routed through the network until being resolved. Studies have been focused on developing automated routing algorithms to route a task to its final resolver as fast as possible [48, 38, 61]. Miao et al. [39] studied a network model and a routing model to jointly simulate the structure and the task routing procedure in a collaborative network. Sun et al. [53] studied the routing behaviors of experts in collaborative networks and modeled their decision making process on where to transfer a task. In this paper, we study a key problem in collaborative networks: expertise modeling and representation, which serves the central role in many applications, such as expert search and expertise comparison.

Expert Search. Expert search aims at finding experts who are knowledgable on a given topic and capable of solving a given problem [15, 2, 47, 19]. Expert search became an important research area since it started at the TREC enterprise

[1]http://www.quora.com/
[2]http://stackoverflow.com/

track [15] in 2005. The crucial problem in expert search is how to rank experts given a query. Many papers in this line consider expertise modeling as a part of expert ranking. No explicit expertise representations are learned [35, 18]. For example, Deng et al. formalized expert ranking by statistical language model and topic-based model [18]. Fang et al. [19] proposed a discriminative learning framework to directly model the conditional probability of relevance between an expert and a query. In this paper, we try to learn a vector representation for each expert, which can be used to compare expertise in a large organization and optimize expert allocation.

Expertise Modeling. In different application scenarios, various types of data are used to model expertise, such as project descriptions and professional articles [2, 41, 47, 19]. For example, Mimno et al. [41] modeled a reviewer's expertise by the papers she has written when matching reviewers with submitted papers. Guan et al. [24] mined the fine-grained knowledge of a Web user, by analyzing their Web surfing data, to facilitate expert search in collaborative environments. In question answering, Zhang et al. [62] used language models to compute user expertise based on the threads a user contributes to, where each thread contains a question post and a number of reply posts. Li et al. [33] incorporated question category into question routing, where question category is used to estimate answerers' expertise. Chang et al. [12] proposed a routing framework that uses compatibility,

availability and expertise of the users to recommend answerers and commenters to a question. Expertise is modeled by using questions an expert has answered and questions' tags annotated by askers.

As social media becomes prevalent, studies (e.g., [60, 25, 55]) on using social media data to derive user expertise, have emerged recently. Yeniterzi et al. [60] employed authority signals such as votes, comments, and follower/following information to estimate user expertise. In [25], Guy et al. provided an extensive study that explores the use of social media to infer user expertise, by evaluating the data users produced through a large survey. Varshney et al. [55] inferred the expertise of employees in the IBM corporation through mining social data that is not specifically generated and collected for expertise inference. Our work on modeling expertise distinguishes itself from these studies: We not only take into account tasks that are solved by an expert, but also those unsolved by her, in order to characterize her expertise.

Expertise Representation. For expertise representation, early approaches built a knowledge base which contained the descriptions of people's skills within an organization [16] or used tags to represent expertise of each expert [29]. More advanced methods are based on topic modeling [41, 45]. Topic Modeling is a standard approach to explain the observed data. Latent Dirichlet Allocation (LDA) [7], which takes a set of documents as input and simultaneously learn

the document-topic and topic-word distributions. For each expert, one can combine the set of tasks solved as a single document. LDA takes the documents of all the experts and outputs the topic distribution for each expert as his/her expertise representation. However, topic distributions learnt by LDA essentially capture experts' historical task distributions but not their true capability on task solving, such as the proficiency level on a specialized area. In contrast, expertise learnt by our models conveys both aspects of human expertise: specialization and proficiency level.

3.7 Communication Frequency and Expertise Closeness

Based on observations in cognitive science studies [52] and previous work [53], we characterize two properties that a good expertise representation should have in collaborative networks.

(1) **Communication Frequency**. Expertise recognition, which is critical for the work efficiency of collaborative platforms, was studied broadly in Transactive Memory Theory [57]. Previous studies found that social communications help experts to understand each others' expertise much better than just reading digital descriptions of experts [52]. Based on analyzing tasks in an IT department, we

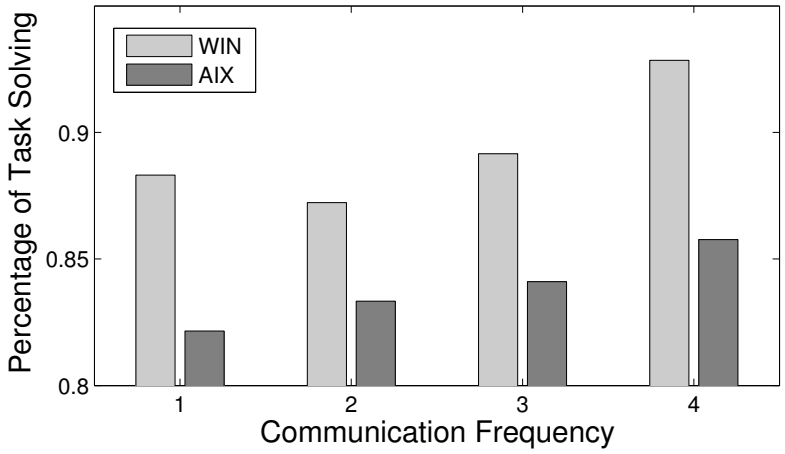


Figure 3.6: An observation in a task resolution dataset. The percentage of tasks solved increases with communication frequency. WIN and AIX are the categories of tasks on MS Windows and AIX operating systems.

also found that frequent communications will help experts understand expertise of each other. As shown in Fig 3.6, with the communication frequency increasing, the percentage of tasks being solved increases. One possible reason is, after a few times of communication, an expert will become much clearer about which tasks his neighbors can or cannot solve.

(2) **Expertise Closeness**. One observation reported by [53] is that an expert tends to transfer tasks to other experts whose expertise is quite different from his own. The reason is obvious as he (with his expertise) has failed to solve the task.

A good expertise representation should meet these constraints. This leads to a question, can we add them to learn a more accurate expertise representation?

In this section we will first describe our tries and then explain why the FAE and ARE models likely have incorporated these constraints to some degree.

As we mentioned earlier, people who communicate frequently will have better understanding of each other's expertise. Therefore if e_i has a frequent communication history with e_j then the task routed from e_i to e_j will be likely closer to e_j's expertise, thus will be more likely solved by e_j. In order to utilize this property, we vary the key parameters in our two models, the radius parameter r in FAE and the margin parameter α in ARE, according to the communication frequency between two experts. For example, for task t transferred from e_i to e_j, the radius parameter r exponentially decreases with respect to the communication frequency f_{ij} between e_i and e_j. To be specific, we keep the same objective formula in Eq. 3.2 but change the boundary function as follows:

$$f(x) = \frac{1}{1 + e^{-(x - r * exp(-\gamma f_{ij}))}}, \tag{3.7}$$

where γ is the exponentially decreasing rate. We can also apply a similar exponential decreasing factor on the margin parameter α in ARE.

For expertise closeness, we penalize communicated expert pairs by adding the following regularization to the objective function of both proposed models:

$$\sum (1 - f(||e_i - e_j||_2)), \tag{3.8}$$

where the summation is over all communicated expert pairs e_i and e_j and function f is the bounded non-linear function introduced in Eq. 3.7. It is easy to see that this penalty term returns 1 if e_i is closed to e_j and returns 0 if e_i is far from e_j.

Intuitively the above modification shall improve the original models, FAE and ARE. However, such improvement is not observed. We suspect that these two properties have already been captured in the tasks transferred between experts. Take the expertise closeness property and FAE as an example. If expert e_i routed task t to e_j and e_j solved it, then the expertise of e_j will be close to $\tilde{t}$. Since e_i failed to solve t, the expertise of e_i will be away from $\tilde{t}$. Therefore the expertise of e_i and e_j will not be close and task t carries this information. We will later show in the experiments that the representation learnt from FAE and ARE indeed have these two properties.

3.8 Conclusion

Expertise modeling is considered as the core content in improving the efficiency of collaborative networks. The goal is to represent experts' abilities in terms of specialization and proficiency level. In this paper, we developed two models to learn distributed representations of expertise which can convey both aspects. The shared insight of these two models is embedding both expertise and tasks that were solved/unsolved by experts in the same space. The embeddings are optimized over

all historical data points: which expert solved which task and which expert failed to solve which task. The first model assumes that an expert only solves tasks matching his/her specialization and proficiency level. Alternatively, in the second model, the expert can solve all the tasks whose difficulty level is equal to or lower than his/her proficiency level in his/her specialized area. Experimental results on real datasets demonstrated that the proposed models can learn meaningful expertise representations and are effective in predicting task resolution.

Chapter 4

Summary

The performance of machine learning methods is heavily dependent on feature representation on which they are applied. As a result, representation learning, which transfers real world data such as graphs, images and texts, into representations that can be effectively processed by machine learning algorithms, has became a new focus in machine learning community. In this thesis, we apply representations learning algorithm on two different tasks. In the first work, we used well-known VG kernel on a newly developed nodes representation. In the second work, we learn distributed representations of experts' abilities in terms of specialization and proficiency level in a collaborative networks. The representations are optimized over all historical data points: which expert solved which task and which expert failed to solve which task. Experimental results on real

datasets demonstrated that the powerfulness of the proposed representation in both applications.

Bibliography

[1] A. Airola, S. Pyysalo, J. Björne, T. Pahikkala, F. Ginter, and T. Salakoski. All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning. *BMC bioinformatics*, 9(Suppl 11):S2, 2008.

[2] Krisztian Balog, Leif Azzopardi, and Maarten De Rijke. Formal models for expert finding in enterprise corpora. In *SIGIR*, pages 43–50, 2006.

[3] Krisztian Balog and Maarten De Rijke. Determining expert profiles (with an application to expert finding). In *Proc. of IJCAI-07*, pages 2657–2662, 2007.

[4] Krisztian Balog, Yi Fang, Maarten de Rijke, Pavel Serdyukov, and Luo Si. Expertise retrieval. *Foundations and Trends in Information Retrieval*, 6(2–3):127–256, 2012.

[5] C Bishop. Pattern recognition and machine learning, 2007.

[6] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - the story so far, 2009.

[7] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[8] D. D. Bonchev and D. H. Rouvray. *Chemical graph theory: introduction and fundamentals*, volume 1. 1991.

[9] K. Borgwardt and H. Kriegel. Shortest-path kernels on graphs. In *ICDM*, 2005.

[10] K. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl 1):i47–i56, 2005.

[11] C. Chang and C. Lin. Libsvm: a library for support vector machines. *TIST*, 2(3):27, 2011.

[12] Shuo Chang and Aditya Pal. Routing questions for collaborative answering in community question answering. In *ASONAM*, pages 494–501, 2013.

[13] J. H. Chen, E. Linstead, S. J. Swamidass, D. Wang, and P. Baldi. ChemDB update – full-text search and virtual chemical space. *Bioinformatics*, 23(17):2348–2351, 2007.

[14] F. Costa and K. D. Grave. Fast neighborhood subgraph pairwise distance kernel. In *ICML*, 2010.

[15] Nick Craswell, Arjen P de Vries, and Ian Soboroff. Overview of the trec 2005 enterprise track. In *TREC*, volume 5, pages 199–205, 2005.

[16] Thomas H Davenport and Laurence Prusak. *Working knowledge: How organizations manage what they know*. Harvard Business Press, 1998.

[17] 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. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991.

[18] Hongbo Deng, Irwin King, and Michael R Lyu. Formal models for expert finding on dblp bibliography data. In *ICDM*, pages 163–172, 2008.

[19] Yi Fang, Luo Si, and Aditya P Mathur. Discriminative models of integrating document evidence and document-candidate associations for expert search. In *SIGIR*, pages 683–690, 2010.

[20] A. Feragen, N. Kasenburg, J. Petersen, M. de Bruijne, and K. Borgwardt. Scalable kernels for graphs with continuous attributes. In *NIPS*, 2013.

[21] H. Fröhlich, J. K. Wegner, F. Sieker, and A. Zell. Optimal assignment kernels for attributed molecular graphs. In *ICML*, 2005.

[22] T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, pages

129–143. 2003.

[23] K. Grauman and T. Darrell. Approximate correspondences in high dimensions. In *NIPS*, 2006.

[24] Ziyu Guan, Shengqi Yang, Huan Sun, Mudhakar Srivatsa, and Xifeng Yan. Fine-grained knowledge sharing in collaborative environments. *TKDE*, 2015.

[25] Ido Guy, Uri Avraham, David Carmel, Sigalit Ur, Michal Jacovi, and Inbal Ronen. Mining expertise and interests from social media. In *WWW*, pages 515–526, 2013.

[26] C. Helma, R. D. King, S. Kramer, and A. Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108, 2001.

[27] S. Hido and H. Kashima. A linear-time graph kernel. In *ICDM*, 2009.

[28] Srikanth Jagabathula, Lakshminarayanan Subramanian, and Ashwin Venkataraman. Reputation-based worker filtering in crowdsourcing. In *NIPS*, pages 2492–2500, 2014.

[29] Ajita John and Dorée Seligmann. Collaborative tagging and expertise in the enterprise. In *WWW*, 2006.

[30] David R Karger, Sewoong Oh, and Devavrat Shah. Efficient crowdsourcing for multi-class labeling. In *ACM SIGMETRICS*, volume 41, pages 81–92, 2013.

[31] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *ICML*, 2003.

[32] N. Kriege and P. Mutzel. Subgraph matching kernels for attributed graphs. In *ICML*, 2012.

[33] Baichuan Li, Irwin King, and Michael R Lyu. Question routing in community question answering: putting category in its place. In *CIKM*, pages 2041–2044, 2011.

[34] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[35] Xiaoyong Liu, W Bruce Croft, and Matthew Koll. Finding experts in community-based question-answering services. In *CIKM*, pages 315–316, 2005.

[36] U. Lösch, S. Bloehdorn, and A. Rettinger. Graph kernels for rdf data. In *the Semantic Web: Research and Applications*, pages 134–148. Springer, 2012.

[37] P. Mahé and J. Vert. Graph kernels based on tree patterns for molecules. *Machine Learning*, 75(1):3–35, 2009.

[38] Gengxin Miao, Louise E Moser, Xifeng Yan, Shu Tao, Yi Chen, and Nikos Anerousis. Generative models for ticket resolution in expert networks. In *SIGKDD*, pages 733–742, 2010.

[39] Gengxin Miao, Shu Tao, Winnie Cheng, Randy Moulic, Louise E Moser, David Lo, and Xifeng Yan. Understanding task-driven information flow in collaborative networks. In *WWW*, pages 849–858, 2012.

[40] S. Mika, B. Schölkopf, A. J. Smola, KR Müller, M. Scholz, and G. Rätsch. Kernel PCA and de-noising in feature spaces. In *NIPS*, 1998.

[41] David Mimno and Andrew McCallum. Expertise modeling for matching papers with reviewers. In *SIGKDD*, pages 500–509, 2007.

[42] M. Neumann, R. Garnett, C. Bauckhage, and K. Kersting. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, pages 1–37, 2015.

[43] M. Neumann, N. Patricia, R. Garnett, and K. Kersting. Efficient graph kernels by randomization. In *Machine Learning and Knowledge Discovery in Databases*, pages 378–393. 2012.

[44] F. Odone, A. Barla, and A. Verri. Building kernels from binary strings for image matching. *IEEE Trans. on Image Processing*, 14(2):169–180, 2005.

[45] Michal Rosen-Zvi, Chaitanya Chemudugunta, Thomas Griffiths, Padhraic Smyth, and Mark Steyvers. Learning author-topic models from text corpora. *TOIS*, 28(1):4, 2010.

[46] B. Schölkopf and A. J. Smola. *Learning with kernels*. The MIT Press, 2002.

[47] Pavel Serdyukov, Henning Rode, and Djoerd Hiemstra. Modeling multi-step relevance propagation for expert finding. In *CIKM*, pages 1133–1142, 2008.

[48] Qihong Shao, Yi Chen, Shu Tao, Xifeng Yan, and Nikos Anerousis. Efficient ticket routing by resolution sequence mining. In *SIGKD*, pages 605–613, 2008.

[49] N. Shervashidze and K. Borgwardt. Fast subtree kernels on graphs. In *NIPS*, 2009.

[50] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. Borgwardt. Weisfeiler-lehman graph kernels. *the Journal of Machine Learning Research*, 12:2539–2561, 2011.

[51] A. Smalter, J. Huan, Y. Jia, and G. Lushington. GPD: a graph pattern diffusion kernel for accurate graph classification with applications in cheminformatics. *TCBB*, 7(2):197–207, 2010.

[52] Chunke Su and Noshir Contractor. A multidimensional network approach to studying team members' information seeking from human and digital knowledge sources in consulting firms. *JASIST*, 62(7):1257–1275, 2011.

[53] Huan Sun, Mudhakar Srivatsa, Shulong Tan, Yang Li, Lance M Kaplan, Shu Tao, and Xifeng Yan. Analyzing expert behaviors in collaborative networks. In *SIGKDD*, pages 1486–1495, 2014.

[54] J. J. Sutherland, Lee A. O'Brien, and D. F. Weaver. Spline-fitting with a genetic algorithm: A method for developing classification structure-activity relationships. *Journal of Chemical Information and Computer Sciences*, 43(6):1906–1915, 2003.

[55] Kush R Varshney, Vijil Chenthamarakshan, Scott W Fancher, Jun Wang, Dongping Fang, and Aleksandra Mojsilović. Predicting employee expertise for talent management in the enterprise. In *SIGKDD*, pages 1729–1738, 2014.

[56] J. Vert. The optimal assignment kernel is not positive definite. *arXiv preprint arXiv:0801.4061*, 2008.

[57] Daniel M Wegner, Toni Giuliano, and Paula T Hertel. Cognitive interdependence in close relationships. In *Compatible and Incompatible Relationships*, pages 253–276. 1985.

[58] Peter Welinder, Steve Branson, Pietro Perona, and Serge J Belongie. The multidimensional wisdom of crowds. In *NIPS*, pages 2424–2432, 2010.

[59] Xiaohui Yan, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. A biterm topic model for short texts. In *WWW*, pages 1445–1456, 2013.

[60] Reyyan Yeniterzi. Effective approaches to retrieving and using expertise in social media. In *SIGIR*, pages 1150–1150, 2013.

[61] Haoqi Zhang, Eric Horvitz, Yiling Chen, and David C Parkes. Task routing for prediction tasks. In *AAMS-Volume 2*, pages 889–896, 2012.

[62] Yanhong Zhou, Gao Cong, Bin Cui, Christian S Jensen, and Junjie Yao. Routing questions to the right users in online communities. In *ICDE*, pages 700–711, 2009.