Chapter 13

TEXT MINING IN SOCIAL NETWORKS

Charu C. Aggarwal
IBM T. J. Watson Research Center
Hawthorne, NY 10532, USA
charu@us.ibm.com

Haixun Wang
Microsoft Research Asia
Beijing, China 100190
haixunw@microsoft.com

Abstract Social networks are rich in various kinds of contents such as text and multimedia. The ability to apply text mining algorithms effectively in the context of text data is critical for a wide variety of applications. Social networks require text mining algorithms for a wide variety of applications such as keyword search, classification, and clustering. While search and classification are well known applications for a wide variety of scenarios, social networks have a much richer structure both in terms of text and links. Much of the work in the area uses either purely the text content or purely the linkage structure. However, many recent algorithms use a combination of linkage and content information for mining purposes. In many cases, it turns out that the use of a combination of linkage and content information provides much more effective results than a system which is based purely on either of the two. This paper provides a survey of such algorithms, and the advantages observed by using such algorithms in different scenarios. We also present avenues for future research in this area.

Keywords: Text Mining, Social Networks

1. Introduction

Social networks are typically rich in text, because of a wide variety of methods by which users can contribute text content to the network. For example, typical social networks such as *Facebook* allow the creation of various text content such as wall posts, comments, and links to blog and web pages. Emails between different users can also be expressed as social networks, which can be mined for a variety of applications. For example, the well known *Enron email database* is often used in order to mine interesting connections between the different characters in the underlying database. Using interesting linkages within email and newsgroup databases in addition to the content [5, 8] often leads to qualitatively more effective results.

Social networks are rich in text, and therefore it is useful to design text mining tools for a wide variety of applications. While a variety of search and mining algorithms have been developed in the literature for text applications, social networks provide a special challenge, because the linkage structure provides guidance for mining in a variety of applications. Some examples of applications in which such guidance is available are as follows:

- **Keyword Search:** In the problem of keyword search, we specify a set of keywords, which are used to determine social network nodes which are relevant to a given query. In the problem of keyword search, we use both the content and the linkage behavior in order to perform the search. The broad idea is that text documents containing similar keywords are often linked together. Therefore, it is often useful to determine closely connected clusters of nodes in the social network which contain specific keywords. This problem is also related to the problem of *expertise search* [34] in social networks, in which we wish to determine the individuals in the social network with a particular kind of expertise. The problem is expertise search is discussed in detail in Chapter 8 of the book.

- **Classification:** In the problem of classification, the nodes in the social network are associated with labels. These labeled nodes are then used for classification purposes. A variety of algorithms are available for classification of text from content only. However, the presence of links often provides useful hints for the purpose of classification. For example, label propagation techniques can be combined with content-based classification in order to provide more effective results. This chapter discusses a number of such algorithms.

- **Clustering:** In the problem of clustering, we would like to determine sets of nodes which have similar content for the purposes of clustering. The linkage structure can also play a very useful role during such a clustering process. In many applications, linkage and content [53] can be combined together for the purposes of classification. This results in clustering which is of much better quality.

- **Linkage-based Cross Domain Learning:** Social networks contain a large amount of linked information between different kinds of objects such as text articles, tags, posts, images, and videos. This linkage information can be used in order to transfer knowledge across different kinds of links. We refer to this class of methods as *transfer learning*.

A common characteristic of the above problems is that they are defined in the content space, and are extensively studied by the text mining community. However, social networks can be considered linked networks, and therefore links can be used to enhance each of the above problems. It is important to note that these problems are not just defined in the context of social networks, but have also been studied *more generally* in the context of any linked network such as the World Wide Web. In fact, most of the earlier research on these problems has been performed more generally in the context of the web, which is also an example of a content-based linked network. Thus, this chapter deals more generally with the problem of combining text content analysis with link analysis; it applies *more generally* to a variety of web domains *including* social networks. Some of these problems also arise in the context of the XML domain, which can also be considered a content-based linked graph. For example, the problem of keyword search has been studied extensively in the context of the XML domain; however the techniques are quite useful for content and link-based keyword search even in the social network domain. Many of the techniques which are designed in these different domains are applicable to one another. We have included the study of this class of methods in this book because of its significant importance to the problem of social network analysis.

The linkage information in social networks also provides interesting hints, which can be adapted for problems such as transfer learning. In transfer learning, we attempt to use the knowledge which is learned in one domain to another domain with the use of bridges (or mappings) between different domains. These bridges may be in the form of similar class structure across the domains, dictionaries which define mappings between attributes (as in cross-lingual learning), or links in network based repositories. The links in a social network provide good bridging information which can be leveraged for the learning process [40]. In this chapter, we will provide a key overview of some of the techniques which are often utilized for linkage-based cross-domain learning.

The chapter is organized as follows. The next section will discuss algorithms for keyword search. In section 3, we will discuss algorithms for classification. Section 4 will discuss algorithms for clustering. Section 5 will study the problem of *transfer-learning*, as it relates to the social network domain. Section 6 contains the conclusions and summary.

2. Keyword Search

Keyword search provides a simple but user-friendly interface for information retrieval on the Web. It also proves to be an effective method for accessing structured data. Since many real life data sets are structured as tables, trees and graphs, keyword search over such data has become increasingly important and has attracted much research interest in both the database and the IR communities.

A social network may be considered as a massive graph, in which each node may contain a large amount of text data. We note that many informal forms of social networks such as blogs or paper-citation graphs also contain a large amount of text graph. Many of these graphs do not have any privacy restrictions in order to implement an effective search process. We would like to determine small groups of link-connected nodes which are related to a particular set of keywords. Even though keyword search is defined with respect to the text inside the nodes, we note that the linkage structure also plays an important role in determining the appropriate set of nodes. It is well known the text in linked entities such as the web are related, when the corresponding objects are linked. Thus, by finding groups of closely connected nodes which share keywords, it is generally possible to determine the qualitatively effective nodes. The problem is also related to that of *expertise location* [34] in social networks, in which we would like to determine groups of closely connected individuals with a particular kind of expertise. The problem of expertise search is discussed in detail in Chapter 8 of this book.

Because the underlying data assumes a graph structure, keyword search becomes much more complex than traditional keyword search over documents. The challenges lie in three aspects:

- **Query semantics**: Keyword search over a set of text documents has very clear semantics: A document satisfies a keyword query if it contains every keyword in the query. In our case, the entire dataset is often considered as a single graph, so the algorithms must work on a finer granularity and return subgraphs as answers. We must decide what subgraphs are qualified as answers. The qualification of a subgraph as a valid answer depends both upon the content in the document and the underlying linkage structure.

- **Ranking strategy**: For a given keyword query, it is likely that many subgraphs will satisfy the query, based on the query semantics in use. However, each subgraph has its own underlying graph structure, with subtle semantics that makes it different from other subgraphs that satisfy the query. Thus, we must take the graph structure into consideration and design ranking strategies that find most meaningful and relevant answers. This is not the case for content-based keyword search in which content-based objective functions can be defined in order to quantify the objective function.

- **Query efficiency**: Many real life graphs are extremely large. A major challenge for keyword search over graph data is query efficiency, which, to a large extent, hinges on the semantics of the query and the ranking strategy. Clearly, the ability to perform efficient search depends upon our ability to perform an effective traversal of the underlying graph structure.

2.1 Query Semantics and Answer Ranking

A query consists of a set of keywords $Q = \{k_1, k_2, \cdots, k_n\}$. We must first define what is a qualified answer to Q, and the goodness (the score) of the answer.

For tree structures such as XML, under the most common semantics, the goal is to find the *smallest* subtrees that contain the keywords. There are different ways to interpret the notion of *smallest*. Several algorithms [51, 27, 50] are based on the SLCA (smallest lowest common ancestor) semantics, which requires that an answer (a least common ancestor of nodes that contain all the keywords) does not have any descendent that is also an answer. XRank [21] adopts a different query semantics for keyword search. In XRank, answers consist of substrees that contain at least one occurrence of all of the query keywords, after excluding the sub-nodes that already contain all of the query keywords. Thus, the set of answers based on the SLCA semantics is a subset of answers qualified for XRank.

We can use similar semantics for keyword search over graphs. For this purpose, the answer must first form trees (embedded in the graph). In many graph search algorithms, including BANKS [7], the bidirectional algorithm [29], and BLINKS [23], a response or an answer to a keyword query is a minimal rooted tree T embedded in the graph that contains at least one node from each S_i, where S_i is the set of nodes that match the keyword k_i.

Next, we must define a measure for the “goodness” of each answer. An answer tree T is good if it is meaningful to the query, and the meaning of T lies in the tree structure, or more specifically, how the keyword nodes are connected through paths in T. The goodness measure adopted by BANKS and the bidirectional algorithm is as follows. An answer tree T is decomposed into edges and nodes, and score of T is the combined score of the edges and nodes of T. Specifically, each edge has a pre-defined weight, and default to 1. Given an answer tree T, for each keyword k_i, we use $s(T, k_i)$ to represent the sum of the edge weights on the path from the root of T to the leaf containing keyword k_i. Thus, the aggregated edge score is $E = \sum_i^n s(T, k_i)$. The nodes, on the other hand, are scored by their global importance or prestige, which is usually based on PageRank [10] random walk. Let N denote the aggregated score of nodes that contain keywords. The combined score of an answer tree is given by $s(T) = EN^\lambda$ where λ helps adjust the importance of edge and node scores [7, 29].

Query semantics and ranking strategies used in BLINKS [23] are similar to those of BANKS [7] and the bidirectional search [29]. But instead of using a measure such as $S(T) = EN^\lambda$ to find top-K answers, BLINKS requires that each of the top-K answer has a different root node, or in other words, for all answer trees rooted at the same node, only the one with the highest score is considered for top-K. This semantics guards against the case where a "hub" pointing to many nodes containing query keywords becomes the root for a huge number of answers. These answers overlap and each carries very little additional information from the rest. Given an answer (which is the best, or one of the best, at its root), users can always choose to further examine other answers with this root [23].

Unlike most keyword search on graph data approaches [7, 29, 23], ObjectRank [6] does not return answer trees or subgraphs containing keywords in the query, instead, for ObjectRank, an answer is simply a node that has high authority on the keywords in the query. Hence, a node that does not even contain a particular keyword in the query may still qualify as an answer as long as enough authority on that keyword has flown into that node (Imagine a node that represents a paper which does not contain keyword *OLAP*, but many important papers that contain keyword *OLAP* reference that paper, which makes it an authority on the topic of *OLAP*). To control the flow of authority in the graph, ObjectRank models *labeled* graphs: Each node u has a label $\lambda(u)$ and contains a set of keywords, and each edge e from u to v has a label $\lambda(e)$ that represents a relationship between u and v. For example, a node may be labeled as a *paper*, or a *movie*, and it contains keywords that describe the paper or the movie; a directed edge from a paper node to another paper node may have a label *cites*, etc. A keyword that a node contains directly gives the node certain authority on that keyword, and the authority flows to other nodes through edges connecting them. The amount or the rate of the outflow of authority from keyword nodes to other nodes is determined by the types of the edges which represent different semantic connections.

2.2 Keyword search over XML and relational data

Keyword search on XML data [19, 15, 21, 31] is a simpler problem than on schema-free graphs. In some cases, documents in social networks are expressed as XML documents as well. Therefore, it is interesting to explore this particular case.

XML data is mostly tree structured, where each node only has a single incoming path. This property has significant impact on query semantics and answer ranking, and it also provides great optimization opportunities in algorithm design [51].

It is straightforward to find subtrees that contain all the keywords. Let L_i be the set of nodes in the XML document that contain keyword k_i. If we pick one node n_i from each L_i, and form a subtree from these nodes, then the subtree will contain all the keywords. Thus, an answer to the query can be represented by $lca(n_1, \cdots, n_n)$, the lowest common ancestor of nodes $n_1, \cdots, n_n$ in the tree, where $n_i \in L_i$.

A keyword query may find a large number of answers, but they are not all equal due to the differences in the way they are embedded in the nested XML structure. Many approaches for keyword search on XML data, including XRank [21] and XSEarch [15], present a ranking method. A ranking mechanism takes into consideration several factors. For instance, more specific answers should be ranked higher than less specific answers. Both SLCA and the semantics adopted by XRank signify this consideration. Furthermore, keywords in an answer should appear *close* to each other, and closeness is interpreted as the the semantic distance defined over the XML embedded structure.

For relational data, SQL is the de-facto query language for accessing relational data. However, to use SQL, one must have knowledge about the schema of the relational data. This has become a hindrance for potential users to access tremendous amount of relational data.

Keyword search is a good alternative due to its ease of use. The challenges of applying keyword search on relational data come from the fact that in a relational database, information about a single entity is usually divided among several tables. This is resulted from the normalization principle, which is the design methodology of relational database schema.

Thus, to find entities that are relevant to a keyword query, the search algorithm has to join data from multiple tables. If we represent each table as a node, and each foreign key relationship as an edge between two nodes, then we obtain a graph, which allows us to convert the current problem to the problem of keyword search over graphs. However, there is the possibility of self-joins: that is, a table may contain a foreign key that references itself. More generally, there might be cycles in the graph, which means the size of the join is only limited by the size of the data. To avoid this problem, the search algorithm may adopt an upper bound to restrict the number of joins [28].

Two most well-known keyword search algorithm for relational data are DBXplorer [4] and DISCOVER [28]. They adopted new physical database design (including sophisticated indexing methods) to speed up keyword search over relational databases. Qin et al [41], instead, introduced a method that takes full advantage of the power of RDBMS and uses SQL to perform keyword search on relational data.

2.3 Keyword search over graph data

Keyword search over large, schema-free graphs faces the challenge of how to efficiently explore the graph structure and find subgraphs that contain all the keywords in the query. To measure the "goodness" of an answer, most approaches score each edge and node, and then aggregate the scores over the subgraph as a goodness measure [7, 29, 23]. Usually, an edge is scored by the strength of the connection, and a node is scored by its importance based on a PageRank like mechanism.

Graph keyword search algorithms can be classified into two categories. Algorithms in the first category finds matching subgraphs by exploring the graph link by link, without using any index of the graph. Representative algorithms in this category include BANKS [7] and the bidirectional search algorithm [29]. One drawback of these approaches is that they explore the graph blindly as they do not have a global picture of the graph structure, nor do they know the keyword distribution in the graph. Algorithms in the other category are index-based [23], and the index is used to guide the graph exploration, and support "forward-jumps" in the search.

2.3.1 Graph Exploration by Backward Search. Many keyword search algorithms try to find trees embedded in the graph so that similar query semantics for keyword search over XML data can be used. Thus, the problem is how to construct an embedded tree from keyword nodes in the graph. In the absence of any index that can provide graph connectivity information beyond a single hop, BANKS [7] answers a keyword query by exploring the graph starting from the nodes containing at least one query keyword – such nodes can be identified easily through an inverted-list index. This approach naturally leads to a *backward search* algorithm, which works as follows.

1 At any point during the backward search, let E_i denote the set of nodes that we know can reach query keyword k_i; we call E_i the *cluster* for k_i.

2 Initially, E_i starts out as the set of nodes O_i that directly contain k_i; we call this initial set the *cluster origin* and its member nodes *keyword nodes*.

3 In each search step, we choose an incoming edge to one of previously visited nodes (say v), and then follow that edge *backward* to visit its source node (say u); any E_i containing v now expands to include u as

well. Once a node is visited, all its incoming edges become known to the search and available for choice by a future step.

4 We have discovered an answer root x if, for each cluster E_i, either $x \in E_i$ or x has an edge to some node in E_i.

BANKS uses the following two strategies for choosing what nodes to visit next. For convenience, we define the distance from a node n to a set of nodes N to be the shortest distance from n to any node in N.

1 ***Equi-distance expansion in each cluster***: This strategy decides which node to visit for expanding a keyword. Intuitively, the algorithm expands a cluster by visiting nodes in order of increasing distance from the cluster origin. Formally, the node u to visit next for cluster E_i (by following edge $u \rightarrow v$ backward, for some $v \in E_i$) is the node with the shortest distance (among all nodes not in E_i) to O_i.

2 ***Distance-balanced expansion across clusters***: This strategy decides the frontier of which keyword will be expanded. Intuitively, the algorithm attempts to balance the distance between each cluster's origin to its frontier across all clusters. Specifically, let (u, E_i) be the node-cluster pair such that $u \notin E_i$ and the distance from u to O_i is the shortest possible. The cluster to expand next is E_i.

He et al. [23] investigated the optimality of the above two strategies introduced by BANKS [7]. They proved the following result with regard to the first strategy, *equi-distance expansion of each cluster* (the complete proof can be found in [24]):

THEOREM 13.1 *An optimal backward search algorithm must follow the strategy of equi-distance expansion in each cluster.*

However, the investigation [23] showed that the second strategy, *distance-balanced expansion across clusters*, is not optimal and may lead to poor performance on certain graphs. Figure 13.1 shows one such example. Suppose that $\{k_1\}$ and $\{k_2\}$ are the two cluster origins. There are many nodes that can reach k_1 through edges with a small weight (1), but only one edge into k_2 with a large weight (100). With distance-balanced expansion across clusters, we would not expand the k_2 cluster along this edge until we have visited all nodes within distance 100 to k_1. It would have been unnecessary to visit many of these nodes had the algorithm chosen to expand the k_2 cluster earlier.

2.3.2 Graph Exploration by Bidirectional Search. To address the problem shown in Figure 13.1, Kacholia et al. [29] proposed a *bidirectional search* algorithm, which has the option of exploring the graph by following

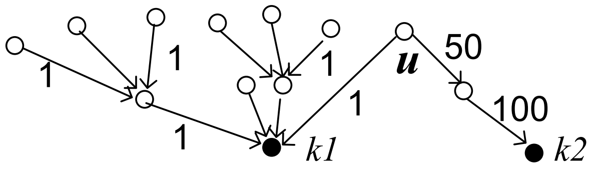

Figure 13.1. Distance-balanced expansion across clusters may perform poorly.

forward edges as well. The rationale is that, for example, in Figure 13.1, if the algorithm is allowed to explore forward from node u towards k_2, we can identify u as an answer root much faster.

To control the order of expansion, the bidirectional search algorithm prioritizes nodes by heuristic *activation factors* (roughly speaking, PageRank with decay), which intuitively estimate how likely nodes can be roots of answer trees. In the bidirectional search algorithm, nodes matching keywords are added to the iterator with an initial activation factor computed as:

$$a_{u,i} = \frac{nodePrestige(u)}{|S_i|}, \forall u \in S_i \tag{13.1}$$

where S_i is the set of nodes that match keyword i. Thus, nodes of high prestige will have a higher priority for expansion. But if a keyword matches a large number of nodes, the nodes will have a lower priority. The activation factor is spread from keyword nodes to other nodes. Each node v spreads a fraction μ of the received activation to its neighbors, and retains the remaining $1 - \mu$ fraction.

As a result, keyword search in Figure 13.1 can be performed more efficiently. The bidirectional search will start from the keyword nodes (dark solid nodes). Since keyword node k_1 has a large fanout, all the nodes pointing to k_1 (including node u) will receive a small amount of activation. On the other hand, the node pointing to k_2 will receive most of the activation of k_2, which then spreads to node u. Thus, node u becomes the most activated node, which happens to be the root of the answer tree.

While this strategy is shown to perform well in multiple scenarios, it is difficult to provide any worst-case performance guarantee. The reason is that activation factors are heuristic measures derived from general graph topology and parts of the graph already visited. They do not accurately reflect the likelihood of reaching keyword nodes through an unexplored region of the graph within a reasonable distance. In other words, without additional connectivity information, forward expansion may be just as aimless as backward expansion [23].

2.3.3 Index-based Graph Exploration. The effectiveness of forward and backward expansions hinges on the structure of the graph and the distribution of keywords in the graph. However, both forward and backward expansions explore the graph link by link, which means the search algorithms do not have knowledge of either the structure of the graph nor the distribution of keywords in the graph. If we create an index structure to store the keyword reachability information in advance, we can avoid aimless exploration on the graph and improve the performance of keyword search. BLINKS [23] is designed based on this intuition.

BLINKS makes two contributions: First, it proposes a new, *cost-balanced* strategy for controlling expansion across clusters, with a provable bound on its worst-case performance. Second, it uses indexing to support forward jumps in search. Indexing enables it to determine whether a node can reach a keyword and what the shortest distance is, thereby eliminating the uncertainty and inefficiency of step-by-step forward expansion.

Cost-balanced expansion across clusters. Intuitively, BLINKS attempts to balance the number of accessed nodes (i.e., the search cost) for expanding each cluster. Formally, the cluster E_i to expand next is the cluster with the smallest cardinality.

This strategy is intended to be combined with the equi-distance strategy for expansion within clusters: First, BLINKS chooses the smallest cluster to expand, then it chooses the node with the shortest distance to this cluster's origin to expand.

To establish the optimality of an algorithm A employing these two expansion strategies, let us consider an optimal "oracle" backward search algorithm P. As shown in Theorem 13.1, P must also do equi-distance expansion within each cluster. The additional assumption here is that P "magically" knows the right amount of expansion for each cluster such that the total number of nodes visited by P is minimized. Obviously, P is better than the best practical backward search algorithm we can hope for. Although A does not have the advantage of the oracle algorithm, BLINKS gives the following theorem (the complete proof can be found in [24]) which shows that A is m-optimal, where m is the number of query keywords. Since most queries in practice contain very few keywords, the cost of A is usually within a constant factor of the optimal algorithm.

THEOREM 13.2 *The number of nodes accessed by A is no more than m times the number of nodes accessed by P, where m is the number of query keywords.*

Index-based Forward Jump. The BLINKS algorithm [23] leverages the new search strategy (*equi-distance* plus *cost-balanced* expansions) as well as indexing to achieve good query performance. The index structure consists of two parts.

- **Keyword-node lists** L_{KN}. BLINKS pre-computes, for each keyword, the shortest distances from every node to the keyword (or, more precisely, to any node containing this keyword) in the data graph. For a keyword w, $L_{KN}(w)$ denotes the list of nodes that can reach keyword w, and these nodes are ordered by their distances to w. In addition to other information used for reconstructing the answer, each entry in the list has two fields $(dist, node)$, where $dist$ is the shortest distance between $node$ and a node containing w.

- **Node-keywordmap** M_{NK}. BLINKS pre-computes, for each node u, the shortest graph distance from u to every keyword, and organize this information in a hash table. Given a node u and a keyword w, $M_{NK}(u, w)$ returns the shortest distance from u to w, or ∞ if u cannot reach any node that contains w. In fact, the information in M_{NK} can be derived from L_{KN}. The purpose of introducing M_{NK} is to reduce the linear time search over L_{KN} for the shortest distance between u and w to $O(1)$ time search over M_{NK}.

The search algorithm can be regarded as index-assisted backward and forward expansion. Given a keyword query $Q = \{k_1, \cdots, k_n\}$, for backward expansion, BLINKS uses a cursor to traverse each keyword-node list $L_{KN}(k_i)$. By construction, the list gives the equi-distance expansion order in each cluster. Across clusters, BLINKS picks a cursor to expand next in a round-robin manner, which implements cost-balanced expansion among clusters. These two together ensure optimal backward search. For forward expansion, BLINKS uses the node-keyword map M_{NK} in a direct fashion. Whenever BLINKS visits a node, it looks up its distance to other keywords. Using this information, it can immediately determine if the root of an answer is found.

The index L_{KN} and M_{NK} are defined over the entire graph. Each of them contains as many as $N \times K$ entries, where N is the number of nodes, and K is the number of distinct keywords in the graph. In many applications, K is on the same scale as the number of nodes, so the space complexity of the index comes to $O(N^2)$, which is clearly infeasible for large graphs. To solve this problem, BLINKS partitions the graph into multiple blocks, and the L_{KN} and M_{NK} index for each block, as well as an additional index structure to assist graph exploration across blocks.

2.3.4 The ObjectRank Algorithm. Instead of returning sub-graphs that contain all the keywords, ObjectRank [6] applies authority-based ranking to keyword search on labeled graphs, and returns nodes having high authority with respect to all keywords. To certain extent, ObjectRank is similar to BLINKS [23], whose query semantics prescribes that all top-K answer trees have different root nodes. Still, BLINKS returns sub-graphs as answers.

Recall that the bidirectional search algorithm [29] assigns activation factors to nodes in the graph to guide keyword search. Activation factors originate at nodes containing the keywords and propagate to other nodes. For each keyword node u, its activation factor is weighted by $nodePrestige(u)$ (Eq. 13.1), which reflects the importance or authority of node u. Kacholia et al. [29] did not elaborate on how to derive $nodePrestige(u)$. Furthermore, since graph edges in [29] are all the same, to spread the activation factor from a node u, it simply divides u's activation factor by u's fanout.

Similar to the activation factor, in ObjectRank [6], authority originates at nodes containing the keywords and flows to other nodes. Furthermore, nodes and edges in the graphs are labeled, giving graph connections semantics that controls the amount or the rate of the authority flow between two nodes.

Specifically, ObjectRank assumes a labeled graph G is associated with some predetermined schema information. The schema information decides the rate of authority transfer from a node labeled u_G, through an edge labeled e_G, and to a node labeled v_G. For example, authority transfers at a fixed rate from a *person* to a *paper* through an edge labeled *authoring*, and at another fixed rate from a *paper* to a *person* through an edge labeled *authoring*. The two rates are potentially different, indicating that authority may flow at a different rate backward and forward. The schema information, or the rate of authority transfer, is determined by domain experts, or by a trial and error process.

To compute node authority with regard to every keyword, ObjectRank computes the following:

- **Rates of authority transfer through graph edges.** For every edge $e = (u \to v)$, ObjectRank creates a forward authority transfer edge $e^f = (u \to v)$ and a backward authority transfer edge $e^b = (v \to u)$. Specifically, the authority transfer edges e^f and e^b are annotated with rates $\alpha(e^f)$ and $\alpha(e^b)$:

$$\alpha(e^f) = \begin{cases} \frac{\alpha(e^f_G)}{OutDeg(u, e^f_G)} & \text{if } OutDeg(u, e^f_G) > 0 \\ 0 & \text{if } OutDeg(u, e^f_G) = 0 \end{cases} \tag{13.2}$$

  where $\alpha(e^f_G)$ denotes the fixed authority transfer rate given by the schema, and $OutDeg(u, e^f_G)$ denotes the number of outgoing nodes from u, of type e^f_G. The authority transfer rate $\alpha(e^b)$ is defined similarly.

- **Node authorities.** ObjectRank can be regarded as an extension to PageRank [10]. For each node v, ObjectRank assigns a global authority denoted by $ObjectRank^G(v)$ that is independent of the keyword query. The global $ObjectRank^G$ is calculated using the random surfer model, which is similar to PageRank. In addition, for each keyword w and each node v, ObjectRank integrates authority transfer rates in Eq 13.2 with PageRank to calculate a keyword-specific ranking $ObjectRank^w(v)$:

$$ObjectRank^w(v) = d \times \sum_{e=(u \to v) or (v \to u)} \alpha(e) \times ObjectRank^w(u) + \frac{1-d}{|S(w)|} \tag{13.3}$$

where $S(w)$ is s the set of nodes that contain the keyword w, and d is the damping factor that determines the portion of ObjectRank that a node transfers to its neighbours as opposed to keeping to itself [10]. The final ranking of a node v is the combination combination of $ObjectRank^G(v)$ and $ObjectRank^w(v)$.

3. Classification Algorithms

The problem of classification has been widely studied in the text mining literature. Some common algorithms which are often used for content-based text classification are the Naive Bayes classifier [36], TFIDF classifier [30] and Probabilistic Indexing classifier [20] respectively. A common tool kit used for classification is Rainbow [37], which contains a variety of different classifiers. Most of the classification algorithms directly use the text content in order to relate the content in the document to the class label.

In the context of social networks, we assume that the nodes in the social network are associated with labels, and each node may contain a certain amount of text content. In the case of social networks, a number of additional challenges arise in the context of text classification. These challenges are as follows:

- Social networks contain a much larger and non-standard vocabulary, as compared to more standard collections such as news collections. This is because of the greater diversity in authorship both in terms of the number and style of different authors.

- The labels in social networks may often be quite sparse. In many cases, some of the label values may be unknown. Thus, social network data is often much more noisy than other standard text collections.

- A particularly useful property of social networks is that they may contain links which can be used in order to guide the classification process. Such links can be very helpful in determining how the labels may be propagated between the different nodes. While pure link-based classification [44, 46, 48] is a well known technique which works effectively in a variety of scenarios, the use of content can greatly improve the effectiveness of the classification process.

The earliest work on combining linkage and content information for classification was discussed in [13]. This technique was designed in the context of the web, though the general idea can be easily extended to the case of social networks. In this paper, a hypertext categorization method was proposed, which uses the content and labels of neighboring web pages for the classification process. When the labels of all the nearest neighbors are available, then a Bayesian method can be adapted easily for classification purposes. Just as the presence of a word in a document can be considered a Bayesian feature for a text classifier, the presence of a link between the target page, and a page (for which the label is known) can be considered a feature for the classifier. The real challenge arises when the labels of all the nearest neighbors are not available. In such cases, a relaxation labeling method is proposed in order to perform the classification. Two methods have been proposed in this work:

- **Completely Supervised Case of Radius one Enhanced Linkage Analysis:** In this case, it is assumed that all the neighboring class labels are known. In such a case, a Bayesian approach is utilized in order to treat the labels on the nearest neighbors as features for classification purposes. In this case, the linkage information is the sole information which is used for classification purposes.

- **When the class labels of the nearest neighbors are not known:** In this case, an iterative approach is used for combining text and linkage based classification. Rather than using the pre-defined labels (which are not available), we perform a first labeling of the neighboring documents with the use of document content. These labels are then used to classify the label of the target document, with the use of *both* the local text and the class labels of the neighbors. This approach is used iteratively for re-defining the labels of both the target document and its neighbors until convergence is achieved.

The conclusion from the work in [13] is that a combination of text and linkage based classification always improves the accuracy of a text classifier. Even when none of the neighbors of the document have known classes, it seemed to be always beneficial to add link information to the classification process. When the class labels of all the neighbors are known, then the advantages of using the scheme seem to be quite significant.

An additional idea in the paper is that of the use of *bridges* in order to further improve the classification accuracy. The core idea in the use of a bridge is the use of *2-hop* propagation for link-based classification. The results with the use of such an approach are somewhat mixed, as the accuracy seems to reduce with an increasing number of hops. The work in [13] shows results on a number of different kinds of data sets such as the *Reuters database*, *US patent database*, and *Yahoo!*. We note that the *Reuters database* contains the least amount of noise, and pure text classifiers can do a good job. On the other hand, the *US patent database* and the *Yahoo! database* contain an increasing amount of noise which reduces the accuracy of text classifiers. An interesting observation in [13] was that a scheme which simply absorbed the neighbor text into the current document performed *significantly worse* than a scheme which was based on pure text-based classification. This is because there are often significant cross-boundary linkages between topics, and such linkages are able to confuse the classifier.

A second method which has been used for text and link based classification is the utilization of the graph regularization approach for text and link based classification [46]. The work presents a risk minimization formulation for learning from both text and graph structures. This is motivated by the problem of collective inference for hypertext document categorization. More details on the approach may be found in [46]. While much of the work in this paper as well as the earlier work in [13] is defined in the context of web data, this is also naturally applicable to the case of social networks which can be considered an interlinked set of nodes containing text content.

There is also some work which deals explicitly with social-network like text interactions. A classic example of this is a set of emails which are exchange between one or more different users. Typically, the emails in a particular "thread" may be considered to be linked based on the structure of the thread. Alternatively, the exchanges in the email between the different users correspond to the links between the different emails. An important problem which arises in the context of email classification is that of the classification of *speech acts* in email. Some examples of speech acts in email could be *request*, *deliver*, *commit*, and *propose*. One method for performing the classification is to use purely the *content* of the email for classification purposes [16]. An important observation in later work is that successive email exchanges between different users can be considered links, which can be utilized for more effective classification. This is because the speech acts in different emails may not be independent of one another, but may significantly influence each other. For example, an email asking for a "request" for a meeting may be followed by one which "commits" to a meeting.

The work in [11] proposes a method for collective classification, which tries to model the relationships between different kinds of acts in the form of a graphical stricture. The links in this graphical structure are leveraged in order to improve the effectiveness of the classification process. The work in [11] proposes an iterative collective classification algorithm. This algorithm us closely related to the implementation of a Dependency Network [25]. Dependency networks are probabilistic graphical models in which the full joint distribution of the network is approximated with a set of conditional distributions that can be learned independently. These probability distributions are calculated for each node given its parent nodes. In the context of an email network, the email messages are the nodes, and the dependencies are the threads which relate the different nodes. The work in [11] shows that the use of linkage analysis provides a much more effective classification analysis, because the class labels between the different nodes are naturally connected by their corresponding linkages.

4. Clustering Algorithms

The problem of clustering arises quite often in the context of node clustering of social networks. The problem of network clustering is closely related to the traditional problem of graph partitioning [32], which tries to isolated groups of nodes which are closely connected to one another. The problem of graph partitioning is NP-hard and often does not scale very well to large networks. The Kerninghan-Lin algorithm [32] uses an iterative approach in which we start off with a feasible partitioning and repeatedly interchange the nodes between the partitions in order to improve the quality of the clustering. We note that that this approach requires random access to the nodes and edges in the underlying graph. This can be a considerable challenge for large-scale social networks which are stored on disk. When the social-network is stored on disk, the random access to the disk makes the underlying algorithms quite inefficient. Some recent work has designed methods for link-based clustering in massive networks [9, 12, 14, 33], though these methods are not specifically designed for the case of disk-resident data. Such methods are typically designed for problems such as community detection [39, 38] and information network clustering [9, 33]. These methods are often more sensitive to the evolution of the underlying network, which is quite common in social networks because of the constant addition and deletion of users from the network. Some recent work [2] has also been proposed for clustering graph streams, which are common representations of different kinds of edge-based activity in social network analysis.

The work discussed above uses only the structure of the network for the clustering process. A natural question arises, as to whether one can improve the quality of clustering by using the text content in the nodes of the social network. The problem of clustering has been widely studied by the text mining community. A variety of text clustering algorithms have been proposed by the data mining and text mining community [3, 18, 42], which use a number of variants of traditional clustering algorithms for multi-dimensional data. Most of these methods are variants of the k-means method in which we start off with a set of k seeds and build the clusters iteratively around these seeds. The seeds and cluster membership are iteratively defined with respect to each other, until we converge to an effective solution. Some of these methods are also applicable to the case of *dynamic text data*, as is the case with social networks. In particular, the methods in [3, 47] can be used in order to cluster text data streams. These methods are designed on the basis of k-means methods which can use non-iterative variants of the k-means methods for clustering streams. Such techniques can be useful for the dynamic scenarios which can arise in the context of social networks. However, these algorithms tend to be at the other end of the spectrum in terms of preserving only content information and no linkage information. Ideally, we would like to use clustering algorithms which preserve both content and linkage information.

The work in [53] proposes a method which can perform the clustering with the use of both content and structure information. Specifically the method constructs a new graph which takes into account both the structure and attribute information. Such a graph has two kinds of edges: *structure edges* from the original graph, and *attribute edges*, which are based on the nature of the attributes in the different nodes. A random walk approach is used over this graph in order to define the underlying clusters. Each edge is associated with a weight, which is used in order to control the probability of the random walk across the different nodes. These weights are updated during an iterative process, and the clusters and the weights are successively used in order to refine each other. It has been shown in [53] that the weights and the clusters will naturally converge, as the clustering process progresses. While the technique is applied to multi-relational attributes, it can also used in a limited way for text attributes, especially when a limited subset of keywords is used. It has been shown in [53] how the method can be used in the context of blog and bibliographic networks, when a number of relational and keyword-based attributes are also utilized for the clustering process. It has been shown in [53] that the combination of structural and attribute information can be used for a more effective clustering process.

A method for leveraging both content and linkage information in the clustering process has been proposed in [43]. While the work in [43] addresses the problem of topic modeling rather than clustering, this process also provides a generative model, which can be used for the purpose of a soft clustering of the documents. A graphical model was proposed to describe a multi-layered generative model. At the top layer of this model, a multivariate Markov Random Field for topic distribution random variables for each document is defined. This is useful for modeling the dependency relationships among documents over the network structure. At the bottom layer, the traditional topic model is used in order to model the generation of text for each document. Thus, the combination of the two layers provides a way to model the relationships of the text and the structure with one another. Methods are proposed to decide the topic structure, as well as the number of topics. The method essentially uses an LDA model in order to construct a topic space, in which the clustering can be performed. This model uses both structure and content information for the modeling process. It has been shown in [43] that this integrated approach is much more useful than an approach which relies purely on the content of underlying documents. In general, model-based clustering methods are very effective, because of being able to model different kinds of data by introducing different model parameters. Another model-based method which has specifically been proposed in the context of social networks is discussed in [22]. The main idea in this method is the use of a latent position cluster model, in which the probability of a tie between two nodes depends on the distance between them in an implicit Euclidean social space, and the location of these nodes in the latent social space which is defined with the use of a mixture of cluster-based distributions. The method in [22] is general enough to incorporate different kinds of attributes in the clustering process. For example, one can associated linkage-behavior and keyword-behavior with different kinds of attributes and apply this approach in order to design an effective clustering. While the work in [22] has specifically not been used with the use of text-based content, the techniques proposed are quite applicable to this scenario, because of the approach used in the paper, which can embed arbitrary data in arbitrary Euclidean social spaces. We note that such latent-space approaches are useful not just for the problem of clustering, but as a tool which can be used for *representation* of the network in an Euclidean space, which is more amenable to data mining problems. A more general piece of work on the latent-space approach may be found in [26].

A major challenge which arises in the context of social networks is that many such networks are heterogeneous, and this makes it difficult to design algorithms which can determine the distances between the nodes in a more robust way. For example, bibliographic networks are classic examples of heterogeneous networks. Such networks contain different kinds of nodes corresponding to authors, keywords, conferences or papers. In general, information networks are a more generalized notion of social networks, which may contain nodes which correspond not just to actors, but also to other kinds of entities which are related to actors. Such networks are far more challenging to cluster as compared to the vanilla social network scenario, because the different kinds of entities may contain different objects such as text, images, and links. A major challenge in the field is to design unified clustering methods which can integrate different kinds of content in the clustering process.

5. Transfer Learning in Heterogeneous Networks

A closely related problem in this domain is that of *transfer learning*. Our discussion in this chapter has so far focussed on how content can be used in order to enhance the effectiveness of typical problems such as clustering or classification in the context of social networks. In this section, we will study how the link in an information network can be used in order to perform cross-domain learning in traditional data mining problems such as clustering and classification. The primary idea in transfer learning is that data mining problems have varying levels of difficulty in different domains either because of lack of data availability or representational issues, and it is often useful to *transfer* the knowledge from one domain to another with the use of mappings. The links in social networks can be used in order to define such mappings. A general survey of transfer-learning methods may be found in [40].

Many social networks contain *heterogeneous content*, such as text, media and images. In particular, text is a common content in such networks. For example, many social media such as *Flickr* contain images which are annotated with keywords. The problem of transfer learning is motivated by the fact that different kinds of content may be more or less challenging for the learning process. This is because different amounts of training data may be available to a user in different domains. For example, it is relatively easy to obtain training data for text content, as well as mapping the features to different classes. This may however not be quite as true in the case of the image domain in which there may be fewer standardized collections. Therefore, a natural approach is to use either the *links in the social network between text and images* or the *implicit links* in terms of annotations as a mapping for the transfer learning process. Such links can be used as a bridge to learn the underlying mapping, and use it for the problem of classification.

A similar observation applies to the case of cross-lingual classification problems. This is because different amounts of training data is available for the documents in different languages. A lot of labeled text content may be available for content in the english language. On the other hand, this may not be the case for other languages such as chinese web pages. However, there may often be a number of links between the documents in the different languages. Such links can be used in order to learn the mappings and use it for the classification process.

A tremendous amount of text-to-image mapping information exists in the form of tag information in social media sites such as *Flickr*. It has been shown in [17], that such information can be effectively leveraged for transfer-learning. The key idea is to construct a mapping between the text and the images with the use of the tags, and then use PLSA in order to construct a latent space which can be used for the transfer process. It has been shown in [17] that such an approach can be used very effectively for transfer learning. A related work [54] discusses how to create connections between images and text with the use of tag data. It shows how such links can be used more effectively for the classification process. These techniques have also been exploited for the problem of image clustering [52]. The work in [52] collects annotated image data from the social web, and uses it in order to construct a text to image mapping. The algorithm is referred to as aPLSA (Annotated Probabilistic Latent Semantic Analysis). The key idea is to unify two different kinds of latent semantic analysis in order to create a bridge between the text and images. The first kind of technique performs PLSA analysis on the target images, which are converted to an image instance-to-feature co-occurrence matrix. The second kind of PLSA is applied to the annotated image data from social Web, which is converted into a text-to-image feature co-occurrence matrix. In order to unify those two separate PLSA models, these two steps are done simultaneously with common latent variables used as a bridge linking them. It has been shown in [52] that such a bridging approach leads to much better clustering results. Clustering is a useful tool for web-based or social-network based image search. Such search results are often hindered by polysemy in textual collections, in which the same work may mean multiple things. For example, the word "jaguar" could either refer to an animal or a car. In such cases, the results of the query are ambiguous as well. A major motivation for clustering is that such ambiguities can be more effectively resolved. In particular when textual features are associated with images, and these are used simultaneously [35] for the clustering process, the overall result is much more effective. Some of the available techniques in the literature use spectral clustering on the distance matrix built from a multimodal feature set (containing both text and image information) in order to get a better feature representation. The improved quality of the representation leads to much better clustering results. Furthermore, when the number of images is very small, traditional clustering algorithms do not work very well either. This approach does not work too well, when only a small amount of data is available on the association between image and text. A different work in [52] treats this as a heterogeneous transfer learning problem by leveraging social annotation information from sites such as *Flickr*. Such sites contain a lot of information in the form of feedback from different users. The auxiliary data is used in order to improve the quality of the underlying latent representation and the corresponding clustering process in work discussed in [52]. In general, social networks provide tremendous information about the data of different types, which can be leveraged in order to enable an effective learning process across the different domains. This area is still in its infancy, and it is expected that future research will lead to further advances in such linkage-based techniques.

6. Conclusions and Summary

In this chapter, we presented a variety of content-based mining algorithms, which combine text and links in order to design more effective methods for a wide variety of problems such as search, clustering, and classification. In many data domains, links encode a tremendous amount of semantic information which can be leveraged in order to improve the effectiveness of a variety of different algorithms. A number of challenges remain for this particular problem domain. These challenges are as follows:

- Many of the techniques are not designed to be scalable for massive data sets. This may be a challenge, because many network data sets are inherently massive in nature.

- The techniques need to be designed for dynamic data sets. This is especially important because the content and links in a social network are highly dynamic. Therefore, it is important to be able to quickly re-adjust the models in order to take into account the changing characteristics of the underlying data.

- Many of the techniques are designed for homogeneous networks in which the nodes are of a single type. Many of the networks are inherently heterogeneous, in which the nodes may be of different types. For example, social networks contain nodes corresponding to individuals, their posts, their blogs, and the use of such other content. The ability to use such information in the knowledge discovery process can be an advantage in many scenarios.

A considerable research opportunity also exists in the area of transfer learning. Transfer learning methods are dependent upon the ability to define *bridges* or *mappings* between different domains for the learning process. The links in a social network can be leveraged in order to define such bridges. There has been some recent work in leveraging the text annotations in social media for the problem of transfer learning of images. Research in this area is still in its infancy, and considerable scope exists to improve both the effectiveness and the efficiency of the underlying algorithms.

Acknowledgements

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] C. C. Aggarwal, H. Wang (ed.) Managing and Mining Graph Data, *Springer*, 2010.

[2] C. C. Aggarwal, Y. Zhao, P. Yu. On Clustering Graph streams, *SIAM Conference on Data Mining*, 2010.

[3] C. C. Aggarwal, P. S. Yu. A Framework for Clustering Massive Text and Categorical Data Streams, *SIAM Conference on Data Mining*, 2006.

[4] S. Agrawal, S. Chaudhuri, G. Das. DBXplorer: A system for keyword-based search over relational databases. *ICDE Conference*, 2002.

[5] R. Agrawal, S. Rajagopalan, R. Srikant, Y. Xu. Mining Newsgroups using Networks arising from Social Behavior. *WWW Conference*, 2003.

[6] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[7] G. Bhalotia, C. Nakhe, A. Hulgeri, S. Chakrabarti, S. Sudarshan. Keyword searching and browsing in databases using BANKS. *ICDE Conference*, 2002.

[8] C. Bird, A. Gourley, P. Devanbabu, M. Gertz, A. Swaminathan. Mining Email Social Networks, *MSR*, 2006.

[9] D. Bortner, J. Han. Progressive Clustering of Networks Using Structure-Connected Order of Traversal, *ICDE Conference*, 2010.

[10] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[11] V. Carvalho, W. Cohen. On the Collective Classification of Email “Speech Acts”, *ACM SIGIR Conference*, 2005.

[12] D. Chakrabarti, R. Kumar, A. Tomkins. Evolutionary clustering. *KDD Conference*, 2006.

[13] S. Chakrabarti, B. Dom, P. Indyk. Enhanced Hypertext Categorization using Hyperlinks, *ACM SIGMOD Conference*, 1998.

[14] Y. Chi, X. Song, D. Zhou, K. Hino, B. L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. *ACM KDD Conference*, 2007.

[15] S. Cohen, J. Mamou, Y. Kanza, Y. Sagiv. XSEarch: A semantic search engine for XML. *VLDB Conference*, 2003.

[16] W. Cohen, V. Carvalho, T. Mitchell, Learning to Classify Email into ÂŞSpeech ActsÂŤ. *Conference on Empirical Methods in Natural Language Processing*, 2004.

[17] W. Dai, Y. Chen, G. Xue, Q. Yang, Y. Yu. Translated Learning: Transfer Learning across different Feature Spaces. *NIPS Conference*, 2008.

[18] D. R. Cutting, J. O. Pedersen, D. R. Karger, J. W. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections, *ACM SIGIR Conference*, 1992.

[19] D. Florescu, D. Kossmann, and I. Manolescu. Integrating keyword search into XML query processing. *Comput. Networks*, 33(1-6):119–135, 2000.

[20] N. Fuhr, C. Buckley. Probabilistic Document Indexing from Relevance Feedback Data. *SIGIR Conference*, pages 45–61, 1990.

[21] L. Guo, F. Shao, C. Botev, J. Shanmugasundaram. XRANK: ranked keyword search over XML documents. *ACM SIGMOD Conference*, pages 16–27, 2003.

[22] M. Handcock, A Raftery, J. Tantrum. Model-based Clustering for Social Networks. *Journal of the Royal Statistical Society*, 170(2), pp. 301–354, 2007.

[23] H. He, H. Wang, J. Yang, P. S. Yu. BLINKS: Ranked keyword searches on graphs. *SIGMOD Conference*, 2007.

[24] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: Ranked keyword searches on graphs. Technical report, Duke CS Department, 2007.

[25] D. Heckerman, D. Chickering, C. Meek, R. Rounthwaite, C. Kadie. Dependency networks for inference, collaborative filtering and data visualization. *Journal of Machine Learning Research*, 1, pp. 49–75, 2000.

[26] P. Hoff, A. Raftery, M. Handcock. Latent Space Approaches to Social Network Analysis, *Technical Report No. 399*, University of Washington at Seattle, 2001.

[27] V. Hristidis, N. Koudas, Y. Papakonstantinou, D. Srivastava. Keyword proximity search in XML trees. *IEEE Transactions on Knowledge and Data Engineering*, 18(4):525–539, 2006.

[28] V. Hristidis, Y. Papakonstantinou. Discover: Keyword search in relational databases. *VLDB Conference*, 2002.

[29] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, H. Karambelkar. Bidirectional expansion for keyword search on graph databases. *VLDB Conference*, 2005.

[30] T. Joachims. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. *ICML Conference*, pages 143–151, 1997.

[31] R. Kaushik, R. Krishnamurthy, J. F. Naughton, and R. Ramakrishnan. On the integration of structure indexes and inverted lists. In *SIGMOD*, pages 779–790, 2004.

[32] B. W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal* (49) pp. 291ÂŰ-307, 1970.

[33] M. S. Kim, J. Han. A Particle-and-Density Based Evolutionary Clustering Method for Dynamic Networks. *PVLDB*, 2(1): pp. 622–633, 2009.

[34] T. Lappas, K. Liu, E. Terzi. Finding a Team of Experts in Social Networks. *ACM KDD Conference*, 2009.

[35] N. Loeff, C. O. Alm, D. A. Forsyth. Discriminating image senses by clustering with multimodal features. *ACL Conference*, pp. 547ÂŰ-554, 2006.

[36] M. Maron. Automatic Indexing: An Experimental Inquiry. *J. ACM*, 8(3), pages 404-417, 1961.

[37] A. McCallum. Bow: A Toolkit for Statistical Language Modeling, Text Retrieval, Classification and Clustering. *http://www.cs.cmu.edu/ mccallum/bow*, 1996.

[38] N. Mishra, R. Schreiber, I. Stanton, R. E. Tarjan, Finding Strongly-Knit Clusters in Social Networks, *Internet Mathematics*, 2009.

[39] M. E. J. Newman, Fast algorithm for detecting community structure in networks, *Phys. Rev.* E 69, 066133, 2004.

[40] S.J. Pan, Q. Yang. A Survey on Transfer Learning, *IEEE Transactions on Knowledge and Data Engineering*, October 2009.

[41] L. Qin, J.-X. Yu, L. Chang. Keyword search in databases: The power of RDBMS. *SIGMOD Conference*, 2009.

[42] H. Schutze, C. Silverstein, Projections for Efficient Document Clustering, *ACM SIGIR Conference*, 1992.

[43] Y. Sun, J. Han, J. Gao, Y. Yu, iTopicModel: Information Network-Integrated Topic Modeling. *ICDM Conference*, 2009.

[44] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *UAI*, pages 485–492, 2002.

[45] Y. Yang. An evaluation of statistical approaches to text categorization. *Inf. Retr.*, 1(1-2):69–90, 1999.

[46] T. Zhang, A. Popescul, and B. Dom. Linear prediction models with graph regularization for web-page categorization. In *KDD*, pages 821–826, 2006.

[47] S. Zhong. Efficient Streaming Text Clustering, *Neural Networks*, 18 (5–6), pp. 790–798, 2005.

[48] D. Zhou, J. Huang, and B. Schölkopf. Learning from labeled and unlabeled data on a directed graph. In *ICML*, pages 1036–1043, 2005.

[49] H. Wang, C. Aggarwal. A Survey of Algorithms for Keyword Search on Graph Data. appears as a chapter in *Managing and Mining Graph Data, Springer*, 2010.

[50] Y. Xu, Y. Papakonstantinou. Efficient LCA based keyword search in XML data. *EDBT Conference*, 2008.

[51] Y. Xu, Y.Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. *ACM SIGMOD Conference*, 2005.

[52] Q. Yang, D. Chen, G.-R. Xue, W. Dai, Y. Yu. Heterogeneous Transfer Learning for Image Clustering vis the Social Web. *ACL*, 2009.

[53] Y. Zhou, H. Cheng, and J. X. Yu. Graph clustering based on structural/attribute similarities. *PVLDB*, 2(1):718–729, 2009.

[54] Y. Zhu, S. J. Pan, Y. Chen, G.-R. Xue, Q. Yang, Y. Yu. Heterogeneous Transfer Learning for Image Classification. *AAAI*, 2010.