LEARNING IN RELATIONAL NETWORKS

by

Tanwistha Saha
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____	Dr. Carlotta Domeniconi, Dissertation Director
_____	Dr. Huzefa Rangwala, Dissertation Co-Director
_____	Dr. Daniel Barbara, Committee Member
_____	Dr. Igor Griva, Committee Member
_____	Dr. Sanjeev Setia, Department Chair
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering

Date: 11/13/2014

Fall Semester 2014
George Mason University
Fairfax, VA

Learning in Relational Networks

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Tanwistha Saha
Master of Science
George Mason University, USA, 2011
Bachelor of Technology
University of Kalyani, India, 2004

Director: Dr. Carlotta Domeniconi, Professor
Department of Computer Science

Fall Semester 2014
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my parents Mr. Arabinda Saha and Mrs. Aruna Saha. Without their love and encouragement this work would not have been possible.

Acknowledgments

In the past six years, I have often wondered how would the acknowledgement page of my thesis look like. I used to check theses written by fellow graduate students and used to put myself in their shoes, just to sense the feeling of joy, sorrow, gratitude - all these emotions together at the same time. But I must say, those feelings that I have imagined earlier can not replicate whatever I am feeling at this moment. Now that I have finally reached the phase when I have to draft the acknowledgement section for my own thesis, I want to look back and think because I have lots of people to thank, and I do not want to miss mentioning anyone here. Without their help and support, none of these chapters of my thesis would have ever been possible.

First, I would like to thank my advisors Dr. Carlotta Domeniconi and Dr. Huzefa Rangwala for guiding me throughout the PhD. Prof. Domeniconi has been very supportive as my academic advisor during the initial years in the program. Her expertise on certain topics in Machine Learning helped me to narrow down my research direction. Prof. Rangwala has been a true guide in all the technical projects that I have tried my hands on during this time. He always gave me lots of feedback/criticism everytime I drafted a paper or proposed an idea, which, to be honest, appeared little frustrating at times. But now in the hindsight, I realize how constructive his comments had been in making me a good researcher. I would like to thank my other two committee members Dr. Daniel Barbara and Dr. Igor Griva for their feedbacks on my research work. Dr. Barbara has always advised me to think "big" in terms of deployment of software, because without a resuable implementation an idea is nothing but a mere paper in the records. Dr. Griva has helped me to look for subtle mathematical insights in my learning models that I have skipped otherwise. While writing my thesis, I realized how valuable their inputs have been.

I want to thank all the professors with whom I have either taken courses or have worked as their Teaching Assistant in all these years. Life teaches us many things in its own small ways. I have learned lots of things from the professors at the Computer Science department of George Mason University. My heartfelt gratitude to our PhD program co-ordinator Ms. Therese Michael and TA administrator Ms. Michele Pieper for helping me out in numerous occasions which I can not even count!

I want to thank my labmates from Data Mining Lab, my friends from George Mason University, my friends from school and college days in India, and friends that I made here in USA while attending numerous conferences, internships, summer schools etc., for their continuous support and encouragement in all these years. The first thing you would need to survive in a grad school is a bunch of good friends. I am fortunate to have these friends who always stood beside me no matter what happened. I would specifically like to name Irina, Rohan, Debdipto, Anindya, Anveshi, Nikhil, Azad, Matt, Xing, Guoxian and Yazhou for their encouragement in all these years - grad school would not have been so much fun without all of you. I am fortunate to have a room mate like Allison who has been my friend and my elder sister at the same time. I can not thank her enough for numerous chit-chats,

advice, and many other small things that I can not name here, but all of which made my life much easier as a PhD student.

Last but not the least, I thank my parents and my younger sister for giving me continuous support, and for encouraging me to go after my dreams. Looking back, I have lots of loving memories with my sister who has always been my dear friend as we grew up together. I will cherish those memories for a long time in my life. I also want to thank Dr. Vivek Chatterjee, my Chemistry teacher from high school days, who inspired me to accomplish something in life. It is because of him that I gave a thought to pursue PhD after spending four years in the industry as a software engineer. A very special thanks goes to my husband, Nilanjan, who has always been by my side in good days and bad days, in my joys and sorrows. I am not good at putting words to my thoughts, but it goes without saying that he has been the main driving force behind my work for the last three years - constantly encouraging me not to give up hope when things got frustrating. His love and encouragement made me believe the age-old adage - "When the going gets tough, the tough gets going". Without him by my side, I would not have been able to complete this work. Thank you for believeing in me!

Table of Contents

List of Tables

List of Figures

Abstract

LEARNING IN RELATIONAL NETWORKS

Tanwistha Saha, PhD

George Mason University, 2014

Dissertation Director: Dr. Carlotta Domeniconi

Classification of nodes in relational networks is an important task because it involves applications in multiple areas that can impact people's lives on a daily basis. The inability to use traditional classification algorithms for classifying nodes in relational networks has encouraged researchers to develop a special class of methods, known as collective classification algorithms. During the training phase, collective classification exploits the structural information embedded in the network for jointly classifying the labels of all test nodes. Any relational model needs good samples for training in order to do better predictions on unseen test data. Hence, to do a fair evaluation of a model we should always make sure that the samples on which the model is trained, are good representation of the original dataset. However, unlike traditional machine learning on non-relational data where randomly selected samples are considered good enough for training a model, relational learning relies heavily on the method of sample selection. This is because, in relational learning information propagates from the training samples to the test samples through the link structure. Hence, a sampling method that is specifically tailored for evaluating collective classification algorithms is required. A remotely related concept to sampling is the process of acquiring informative labeled data for training. Labeled data comes with a cost because it involves human interaction. In order to minimize this cost, numerous active learning algorithms

have been proposed by researchers. Although active learning methods have evolved over the years, not much had been done to deal with relational networks which are very common representation of many real-world datasets.

Relational network analysis can also be applied to improve the recommendation system framework. In traditional setting recommender system learns a model based on past ratings of users in the user-item rating matrix, and predicts the items to be rated by target users in the future. The most popular recommender system models are either neighborhood-based or latent factor based, both of which have witnessed a lot of advancements over the past few decades. Although the idea of incorporating concepts from relational classification into recommender systems seems far-fetched, in my thesis I have shown that it is quite feasible and effective in improving overall performance of the system.

There are few research gaps in the areas related to collective classification, which I have aimed to address in my thesis. To begin with, there have been a few models for multi-label collective classification, but they treat all the neighbors of a target node equally during label prediction. This is unfair because labels from influential training nodes are bound to have more effect on the target nodes, in comparison with other neighboring nodes. Sampling algorithms in networks are often aimed towards collecting samples that inherit the key characteristics of the original network. However, which feature is the "key" depends on the task we are trying to solve. Hence, instead of developing a generalized sampling method, we should propose an algorithm that can propagate label information from training nodes to test nodes more effectively, thereby improving classification performance. Active learning for network datasets studied by researchers mostly involves single-labeled networks, without any insights on how those methods can be extended to deal multi-labeled networks. Additionally, querying multiple labels of a node in multi-labeled network involves more cost. Finally, the use of short texts or tags in recommender systems has been well-explored except that, researchers often assumed the availability of these tags during training a model. In my thesis, I have predicted tags for users as "preferences" and tags for items as "descriptors" using collective classification in order to improve the overall performance of the system. In the past, there has been no work on incorporating concepts from relational learning into

this paradigm to improve item recommendations. I aimed to bridge this gap by integrating predicted tags into state-of-the-art recommender systems.

Although a lot of research have been pursued in different directions to address all these previously discussed concerns individually, in my thesis I have tried to come up with a unified approach. I have developed a rank based influential neighbor selection method for collective classification in multi-labeled networks. This method ensures that all the labeled neighbors are not treated equally while assigning labels to target nodes. To address the sampling issue, I have developed an approach tailored for improving collective classification in single-labeled networks. The active learning algorithms proposed in my thesis, work for both single- and multi-labeled networks that do not have node features accessible during training. This is an important property of the algorithm because sample features are often not accessible in network due to privacy issues. I have also developed algorithms that use cost-per-label concept for querying labels from multi-labeled nodes during active learning. Going further, I have used collective classification of tags for user preference prediction and item descriptor prediction in recommender system. Often the number of users and items with known tags are very few. In such situations, I have successfully used my active learning models to learn classifiers for predicting multiple tags for both the users and the items in the system. I have tested all the models on several real world networks and the extensive experimental results show statistically significant improvements over multiple state-of-the-art baseline methods.

Chapter 1: Introduction

1.1 Motivation

Relational Network Classification is a well-researched topic since the past few decades due to the massive rise in the development and use of web services that allow users to share different forms of data like, articles (web-blogs), pictures (Flickr.com), videos (Youtube.com), social life (Facebook), and status updates (Twitter.com). The data embed interaction patterns between individuals taking part in different activities on the internet. Virtual friends on the Facebook network tend to influence each other in different areas of life. Hence, it would be useful to track the effect of this influence among friends. Individuals in social network usually have a variety of interests. Classifying these interests into groups and labeling those, can be accomplished using the individuals' social network profile information and their interactions with friends. This classification task can be profitable for advertising agencies, in a sense that they can use this class information to recommend new products that are specific to a class. Hence, product recommendation systems in a social network can be facilitated with a model that can provide product information specific to groups in which an invidual belong. This gives a whole new perspective of classifying individuals into groups in a social network.

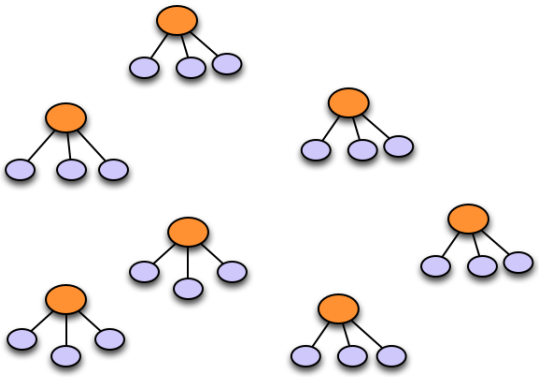
Traditional classification deals with datasets which do not have link structure between the samples. Hence, classification algorithms that perform well on non-relational datasets can not be directly applied to classify nodes in network. Figure 1.1(a)-(b) gives an idea how these two types of data look like. Each sample in the dataset has three features (colored in orange). These are referred as *node features* or *node attributes* when mentioning relational data. Figure 1.1(c)-(d) gives an idea how training nodes and test nodes are present in the dataset. In non-relational datasets it is safe to assume that test nodes are independent of

the properties of the training nodes. However, this is not the case with relational datasets because certain test nodes can be linked to training nodes. If we employ traditional classification algorithms to classify the test nodes in relational datasets, we are bound to lose this additional link information, which in turn will yield poor classification performance. Due to this reason, researchers have come up with *collective classification* algorithms that *jointly* classifies *all* the test nodes in the network to maximize the likelihood of the data. Although a significant amount of work has been done on single-labeled networks, not much has been done for multi-labeled network classification.

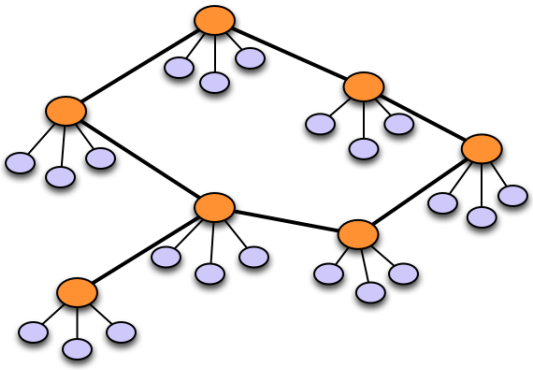
Multi-label classification in non-relational datasets has been a popular research direction these days, although little has been done to deal with multi-labeled networks. To add, most of the methods that work with networks, assume the accessibility of node features to the learning model. However, given the problem domain (e.g., social networks), the node features are often not accessible due to privacy concerns. Hence, the need developed to come up with classification models that can work using only the relational structure of the data, which is a challenging work.

Collective classification algorithms rely heavily on the labels of training nodes in order to efficiently estimate the labels of test nodes. If a test node is linked with multiple training nodes then the effect of all the training nodes (neighboring nodes) on that test node is treated equivalently during the learning process. On the other hand, there might be a subset of training nodes which are more *influential* in deciding the label(s) of the test node. Measuring influence of a node has been an important aspect of viral marketing, where the focus is to target those individuals for product advertisement who have maximum influence in the network. But to the best of my knowledge, no work has been done to use influence based scoring mechanism for improving collective classification models.

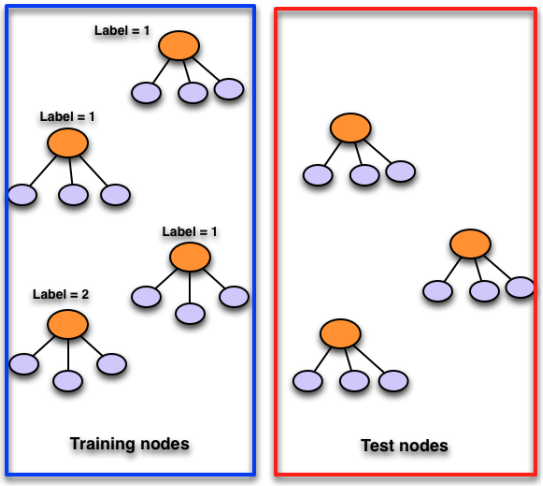
Another important question in evaluating a collective classifier is: how to select samples that are good representation of the overall network? Since the essence of collective classification is to propagate label information from training nodes to test nodes in a network dataset, the samples used to train a classifier play a very important role in inference. In



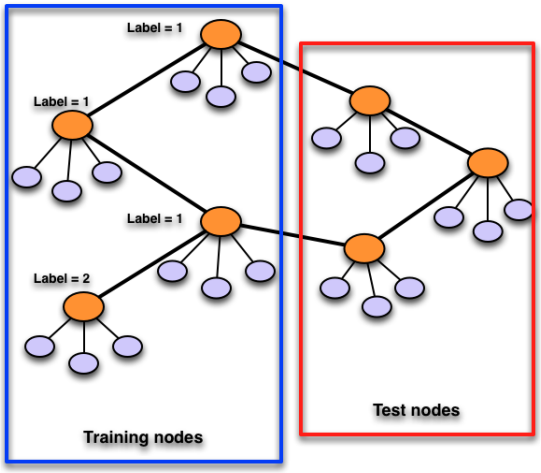
(a) Non-relational data



(b) Relational data



(c) Training - Testing split in non-Relational data



(d) Training - Testing split in Relational data

Figure 1.1: Non-relational and Relational datasets

traditional learning theory, randomly chosen samples are assumed to represent the key characteristics of the original dataset. However, in network analysis, researchers have shown that random samples are not good representatives of the network. Many competitive approaches, e.g., forest fire sampling, node sampling, edge sampling and edge sampling with graph induction have been proposed. Most of these methods have either taken a generalized approach towards sampling, or have considered networks that evolve over time.

The first step towards building a classification model is to acquire a representative training set. However, acquiring training samples can be expensive due to the cost of querying labels through interactions with a human or *oracle*. Active learning aims to learn a model with minimal querying cost and can also prioritize the acquisition of labeled samples under budget constraints. Even though active learning approaches for single-labeled datasets have been extensively studied, algorithms for multi-labeled datasets have not yet been explored. The task becomes even more challenging for multi-labeled networks because traditional active learning strategies do not take into account the explicit relational information.

In the beginning of this chapter I have discussed how classifying individuals in a network can eventually help to improve the recommendation systems. Collective classification of users in social networks within online product recommender systems is a major application area for relational learning. In traditional recommendation system the aim is to predict ratings of new items for an user based on the items he or she has rated in the past. Recommender systems work mostly with the user-item rating matrix. However, if relational network between the users is present in the system, then the contemporary algorithms often fail to make use of that additional information. Also, not much work has been done to build recommender system models when there are *tags* available as side information. These tags can be considered as meta-labels which a trained relational classifier can predict, given an implicit (or explicit) network structure between the users or items in the system. To the best of my knowledge, relational learning has been used earlier on tag recommendation systems, but not on product recommendation systems. Using tags for item recommendation is expected to improve the overall performance of the recommendation system. In my thesis,

I have aimed to bridge this gap between the application of relational learning and collaborative filtering recommendation system, by developing a tag-based item recommendation model.

1.2 Problem Statement

This dissertation is focussed on narrowing the gaps in the four directions related to collective classification that has been discussed in the previous section. Specifically, I have designed models that can improve multi-label collective classification for relational networks under different problem scenarios (influential neighbor selection, sampling from the network for evaluating the classifier, acquiring informative training samples for budgeted pool-based active learning) and proposed to use multi-label collective classification in order to predict multiple tags for users as well as items. These tags have been used to improve the overall performance of state-of-the-art recommendation systems.

1.3 Contributions

To solve the problem stated in previous section for influential neighbor selection, I have defined a metric (*rank of a node*) that can give an estimate of static influence score for the training nodes, and select nodes based on higher scores. These influential training nodes are further used for collective classification model for multi-labeled networks.

To address the sampling problem for collective classification, I have deviated from the standard approach and have developed a method to sample key nodes for training a collective classification model in single-labeled network datasets.

The previous two paragraphs states my work on two different paradigms: (*i*) multi-label collective classification using node centric influence measure for selecting neighbors for relational feature computation; (*ii*) sampling for collective classification in single-labeled network. In order to bridge the gap between these two areas of work, I have focused onto

exploring active learning methods for network data which do not have node features accessible to the learning model. I have developed active learning strategies using the collective classification paradigm that can work both on single-labeled and on multi-labeled networks under restricted budget conditions.

Using tags as side information in recommender systems is a comparatively new area of research. I have developed a model that uses collective classification on user-user network (implicit or explicit) and item-item network (implicit) to predict user preferences and item descriptors in the form of tags. These tags can be combined into a collaborative filtering recommender system model to improve the overall performance. To summarize, the contributions in my thesis are as follows:

1. A rank based influence measure for selecting neighbors in multi-label collective classification of relational networks (ICML_Rank).

2. An adaptive forest fire sampling (AFS) to sample from a single-labeled relational network for evaluating collective classification algorithm.

3. A novel active learning strategy for single labeled and multi-labeled networks (FLIP).

4. A novel active learning strategy for querying a subset of labels of an instance for multi-labeled network (FLIP-per-label).

5. A novel method for exploiting implicit link structure between users and between items through collective classification for predicting tags that can be used to improve recommendation system's performance (User-Item-Tag-KNN and User-Item-Tag-FM).

The remaining chapters of this dissertation are organized as follows: Chapter 2 discusses background of collective classification and technical notations used throughout the thesis, Chapter 3 discusses related work that have been done so far in all the directions relevant to the models I have developed, Chapter 4 introduces my model for multi-label collective classification using influential neighbors, Chapter 5 introduces the sampling algorithm I have

developed to improve collective classification in single-labeled networks, Chapter 6 introduces the active learning paradigms for single-labeled and multi-labeled networks, Chapter 7 discusses the use of multi-label collective classification model for predicting tags and using those in recommendation system. Finally, Chapter 8 summarizes the key points of this dissertation with some ideas for future work.

Chapter 2: Background

In this chapter, I will introduce the background for solving the problems which I have addressed in my thesis, with some notations that will be used throughout the rest of the document.

2.1 Collective Classification

In machine learning, classification refers to the task of identifying the category to which a new observation belongs. Classification problems for different types of datasets have been studied for many decades. In earlier years, the research methodology was to simulate real-world data, and thereafter, build a model that can represent this data. As the complexity of the data kept increasing, the need for developing sophisticated machine learning algorithms also continued to rise. *Relational Learning* is a sub-branch of machine learning which studies data that has some sort of relational characteristics in it. In recent years, relational learning has gained popularity because of the ability to represent many complex real world datasets as a graph representing the interaction patterns between the instances. Social networks of individuals, protein-protein interaction networks in biological domain and citation networks of scientific articles are only a few examples of this representation. For all these networks, the complicated relationship between the entities are not explicit from the structural properties. Relational learning is an active research topic which involves mining this latent relational information from the data. One of the objectives of relational learning is to efficiently and accurately classify nodes in a network by using the latent information in the link structure, and thus arises the need of sophisticated classification algorithms. Learning these interactions can unveil plethora of rich information which are otherwise hidden. This gives a whole new perspective to classification of nodes (or entities) in networks which is

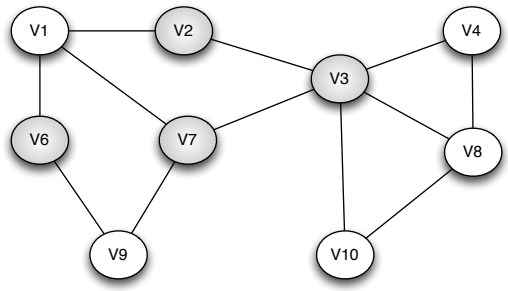


Figure 2.1: Collective Classification of a network $G = (V, E)$; shaded nodes represent observed variables.

accomplished through the development of a special class of algorithms known as *Collective Classification*.

Collective classification is defined as a class of algorithms that can learn a model from the characteristic properties as well as topological properties of nodes linked to each other. Given a network, three possible correlations are relevant while determining the labels of nodes [53]: (i) the correlation between the label of a node and the observed attributes of the node, (ii) the correlation between the label of a node and the observed attributes and labels of neighboring nodes, and (iii) the correlation between a known label of a node and the unobserved labels of neighboring nodes. Collective classification methods jointly classify a set of related (linked) nodes by exploiting the above underlying correlations [20]. Let us consider a network G in Figure 2.1. It represents a network of 10 nodes $V = (v_1, v_2, \cdots, v_{10})$ having $E = (E_1, E_2, \cdots, E_{12})$ edges amongst them. For the shaded (training) nodes v_2, v_3, v_6 and v_7, we know the classes (labels) to which they belong. Hence, the task is to predict the class assignments of the unshaded (test) nodes. In addition to class labels, sometimes we also have node features $\mathbf{X}$ (i.e., characteristic properties) for the training nodes, which are used while building the classification model. Given all these information, we can define *collective classification* as a combinatorial optimization problem on a given graph $G = (V, E, \mathbf{X}, \mathbf{Y}, L, K)$, such that, $V = \{v_1, v_2, v_3, \cdots, v_n\}$ is the set of nodes, E is the set of edges, $\mathbf{X}$ is the node feature matrix and each $\mathbf{x}_i \in \mathbf{X}$ is the feature

vector for node $v_i \in V$,($|V| = n$) and L is the set of training nodes and K is the total number labels (or classes). The label(s) of any node v_i in the graph is represented as a vector $\mathbf{y}_i = (y_{i1}, \cdots, y_{iK})$ where $y_{ik} = 1$ if node v_i belongs to the class k ($k \in \{1, 2, \cdots, K\}$). For single-labeled network, a node can belong to only one class. Hence, only one element, say y_{ik}, of the label vector $\mathbf{y}_i$ can have value 1 (if v_i belongs to the class k) and the rest of the values in $\mathbf{y}_i$ are 0. Given the set of labeled nodes L and the set of unlabeled nodes U, the objective of *collective classification* is to predict the labels of *all* the unlabeled nodes [53]. $\mathbf{Y} = \{\mathbf{y}_1, \cdots, \mathbf{y}_n\}$ represents the set of label vectors of all the nodes in network G. l and u represent the indices of a labeled node and an unlabeled node respectively, such that $l \in L$ and $u \in U$. $\mathbf{Y}_L$ is the label matrix for training data, where each row is the label vector $\mathbf{y}_l$, $\forall l \in L$. $\mathbf{Y}_U$ is the predicted output label matrix of the unlabeled data obtained through collective classification, where each row is the predicted label vector $\mathbf{y}_u$, $\forall u \in U$.

In collective classification, the main challenge is to leverage the information from the link between training samples and test samples in the network. Most of the research in collective classification has been centered around single-labeled relational networks [53] which has node attributes (features) available to the learner during training. These approaches are either iterative in nature (local neighborhood based models), or they try to optimize a global objective function (global methods). Since my thesis is based on the local methods, I will discuss it in details. Overview of global methods will be given in Chapter 3.

Local neighborhood based methods start off with training a traditional binary classifier (e.g., logictic regression, Support Vector Machine [62]) using the node attributes of the training samples. In the first phase, this trained classifier is used to predict the initial labels for test samples. In the subsequent phases, the *link attributes* or *relational attributes* for each of the nodes (train and test) are derived using either one of the following aggregators: (*i*) mode-link, which computes the mode of the classes to which the linked neighboring nodes belong; (*ii*) count-link, which computes the frequency of the classes of the linked neighboring nodes; (*iii*) binary-link, which is a middle ground between the previous two options where 1 or 0 indicates if a particular class is present or absent among the linked

neighbors. The classifier is further retrained, and label predictions are made for test nodes. This process continues iteratively until convergence.

Following the notations by Lu and Getoor [32], let $\mathbf{x}_i$ respresents the concatenated node attributes and relational attributes (derived using one of the previously mentioned aggregators) for a node $v_i \in V$. For a binary classification problem, we can represent the conditional probability distribution $P(y_{ik} = 1|\mathbf{x}_i)$, as a logistic regression where $y_{ik} = 1$ if node v_i belongs to the positive class, and -1 otherwise ($k \in \{1, 2\}$). Let $\mathbf{w}$ be the weights (or parameters) associated with the attribute variables $\mathbf{x}_i$. Given the labeled training sample $(\mathbf{x}_i, \mathbf{y}_i)$, we can train a binary logistic regression model [5,13] using the following equation:

$$P(y_{ik} = 1|\mathbf{x}_i, \mathbf{w}) = \frac{1}{\exp(-\mathbf{w}^T \mathbf{x}_i y_{ik}) + 1} \tag{2.1}$$

With a set of training samples $(\mathbf{x}_i, \mathbf{y}_i)$ where $i = 1, 2, \cdots, n$, the optimal $\mathbf{w}$ for this discriminative function can be obtained by minimizing the following regularized logistic regression problem:

$$\underset{\mathbf{w}}{\operatorname{arginf}} \frac{1}{n} \sum_{i=1}^{n} \ln(1 + \exp(-\mathbf{w}^T \mathbf{x}_i y_{ik})) + \lambda \mathbf{w}^T \mathbf{w} \tag{2.2}$$

where λ is the regularization co-efficient. Using the optimal $\mathbf{w}$ and the logistic regression, we can predict the initial labels for all the test nodes and compute the relational attributes to iteratively recompute the labels of test nodes again. These steps continue until maximun number of iterations has been reached or the labels of the test nodes have stopped changing across consecutive iterations. The basic iterative classification algorithm (ICA) as proposed by Neville and Jensen [38] is described in Algorithm 1.

In many real world situations, node attributes of samples in relational network are often not available for training due to privacy issues. The ICA algorithm and its other sophisticated improvements [38, 39, 52] worked on relational datasets which have node attributes of samples during training, until Macskassey [33] proposed the Weighted Vote Relational

Algorithm 1 `Iterative Classification Algorithm (ICA)`

```
Use training samples to build a model classifier.
Apply trained classifier to test samples.
for each iteration i = 1 to maxiter do
    Calculate dynamic relational attributes for training and testing samples.
    Use model to predict class labels for test samples.
    Sort the probability scores for the classes for each test sample.
    Assign the class label which has maximum probability, to the test sample.
end for
Output final predictions of the model classifier made on the test samples.
```

Neighbor (wvRN) classifier that can perform using only the topological structure of the network. Given $v_i \in U$, the wvRN classifier estimates the probability of node v_i having class label k (i.e., $y_{ik} = 1$) as a weighted average of the labels of it's neighboring nodes.

$$P(y_{ik} = 1|N_i) = \frac{1}{Z} \sum_{v_j \in N_i} w_{ij} \cdot P(y_{jk} = 1|N_j). \tag{2.3}$$

N_i is the set of neighbors of v_i in G, Z is the normalization constant ($Z = \sum_{k=1}^{K} P(y_{ik}|N_i)$) and w_{ij} is the weight on the edge, e_{ij}. In the collective classification algorithm, the *bootstrap* phase assigns a class probability to all the test nodes $v_i \in U$ by estimating the class prior probability as:

$$P(y_{ik} = 1) = \frac{1}{|L|} \sum_{j \in L} \mathbb{I}(y_{jk} = 1), \tag{2.4}$$

where $\mathbb{I}(\cdot)$ is an indicator function with value 1 if the arguement is true.

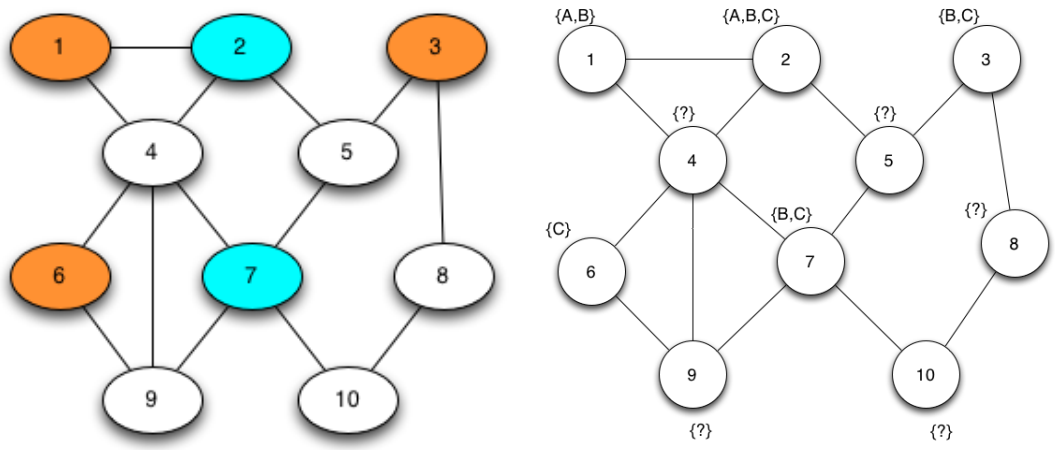
Relaxation labeling is a collective inference method based on the approach by Chakrabarti *et al.* [7]. During each iteration of collective inference, relaxation labeling [7,33] is used to update the prediction probability estimates from previous iterations. At step $t+1$ the predicted labels of *all* the test nodes are updated based on their estimation at step t. The update rule is given by [33, 63]:

$$\mathbf{P}_i^{(t+1)} = \beta^{(t+1)} \cdot \mathcal{M}_{\mathcal{R}}(v_i^{(t)}) + (1 - \beta^{(t+1)}) \cdot \mathbf{P}_i^{(t)} \tag{2.5}$$

A simulated annealing based technique [33] is used to reduce the influence of neighbors by giving more weight to a node's current estimate. In Equation (2.5), $\mathcal{M}_{\mathcal{R}}(\cdot)$ is the relational model (here, wvRN), $\beta^1 = \gamma$ and $\beta^{(t+1)} = \beta^t \cdot \alpha$, where γ and α are constants in the range $(0, 1]$, both γ and α values are chosen closer to 1, $\mathbf{P}_i^{(t)}$ is a vector with class probability values for node v_i at step t.

Besides the recently proposed work of Kong *et al.* [24], not much has been done towards the development of multi-labeled collective classification algorithms. Multi-label classification focuses on the prediction of multiple labels for each test instance simultaneously. For example, consider Figure 2.2 where 2.2(a) shows a single-labeled network in which nodes can belong to either one of the two classes (orange and blue classes), and 2.2(b) shows a multi-labeled network where each node can belong to one or more of the 3 possible classes (A,B or C classes). As per the notations, each label of a node v_i in a multi-labeled network is given by $y_{ik} = 1 \ \forall k \in M$, where M is the set of classes ($M = 3$ for Figure 2.2(b)) to which this node belongs ($|M| \leq K$). In case of multi-label collective classification when the labels are correlated and there is explicit link structure between the instances of the dataset, it becomes challenging to leverage these multiple correlations during training [67]. In my thesis, I have incorporated wvRN's approach into a one-vs-rest algorithm to come up with a multi-label weighted vote relational neighbor (ML-wvRN) classifier. Relaxation labeling has been used for the collective inference, and probability for a test node to belong to K different classes is estimated. Only those classes are selected that have higher probabilities (the number of selected classes is equal to $|M|$ where $|M| \leq K$) and the corresponding labels are assigned to the test node. A similar approach was taken by Tang *et al.* [58] in multi-label classification. This classifier is referred to as ML-wvRN-RL in my thesis.

Collective classification is a semi-supervised algorithm which relies on the labels of the neighboring nodes in order to predict the label of a test node. However, in most of the methods proposed for both single-labeled and multi-labeled collective classification (including the state-of-the-art algorithms), all the neighbors of a test node are treated equivalently.



(a) Single-labeled network with 2 classes (b) Multi-labeled network with 3 classes

Figure 2.2: Single-labeled and Multi-labeled Networks.

In reality, some neighbors are more *influential* than the others. I proposed a method to exploit the information hidden in the interactions between entities by using properties of graphs. In addition to capturing information from the topology of the graph, this method also captures the *association* between a node and its label. It *prunes* the neighborhood of a test node based on the "*ranks*" of the training nodes, so that, while predicting the label of the test node, we can only trust the label information propagated from the *influential* neighbors. Details of this approach will be discussed in Chapter 4.

2.2 Sampling for Collective Classification

Collective classification has been a well researched topic for years, but very little has been done to actively select the labeled nodes from the network to be used for training. Recently, Ahmed et al. [2] have shown that, forest fire sampling has an overall good performance for classification in relational data. The goal of any sampling method is to preserve key features (e.g., sufficient statistics) of the underlying distribution. Which feature is a *key*, depends on the problem we are trying to solve. Hence, instead of developing another generalized

sampling algorithm for any learning method (supervised, semi-supervised or unsupervised), I have developed [46] an adaptive version of forest fire sampling in order to facilitate collective classification in network datasets. This is different from the *active sampling* method proposed recently [40], where the goal is to draw only those samples from the network that have high probability of having a specific label. In comparison to all these methods, the proposed algorithm can be categorized as *active labeling*. It is similar to a random walk based approach, where the selection process is recursive and it halts when no new nodes are selected, or when the required sample size has been reached. Since collective classification propagates the label information from the training nodes to the test nodes in a network during the inference process, it would be helpful if a test node is connected to at least one training node in order to use the training node's characteristics during the training phase of the classifier. Such a sampling algorithm is an adaptive version of forest fire sampling and is proposed in our work [46].

The goal of Forest fire sampling proposed by Leskovec *et al.* [29] is to obtain a subset of nodes from a large graph using a randomized procedure such that the selected subset of samples will obey the heavy-tailed distribution for in- and out-degree, the Densification Power law and the shrinking diameter. The basic forest fire sampling is described in Algorithm 2. Two parameters - *forward burning probability* p and *backward burning ratio* r, are needed to start with. Forest fire sampling is very good sampling algorithm for extracting

Algorithm 2 `Forest Fire Sampling`

Input: p: forward burning probability; r: backward burning ratio; G_t: Graph constructed at time $t = 1$ with 1 node

1: Let G_t be the graph constructed thus far.
2: Consider a node v joining the network at time $t > 1$. Node v will form outlinks to nodes in G_t according to the following:
3: Select a node w uniformly at random and form a link to w.
4: Generate 2 random numbers from geometric distributions with mean $(1-p)^{-1}$ and $(1-rp)^{-1}$ respectively.
5: Node v selects x outlinks and y inlinks that were not yet visited. Let $w_1, w_2, \cdots, w_{x+y}$ denote the other ends of these links.
6: Node v links out to $w_1, w_2, \cdots, w_x$ and then applies step 4 recursively to $w_1, w_2, \cdots, w_x$ without creating any cycle (i.e., by not revisiting a node).

samples from large graphs but it lacks the motivation for using it in collective classification

algorithms. In collective classification, the links from test nodes to the training nodes (nodes obtained through sampling) are important for label propagation, but no such constraint has been enforced in the forest fire algorithms.

Let graph $G = (V, E)$ represent a single-labeled relational dataset $D = (\mathbf{X}, \mathbf{Y})$, where $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$ is the set of node attributes and $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$ is the set of corresponding labels for the $|V| = n$ nodes in the network. Let K be the number of classes, denoted by $k = 1, 2, \cdots, K$, to which the nodes can belong, i.e., ($y_{ik} = 1$, where $i = 1, 2, \cdots, n$) and if v_i belonged to class k. Let $\mathbf{Q}_{|E|\times 2}$ be an edge-vertex incidence matrix where row denotes an edge $e \in E$. The column of $\mathbf{Q}$ indicates the two vertices with which the edge e is associated. Let $\mathbf{A}_{n\times n}$ be the weighted adjacency matrix representing the interactions between any pair of nodes in the network. The problem definition is to find a sample subgraph $G'_s = (V'_s, E'_s)$ from the network $G = (V, E)$ (where $V'_s \subset V$ and $E'_s \subset E$) using a sampling algorithm, in order to train a collective classifier. Chapter 5 will give the detailed overview of the algorithm followed by the results.

2.3 Active Learning for Relational Network Datasets

Active Learning is a sub-field of machine learning which deals with learning models under restricted budget conditions. The main hypothesis of active learning is that, if a learner is allowed to choose the training samples, then it performs better with fewer training samples. For any supervised learning algorithm to perform well, it requires considerable amount of training samples. Acquiring these training examples involve human interaction or querying *oracle* that comes with a cost. Often the total cost of all the queries is restricted by a fixed *budget*. Obtaining labeled data is expensive, but unlabeled data is abundant and can be exploited at will. Active learner aims to utilize the budget as efficiently as possible by selecting *informative* samples from unlabeled data and *querying* their labels from *oracle*, that the learner thinks are harder to classify, and hence, more helpful in training. The normal setup starts off with training a weak learner using few training examples. In subsequent

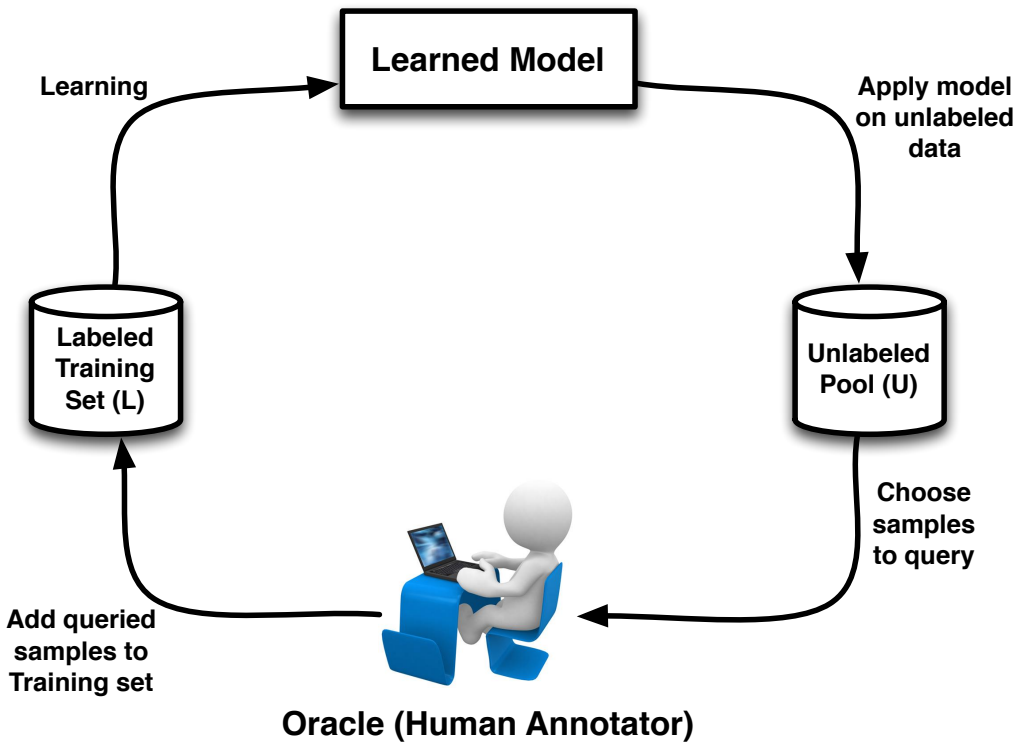


Figure 2.3: Pool-based Active Learning.

rounds, samples from an *unlabeled pool* are queried and added to the set of training samples. Figure 2.3 (similar to the one described in the survey by Settles [54]) gives an overview of a pool-based active learning system and Algorithm 3 details out the steps. Most of the active learning strategy differs in formulating the query strategy. The most primitive technique is the *uncertainty sampling* [30]. It selects those samples for which the classifier is most uncertain about the class conditional probability:

$$S[i] = 1 - \underset{k}{\operatorname{argmax}} P(y_{ik} = 1 | \mathbf{x}_i, \mathbf{w}) \tag{2.6}$$

where $S[i]$ is the *utility* score for sample x_i (or node v_i in a network dataset). The more uncertain the classifier is about the class of the sample, the lower will be the $P(y_{ik} = 1|\mathbf{x}_i, \mathbf{w})$ value and the higher will be the utility score $S[i]$. So, if we select either one sample (for

Algorithm 3 `Pool-based Active Learning`

Input: L: a labeled set of samples , U: and unlabeled pool of samples , $\mathcal{F}$: a learner, an oracle, B: total number of queries or *budget* , b: number of queries to the oracle at each step

Output: an improved learner $\mathcal{F}$

1: Create a learner $\mathcal{F}$ with labeled training samples L.
2: **while** $|L| < B$ **do**
3: $\mathbf{P}_U \leftarrow \mathcal{F}(U)$.
4: Compute S for each sample $i \in U$ according to Equation (2.6).
5: Select b samples which have top S values.
6: Add b samples to the training set L
7: Retrain the learner with new training set L.
8: $|L| \leftarrow |L| + b$
9: **end while**

stream-based active learning) or a *batch* of samples (for batch-mode active learning) according to the highest utility scores then those samples are deemed to be more informative than the other samples in the pool. Most of the active learning strategies that have been proposed in the past three decades followed this framework by Lewis and Gale [30]. Settles provides a comprehensive survey [54] on active learning for traditional classification settings, all of which are undoubtedly very good for non-relational datasets. However, the additional link information present in the relational datasets demands more sophisticated approaches than the ones which have already worked wonders for non-relational datasets. Active learning aims to learn a model with minimal querying cost and can also prioritize the acquisition of labeled samples under budget constraints. Previously, different pool-based active learning strategies have been developed for selecting the most informative sample(s) to improve the generalization performance of a classifier. Even though active learning approaches for single-labeled datasets have been extensively studied, algorithms for multi-labeled datasets have not yet been explored. The task becomes even more challenging for multi-labeled networks because traditional active learning strategies do not take into account the explicit relational information. In my thesis I have built upon the work proposed by Bilgic *et al.* [4] for single-labeled networks. But unlike them, I have tried to come up with a prototype that works for both single- and multi-labeled networks [47].

Following the terminology discussed in Section 2.1, let graph $G = (V, E, \mathbf{X}, \mathbf{Y}, L, K)$

be the network in question and L and U be the set of labeled and unlabeled nodes in the network, respectively. For active learning setup, we denote $\mathbf{P}_U$ as the class probability matrix of all the unlabeled nodes such that each row of $\mathbf{P}_U$ is a vector $\mathbf{p}_u = (p_{u1}, \cdots, p_{uK})$ and denote $\mathbf{P}_L$ as the class probability matrix of all the labeled nodes in training set L, such that $\mathbf{P}_L = \mathbf{Y}_L$. p_{uk} is equal to $P(y_{ik} = 1|N_i)$, where $i = u$ and N_i is the set of neighboring nodes of node v_i. PO represents the set of indices of all the nodes in unlabeled pool. The class conditional probability $\mathbf{P}$ of all the labeled and unlabeled nodes is given by, $\mathbf{P} = [\mathbf{P}_L; \mathbf{P}_U]$, where $\mathbf{P}_U$ is predicted by active learner.

The previous two sections 2.1 and 2.2, discussed the research on two different paradigms: (i) multi-label collective classification using node centric influence measure for selecting neighbors for relational feature computation; (ii) sampling (i.e., active labeling) for collective classification in single-labeled network. In order to bridge the gap between these two areas of work, my research focused on exploring active learning methods for networked data. On one hand, multi-label collective classifiers that are in practice (and the one used in my thesis [45]), use one-vs-rest approach which is computationally very inefficient when the number of labels is very high (≥ 10). On the other hand, active labeling or sampling from network [46], is very similar to situations that arise during active learning when number of samples to be acquired for training is restricted by a fixed budget. In order to bridge the gap between these two paradigms, I have developed an active learning strategy for a multi-label collective classification algorithm that can select informative samples under restricted budget, even for network datasets which do not have node attributes of samples available to the learner during training. There had been some work done on active learning in networks which we will go through briefly in Chapter 3 as a precursor to my models which will be discussed in Chapter 6.

2.4 Tag-based Recommendation Systems

Collaborative Filtering (CF) is a method applied to user-item rating matrix for predicting new items for users, using their past rating history. State-of-the-art collaborative filtering methods, as mentioned in the comprehensive survey by Su and Khoshgoftaar [57], can be categorized into two types: (i) neighborhood-based models; (ii) latent factor based models. Neighborhood-based models can be of two types: (a) User-based; (b) Item-based. These approaches recommend products to a target customer either using his/her similarity with other customers (user-based) or using the similarity with the items he/she has rated in the past (item-based). Let us consider two users x and y in the system where there are I items. Similarity measures between these two users can be computed using the following two metrics:

- Correlation: Similarity between two users x and y is measured by the *Pearson correlation coefficient* which is computed as:

$$corr_{x,y} = \frac{\sum_i (R_{xi} - \bar{R_x})(R_{yi} - \bar{R_y})}{\sqrt{\sum_i (R_{xi} - \bar{R_x})^2 \sum_i (R_{yi} - \bar{R_y})^2}} \tag{2.7}$$

  where R_{xi} gives the rating by user x for item $i \in I$ and R_{yi} gives the rating by user y for item $i \in I$ - both of which are obtained from the $|U| \times |I|$ user-item rating matrix R (U being the total number of users in the system).

- Cosine: In this case x and y are considered as two vectors in the $|I|$ dimensional product space. The similarity is computed as the cosine of the angle between the two vectors:

$$cos(\overrightarrow{x}, \overrightarrow{y}) = \frac{\overrightarrow{x} \cdot \overrightarrow{y}}{\|\overrightarrow{x}\|_2 * \|\overrightarrow{y}\|_2} \tag{2.8}$$

  where "$\cdot$" denotes the scalar product between two vectors.

The final step of a CF-based recommendation system using neighborhood based approach is to derive the *top-N* recommendations from the neighborhood of users. This can be done in the following two ways:

- *User-based top-N recommendation*: This type of approaches identify K most similar users (K nearest neighbors) to the target user, using either Pearson correlation or cosine-based similarity metric. After K most similar users are identified, the items they have rated are aggregated from the user-item rating matrix R along with their purchase frequency in this group of users. From this aggregated set of items, user-based CF technique then selects *top-N* most frequent items and recommends it to the target user.

- *Item-based top-N recommendation*: This algorithm first identifies K most similar items to the items already rated by target user, using the similarity metrics. Then it creates the set I_{sel} by removing the items already rated by the user earlier. It then computes the similarity of each item in I_{sel} to the items that has been rated before by target user. Based on the similarity scores, *top-N* items from the I_{sel} are finally recommended to the target user.

The other prominent CF techniques in the post-Netflix prize competition era are the latent factor models involving matrix factorization based methods. These models are based on Singular Value Decomposition (SVD) type approaches which decomposes the user preferences and item characteristics into a latent subspace. For example, for $|U|$ users and $|I|$ items, the user-item rating matrix R is given as the product of $|U| \times D$ user coefficient matrix U^T and $D \times |I|$ factor matrix V. SVD finds the low rank matrix $\hat{R} = U^T V$ which minimizes the sum squared distance to the target matrix R. Even though it looks like a minor modification, but it results in a non-convex optimization problem which is difficult to solve, especially when R is very sparse. To address this problem, matrix facorization methods came into prominence during the Netflix grand prize competition. Minh and Salakhutdinov [37] proposed a probabilistic extension of SVD (probabilistic matrix factorization or PMF) which

models the user-item rating matrix as a product of two low rank user matrix and item matrix. The sum squared distance objective function for PMF is as follows:

$$f(U,V) = \frac{1}{2}\sum_{i=1}^{|U|}\sum_{j=1}^{|I|} I_{ij}(R_{ij} - U_i^T V_j)^2 + \frac{\lambda_U}{2}\sum_{i=1}^{|U|}\|U_i\|_{Fro}^2 + \frac{\lambda_V}{2}\sum_{j=1}^{|I|}\|V_j\|_{Fro}^2 \tag{2.9}$$

where $\lambda_U = \sigma^2/\sigma_U^2$ and $\lambda_V = \sigma^2/\sigma_V^2$ and $\|\cdot\|_{Fro}^2$ denotes Frobenius norm.

Both the neighborhood based methods and matrix factorization approaches work well for single aspect rating prediction for users who have rich information in the system. However, very little has been done to address the situation when users' tagging behaviour can be used as side information in improving the models and not all users have these tags readily available in the system. Users often associate multiple aspects while rating an item. These aspects are represented by an user as short text snippets or *tags* which usually signify what the user thinks about a particular item. For example, in Tripadvisor website, consider an user who rates a hotel and tags it as *solo* and *couple*, where the tags are representing the user's perspective about that hotel. If another user also tags the same hotel with same tags or a subset of these tags, then these two users are similar in the system. Hence, in order to make personalized recommendations, tags can facilitate finding similar users or items w.r.t. a target user or an item, respectively.

In relational learning, *collective classification* methods are quite popular in classifying nodes in network datasets. As described in Section 2.1, these methods jointly classify *all* the test nodes in a network by leveraging the complex and implicit correlations between multiple entities and their labels. They are applicable towards networks which have topological features [33, 53], but may or may not have node features [33], and can also be applied on multi-labeled networks [24, 45]. Recently, researchers have used collective classification in *tag-recommendation* systems with promising results [41]. Inspired by their approach, in my thesis I have proposed an item recommendation model by employing multi-label collective classification on the implicit user-user network and item-item network (derived from the

user-item rating matrix) to predict *tags* (or labels) for users and items. If the tags are not available for *some* of the users in the system, then we employ active learning to incrementally learn a collective classifer and predict tags for those users. Furthermore, I have used these predicted tags as *user attributes* and *item-attributes*, and have introduced a neighborhood- and a latent factor-based collaborative filtering method. Hence, my contributions can be summarized as follows [48]:

- Using *collective classification* on user and item networks to predict tags for users. For those users who do not use tags, active learning with collective classifier is used to predict tags for them.
- Using tags for users as *user attributes* and tags for items as *item attributes*, thereby, computing *tag-induced pairwise similarities* between users and between items for neighborhood based collaborative filtering model.
- Incorporating user attributes and item attributes into a latent factor model to improve the performance of item recommendation.

Latent factor models (probabilistic matrix factorization, SVD++) discussed earlier, are the state-of-the-art methods in recommender systems. But their formulation [37, 49] do not involve seamless integration of side information that are sometimes available in the system. To address this problem, Factorization Machines (FM) were proposed as a new model class of general predictors. These are similar to support vector machines, that can work with any feature vector. However, unlike SVMs, factorization machines can model all pairwise nested interactions between the variables even for very sparse input data. Rendle [43] proposed factorization machine model for a set of tuples $(\mathbf{x}, f(\mathbf{x}))$ where $\mathbf{x} \in \mathbb{R}^D$ is a feature vector and $f(\mathbf{x})$ is the corresponding target. A factorization machine that models all interactions upto order $d = 2$ between the D input variables is defined as:

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{D} w_j x_j + \sum_{j=1}^{D} \sum_{j'=j+1}^{D} x_j x_{j'} \sum_{f=1}^{F} v_{j,f} v_{j',f} \tag{2.10}$$

where F is the dimensionality of the factorization, and the model parameters $\Theta = \{w_0, w_1, \cdots, w_D,$

$v_{1,1}, \cdots, v_{D,F}\}$ are: $w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^D, V \in \mathbb{R}^{D \times F}$. The main difference between the second part of Equation (7.5) and polynomial regression, as mentioned by Rendle [43], is that the interaction is not modeled as an independent parameter $w_{j,j'}$. Rather, it is modeled as a factor $w_{j,j'} \approx \langle \mathbf{v}_j, \mathbf{v}_{j'} \rangle = \sum_{f=1}^{F} v_{j,f} v_{j',f}$.

Using the FM tool, I have integrated the tags for users and items into the model, thereby improving the performance of product recommendation. Furthermore, if tags for large number of users/items are not known beforehand, I have proposed the use of active learning on collective classification for tag prediction. Details of the experimental setup and results on several real world datasets will be shown in Chapter 7.

Chapter 3: Related Work

This chapter gives an overview of all the work that is related to my thesis.

3.1 Collective Classification

Collective classification is a semi-supervised method, first proposed almost ten years ago [16, 38]. State-of-the-art collective classification methods on single labeled datasets are described in a comprehensive survey [53]. Conventional collective classification approaches focus on classifying single labeled datasets, and can be broadly categorized into *local* and *global* methods. In local methods, a *local classifier* predicts labels for each node whose label is unknown, using either both the node features and relational features or just relational features. The local classifier used here can be logistic regression [16, 32], relational dependency networks [38, 39] or simply a weighted nearest neighbor (wvRN) [33]. McDowell *et al.* [36] proposed a *cautious* approach to exploit intermediate relational data which can be noisy. A local classifier uses node features and relational features (whenever available) to predict the labels of unobserved nodes in the network. In an iterative inference algorithm, the label information from observed (training) nodes is propagated and combined with the previously predicted labels of unobserved (test) nodes across the network. This process continues to iterate until convergence (i.e., until the labels of unobserved nodes have stopped changing or the maximum number of iterations have been reached). Examples of these types of algorithms for single labeled network include Iterative Classification Algorithm (ICA) [32, 38] and Gibbs Sampling [33, 36, 39]. Macskassy *et al.* have developed a toolkit named NetKit-SRL that combines different components, e.g., use of any local classifier with any collective inference algorithm [33]. Global methods optimize a single objective function

based on a Markov Random Field (MRF) representing the relational network dataset. Examples of this type of approaches include Loopy Belief Propagation (LBP) and Mean Field Relaxation Labeling (MF). In LBP, each random variable in the MRF sends a message to a neighboring random variable, depicting its belief of what the label of the random variable should be. In MF, each node asks its neighbors for their probability distribution over a set of all possible class labels. Sen et al. [53] and Tasker et al. [59] have discussed more on these approaches.

Collective classification has been shown to perform poorly when the amount of labeled data is sparse, to the extent that the classifiers perform better without it [20, 39]. To deal with fewer labeled instances and still aiming for good performance of the classifier, another approach [15] attempts to find features (computed from network topology only) that are independent of node labels and analyzes the classifier's performance using those label-independent features. Since, label-independent features are less sensitive to the specific label assignment of the nodes in a graph, classifiers that use these features exhibit better performances [15]. Such features, derived from network's structural properties (e.g., node and link counts, betweenness centrality), are found to be most informative for classification in network data [15]. Collective classification has been observed to work well on single labeled data, whereas, in many real world scenarios the networks are multi-labeled.

Since, the cardinality of the label sets are usually very high, the task of multi-label learning becomes challenging in network datasets. Recently some work has been focused on multi-label collective classification that trains K independent one-vs-rest collective classifiers (K is the total number of possible labels), and thereafter, combines the outcomes from all the K models in predicting the multiple classes of a test node [24]. However, this approach has three major drawbacks: (i) relational feature computation in this method does not distinguish between the influence of different labels of the neighboring nodes; (ii) this method uses Support Vector Machine [62] classifier that requires rich node features in order to make good predictions; and (iii) this method faces scalability issues while dealing with one-vs-rest approach for large number of classes (K). Another algorithm was proposed by

Wang and Sukthankar [63] that addressed the problem of multi-label collective classification using only social context features derived from the network structure. Even though their method does not require any node feature for classification, still it has some drawbacks in the context of dealing with different types of correlations between multi-labeled instances while doing inference across the network. Besides the recently proposed work of Kong *et al.* [24] and Wang and Sukthankar [63], not much has been done towards the development of multi-labeled collective classification algorithms.

3.2 Multi-label Classification

Multi-label learning focuses on the prediction of multiple labels for each test instance simultaneously. Several algorithms have been proposed in this area (see the survey in [61]). Zhang *et al.* developed a k-nearest neighbors based method for multi-label classification, which treats each label independently, and then computes the probability of assigning a particular label [66]. Another method proposed by Zhang uses *label specific* features for classifying instances [64]. Zhang *et al.* also proposed a Bayesian network based approach to capture the conditional dependencies between labels as well as the feature set [65]. A recently developed algorithm simultaneously ranked and classified objects within the DBLP citation data [22]. This algorithm was designed for a heterogeneous network consisting of different types of objects (nodes), and computes the *within-class* ranking of these objects. In my thesis, I have discussed an approach for multi-labeled collective classification within a homogeneous network. The multi-label network classification method developed in this thesis uses the word *rank* as a measure of *influence* in order to impose a pruning strategy for determining the neighboring nodes that would affect the prediction of the label of the target node. This is different from the approaches that study influence of a node within a relational network in the context of viral marketing [17].

3.3 Sampling for Collective Classification

Given the aim of collective classification in Chapter 2, an important question that comes to mind is - how to evaluate a collective classification model? In traditional learning theory, samples (set of instances) chosen randomly (with or without replacement) from the entire set of data points are assumed to represent the key characteristics of the original dataset [13]. In these cases, randomly chosen instances are good enough for learning models which does not make use of link structure in the dataset. However, in network analysis, researchers have shown that random samples are not good representatives of the original network [2]. Many competitive approaches, e.g., forest fire sampling [28], node sampling, edge sampling and edge sampling with graph induction [1] have been proposed to sample from network datasets. Most of the proposed methods have either taken a generalized approach towards sampling or have considered networks that evolve over time. Recently, Ahmed et al. [2] have shown that, forest fire sampling has an overall good performance for classification in relational data. The goal of any sampling method is to preserve key features (e.g., sufficient statistics) of the underlying distribution. Which feature is a *key* depends on the problem we are trying to solve. Hence, instead of developing another generalized sampling algorithm for any learning method (supervised, semi-supervised or unsupervised), I have proposed an adaptive version of forest fire sampling in order to evaluate the performance of collective classification in network datasets. This is different from the *active sampling* method proposed recently [40], where the goal is to draw samples from the network that have high probability of having a specific label. My work can be categorized as *active labeling*, where both the value of the label as well as the local structural information of the labeled instance are acquired.

3.4 Active Learning for Relational Nework Datasets

Previous active learning algorithms [3, 4, 9, 35] determine informative examples to query based on one or more of the following properties: (i) maximum entropy based on classifier's

prediction of its label; (*ii*) least confidence of the classifier on its label; (*iii*) maximum disagreement between multiple classifiers (e.g., ensemble) predicting its label. Zhu *et al.* [71] have deviated from this approach by combining active learning with semi-supervised learning using Gaussian field and harmonic functions, and employing Empirical Risk Minimization (ERM) framework. Macskassy proposed a method [34] that uses graph-based metrics (e.g., clustering co-efficient, betweenness centrality and degree of a node) to identify a set of informative examples and then apply ERM on that same set of examples to identify the *single most informative* instance from the pool. Most of these methods [34, 71] have the benefit of using the features of the instances during training phase. This is often a challenge for relational networks because the features may not be available for mining due to privacy concerns. In my thesis, I have focussed on developing methods that learn from the structural properties of networks.

The algorithm developed in my thesis is inspired by Zhu *et al.* and Macskassy [34, 71], but unlike their approaches which focus on matrix based methods (that rely heavily on rich instance features), my method explores the use of the collective inference procedure within the active learning strategy. Bilgic *et al.* [4] were the first to propose an active learning approach that uses the disagreement score between a content-only classifier (i.e., trained using node features) and a collective classifier. Shi *et al.* [55] proposed a batch algorithm that combines node features and link information for active sample selection strategy. The algorithm by Ji *et al.* [21] selects instances that minimize the total variance of the distribution of the unlabeled samples and the total prediction error. Kuwadekar and Neville [27] use a probabilistic relational model [39] to select informative instances. All these methods rely on the use of both the node features and the structure of the network, whereas in my thesis, I have aimed to build an active learning model that does not require node features during learning.

3.5 Tag-based Recommendation Systems

Earlier models on user-based collaborative filtering identify users who are most similar to the target user, and estimate the rating of an item as the average of the ratings given by these similar users [6]. In item-based collaborative filtering [50], items that are similar to *previously rated* items by the target user, are identified first. The predicted rating for each such similar item is computed as an average of the ratings of other items that are *most similar* to it. Several enhancements have been made on these user-based CF methods (e.g., incorporating item popularity while making predictions [42]) and item-based CF methods (e.g., proposing different model building techniques and similarity metrics [11]). However, very little have been done so far to incorporate side information from users into these neighborhood-based models.

Latent factor models (e.g., SVD++, Probabilistic Matrix Factorization a.k.a. PMF, Bayesian Probabilistic Matrix Factorization a.k.a. BPMF) are the most popular factorization based approaches in recommendation systems [26, 37, 49] after their success in the Netflix prize competition [25]. In order to generalize matrix factorization based methods into a predictive framework that can combine the advantages of Support Vector Machines [62] into factorization models, Rendle [43] came up with a new framework known as Factorization Machines (FM). It has been experimentally shown that FM can outperform other matrix factorization methods (SVD++, PMF, BPMF) for large datasets [44].

Using active learning to address new user problem in recommender systems has been studied by researchers from a few different perspectives. Schein *et al.* [51] have developed an Expectation-Maximization [10] based method for dealing with the new item scenario only. Other researchers have proposed Bayesian approaches and the use of the Aspect model [19] to improve the performances for the new users in the system [18, 23]. Enrich *et al.* [12] used *tags* assigned by users to items as *cross-domain* information for collaborative filtering while predicting ratings for new users in the system. Collective classification algorithms are used for jointly classifying all test nodes in a network leveraging both relational

attributes and node attributes [53]. Irrespective of being a popular network classification algorithm, collective classification has not been used in product recommendation systems so far. Furthermore, the ability to seamlessly integrate active learning strategy in collective classification framework gives us the scope to address situations when tag information is not present for most of the users in the system.

Using tags to improve the performance of recommendation systems have recently become popular among researchers [68]. If an user tags an item then it is assumed that he prefers those tags. Hence, a set of tags used by an user can be considered as *user preferences*. Similarly, a set of tags used on an item by either one or more users, can be considered as *item descriptors*. Zhen *et al.* [70] have used this representation of users and items while incorporating the tag-based similarity computation in the framework of probabilistic matrix factorization. Tso-Sutter *et al.* [60] proposed an approach that aims to fuse the user-based and item-based collaborative filtering by combining the ratings from these two methods. They used an extended user-item rating matrix generated from user-tag matrix and item-tag matrix. Shi *et al.* [56] proposed a generative model to tackle tag-induced cross-domain collaborative filtering. However, all these methods lack in one aspect: they assume tags for *all* the users are readily available in the recommendation systems.

Chapter 4: Multi-labeled Collective Classification using Adaptive Neighborhoods

4.1 Introduction

Traditional multi-label classification algorithms (except a few [64–66]) treat every node independently and assume conditional independence between all its labels. As such, these methods perform poorly on multi-labeled relational datasets. In this thesis, I have developed a neighborhood ranking method for multi-label classification, which can be further used in the multi-label collective classification framework. I have tested the methods on real world datasets and also discussed the relevance of this approach for other multi-labeled relational data. Experimental results show that the use of ranking in neighborhood selection for collective classification improves the performance of the classifier.

4.2 Problem Statement

Most collective classification methods are applicable to single-labeled relational data [16, 33, 36, 38]. Recently, a method was proposed for the multi-label collective classification problem [24]. However, none of these approaches discuss how to differentiate between the influence of labels of "closely-related" neighbors from that of "unrelated" neighbors. This thesis approaches the problem of multi-labeled collective classification by combining three different types of information. Firstly, it uses the information that is associated with a given node in the form of attributes or features. Secondly, it uses information associated with a given node in the form of its "other" labels. Finally, within the collective classification paradigm, it uses the label information provided by the node's neighborhood. The primary contribution is in realizing that, for a given node, not every neighboring node has the same

influence while predicting the node's multiple labels. As such, I have developed a method to *rank* the neighboring nodes using the pairwise interaction information coupled with the association of neighboring nodes with different labels. In the next step, I used a method for *pruning* the node's neighborhood using the *rank* values, and incorporated it within the standard collective classification algorithms.

4.3 Methodology

Given the set L of training nodes, the training data $D_L = (\mathcal{X}_L, \mathcal{Y}_L)$, the set U of unlabeled test nodes and the set of node attributes $\mathcal{X}_U$ for the test nodes in U; the goal is to predict the label vector $\mathbf{y}_u = (y_{u1}, \cdots, y_{uk}, \cdots, y_{uK})$ of each test node $u \in U$, where $y_{uk} \in \{0, 1\}$. For convenience, we denote $\hat{\mathbf{Y}}_U = \{\hat{\mathbf{y}}_1, \cdots, \hat{\mathbf{y}}_u, \cdots, \hat{\mathbf{y}}_{|U|}\}$ as the set of label vectors predicted by our models, and $\mathbf{Y}_U = \{\mathbf{y}_1, \cdots, \mathbf{y}_u, \cdots, \mathbf{y}_{|U|}\}$ as the set of true label vectors for all the test nodes in U. This problem is similar to collective classification in a single labeled dataset, except that, in this case, we have to predict multiple labels for each of the test nodes. This thesis aims to exploit the information hidden in the interactions between entities by using properties of graphs. In addition to capturing information from the topology of the graph, this method also captures the *association* between a node and its label. *Pruning* the neighborhood of a test node is based on the "*ranks*" of the training nodes, so that, while predicting the label of the test node, the algorithm only trusts the label information propagated from the *influential* neighbors.

4.3.1 RankNN Algorithm

The proposed RankNN algorithm (Algorithm 4) estimates the labels of test nodes using a naive Bayes approach. Assuming that a node's label is influenced by that of its neighbors, a node's posterior label assignment might change based on the beliefs shared by its neighborhood. Given a set L of training nodes and K labels, RankNN computes the ranking (importance) for each of the labels. The rank r_k for label k is given by (line 3 in

Algorithm 4),

$$r_k = \frac{|L_k|}{|L|} \tag{4.1}$$

where L_k is the set of training instances with label k. Using Equation 4.1, the label rank vector $(r_1, r_2, \cdots, r_k, \cdots, r_K)$ of all the K labels is determined. The rank of a labeled node l is computed as follows (line 5 in Algorithm 4),

$$rank_l = \sum_{j \in N_l} C_{lj} A_{lj} + \sum_{k=1}^{K} B_{lk} r_k \tag{4.2}$$

where N_l is the set of labeled neighbors of node l. $\mathbf{A}_{n \times n}$ is the weighted adjacency matrix representing the number of interactions between any pair of nodes in the network. The weighted adjacency matrix is used for training nodes only. $\mathbf{C}_{n \times n}$ is the pairwise cosine similarity measure between the attributes of the entities. $\mathbf{B}_{n \times K}$ is a matrix representing the association of a node, say l, with a label, say k. Equation 4.2 can be explained as: (i) The first term captures the level of interactivity of an individual. This is measured by the similarity the author has with his neighbors (captured by the cosine similarity matrix $\mathbf{C}_{n \times n}$), weighted by the number of interactions (co-authored papers) the author has had with them; (ii) The second term measures the reputation of the individual in terms of his association with different groups (or labels), and the author's contribution towards those groups (captured by $\mathbf{B}_{n \times K}$); (iii) We define the *rank of a label* as the frequency of entities in the training data that has that corresponding label (e.g., r_k in Equation (4.1) is rank of label k). A highly reputed group will tend to have a large number of members, and hence will be ranked higher. For example, in the DBLP computer science researchers' collaboration network, the association between an author and a research area is determined by the number of publications he/she has in the conferences under that research track. If the author is a prolific researcher but does not publish too many papers in a highly ranked conference, then the second term will weigh down his/her rank. Similarly, if the author has several

publications in reputed conferences, which are not related to the research area (label) we are investigating, then the second term will be under weighted. The rank of the conference in an unrelated research area will be globally low, as determined by Equation (4.1). On the other hand, if an author has published few papers in highly ranked conferences, and has collaborated with too many researchers, then the first term will decrease the author's rank compared to the rank of other researchers who have many co-authors and have published in highly ranked conferences.

Algorithm 4 `RankNN`

Input: A graph $G = (V, E)$, weighted adjacency matrix $\mathbf{A}_{n\times n}$, similarity matrix $\mathbf{C}_{n\times n}$, node and label association matrix $\mathbf{B}_{n\times K}$, total number of classes K, probability threshold θ, number of iterations T, training percentage η for sampling the data into training set L and test set U

Output: Set of label vectors of nodes in test set U i.e. $\hat{\mathbf{Y}}_U$

```
for t = 1 to T do
    Sample training set L and test set U based on training percentage η
    Compute ranks of labels ∀k ∈ K according to Equation (4.1)
    for each node l in L do
        Compute the rank of node l according to Equation (4.2)
        Set threshold δ as median of the rank values
    end for
    for each node u in U do
        Compute prior probability Π̂_u from Equation (4.3)
        Find all influential neighbors N_u^i (of node u) s.t. ∀j ∈ N_u^i, rank_j ≥ δ
        for k = 1 to K do
            Compute likelihood Pr[Z_u = k | y_uk = 1] from Equation (4.4)
            Calculate the posterior probability of y_uk = 1 from Equation (4.5)
        end for
    end for
    for each node u in U do
        Normalize the posterior probabilities of y_u across all K
        if Pr[y_uk = 1 | Z_u = k] > θ then
            Set ŷ_uk = 1
        else
            Set ŷ_uk = 0
        end if
    end for
end for
```

Next, the prior probabilities $\hat{\mathbf{\Pi}}_u$ for each unlabeled test node u is computed from the information contained in the local neighborhood. $\hat{\mathbf{\Pi}}_u = (\hat{\pi}_{u1}, \hat{\pi}_{u2}, \cdots, \hat{\pi}_{uK})$ is a K dimensional vector of probabilities, where each element of the vector corresponds to the probability of a label. Let N_u be the set of labeled neighbors of an unlabeled node u. Let M_j be the set

of labels of the labeled neighbor $j \in N_u$. Let N_{uk} be the set of labeled neighbors of node u that has label k. Then the prior probability of label k for node u is given by (line 9 in Algorithm 4):

$$\hat{\pi}_{uk} = \Pr\left[y_{uk} = 1\right] = \frac{|N_{uk}|}{|\cup_{j \in N_u} M_j|} \tag{4.3}$$

However, if the unlabeled node u does not have labeled neighbors, then an equal probability $\frac{1}{K}$ is assigned to all the labels of that node. This is dependent on the percentage of available labeled data, i.e., if there are very few labeled nodes then there is a high chance that an unlabeled test node will *not* have any labeled neighbors. The performance of the algorithms is assessed by varying the sampling ratios.

Algorithm RankNN estimates the posterior probability of each label for each test node. Let N_u^i be the set of influential neighbors (of a test node u) having rank values above a threshold δ (set dynamically as median of the calculated rank values, although other values of δ are also possible). Let Z_u be a random variable defined over the distribution of all possible data labels, and defined as follows: $Z_u = k$ if k is the most frequent label within the neighborhood of u. Let M_j^i be the set of labels of the labeled influential neighbor $j \in N_u^i$, where N_u^i is the set of influential neighbors of node u. Let N_{uk}^i be the set of influential nodes in the neighborhood of node u that have label k (includes node u based on the assumption that node u has label k), then the probability that u will also have label k depends on the following likelihood (line 12 in Algorithm 4):

$$\Pr\left[Z_u = k | y_{uk} = 1\right] = \frac{|N_{uk}^i|}{|\cup_{j \in N_u^i} M_j^i|} \tag{4.4}$$

The posterior probability used for determining whether node u has label k, i.e., y_{uk} (line 13 in Algorithm 4) is:

$$\Pr\left[y_{uk} = 1 | Z_u = k\right] \propto \Pr\left[y_{uk} = 1\right] \times \Pr\left[Z_u = k | y_{uk} = 1\right] \tag{4.5}$$

The computational complexity of RankNN is $O(|U|KT)$, (if $|L| \leq |U|$). $|U|$ is the number of unlabeled nodes, K is the number of labels and T is the number of iterations.The two input parameters for RankNN are: (i) rank threshold δ and (ii) the probability threshold θ. The rank threshold δ, controls the number of influential neighbors that should be selected while determining the labels for a test node. The RankNN algorithm produces a probabilistic output for the different class labels. As such, to convert the probability scores into hard label assignments (0/1), the probability threshold parameter (θ) has been used.

Algorithm 5 `ICML_Rank`

Input: A graph $G = (V, E)$, weighted adjacency matrix $\mathbf{A}_{n\times n}$, similarity matrix $\mathbf{C}_{n\times n}$, node and label association matrix $\mathbf{B}_{n\times K}$, total number of classes K, number of iterations T, training percentage η for sampling the data into training set L and test set U

Output: Set of label vectors of nodes in test set U i.e. $\hat{\mathbf{Y}}_U$

1: **for** $t = 1$ to T **do**
2: Training:
3: Sample the dataset into training L and test U based on training percentage η
4: **for** $k = 1$ to K **do**
5: Compute the rank of label k from Equation (4.1)
6: Compute the ranks of each node $l \in L$ from Equation (4.2)
7: Set the threshold δ, such that nodes with ranks above δ are *influential*
8: Find the relational features of each node $l \in L$ considering only influential neighbors
9: Construct an extended training set $D_L^k = (X_L^k, \mathbf{Y}_L^k)$ by combining attribute x_l of each node l with its relational features, to give x_l^k
10: Train a local classifier using D_L^k
11: Let $f^k = f(D_L^k)$ be the learned local model for label k
12: **end for**
13: Bootstrapping:
14: **for** each node $u \in U$ **do**
15: Estimate the label $\hat{\mathbf{y}}_u = (\hat{y}_{u1}, \hat{y}_{u2}, \cdots, \hat{y}_{uK})$ such that, $\hat{y}_{uk} = f^k(\mathbf{x}_u, \mathbf{0})$ using only node attributes and no relational features
16: **end for**
17: Iterative Inference:
18: Compute the initial ranks of test nodes by averaging the ranks of labels predicted in the bootstrapping phase
19: **repeat**
20: Construct an extended test set $D_U^k = (X_U^k, \hat{\mathbf{Y}}_U^k)$ (same method as in training) using the ranked influential neighbors
21: Update the estimated values of $\hat{\mathbf{y}}_u$ for $\mathbf{y}_u$ on each test node $u \in U$ as, $\hat{y}_{uk} = f^k(D_U^k)$
22: Update the ranks of the labels (equation (4.1)) and ranks of *all* the nodes (equation (4.2)), based on this new label assignment $\hat{\mathbf{y}}_u$ of each node $u \in U$
23: **until** convergence criteria OR maximum number of iterations
24: Return the label vector $\hat{\mathbf{y}}_u$ for each test instance $u \in U$
25: **end for**

Table 4.1: Description of datasets.

Dataset	#Nodes	#Classes	#Degree/node	#Classes/node
DBLP_A	10334	7	7.1608	1.6205
DBLP_B	6379	6	5.6247	1.1130

4.3.2 ICML_Rank Algorithm

In order to improve the multi-label collective classification approach proposed by Kong *et al.* [24], ICML_Rank algorithm incorporates the node ranking function within the collective classification framework. Instead of blindly considering all the neighbors, *pruning* of the neighborhood of a test node is done based on the ranks of its neighbors while computing the relational features for that node (line 7 in Algorithm 5). The threshold δ is dynamically set as the median of the obtained rank values. LIBSVM [8] with a linear kernel is used for learning classifiers for each of the K labels. In ICML_Rank, a model is learned with training samples for each of the labels $k = 1, 2, \cdots K$. An initial set of labels for test nodes is predicted after feeding the test instances to the trained models. This initial label set for test nodes is used to recompute the ranks of *all* the nodes in the network. Inference of the labels for the test nodes is carried out through an iterative process (in a similar way as ICA algorithm discussed in Chapter 2). Algorithm 5 gives the pseudo-code of ICML_Rank.

4.4 Experimental Setup

4.4.1 Datasets

For the experiments, I have filtered the DBLP citation network data[1] and created two datasets which are denoted as: DBLP_A and DBLP_B. The authors in DBLP_A have published two or more papers from year 2000 to year 2010 in seven research areas identified by the conference names: Databases (SIGMOD, VLDB, EDBT, ICDE, PODS), Data Mining (KDD, SDM, ICDM), Artificial Intelligence (AAAI, IJCAI, AAMAS, UAI), Software

[1] downloaded from http://arnetminer.org/

Engineering (ICSOFT), Computer Vision (CVPR), Information Retrieval (SIGIR, ECIR), Machine Learning (ICML, ECML). The DBLP_B dataset consists of authors who have published papers in six different research areas: Algorithms & Theory (FOCS, STOC, SODA, COLT), Natural Language Processing (ACL, ANLP, COLING), Bioinformatics (ISMB, RECOMB), Networking (SIGCOMM, MOBICOM, INFOCOM), Operating Systems (SOSP, OSDI), Distributed & Parallel Computing (PODC, ICDCS). Table 4.1 provides key statistics for both the datasets. A graph $\mathcal{G}$ represents the DBLP network data containing labeled and unlabeled nodes. Each node in the graph represents an author; node attributes correspond to the titles of the papers that an author has published (attributes are obtained as *tf-idf* measure and are collectively treated as *a document vector* for the author node; relational features are computed following Kong *et. al* [24]). Labels of each node (or author) correspond to the research areas the author has been working on. Since each author may be interested in more than one research area, authors can have multiple labels. A link between a pair of nodes exists if the corresponding authors have co-authored at least one paper. The weights on the edges of the adjacency matrix represent the number of papers they have co-authored. We also have a weight matrix that captures the pairwise cosine similarity between the titles of the papers that a given pair of authors have co-authored. Using the structure of the collaboration network, given the titles of the published papers and multiple labels of authors in the training set, the goal is to predict the multiple labels for authors in the test set.

4.4.2 Validation Protocol

To create an unbiased validation set, all papers (titles) from the training set that are published by authors within the test set were removed. As such, the authors in the training set do not have details about some of the papers that they have published. This leads to a modification in the individual labels for nodes within the training set. The test nodes do not possess any information that is shared with the training nodes. Most of our empirical evaluations are performed on this set, referred to as the "filtered" dataset. In previous

work, this filtering process was not done [24,32,38,65]. As such, the test and train instances would share the co-authored publications. To understand the nature of biased evaluation, the performance of the algorithms on the original dataset was measured and referred to as "unfiltered".

4.4.3 Evaluation Metrics

Following multi-label classification metrics [24] were used for evaluation:

Hamming Loss (HL): measures the fraction of incorrectly predicted labels, averaged across the set U of test nodes. HL is given by:

$$HL(\hat{\mathbf{Y}}_U, \mathbf{Y}_U) = \frac{1}{|U|}\sum_{u=1}^{|U|}\frac{1}{K}\|\hat{\mathbf{Y}}_u \oplus \mathbf{Y}_u\|_1 \tag{4.6}$$

where $\oplus$ denotes the XOR–operation, $\|\cdot\|_1$ denotes l_1-norm.

Subset Loss (SL): evaluates if the predicted label vector is *exactly* identical to the true label vector (macro-label). SL is given by:

$$SL(\hat{\mathbf{Y}}_U, \mathbf{Y}_U) = \frac{1}{|U|}\sum_{u=1}^{|U|} I(\hat{\mathbf{Y}}_u \neq \mathbf{Y}_u) \tag{4.7}$$

where $I(\cdot)$ is an indicator function which outputs 1 if the argument is true and 0 otherwise.

Micro-F1 score (MI–F1): measures the harmonic mean of precision and recall across individual micro-labels, averaged across set U of test nodes. MI–F1 is expressed as follows:

$$MI - F1(\hat{\mathbf{Y}}_U, \mathbf{Y}_U) = \frac{2 \times \sum_{k=1}^{K}\sum_{u=1}^{|U|}\hat{y}_{uk}y_{uk}}{\sum_{k=1}^{K}\sum_{u=1}^{|U|}\hat{y}_{uk} + \sum_{k=1}^{K}\sum_{u=1}^{|U|}y_{uk}} \tag{4.8}$$

Macro-F1 score (MA–F1): computes the average of the F1 measure on the predictions of the macro-labels. MA–F1 is given by:

$$MA - F1(\hat{\mathbf{Y}}_U, \mathbf{Y}_U) = \frac{1}{K}\sum_{k=1}^{K} \frac{2 \times \sum_{u=1}^{|U|} \hat{y}_{uk} y_{uk}}{\sum_{u=1}^{|U|} \hat{y}_{uk} + \sum_{u=1}^{|U|} y_{uk}} \tag{4.9}$$

4.5 Results

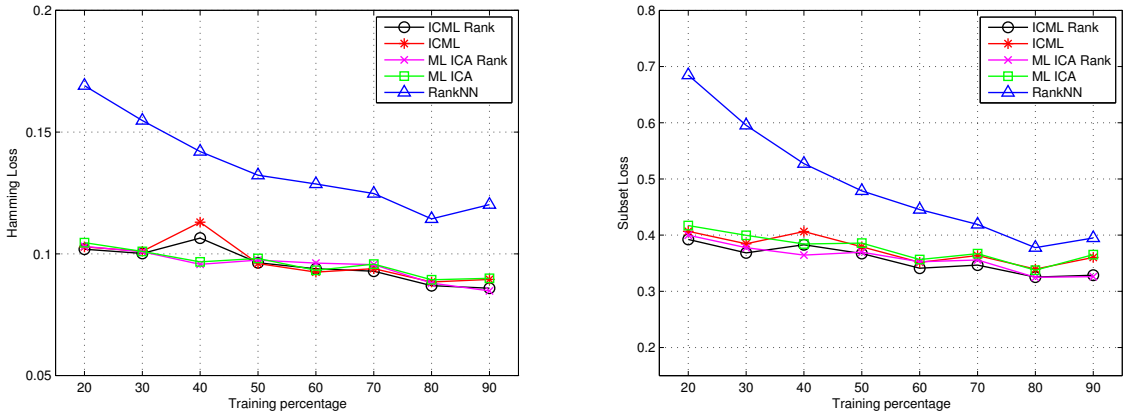
The performance of ICML_Rank and RankNN has been assessed with varying parameters across the four multi-label evaluation metrics. Unless otherwise stated, the default parameter for ICML_Rank is a pruning threshold ($\delta = 50\%$), and for RankNN (to convert the posterior probabilities into hard 0/1 label assignments) is $\theta = 0.25$. The performance of this algorithm is compared against two previously developed approaches (discussed in [24]): (i) **Iterative Classification of Multiple Labels (ICML):** ICML_Rank algorithm is inspired by this approach. ICML does not prune the neighborhood of a node while computing its relational features. (ii) **Multi Label Iterative Classification (ML_ICA):** This algorithm is similar to ICML except that it does not use the intra-instance cross label dependencies. Following a similar approach as in ICML_Rank, the neighborhood pruning technique is incorporated within the ML_ICA framework; the resulting approach is referred as **ML_ICA_Rank**.

4.5.1 Filtered Validation Set

Table 4.2 reports the average of 10 runs with 70% training samples for HL, SL, MI–F1 and MA–F1 for DBLP_A and DBLP_B "filtered" and "unfiltered" sets. For all the experimented algorithms, it was observed that the "unfiltered" set produces lower loss and higher F1 scores in comparison to the "filtered" set. This was expected, because the "unfiltered" set is biased and uses information available from the training nodes directly in the test nodes. Henceforth, the results reported and discussed here are only for the unbiased "filtered"

Table 4.2: Prediction performance (↑ means higher the better, ↓ means lower the better) with filtered & unfiltered label sets for DBLP_A & DBLP_B datasets ($\theta = 0.25$ for RankNN).

DBLP_A					
Labels	Method	HL ↓	SL ↓	MI-F1 ↑	MA-F1 ↑
Filtered	RankNN	0.1543	0.5946	0.5698	0.4851
	ML_ICA	0.1012	0.4023	0.7465	0.6278
	ML_ICA_Rank	**0.1001**	0.3847	**0.7529**	0.6377
	ICML	0.1040	0.3963	0.7451	0.6345
	ICML_Rank	0.1023	**0.3826**	0.7518	**0.6399**
Unfiltered	RankNN	0.1503	0.5672	0.5905	0.5083
	ML_ICA	0.0957	0.3266	0.7788	0.6596
	ML_ICA_Rank	**0.0966**	0.3182	0.7785	0.6638
	ICML	0.0960	0.3169	0.7809	0.6617
	ICML_Rank	0.0965	**0.3107**	**0.7810**	**0.6661**
DBLP_B					
Filtered	RankNN	0.1096	0.4199	0.7074	0.6859
	ML_ICA	**0.07951**	0.2630	**0.8037**	0.7474
	ML_ICA_Rank	0.0806	**0.2543**	0.8028	**0.7475**
	ICML	0.1213	0.3598	0.7025	0.6212
	ICML_Rank	0.1170	0.3451	0.7148	0.6409
Unfiltered	RankNN	0.1102	0.4287	0.7034	0.6797
	ML_ICA	0.0571	0.1657	0.8397	0.7598
	ML_ICA_Rank	0.0584	0.1609	0.8377	0.7605
	ICML	**0.0570**	0.1608	0.8411	0.7608
	ICML_Rank	0.0588	**0.1560**	**0.8479**	**0.7618**



(a) Hamming Loss

(b) Subset Loss

Figure 4.1: Performance vs training percentage (η) for DBLP_A data.

validation set.

4.5.2 Rank Threshold (δ)

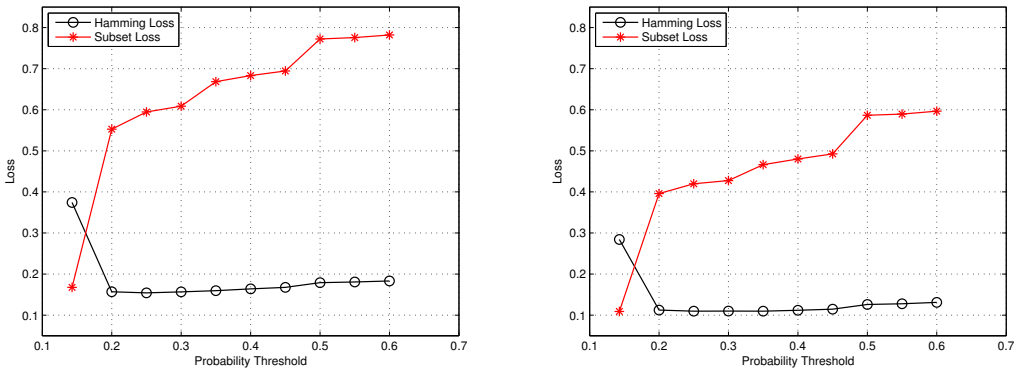
RankNN and ICML_Rank use a dynamically selected threshold for the rank value, which determines the *influential* neighbors to be trusted while determining the label of a test node through collective classification. Table 4.3 shows the multi-label classification performance with varying δ parameter values for RankNN and ICML_Rank across the two DBLP datasets. A threshold (δ) of 25% signifies that 25% of the labeled nodes within the neighborhood of the unlabeled test node are pruned or removed. The ICML_Rank algorithm with a δ value of 0% does not prune any neighboring nodes and is identical to the ICML algorithm.

4.5.3 Sampling Ratio

To evaluate the sensitivity of our algorithms with regards to the training set size, I have performed experiments that vary the training percentage from 20% to 90%. Figure 4.1 shows the variation for different evaluation metrics with respect to the training percentage

Table 4.3: Performance with different rank thresholds (δ).

DBLP_A					
δ	Method	HL ↓	SL ↓	MI-F1 ↑	MA-F1 ↑
0%	RankNN	0.1539	0.6044	0.5663	0.4699
	ICML_Rank	0.0968	0.3727	0.7548	**0.6449**
	ML_ICA_Rank	0.0972	0.3780	0.7612	0.6428
10%	RankNN	0.1545	0.5999	0.5664	0.4900
	ICML_Rank	**0.0948**	**0.3588**	**0.7619**	0.6399
	ML_ICA_Rank	0.0989	0.3615	0.7609	0.6378
25%	RankNN	0.1543	0.5971	0.5676	0.4890
	ICML_Rank	0.1094	0.3921	0.7366	0.6369
	ML_ICA_Rank	0.0971	0.3706	0.7619	0.6414
50%	RankNN	0.1543	0.5946	0.5698	0.4851
	ICML_Rank	0.1023	0.3826	0.7518	0.6399
	ML_ICA_Rank	0.1001	0.3847	0.7529	0.6377
75%	RankNN	0.1572	0.5961	0.5631	0.4657
	ICML_Rank	0.1024	0.3859	0.7508	0.6295
	ML_ICA_Rank	0.1013	0.3935	0.7486	0.6288
DBLP_B					
0%	RankNN	0.1086	0.4198	0.7077	0.6895
	ICML_Rank	0.1143	0.3335	0.7225	0.6641
	ML_ICA_Rank	0.0762	0.2386	0.8145	0.7612
10%	RankNN	0.1104	0.4269	0.7030	0.6842
	ICML_Rank	0.1082	0.3231	0.7361	0.6740
	ML_ICA_Rank	0.0803	0.2509	0.8046	0.7516
25%	RankNN	0.1103	0.4237	0.7056	0.6797
	ICML_Rank	0.1052	0.3181	0.7447	0.6920
	ML_ICA_Rank	**0.0756**	**0.2336**	**0.8147**	**0.7637**
50%	RankNN	0.1096	0.4199	0.7074	0.6859
	ICML_Rank	0.1170	0.3451	0.7148	0.6409
	ML_ICA_Rank	0.0806	0.2543	0.8028	0.7475
75%	RankNN	0.1104	0.4134	0.7075	0.6820
	ICML_Rank	0.1282	0.3704	0.6870	0.6046
	ML_ICA_Rank	0.0835	0.2745	0.7929	0.7248



(a) Loss vs θ for DBLP_A

(b) Loss vs θ for DBLP_B

Figure 4.2: Varying θ for RankNN.

(η) using the filtered DBLP_A set. The ICML_Rank algorithm performs better than the other methods even when the training percentage is very low, i.e., 20%. This shows the robustness of the algorithm, especially when the training data size is small.

4.5.4 RankNN Probability Threshold (θ)

The RankNN algorithm produces a probabilistic output for the different class labels. Probability threshold (θ) was used to convert the probability scores into a hard label assignments. Figure 4.2 shows the hamming loss and subset loss for varying θ values. When θ was set to small values, several more labels get assigned per node; this results in a higher hamming loss. However, setting θ to a larger value will result in fewer assignments of labels, thereby leading to a higher subset loss. From Figure 4.2 it can be inferred that setting the value of θ between 0.2 and 0.45 seems to be optimal.

Chapter 5: Adaptive Forest Fire Sampling

5.1 Introduction

Collective classification involves *collective prediction* of the unknown labels of *all* the test nodes in the network using label information of the training nodes. Even though this has been a well researched topic for years, very little has been done to actively select a good subset of nodes from the network to be used for training a good classifier. This chapter will discuss such a sampling method that I have developed, in order to improve the performance for single labeled collective classification algorithms on real world network datasets.

5.2 Problem Statement

Networks are usually represented as graphs where the vertices are entities and the edges represent interaction between these entities. Of numerous aspects of computational social science that are in practice, my research is primarily focused onto learning supervised models that can be used for making accurate predictions on unseen data within the network. Although major work has been done on finding efficient learning models, an important challenge arises while trying to find the right samples (i.e, instances that represent the distribution from which the original dataset was generated), using which we can learn our models.

The rationale behind collective classification stems from the fact that an entity in a network (or relational data) is most likely influenced by the neighboring entities, and can be classified accordingly, based on the class assignment of the neighbors. Hence, it is very important to have a good training set that can help learning a good model. The contribution in this chapter is the development of an algorithm to *sample* subgraphs from the full

network (a.k.a. *network sampling* [2]). The notion of sampling is explained in detail in section 5.3.1. Two benchmark network datasets: (i) Cora (directed graph representing citation network of scholarly papers)[1] and (ii) DBLP (undirected graph representing co-authorship network of computer science researchers)[2], were used to show that the proposed sampling technique, along with a state-of-the-art collective classification method [33], can improve the overall classification accuracy.

5.3 Methodology

Let $G = (V, E, \mathbf{X}, \mathbf{Y}, L, K)$ represent a relational dataset as defined in Chapter 2. Let $\mathbf{A}_{n\times n}$ be the weighted adjacency matrix and $\mathbf{Q}_{|E|\times 2}$ be an edge-vertex incidence matrix. The goal is to find a sample subgraph $G'_s = (V'_s, E'_s)$ (such that $|V'_s| = |L|$) from the network $G = (V, E)$ (where $V'_s \subset V$ and $E'_s \subset E$) using a sampling algorithm *Adaptive Forest Fire Sampling* (Algorithm 6), in order to train a classifier.

5.3.1 Adaptive Forest Fire Sampling

Forest fire sampling as proposed by Leskovec et al. [28,29], starts by choosing a random seed node and burning a fraction of its outgoing links to neighboring nodes attached to it. This fraction is chosen from a geometric distribution with a predefined mean. It is similar to a random walk based approach, where the selection process is recursive and it halts when no new nodes are selected, or when the required sample size has been reached. Since collective classification propagates the label information from the training nodes to the test nodes in a network during the inference process, it would be helpful if a test node is connected to at least one training node in order to make use of the training node's characteristics for the test node during the training phase of the classifier. The sampling algorithm proposed in this thesis, is an adaptive version of forest fire sampling and is described in detail in Algorithm 6. The complexity of the adaptive forest fire sampling algorithm mostly depends on the time

[1] http://www.cs.umd.edu/∼sen/lbc-proj/LBC.html
[2] http://www.informatik.uni-trier.de/∼ley/db/

Algorithm 6 `Adaptive Forest Fire Sampling`

Input: A graph $G = (V, E)$, edge-vertex incidence matrix $\mathbf{Q}_{|E|\times 2}$, training ratio δ

Output: A sampled subgraph $G'_s = (V'_s, E'_s)$

```
V'_s ← ∅, E'_s ← ∅, E_p ← ∅
Sample size s = ⌈|V| × δ⌉
Randomly select a seed node p from the graph G
repeat
    V'_s ← V'_s ∪ p
    d_p ← degree(p) /* degree(p) returns degree of node p */
    Select ⌈d_p × δ⌉ edges incident on node p and add to E_p
    for each edge e ∈ E_p do
        (p, v) ← Q_e
        E'_s ← E'_s ∪ e
        if v ∉ V'_s then
            V'_s ← V'_s ∪ v
            Add node v in the end of the queue Q
        end if
    end for
    if Queue 𝒬 is not empty then
        p ← first node at the head of the queue 𝒬
    else
        Randomly select a seed node p from the graph G such that p ∉ V'_s
    end if
    E_p ← ∅
until |V'_s| = s
```

Table 5.1: Description of datasets.

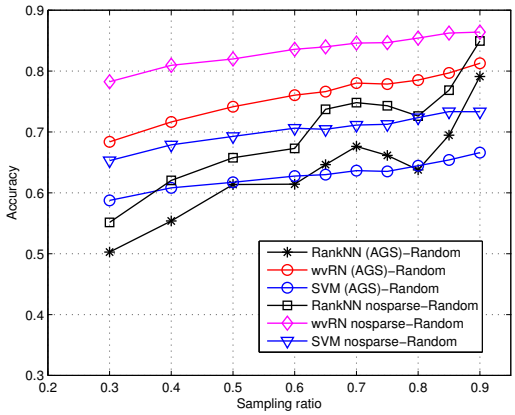
Dataset	$\|V\|$	K	$\|E\|$
Cora	2708	7	5429

for adding/removing elements (edges or nodes) from the queue $\mathcal{Q}$ which is run by the inner *for-loop* (lines $8-15$) in Algorithm 6 with $O(\lceil d_p \times \delta \rceil)$ time. Hence, the time complexity is a function of the degree of the seed nodes chosen randomly by the algorithm. Algorithm 6 aims to find at least one labeled node for each test node in the graph and also ensures that the subgraph is connected. To achieve this goal, the algorithm performs a check at each step by looking forward from the current node and selecting a certain percentage (equal to the percentage of training samples) of edges to be included in the sampled subgraph. It considers the nodes connected to those selected edges as *training nodes* for the sampled subgraph. This process continues until the required training sample size (of nodes) has been reached. We refer to this approach as *adaptive forest fire*, because at each step the algorithm *adapts* towards selecting the edges incident on that particular node residing at the head of the queue that keeps track of the processed nodes. In case there are multiple connected components in the graph, then this algorithm is repeated for each connected component, until the overall sample size for the entire graph has been reached. Selecting training nodes by traversing certain percentage of edges only, ensures that the nodes (that end up being test nodes) at the other end of the unselected edges are connected to at least one labeled node. However, readers should note that, by "sampling" from a graph we mean to extract a subset of nodes whose labels would be made available to the learning algorithm during the training phase such that it helps collective classification. To prove whether this sampled subgraph retains the essential properties of the original graph, not in scope of this work.

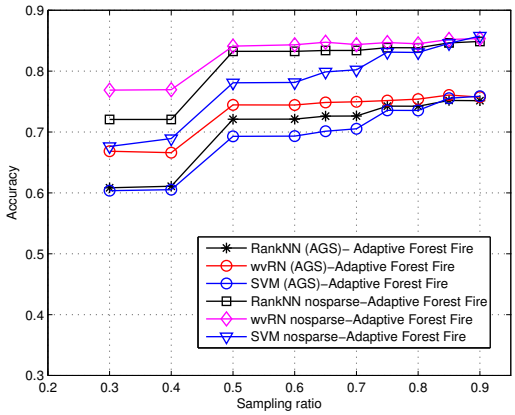
5.4 Results

A real world citation network dataset, Cora, was processed for testing the methods. The statistics of the datasets are given in Table 5.1. Cora is a directed citation network where each node represents a paper and each edge represents a citation link from one paper to another. Each paper can belong to one of seven research areas: Case-Based Reasoning, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning and Theory.

Figures 5.1(a) and 5.1(b) show the performance of different classifiers with random node sampling and adaptive forest fire sampling on the sparsified networks created from Cora dataset. Readers should note here that, Adaptive Global Sparsifier (AGS), Global Sparsifier (GS) and Local Sparsifier (LS) are three different graph sparsification algorithms (AGS was proposed by us to improve the network classification performance and time [46]) for reducing the number of edges in the graph. Since, graph sparsification is out of context for this thesis, I have omitted details of these algorithms. On non-sparsified networks, the wvRN classifier performs best with random sampling. Immediately following this, is the performance of the wvRN classifier used on networks sparsified by our adaptive global sparsification (AGS) algorithm [46]. However, when adaptive forest fire sampling is used, all the three classifiers using non-sparsified networks, perform better in comparison with the classifiers using sparsified networks (which is natural, since sparsification of networks causes loss of information).



(a) Cora - random sampling



(b) Cora - adaptive forest fire sampling

Figure 5.1: Collective Classification performance using graphs sparsified by AGS, GS and LS algorithms [46].

Chapter 6: Active Learning for Network Classification

6.1 Introduction

In this chapter, I have investigated the problem of active learning for both single- and multi-labeled relational network classification in the absence of node features during training. The inability to use a traditional learning setup for classification of relational data, has motivated researchers to propose *Collective Classification* algorithms that jointly classifies *all* the test nodes in a network by exploiting the underlying correlation between the labels of a node and its neighbors. To address this problem, I have proposed active learning algorithms based on different query strategies using a collective classification model where each node in a network can belong to either one class (single-labeled network) or multiple classes (multi-labeled network). I have evaluated the algorithms on both single-labeled and multi-labeled networks, and the results are promising in both the cases for several real world datasets.

6.2 Problem Statement

The first step towards building a classification model is to acquire a representative training set. However, acquiring training samples can be expensive due to the cost of querying labels through interactions with a human or *oracle*. Active learning aims to learn a model with minimal querying cost and can also prioritize the acquisition of labeled samples under budget constraints. Previously, different active learning strategies have been developed for selecting the most informative sample(s) to improve the generalization performance of a classifier. Even though active learning approaches for single-labeled datasets have been extensively studied, algorithms for multi-labeled datasets have not yet been explored. The task becomes even more challenging for multi-labeled networks because traditional active

learning strategies do not take into account the explicit relational information. To address this problem, I have developed a pool-based active learning strategy for single- and multi-labeled networks based on the intuition that, unlabeled instances which are harder to classify undergo multiple changes in their predicted labels during consecutive iterations of collective inference. This changing of labels of an instance is referred as *flipping*, and the method as `FLIP`. I have also investigated the situation when *only a subset of the labels* of a multi-labeled instance can be queried during each round of active learning, thereby, creating a challenge regarding which subset of labels to choose. I have developed a method called `FLIP-per-label` for pool-based active learning to address this real-world situation. My contributions in this chapter can be summarized as follows:

1. Active learning strategy for single labeled and multi-labeled networks (`FLIP`)

2. Active learning strategy for querying a subset of labels of an instance for multi-labeled network (`FLIP-per-label`)

The experiments with six real world single-labeled networks and three multi-labeled networks show statistically significant improvements over random sampling and other baselines for most of the datasets.

6.3 Methodology

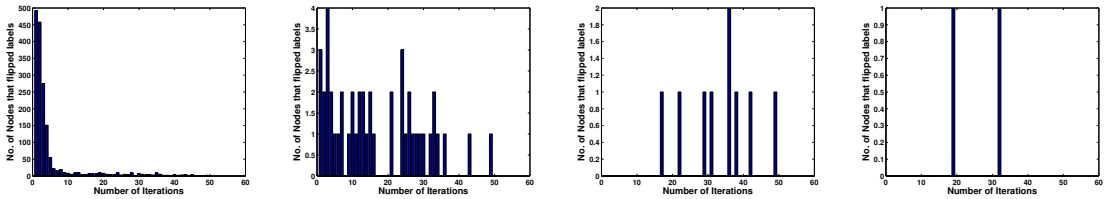
Given a network G=(V, E) where V is the set of nodes and E is the set of edges, each node $v_i \in V$ ($|V| = n$ is the total number of nodes in the network) can have either one label (for single-labeled network) or multiple labels (for multi-labeled networks). Table 6.1 lists the notations used across this chapter.

6.3.1 Active Learning using Iterative Classification Algorithm

For collective classification using iterative inference [32, 38] in relational networks, the label information is propagated from the training node to the test nodes through multiple iterations. This causes the predicted label of a test instance to undergo multiple changes

Table 6.1: Notations

Symbol	Description
$\mathcal{V} = \{v_1, \cdots, v_n\}$	set of n nodes in the network, v_i is node i
$\mathcal{E} = \{e_{ij}\}$	set of edges or links, $e_{ij} \in \mathcal{V} \times \mathcal{V}$ s.t. $v_i, v_j \in \mathcal{V}$
K	total number of classes
N_i	set of all directly linked neighbors of v_i
$\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$	set of label vectors of all the n nodes in network
L and U	index set of labeled nodes and unlabeled nodes such that $L \cup U = \{1, 2, \cdots, n\}$
l and u	index of a labeled node and an unlabeled node such that $l \in L$ and $u \in U$
$\mathbf{y}_u = (y_{u1}, y_{u2}, \cdots, y_{uc}, \cdots, y_{uK})$	label vector for the unlabeled node v_u, $y_{uc} \in \{0, 1\}$
$\mathbf{Y}_U$	predicted output label matrix where each row is the predicted label vector of an unlabeled node, say v_u, $\forall u \in U$
$\mathbf{P}_U$	class probability matrix of all the nodes in set U, such that each row of $\mathbf{P}_U$ is a vector $(p_{u1}, p_{u2}, \cdots, p_{uK})$
$\mathbf{P}_L$	class probability matrix of all the nodes in set L, such that each row of $\mathbf{P}_L$ is a vector $(p_{l1}, p_{l2}, \cdots, p_{lK})$
PO	set of indices of all the nodes in pool



(a) Round 1 (b) Round 2 (c) Round 3 (d) Round 4

Figure 6.1: Distribution of number of Instances which changed labels across number of Iterations in wvRN algorithm's collective inference step during Active Learning.

Algorithm 7 `FLIP`

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: the network, CC: collective classifier, $maxiter$: number of iterations for the approximate inference of CC, b: batchsize, B: budget, PO: pool, T: Initial training set with $l\%$ of $|\mathcal{V}|$ as labeled samples

Output: L: updated training set

1: Initialize $L \leftarrow T$
2: **while** $|L| < B$ **do**
3: Run CC with L as labeled data and PO as unlabeled data to predict labels for instances in the pool PO
4: $S \leftarrow \mathbf{0}$
5: **for** each node v_i s.t. $i \in PO$ **do**
6: $S[i] = \sum_{t=1}^{maxiter} \sum_{k=1}^{K} |y_{ik}^{t} - y_{ik}^{t-1}|$ where y_{ik}^{t} is the predicted label of node v_i in the t-th iteration of inference through CC, such that $y_{ik}^{t} \in \{0, 1\}$ $\forall k \in \{1, 2, \cdots, K\}$
7: **end for**
8: Sort S in descending order of values
9: Pick b nodes from pool PO having top b values in S
10: Add the indices of these b nodes to set L, and remove these indices from pool PO
11: $|L| \leftarrow |L| + b$
12: **end while**

before it finally converges to a particular label. In this work I have monitored the frequency of label change for all the unlabeled instances in the pool set to identify the most informative ones. During the inference steps, I observed that certain nodes that are harder to classify, change their label(s) more frequently than others. I have run multiple iterations of inference on the pool set and aggregated the total changes in labels for each node in the pool set. This aggregation or frequency of label changes is defined as *FLIP score* and an instance with a high *FLIP score* is considered to be a likely candidate for selection by the active learning algorithm (due to it's uncertainty in converging to a fixed label(s)). This approach of picking a batch of instances based on their *FLIP scores*, is referred as `FLIP` and described it in Algorithm 7. For single-labeled networks, wvRN with relaxation labeling (wvRN-RL) is used as the collective classifier (CC) in Algorithm 7. For multi-labeled networks, ML-wvRN-RL is used (see Chapter 2 for background information).

Case Study

A case study was performed on a derived co-authorship network from DBLP[1]. Details of this single-labeled 2-class network and the experimental setup is mentioned in Section 6.4.1.

[1] `http://www.informatik.uni-trier.de/~ley/db/`

I have randomly sampled 2% of the nodes as training and 30% as testing sets, and left the rest in the pool. wvRN-RL [33] was used as collective classifer with training and pool sets; and performed collective inference on nodes in the pool. Figure 6.1 shows the distribution of instances in the pool that changed labels across four rounds of active learning. For each round I have chosen 100 instances from pool with the highest *FLIP scores* across 50 iterations of collective inference and added them to the training set. For each round, the total number of instances that flipped labels, gradually decreased over consecutive iterations of the inference phase. These plots motivated me to use the inference steps of collective classification in order to identify informative instances from pool.

6.3.2 Active Learning for Multi-labeled Networks

The basic steps of `FLIP` algorithm is used for multi-labeled networks with ML-wvRN-RL as the collective classifier. Each label's *FLIP scores* are aggregated and those instances which have highest total *FLIP score* are selected. Rest of the steps are same as shown in Algorithm 7. This method is also referred by `FLIP` in the experiments with multi-labeled networks.

Active Learning with per-label cost

I have developed a method to optimize the cost associated with querying *each label* within the multi-labeled relational networks. Specifically, in each iteration either a single label or a subset of labels of selected samples are queried, while the labels of the remaining instances are inferred in the subsequent rounds. Querying a *subset* of labels of a multi-labeled instance makes more efficient utilization of budget compared to querying all the labels. This method is referred as `FLIP-per-label` (Algorithm 8). Intuitively, this method iteratively chooses a (node,label) pair for which the label has flipped maximum number of times.

The number of (node,label) pairs chosen in each round of active learning depends on a parameter *batchsize*, i.e., the total number of *labels* to be queried. Unlike `FLIP`, in this case array S contains the *FLIP scores* for all possible (node,label) pairs. Additionally, a

Algorithm 8 `FLIP-per-label`

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: the network, K: number of classes, CC: collective classifier, $maxiter$: number of iterations for the approximate inference of CC, b: batchsize, B: budget, PO: pool, T: Initial training set with $l\%$ of $|\mathcal{V}|$ as labeled samples, $\mathbf{Y}_L$: Label matrix of nodes in training set T, l_{tr}: lower threshold of probability score, u_{tr}: upper threshold of probability score

Output: L: updated training set

1: Initialize $L \leftarrow T; \mathbf{P}_L = \mathbf{Y}_L; \mathbf{P}_U = 0.5 \times \mathbf{1}_{|PO| \times K}; total \leftarrow \emptyset$
2: **while** $total < B$ **do**
3: Run CC with L as labeled data and $U = PO$ as unlabeled data to predict labels for unlabeled instances in the pool PO, update $\mathbf{P}_U$ from Equations (2.3)-(2.5)
4: $S \leftarrow \mathbf{0}, index \leftarrow 1, PAIR \leftarrow \mathbf{0}$
5: **for** each node v_i s.t. $i \in PO$ **do**
6: **for** each label $k \in \{1, 2, \cdots, K\}$ **do**
7: $S[index] = \sum_{t=1}^{maxiter} |y_{ik}^t - y_{ik}^{t-1}|$ where y_{ik}^t is the predicted label of node v_i in the t-th iteration of inference through CC, such that $y_{ik}^t \in \{0, 1\}$
8: $PAIR[index] \leftarrow (i, k)$ /* store the (node,label) pair */
9: $index \leftarrow index + 1$
10: **end for**
11: **end for**
12: $SS \leftarrow Sort(S, descend)$
13: $pair \leftarrow (i, k)$ pairs selected from $PAIR$ s.t. $pair[1] = (i, k)$ has the *FLIP score* $= SS[1]$ and $pair[index] = (i', k')$ has *FLIP score* $= SS[index]$
14: $count \leftarrow 0$
15: **for** each $(i, k) \in pair$ **do**
16: Get $P(y_{ik} = 1|N_i)$ from updated $\mathbf{P}_U$ computed in line 3 using Equations (2.3)-(2.5)
17: **if** $P(y_{ik} = 1|N_i) \geq u_{tr}$ or $P(y_{ik} = 1|N_i) \leq l_{tr}$ **then**
18: Query label k of node v_i in the pool PO
19: Add node v_i to L with v_i's k-th label disclosed to CC, remove i from pool PO, Update $\mathbf{P}_L$ with the queried label information of v_i
20: $count \leftarrow count + 1$
21: **if** $count == b$ **then**
22: **break**
23: **end if**
24: **end if**
25: **end for**
26: $total \leftarrow total + b$
27: **end while**

list called $PAIR$ (of (node,label) pairs) is maintained. I have initialized $\mathbf{P}_U = 0.5 \times \mathbf{1}_{|U| \times K}$, assuming all unobserved or *missing* labels are equally likely with probability score = 0.5. I have set $|U| = |PO|$, i.e., number of unlabeled nodes in the pool. In line 3 of Algorithm 8, the learner predicts the probability scores of all the classes (and thereby, labels) for each of the instances in the pool. It queries labels based on the probability values in $\mathbf{P}_U$ (line 17), thereby, updating matrix $\mathbf{P}_L$ (line 19) depending on which instances have their specific label(s) queried. If the predicted class conditional probability of an instance lies between lower and upper probability thresholds (l_{tr} and u_{tr} respectively), then the corresponding class label is assumed to be non-informative, and hence, not queried. I have set $l_{tr} = 0.3$ and $u_{tr} = 0.7$, assuming that any label k of node v_i having probability score $P(y_{ik} = 1|N_i) \leq 0.3$ can not be the true class of v_i. Similarly, any label k of node v_i having probability score $P(y_{ik} = 1|N_i) \geq 0.7$ can be considered as the true label of v_i. In either of the two cases, the label k is queried by the learner. The algorithm is referred as `FLIP-PL` in the plots of Section 7.4.

6.3.3 Variants of `FLIP` for Single- and Multi-labeled Networks

The active learning algorithm relies on computing the *FLIP scores* for each of the instances in the pool during the inference phase. However, several instances may end up having the same *FLIP scores*. As such, I have developed different tie-breaking strategies and variations of the `FLIP` algorithm.

Betweenness Centrality

Macskassy [34] proposed several graph-based metrics to select informative instances for single labeled datasets, out of which betweenness centrality metric was found to be most useful. For a node v_i, the shortest path betweenness $c_B(v_i)$, is defined as follows [34]:

$$c_B(v_i) = \sum_{v_a, v_b \in \mathcal{V}} \frac{\sigma(v_a, v_b|v_i)}{\sigma(v_a, v_b)} \tag{6.1}$$

where $\sigma(v_a, v_b|v_i)$ represents the number of shortest paths that pass through node v_i and $\sigma(v_a, v_b)$ represents number of shortest paths between any pair of nodes v_a, v_b in the graph. Nodes with high betweenness centrality scores can be considered as *information hubs* in the network (making them important for collective inference). Upon encountering a tie on *FLIP scores*, nodes with high betweenness centrality are finally chosen. This is referred by `FLIP-BC`.

Hops from Training nodes

For the proposed active learning algorithm, the objective is to select instances that can disperse the labeled information over the entire network. To this end, upon encountering a tie in *FLIP score* I have chosen only those nodes that are at a greater distance (hops) away from *any* of the training nodes. This method is referred as `FLIP-H`.

Absolute difference in Probability score

For each node in the pool, the active learner predicts the class conditional probabilities with values in $[0, 1]$ (see Equation (2.3)). However, when the learner is most uncertain about the class of an instance, it is more likely to predict a score ≈ 0.5 for each of the K classes. Instead of computing *FLIP score* for v_i as the count/frequency of *flips*, this method computes the deviation of class conditional probabilities from 0.5 for each of the labels at the end of the maximum allowed iterations ($t = maxiter$) and assign the score to $S[i]$.

$$S[i] = \sum_{k=1}^{K} |P^{maxiter}(y_{ik} = 1|N_i) - 0.5| \tag{6.2}$$

where $P^t(y_{ik} = 1|N_i)$ at $t = 0$, is initialized according to Equation (2.4). We query those instances, which have lowest scores in S. If $P^{maxiter}(y_{ik} = 1|N_i) \approx 0.5$ then learner is uncertain about its true label and $S[i] \approx 0$ and v_i can be considered as an informative sample. Intuitively, this approach is very similar to `FLIP`, because $P^{maxiter}(y_{ik} = 1|N_i) \approx 0.5$ will most likely cause node v_i to flip its labels several times. This method is referred as `FLIP-A`.

Entropy

Entropy is a measure of uncertainty in any system. Since, we are predicting the labels of the instances in the pool, I assume that instances which have the highest entropy due to their predicted labels, are most difficult to classify and are expected to be the most informative instances for learning a model. For a node v_i in the pool, the entropy $c_E(v_i)$ due to collective inference is:

$$c_E(v_i) = -\sum_{k=1}^{K} P^{maxiter}(y_{ik} = 1|N_i) \cdot \log(P^{maxiter}(y_{ik} = 1|N_i)) \tag{6.3}$$

This entropy is used as utility score in identifying informative nodes from the pool. This baseline is referred as `Entropy`.

6.3.4 Variants of `FLIP-per-label` for Multi-labeled Networks

The subtle difference between `FLIP` and `FLIP-per-label` required us to use a different set of variants compared to that defined in Section 6.3.3.

Cumulative *FLIP score*

This is a two-phase selection procedure. In first phase, node v_i is selected based on its overall *FLIP score* computed as $S[i] = \sum_{t=1}^{maxiter} \sum_{k=1}^{K} |y_{ik}^{t} - y_{ik}^{t-1}|$ (line 6, Algorithm 7). In second phase, any label k of v_i that has probability score $P^{maxiter}(y_{ik} = 1|N_i) \leq l_{tr}$ or $P^{maxiter}(y_{ik} = 1|N_i) \geq u_{tr}$ was selected for annotation ($P^{maxiter}(y_{ik} = 1|N_i)$ is the probability of class k after *maxiter* iterations during collective classification). This method is referred as `FLIP-PL-ALL`.

Betweenness Centrality

The nodes with high betweenness centrality scores (computed using Equation (6.1)) in the pool are selected first. For each such node v_i, the labels which have probability scores $P^{maxiter}(y_{ik} = 1|N_i) \leq l_{tr}$ or $P^{maxiter}(y_{ik} = 1|N_i) \geq u_{tr}$ are selected for annotation. This is referred by `BC-FLIP-PL`. I have also proposed a baseline method that uses betweenness

centrality score to identify nodes from pool, and then for each such node v_i, randomly selects $\lceil 0.5 \times K \rceil$ labels which have high *FLIP scores*. This method is referred as `BC-RAND`.

Entropy

For a node v_i in the pool, the entropy $c_E(v_i)$ is measured according to Equation (6.3) (Section 6.3.3). This is used as a utility score in identifying informative nodes from the pool. For each such multi-labeled node v_i, $\lceil 0.5 \times K \rceil$ labels which have high *FLIP scores* are queried. This method is referred by `Ent-FLIP-PL`.

6.3.5 Analysis and Time Complexity of `FLIP` and `FLIP-per-label`

The *FLIP score* in `FLIP` (Algorithm 7) is bounded by the following values,

$$\begin{aligned} 0 \leq S[i] \leq maxiter, \text{for single labeled node } v_i \\ 0 \leq S[i] \leq K \times maxiter, \text{for multi-labeled node } v_i \end{aligned} \tag{6.4}$$

The time complexity of `FLIP` primarily depends on the *FLIP score* computation of all the nodes in the pool. Collective classification (with wvRN-RL) for single labeled networks and multi-label networks have time complexity $\mathcal{O}(n \times maxiter)$ and $\mathcal{O}(n \times K \times maxiter)$, respectively. After collective classification, sorting of FLIP scores for all the samples in the pool is done. The time complexity for sorting is $\mathcal{O}(n \times \log n)$. Multi-labeled networks requires sorting of FLIP scores for all labels of an instance, which has time complexity $\mathcal{O}(K \times \log K)$. Hence, the time complexity of the `FLIP` algorithm is $\mathcal{O}(\lceil B/b \rceil \times n \times (maxiter + \log n))$ for single labeled networks, and $\mathcal{O}(\lceil B/b \rceil \times n \times (K \times maxiter + \log n + K \times \log K))$ for multi-labeled networks.

In `FLIP-per-label` (Algorithm 8) the *FLIP score* for a (node,label) pair is bounded by $0 \leq S[index] \leq maxiter$. The collective classifier takes $\mathcal{O}(n \times K \times maxiter)$, and the sort step across $n \times K$ *FLIP scores* of (node,label) pairs takes $\mathcal{O}(n \times K \times \log(n \times K))$. Hence, time complexity of `FLIP-per-label` is $\mathcal{O}(\lceil B/b \rceil \times n \times K \times (maxiter + \log(n \times K)))$.

Table 6.2. Description of datasets (single-labeled and multi-labeled networks) .

Single-label								
Type	Name	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	K	ADN†	ACC†	ALN †	(+)/(-)†
Binary	DBLP(B)-binary	5329	21880	2	7.2117	0.8127	1	2935/2394
	IMDB - prod	1176	37174	2	63.2211	0.3963	1	564/612
Multi-class	Cora	2708	5278	7	3.8981	0.2407	1	NA
	DBLP(B)-multiclass	5329	21880	6	7.2117	0.8127	1	NA
	Industry - pr	2189	11666	12	10.6587	0.5425	1	NA
	Flickr	7971	478980	7	120.1807	0.2955	1	NA
Multi-label								
NA	DBLP(A)	10314	47200	6	8.1526	0.9999	1.6191	NA
	DBLP(B)	5329	21880	6	7.2117	0.8127	1.2211	NA
	IMDB - actor	2411	12255	22	10.1697	0.4720	3.6838	NA

†ADN = Average degree per node, ACC = Average clustering co-efficient, ALN = Average number of labels per node, (+)/(-) = Numbers of instances in positive/negative class

6.4 Experimental Setup

6.4.1 Datasets

The performance of active learning algorithms were evaluated on six single-labeled datasets and three multi-labeled datasets. All datasets used in this thesis can be downloaded from the website[2]. Characteristics of these datasets are provided in Table 6.2. Validation of the results were done using the framework described by Bilgic *et al.* [4]. First, 30% of the nodes are randomly selected from each network as test samples, and kept aside during active learning along with the edges connected to these nodes. The remaining nodes are split into pool and training set. I have found that instead of choosing samples for training randomly, if we choose samples that have high betweenness centrality score then the performance of the classifier improves. The performance of the learner is measured w.r.t. the test set after putting the test nodes and edges back in the network. I have chosen only 2% of the total number of instances having high betweenness centrality scores as the initial training set.

[2] `http://www.cs.gmu.edu/~tsaha/Projects/`

I have have extracted four different co-authorship networks (two single-labeled networks named as *DBLP(B)-binary*, *DBLP(B)-multiclass* and two multi-labeled networks named as *DBLP(B)* and *DBLP(A)*) of computer science researchers from the DBLP[3] bibliographic database, following Kong *et al.* [24]. *DBLP(B)* is a dataset consisting of authors publishing in different computer science areas whereas *DBLP(A)* consists of authors from specific disciplines of computer science (Data Mining/AI). *DBLP(B)-binary* dataset is derived from *DBLP(B)* by considering "Networking" area as the positive class and rest of the classes as negative class. *DBLP(B)-multiclass* is derived by labeling each author (node) in *DBLP(B)* with the research area (label) in which the author has published most of his/her papers. I have created an undirected version of the *Cora* citation network of papers (without any node features) belonging to one of the seven AI related research areas from the original dataset used by Lu and Getoor [32]. *Industry-pr* network comprises of 2189 companies that co-occurred with at least one other company in the PR Newswire release dataset[4]. The labels of the companies are based on Yahoo!'s 12 industry sectors. *IMDB-prod* network contains movies (a link exists if two movies shared a production company) released in the United States between 1996 and 2001, with class labels identifying whether the opening weekend box-office receipts will exceed 2 million dollars or not [33]. Another network of actors (referred as *IMDB-actor*) who acted in movies released between $1990 - 2012$, was created by us. There is an edge between two actors if they have acted together in a movie. The labels of an actor are the multiple genres of the movies in which that actor acted. Since this network is multi-labeled, each actor has one or more genre(s) as his/her label. *Flickr* network was created by sampling the seven most populated classes from the original network by Tang *et al.* [58]. This is a single-labeled connected network of 7971 individuals belonging to 7 specific interest groups (classes).

[3] `http://www.informatik.uni-trier.de/~ley/db/`
[4] `http://netkit-srl.sourceforge.net/data.html`

Table 6.3. Comparing methods for `FLIP` (single-labeled and multi-labeled networks) and `FLIP-per-label` (multi-labeled networks) .

Type of Method	Algorithm	Type of Classification	Publication
Variants of `FLIP`	`FLIP`	Single and multi-label	This thesis (Algorithm 7)
	`FLIP-BC`	Single and multi-label	This thesis (Section 6.3.3)
	`FLIP-H`	Single and multi-label	This thesis (Section 6.3.3)
	`FLIP-A`	Single and multi-label	This thesis (Section 6.3.3)
	`BC`	Single and multi-label	Baseline ([34])
	`Entropy`	Single and multi-label	Baseline (Section 6.3.3)
	`Random`	Single and multi-label	Baseline ([4, 34])
Variants of `FLIP-per-label`	`FLIP-PL`	Multi-label	This thesis (Algorithm 8)
	`FLIP-PL-ALL`	Multi-label	This thesis (Section 6.3.4)
	`BC-FLIP-PL`	Multi-label	This thesis (Section 6.3.4)
	`Ent-FLIP-PL`	Multi-label	This thesis (Section 6.3.4)
	`BC-RAND`	Multi-label	Baseline (Section 6.3.4)
	`Random`	Multi-label	Baseline (Section 6.4.2)

6.4.2 Comparative Methods

Both `FLIP` and `FLIP-per-label` algorithms and their variants are compared w.r.t. several baseline methods listed in Table 6.3. `BC` is the baseline method that chooses instances with high betweenness centrality score as informative samples. A tie is resolved by random selection. For single-labeled networks, `Random` baseline model randomly selects a batch of nodes as informative samples, and queries any label for each of those nodes. For multi-labeled networks, $\lceil 0.5 \times K \rceil$ labels were chosen for querying in `Random`. I have also experimented the active learning paradigm with a link based classifier [32] without using any node features, but the results were considerably poor, because such classifiers use both node and topological features to learn a model.

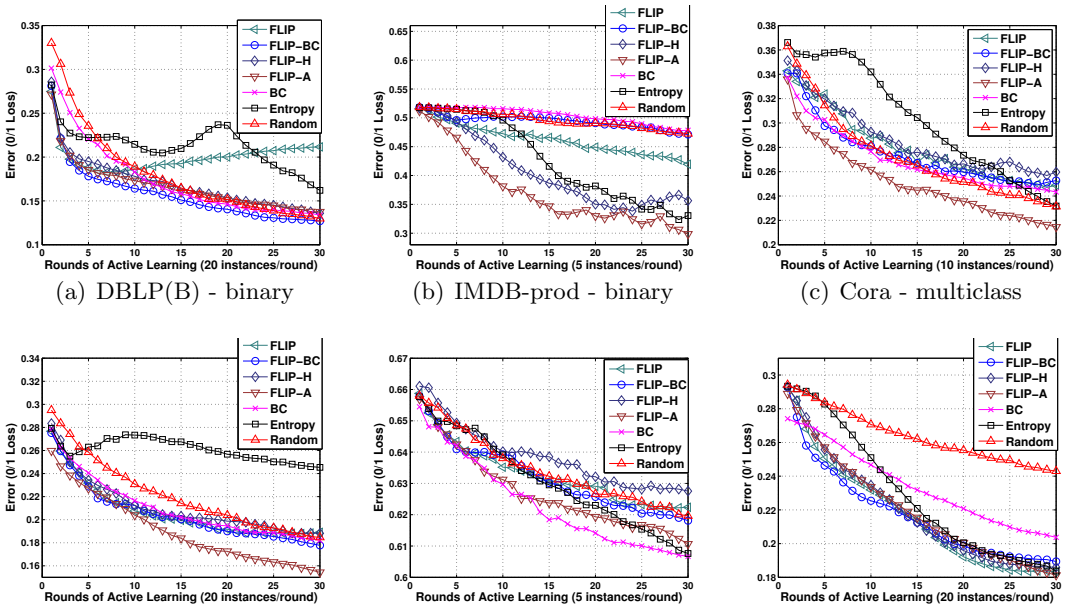
6.4.3 Validation Protocol

For single-labeled networks, I have reported the 0/1 loss (error) on the test set for all the comparing methods. The performances reported for all single-labeled and multi-labeled networks are an average of 10 independent runs. For each round of active learning in single-labeled networks, I have chosen batch size $b = 5$ for IMDB-prod and Industry-pr networks, $b = 10$ for Cora network, and $b = 20$ for DBLP(B) and Flickr networks. The value of batch size, b was determined depending on the total number of nodes in the network. For multi-labeled networks I have used hamming loss (lower the better) and micro-F1 score (higher the better) as evaluation metrics [24, 45, 47]. For each round of active learning using `FLIP`, I have chosen the batchsize $b = 5$ for IMDB-actor, $b = 20$ for DBLP(B) and $b = 30$ for DBLP(A) networks, respectively. For `FLIP-per-label`, $b = 110$ for IMDB-actor, $b = 60$ for DBLP(B) and $b = 90$ for DBLP(A) are used. I have conducted 30 and 50 rounds of active learning for single-labeled and multi-labeled networks, respectively, in order to observe the convergence of all the comparing methods.

6.5 Results

6.5.1 Active Learning for Single-Labeled Networks

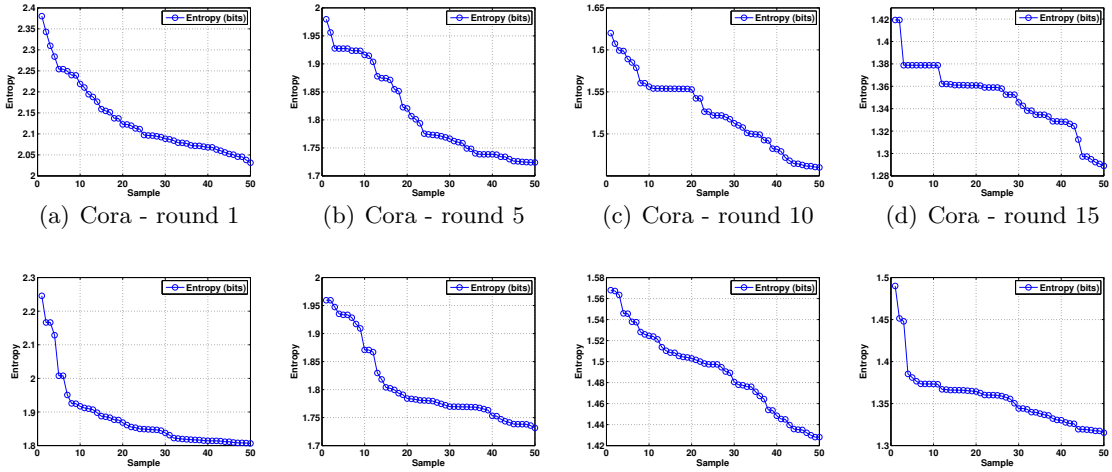
Figure 6.2 shows the performance of all the active learning methods on six single-labeled networks. The X-axis represents the rounds of active learning that have been carried out and Y axis reports the classification error (or 0/1 loss) on the test nodes. We can see that most of the active learning strategies perform well compared to random sampling and other two baseline methods (`BC` and `Entropy`). `Entropy` particularly performs poorly for the Cora and DBLP(B) datasets. Further analysis on the individual entropy scores of the selected nodes at each round of active learning revealed that, after a few rounds of active learning entropy fails as a discriminative criteria for these two datasets. For example, in Figure 6.3, the entropy values of the first 10 samples selected from pool in Cora network are quite different from each other. But, after round 5, the entropy values of the selected samples started to get more closer (sometimes the difference being only in the second or



(a) DBLP(B) - binary (b) IMDB-prod - binary (c) Cora - multiclass

(d) DBLP(B) - multiclass (e) Industry-pr - multiclass (f) Flickr - multiclass

Figure 6.2: Performance of active learning methods on all single-labeled networks (best viewed in color print).



(a) Cora - round 1 (b) Cora - round 5 (c) Cora - round 10 (d) Cora - round 15

(e) DBLP(B) multiclass - round 1 (f) DBLP(B) multiclass - round 5 (g) DBLP(B) multiclass - round 10 (h) DBLP(B) multiclass - round 15

Figure 6.3: Entropy of samples at different rounds of active learning.

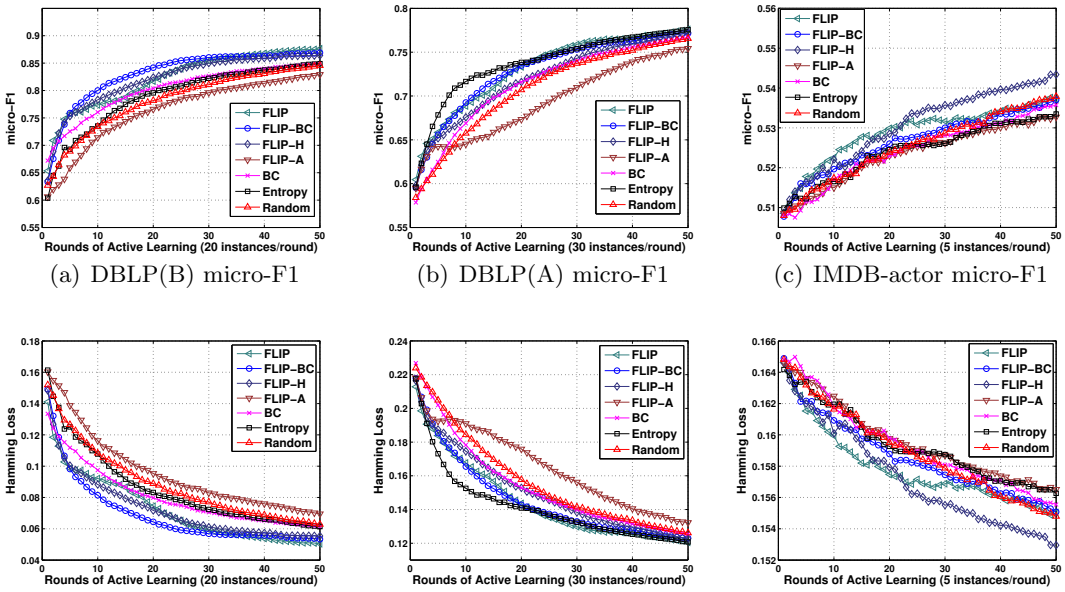
third place after decimal). Figure 6.3(d) shows the decreasing entropy values of the first 10 samples from the pool in Cora are not very different from one other. The distribution pattern of the entropy values stays the same in the subsequent rounds until the end. In Figure 6.2(c) for Cora network, we observe that the loss gradually starts increasing after round 5 onwards. This means that samples that were selected from round 5 onwards are no longer adding relevant information to improve the performance of the learner, which supports the explanations regarding entropy. Figures 6.3(e)-(h) show the entropy values for top 50 samples in the pool of DBLP(B) network. Like Cora, we can see that in DBLP(B), the 20 samples that have highest entropy are somewhat closer in values in round 10 as well as in 15. Hence, we conclude that for DBLP(B) also, entropy loses its discriminative property to choose informative samples after 5 rounds. Figure 6.2(d) for DBLP(B) dataset, shows similar trends to support these claims.

For the IMDB-prod, DBLP(B)-multiclass and Flickr networks, all the active learning algorithms outperform random sampling. These three networks consist of instances that belong to uncorrelated classes, but still have a fully connected structure that can propagate label information over the entire network. So, when *FLIP score* is the only criterion for identifying informative samples in this network, then nodes in the pool undergo multiple changes in their labels through consecutive iterations which causes `FLIP`-based active learning methods to perform well. The high error values for Industry-pr network is because it has 12 classes and the number of samples in the dataset is only 2189, resulting on an average ≈ 182 instances per class. Hence, the classification task is harder for this dataset.

6.5.2 Active Learning for Multi-Labeled Networks

Performance of FLIP

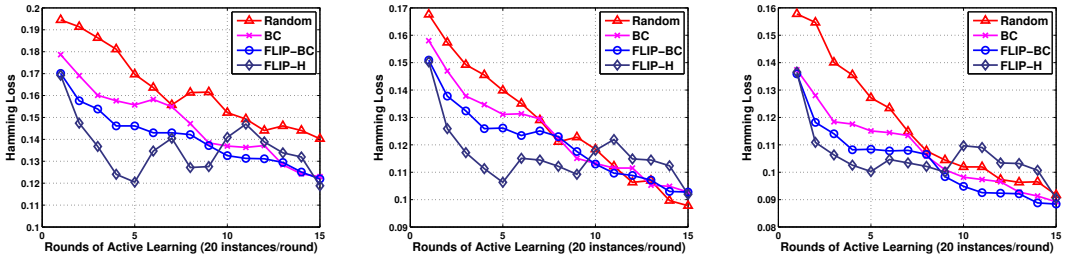
Figures 6.4 shows the performance of different methods with respect to micro-F1 scores and hamming loss for DBLP and IMDB-actor networks, respectively. For DBLP(B) network, the performance of all the active learning methods are better in comparison to the baselines. The good performance of all `FLIP`-based methods (except `FLIP-A`) on this dataset is due to



(a) DBLP(B) micro-F1 (b) DBLP(A) micro-F1 (c) IMDB-actor micro-F1

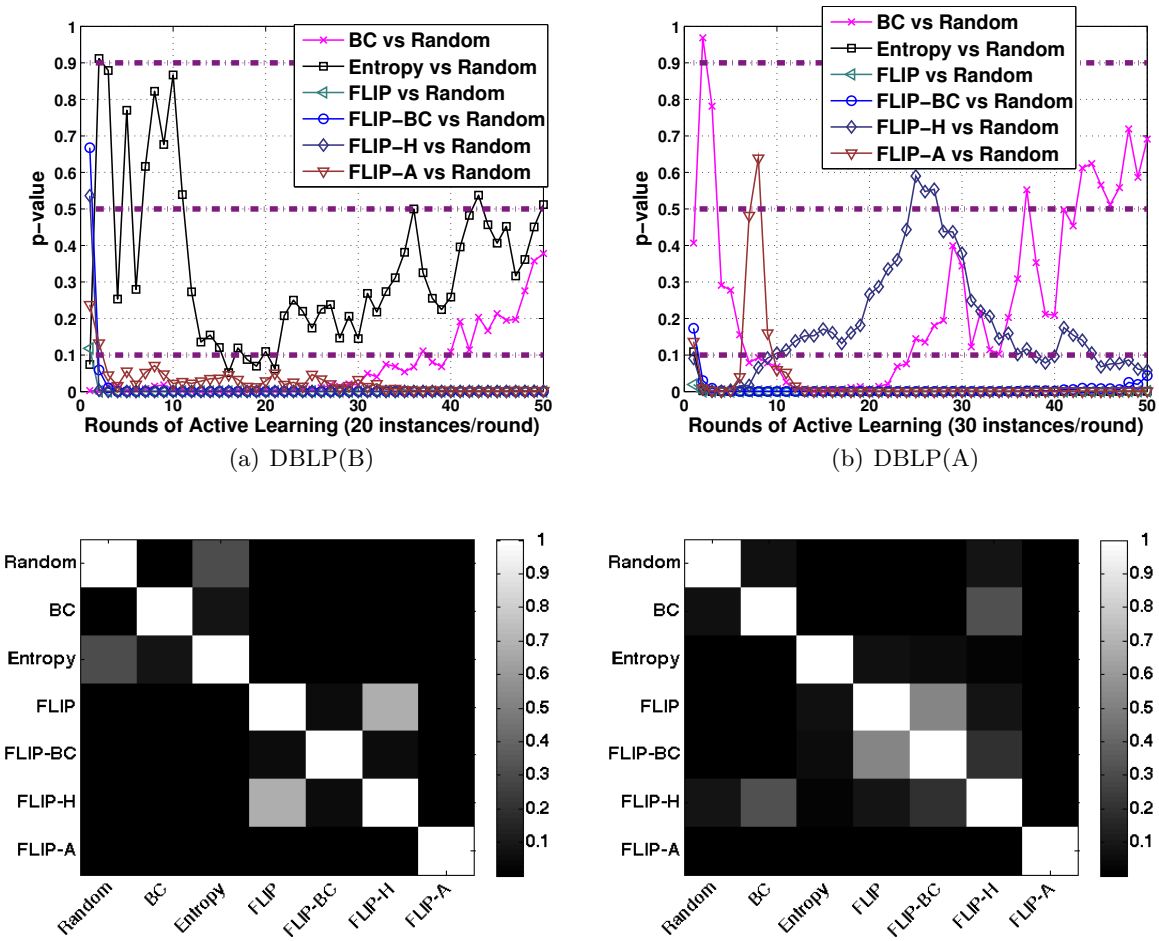
(d) DBLP(B) hamming loss (e) DBLP(A) hamming loss (f) IMDB-actor hamming loss

Figure 6.4: Performance of FLIP on multi-labeled networks w.r.t. micro-F1 score (↑) and Hamming Loss (↓) (best viewed in color print).



(a) $\beta = 0.3$ (b) $\beta = 0.5$ (c) $\beta = 0.8$

Figure 6.5: Performance of FLIP on DBLP(B) multi-labeled network w.r.t. Hamming Loss (↓) using different β (best viewed in color print).



(a) DBLP(B)

(b) DBLP(A)

(c) DBLP(B)

(d) DBLP(A)

Figure 6.6: p-value plots and heatmaps of hamming loss for multi-labeled networks with FLIP: $p <= 0.1$ denotes random sampling is significantly worse, $p >= 0.9$ denotes random sampling is not significantly worse compared to other methods at 10% significance level (best viewed in color print).

the lack of correlation between the classes to which nodes of this network belong. Hence, when we are using only *FLIP* score to query *all* the labels of an instance, the learner is abruptly fed with lots of information that help improve its performance. `FLIP-A` selects instances which have class conditional probability values closer to 0.5 for *all* the classes. For single-labeled network, this is a good metric to identify informative instances, however, for multi-labeled networks this is misleading because it ends up choosing those instances for which *all* the classes have conditional probability values ≈ 0.5. For DBLP(A) network, since all the labels of this network are highly correlated, we see a less promising performance from the `FLIP`-based active learning methods in the first few rounds. For the IMDB-actor network, `FLIP-H` is the best performer. The sparsity of this network and fewer instances per class, poses a hard task for the classifier. When the hops from a training node are considered as tie-breaking criterion for the *FLIP score*, it enables nodes from farther apart zones in the network to get added to the training set, thereby improving the overall diversity of the training set.

One question is how to choose the relaxation labeling parameter β in collective classification (Chapter 2, Section 2.1, Equation (2.5)). I have tested with $\beta = 0.3, 0.5, 0.8$ and it appears from Figure 6.5 that $\beta = 0.8$ gives the best results amongst the three. In general, it is advisable that we choose a value of $\beta \geq 0.5$ to get optimal performance. This ensures that we put more and more weight on the predictions from current iteration compared to that obtained from the previous iteration, thereby, forcing the predictions to converge eventually.

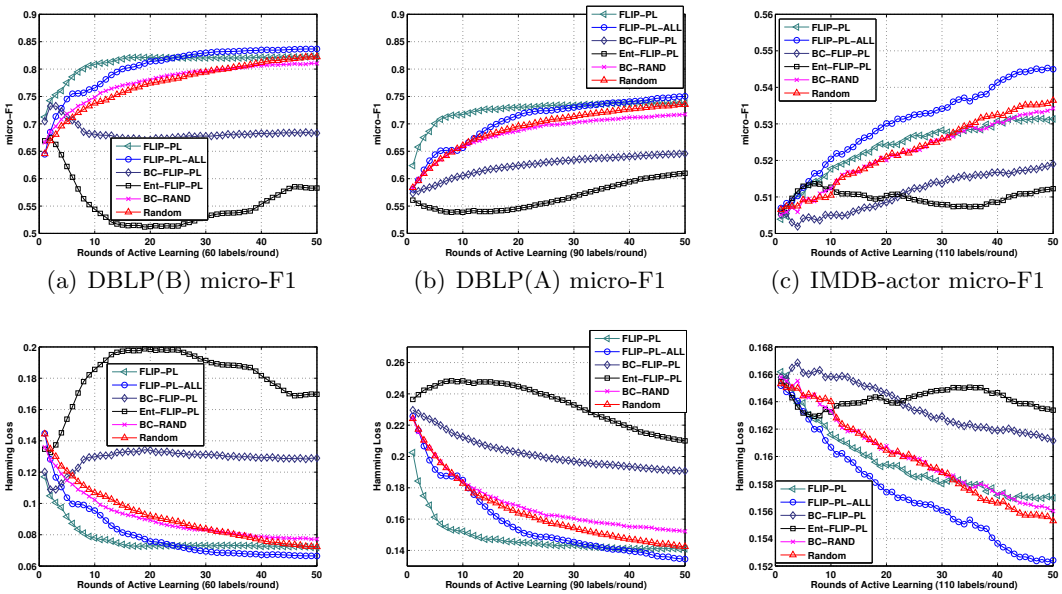
To assess the statistical significance of the results and to compare between the different methods, I have performed paired t-tests following the work of Bilgic *et al.* [4]. Figures 6.6(a)-(b) show the p-value plots for DBLP(B) and DBLP(A) networks. The X axis corresponds to the rounds of active learning, and the Y axis corresponds to the p-value for the paired t-test of the hamming loss resulting from 10 independent runs of the corresponding pair of methods at each round of active learning. If the p-value for a A vs B plot lies below 0.1 then model A wins over model B at 10% significance level.

In order to observe how each of the active learning approaches are performing individually, I have created a heatmap of p-values comparing the algorithms in a pairwise fashion. For example, consider Figure 6.6(c) where each block represents a p-value obtained from the corresponding pair of algorithms (along the rows and columns of the figure). For each algorithm, the performance measures obtained across all the active learning rounds (50 rounds for multi-labeled networks) were aggregated for each of the 10 independent runs. Darker intensity colored boxes indicate that p-value lies below 0.1 and the corresponding pair of methods show significantly different results from one another. Lighter intensity colored boxes suggest otherwise.

I have performed pairwise t-tests on each pair of models for the hamming loss (Figures 6.6(a)-(b) for comparison with `Random` model and Figures 6.6(c)-(d) for comparison with every other method). For brevity, we include results for IMDB-actor network in the submitted supplementary file. Figure 6.6(c)-(d) shows that, for both the DBLP(B) and DBLP(A) networks, all the active learning methods are significantly better than random sampling (`Random` model) and also statistically significant in comparison to one another.

Performance of FLIP-per-label:

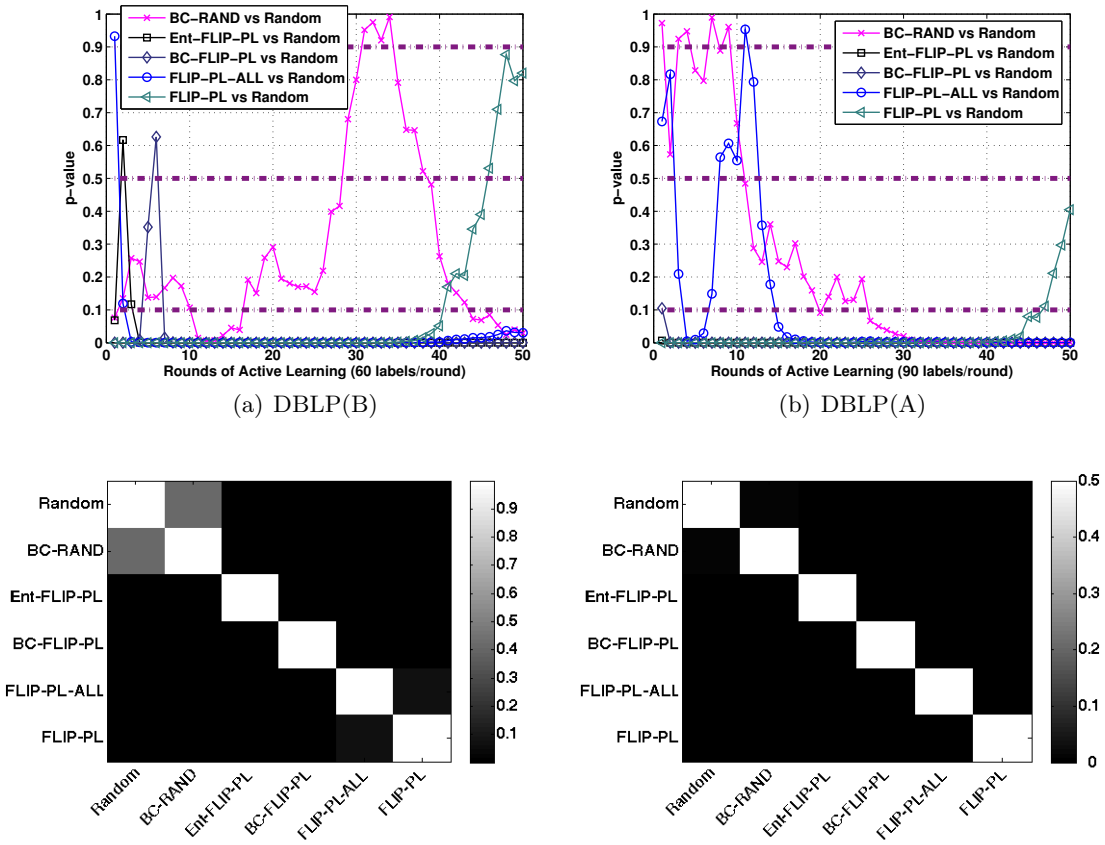
At each round of active learning, instead of querying *all* the labels of an instance, this algorithm queries labels based on (node,label) pairs that were identified as top candidates by the `FLIP-per-label` algorithms. Figure 6.7 shows the performance of `FLIP-PL` and `FLIP-PL-ALL` algorithms for three multi-labeled networks w.r.t. other methods. The poor performance of `BC-FLIP-PL` and `Ent-FLIP-PL` is due to the fact that we initialized the class conditional probability of all pooled nodes with 0.5 in the beginning of active learning. This accounts for high entropy values but low overall *FLIP scores* in the first few rounds, thereby choosing samples that are not quite informative for both `BC-FLIP-PL` and `Ent-FLIP-PL` methods. Figure 6.8 supports these through the corresponding p-value plots and respective heatmaps that compare the performance of different models against random sampling, and also against one another for DBLP datasets.



(a) DBLP(B) micro-F1 (b) DBLP(A) micro-F1 (c) IMDB-actor micro-F1

(d) DBLP(B) hamming loss (e) DBLP(A) hamming loss (f) IMDB-actor hamming loss

Figure 6.7: Performance of FLIP-per-label on multi-labeled networks w.r.t. micro-F1 (↑) and Hamming Loss (↓) (best viewed in color print).



(a) DBLP(B)

(b) DBLP(A)

(c) DBLP(B)

(d) DBLP(A)

Figure 6.8: p-value plots and heatmaps of hamming loss for DBLP(B) and DBLP(A) networks with FLIP-per-label method: $p <= 0.1$ denotes random sampling is significantly worse, $p >= 0.9$ denotes random sampling is not significantly worse compared to other methods at 10% significance level (best viewed in color print).

Chapter 7: Tag-based Collaborative Item Recommendation

7.1 Introduction

In this chapter, I have investigated the problem of improving the performance of state-of-the-art recommendation system algorithms by utilizing concepts from relational learning. Real world social recommendation systems often have rich side information available. This information consists of network structure between users and between items in the system and tags used by the users on the items. Traditional recommendation system exploits the information present in the user-item rating matrix only. Even though tag-recommendation is a very well-research topic related to blogging, online cataloguing etc., it has not been extensively used in combination with item recommendation systems. In this chapter, I will discuss the details of such a model that takes into account the user tags and item tags in the system, in order to be able to predict products to users according to their preferences.

7.2 Problem Statement

While lot of work has been done in both the directions, the usage of tags as *side information* in recommendation systems still has lot of scope for improvement [31]. Some examples of side information in personalized recommender systems are user profile information and location based information [14, 69]. Users often associate multiple aspects while rating an item. These aspects are represented as short text snippets or *tags* which usually signify what the user thinks about a particular item. For example, in Figure 7.1, consider the four users in a user-movie recommendation system tagging the same movie "The Shawshank Redemption". Each of the users assign tags to the movie which represents their perspective. User 1 and 4 uses the same tag "feel-good" which says that both the users think "The Shawshank

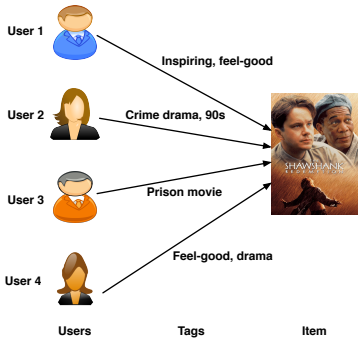


Figure 7.1: Users Tagging Items in Tag-based Recommendation Systems.

Redemption" as a "'feel-good" movie. Hence, while computing user-user similarity using tags as side information, user 1 and user 4 will be more similar to each other compared to other users.

In relational learning, *collective classification* methods are quite popular in classifying nodes in network datasets. These methods jointly classify *all* the test nodes in a network by leveraging the complex and implicit correlations between multiple entities and their labels. They are applicable towards networks which have topological features [33, 53], but may or may not have node features [33], and can also be applied on multi-labeled networks [24, 45]. Recently, researchers have investigated the cold-start problem with new user and new items in the context of *tag-recommendation* using collective classification framework with promising results [41]. I have developed an item recommendation model by employing multi-label collective classification on the implicit user-user network and item-item network (derived from the user-item rating matrix) in order to predict *tags* (or labels) for users and items, respectively. If the tags are not available for *some* of the users in the system then I have employed active learning to incrementally learn a collective classifer and predict tags for those users. Furthermore, I have used these predicted tags as *user attributes* and *item attributes*, and have introduced a neighborhood- and a latent-based collaborative filtering method. Hence, my contribution can be summarized as follows:

- Using *collective classification* on user and item networks to predict tags for users and

items, respectively. I have also employed active learning with a collective classifier when tag information is available only for few users/items.

- Using tags as *user attributes* and *item attributes*, thereby compute *tag-induced pairwise similarities* between users and between items for neighborhood based collaborative filtering model.

- Incorporating user attributes and item attributes into a latent factor model, thereby improving the performance of item recommendation.

Let U be the set of users, I be the set of items and P represent the set of preference tags or multi-criteria aspects that a user can use to provide additional feedback while rating an item. A four-tuple $\langle u, i, p, R_{u,i} \rangle$ represents that a user $u \in U$ has provided a final rating of $R_{u,i}$ and associated a preference tag $p \in P$ for an item $i \in I$. $R_{u,i}$ can be real valued (if explicit rating information exists) or binary (0 or 1) when there is no explicit rating captured, as in tagging or transaction related datasets. I have denoted $\mathbf{R}$ of dimensions $|U| \times |I|$ as the user-item rating matrix.

Let $U_{te} \in U$ be users in the system for whom we do not have the tag information available. These users are referred as *test users* in the tag classification model. I have used multi-label collective classification [24, 47] to predict the tags for these test users. Let $U_{tr} \in U$, be the set of *training users* for whom we know *all* the tags they have used ($U_{tr} \cup U_{te} = U$). For each user $u \in U$, I have hidden some of his rated items during the training of the recommender systems. This generates a partially complete user-item rating matrix for training where the number of users and number of items are same as in the original matrix, but the rating entries for an user depend on the items that are not hidden. The task of the recommender system model is to be able to predict some or all of these hidden items. From the user-item rating matrix for all users in U and the items in I, we can construct:

- a user-user network $G_U = (V_U, E_U)$ where V_U is the set of user nodes ($|V_U| = |U|$) and E_U is the set of edges in G_U.

- an item-item network $G_I = (V_I, E_I)$ where V_I is the set of item nodes ($|V_I| = |I|$) and E_I is the set of edges in G_I.

To create the network I have used the following criteria:

- If user x and user y both rate at least one item then there is a link between these two users in the user network G_U.
- If item w and item v are both bought by at least one user then there is a link between these two items in the item network G_I.
- The label(s) of a training user in the user-user network are the tag(s) he has used to rate items, and the label(s) of an item in an item-item network are the tag(s) that were used previously by users on that item.

Since, there are some users and some items for which we do not know the tags associated with them, the model needs to predict those tags. The specific objectives of this study are two-fold: (i) to predict the concept tags a user is most likely to use and also predict the tags that get associated with a given item; and (ii) incorporate these predicted user- and item-associated tags as side information within standard recommender system models to predict either the rating a user will provide for an item or identify the relevant items for a user.

7.3 Methodology

Figure 7.2 provides an overview of the model for tag-based item recommendation system. In this approach, first I have used collective classification approaches (and extensions) to predict the preference tags that are associated with a given user and also identify descriptive tags for a given item. I have integrated these "predicted tags" as side information within neighborhood-based and latent factor based recommendation systems to identify the most relevant items for a user and the rating of an item provided by a user, respectively. The intuition behind this approach is that tag information provides information about user preferences and description/context about items, and utilizing these tags assists in developing more effective recommender systems.

7.3.1 Collective Classification

For collective classification, I have used the iterative classification algorithm (ICA) [38] with the multi-label weighted vote relational neighbor (wvRN) as the base classifier [33,47]. wvRN is a weighted k-nearest neighbor approach that assigns labels to test nodes based on the labels of training nodes in the neighborhood. The readers are referred to our previous paper on collective classification [47] for more details.

Given a user-item rating matrix for all users in U and for all items in I, I have constructed: (i) a user-user network G_U and (ii) an item-item network G_I. To create these relational networks we use the following criteria: (i) If a pair of users rate at least one common item, then there is a link between these two users in the user network G_U; (ii) If a pair of items are rated/bought by at least one common user, then there is a link between these two items in the item network G_I (Figure 7.2).

The label(s) of a training user in the user-user network are the tag(s) that are used with items, and the label(s) of an item in an item-item network are the tag(s) that were used previously by users on that item. There are several users and items for which we do not know their tag information, and I have utilized collective classification to predict those tags.

Active Learning for Tag Prediction

To address the common real world scenario of having tag information for very few users and items, I have also used pool-based active learning [54] to train the multi-label collective classifier [47]. Active learning involves re-training of a classifier through consecutive iterations, when very few labeled samples are available initially for training, thereby adding informative samples to the training set in every round. Active learning approaches also provide e-commerce websites with a strategy for crowd-sourcing additional tag feedback for a selected set of users and items only. In the iterative collective classification algorithms, the labels of all the unlabeled samples in the pool undergo multiple changes through the consecutive iterations of the inference procedure. Nodes that undergo more changes or *flips* in their labels, are considered to be harder to classify. Following our previous work [47], I

have used *FLIP-score* for each node as the criteria to select informative samples. Let $S[z]$ denote the FLIP score for node v_z (say, a node from the user-user network or the item-item network) computed using the FLIP algorithm [47]:

$$S[z] = \sum_{j=1}^{maxiter} \sum_{l=1}^{|P|} |\hat{t}_{zp}^{j} - \hat{t}_{zp}^{j-1}| \tag{7.1}$$

where $\hat{t}_{zp}^{j}$ is the predicted label (tag) of node v_z in the j-th iteration of inference through collective classification, such that $\hat{t}_{zp}^{j} \in \{0, 1\}$ $\forall p \in \{1, \cdots, |P|\}$. The informative nodes (from user and item networks) are selected for querying tags based on their $S[z]$ scores (higher the better). Once the tags of the informative user/item samples are queried, they are added to their respective training set for collective classification.

7.3.2 Recommender Systems

If $\mathbf{t}_x$ and $\mathbf{t}_y$ be the tag vectors for any pair of users x and y respectively, then pairwise user similarity matrix $\mathbf{S}^U$ can be pre-computed as cosine similarity between the tag vectors $\mathbf{t}_x$ and $\mathbf{t}_y$:

$$S_{x,y}^{U} = \frac{\mathbf{t}_x {\mathbf{t}_y}^T}{||\mathbf{t}_x||_2 ||\mathbf{t}_y||_2} \tag{7.2}$$

Similarly, the pairwise item similarity matrix $\mathbf{S}^I$ can be precomputed using item tag vectors $\mathbf{t}_w$ and $\mathbf{t}_v$ for items w and v respectively, and using Equation (7.2). Once the tag vectors for test users are determined through collective classification, we can employ the following collaborative filtering models depending on the type of dataset we have. Figure 7.2 gives an overview of the tag-based recommendation systems with m users and n items.

Neighborhood-based Model

These models are the most common form of CF approaches [57]. These models are either *user-based* or *item-based*. In their original form, user-based neighborhood models estimate unknown ratings based on past ratings of "like-minded" users. To estimate the rating $\hat{R}_{u,i}$

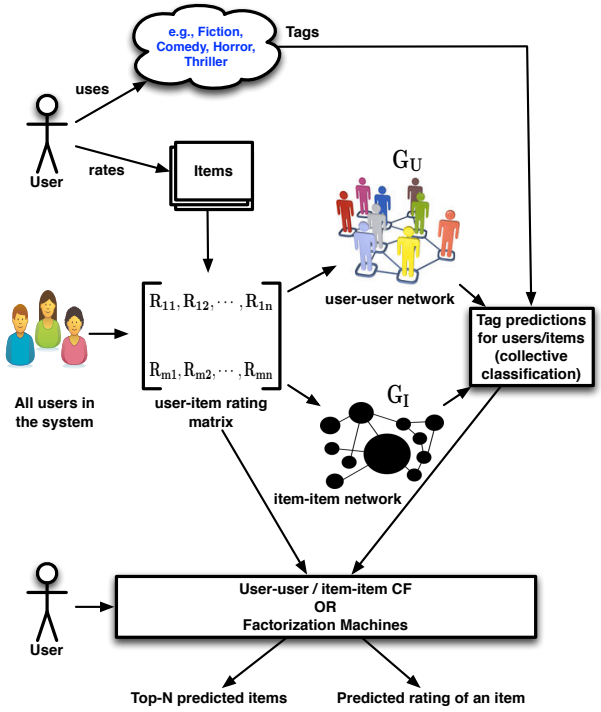


Figure 7.2: Tag-based Item Recommendation Systems.

for a user u on a specific item i, these approaches identify the neighborhood N_u (users similar to u), who have rated item i. The predicted rating $\hat{R}_{u,i}$ is given by:

$$\hat{R}_{u,i} = \frac{\sum_{v \in N_u} S_{u,v} R_{v,i}}{\sum_{v \in N_u} S_{u,v}} \tag{7.3}$$

where $S_{u,v}$ denotes the similarity between users u and v, and is used for determining the neighborhood N_u for computing the weighted average. With only the rating matrix $\mathbf{R}$, choices for computing similarity include cosine similarity and Pearson correlation coefficient. An alternative approach to the user-oriented method is the item-oriented approach [50], where the predicted rating utilizes the known ratings made by the same user on similar (or neighborhood of) items. The predicted rating $\hat{R}_{u,i}$ for a user u on an item i is given by:

$$\hat{R}_{u,i} = \frac{\sum_{j \in N_i} S_{i,j} R_{u,j}}{\sum_{j \in N_i} S_{i,j}} \tag{7.4}$$

where N_i is the set of items that are "most similar" to item i. $S_{i,j}$ is the similarity between items i and j computed in the user space.

In this formulation I have aimed to use the tags for a user and/or item as side information (beyond the rating matrix) within these neighborhood models. For each user u, I have denoted the predicted tag vector as $\hat{\mathbf{t}}_u = (\hat{t}_{u1}, \ldots, \hat{t}_{u|P|})$ where $\hat{t}_{up} \in \{0, 1\}$ denotes the absence or presence of the p-th tag. Instead of using the rating matrix $\mathbf{R}$ to define the neighborhood for a user u in user-based approach, we compute the neighborhood N_u using the predicted tag vectors for the users. Given a pair of users x and y with predicted tag vectors $\hat{\mathbf{t}}_x$ and $\hat{\mathbf{t}}_y$ respectively, I have computed the cosine similarity between the two tag vectors. Given the neighborhood N_u computed using the user tag vectors, this model identifies all the items rated by users in that neighborhood and denotes the set by I_{N_u} and estimates their ratings using Equation (7.3). The top-N most popular items are selected from the set I_{N_u}, and this approach is referred as User-Tag-KNN (**UT-KNN**).

Analogous to the UT-KNN model, I have developed an item oriented approach which we refer as Item-Tag-KNN (**IT-KNN**). In this case, every item i is associated with a predicted tag vector and is denoted by $\hat{\mathbf{t}}_i$ and predict the final rating using the neighborhood of candidate items, following Equation (7.4).

I have also combined the user- and item-associated predicted tag vectors in a neighborhood model and referred to it as User-Item-Tag-KNN (**UIT-KNN**). In this case the similarity score between user u and item i is computed using the cosine similarity between the predicted tag vectors of user u and item i. The rest of the steps are same as the UT-KNN model. This allows us to use the side information with predicted tags in both the user and item space.

Latent Factor Models

Latent factor models (probabilistic matrix factorization, SVD++) are state-of-the-art methods in recommender systems. But their formulation [37,49] do not incorporate side information in the models. To address this problem, Factorization Machines (FM) were proposed as a new model class of general predictors. These are similar to support vector machines in a sense that, they can work with any feature vector. However, unlike SVMs, factorization machines can model all pairwise nested interactions between the variables even for very sparse input data. Rendle [43] proposed factorization machine model for a set of tuples $(\mathbf{x}, f(\mathbf{x}))$ where $\mathbf{x} \in \mathbb{R}^D$ is a feature vector and $f(\mathbf{x})$ is the corresponding target. A factorization machine that models all interactions upto order $d = 2$ between the D input variables is defined as:

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{D} w_j x_j + \sum_{j=1}^{D} \sum_{j'=j+1}^{D} x_j x_{j'} \sum_{f=1}^{F} v_{j,f} v_{j',f} \tag{7.5}$$

where F is the dimensionality of the factorization, and the model parameters $\Theta = \{w_0, w_1, \cdots, w_D, v_{1,1}, \cdots, v_{D,F}\}$ are: $w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^D, V \in \mathbb{R}^{D \times F}$. The main difference

between the second part of Equation (7.5) and polynomial regression, as mentioned by Rendle [43], is that the interaction is not modeled as an independent parameter $w_{j,j'}$. Rather, it is modeled as a factor $w_{j,j'} \approx \langle \mathbf{v}_j, \mathbf{v}_{j'} \rangle = \sum_{f=1}^{F} v_{j,f} v_{j',f}$.

- **User-Item-Tag-FM** Assuming we have categorical data in the form of user-item rating matrix $\mathbf{R}$, then it is easy to represent this information as a feature vector $\mathbf{x}$ in the factorization machine framework. Going further, if we want to incorporate the predicted user attribute vector as the tag vector $\hat{\mathbf{t}}_u = (\hat{t}_{u1}, \hat{t}_{u2}, \cdots, \hat{t}_{u|L|})$ for an user u, and the predicted item attribute vector as the tag vector $\hat{\mathbf{t}}_i = (\hat{t}_{i1}, \hat{t}_{i2}, \cdots, \hat{t}_{i|L|})$ for an item i in $d = 2$ way factorization machine, then the feature vector $\mathbf{x}$ for the $\langle u, i \rangle$ pair will look like:

$$\mathbf{x} = (\underbrace{0, \cdots, 1, \cdots, 0}_{|U|}, \underbrace{0, \cdots, 1, \cdots, 0}_{|I|}, \underbrace{\hat{t}_{u1}, \hat{t}_{u2}, \cdots, \hat{t}_{u|L|}}_{\text{attributes of user } u}, \underbrace{\hat{t}_{i1}, \hat{t}_{i2}, \cdots, \hat{t}_{i|L|}}_{\text{attributes of item } i}) \quad (7.6)$$

where the first $|U| + |I|$ features has the value 1 only at two positions corresponding to the active user u and active item i, i.e., the $\langle u, i \rangle$ pair for which we have to predict the rating. In this formulation, I have used the tags of the users as user attributes and tags associated with the items as item attributes. Hence, both types of attributes have dimensionality $|L|$ because there are $|L|$ number of total tags. The model for predicting target rating $\hat{R}_{ui} = f(\mathbf{x})$ correponding to the user-item pair $\langle u, i \rangle$ with their attribute tag vectors, will have the following representation:

$$f(\mathbf{x}) = w_0 + w_u + w_i + \sum_{s=1}^{|L|} \hat{t}_{us} w_s + \sum_{s=1}^{|L|} \hat{t}_{us} \langle \mathbf{v}_i, \mathbf{v}_s \rangle + \sum_{s=1}^{|L|} \sum_{s'>s}^{|L|} \hat{t}_{us} \hat{t}_{us'} \langle \mathbf{v}_s, \mathbf{v}_{s'} \rangle +$$
$$\sum_{q=1}^{|L|} \sum_{q'>q}^{|L|} \hat{t}_{iq} \hat{t}_{iq'} \langle \mathbf{v}_q, \mathbf{v}_{q'} \rangle + \sum_{q=1}^{|L|} \hat{t}_{iq} w_q + \sum_{q=1}^{|L|} \hat{t}_{iq} \langle \mathbf{v}_u, \mathbf{v}_q \rangle + \sum_{f=1}^{F} v_{uf} v_{if} \quad (7.7)$$

where the feature vector $\mathbf{x}$ is replaced by the expression in Equation (7.6). The dimension (i.e., D in Section 7.3.2) of the feature vector $\mathbf{x}$ for this case is, $|U| + |I| + 2 \times |L|$. The fifth term represents "attribute mapping" that uses linear regression to map user attributes to factors. The sixth and seventh terms are 2-way interactions between user attributes and item attributes, respectively. The ninth term is to map the item attributes to factors. The fourth, eighth and the last terms capture 1-way interactions. This method is referred as UIT-FM.

- **User-Tag-FM**: This is similar to the model described in the previous section except that, only user attributes (tags) are used in the formulation. The feature vector $\mathbf{x}$ looks like :

$$\mathbf{x} = (\underbrace{0, \cdots, 1, \cdots, 0}_{|U|}, \underbrace{0, \cdots, 1, \cdots, 0}_{|I|}, \underbrace{\hat{t}_{u1}, \hat{t}_{u2}, \cdots, \hat{t}_{u|L|}}_{\text{attributes of user } u}) \quad (7.8)$$

The predictive model $f(\mathbf{x})$ is formulated as before, except that there are no item attributes (tags) present in this model. The dimension of the feature vector $\mathbf{x}$ for this case is, $|U| + |I| + |L|$. This method is referred as UT-FM.

- **Item-Tag-FM**: This is similar to the models described earlier, except that only item attributes (tags) are used in the formulation. The feature vector $\mathbf{x}$ looks like :

$$\mathbf{x} = (\underbrace{0, \cdots, 1, \cdots, 0}_{|U|}, \underbrace{0, \cdots, 1, \cdots, 0}_{|I|}, \underbrace{\hat{t}_{i1}, \hat{t}_{i2}, \cdots, \hat{t}_{i|L|}}_{\text{attributes of item } i}) \quad (7.9)$$

The predictive model $f(\mathbf{x})$ is formulated as before, except that there are no user attributes (tags) present in this model. The dimension of the feature vector $\mathbf{x}$ for this case is, $|U| + |I| + |L|$. This method is referred as IT-FM.

Table 7.1: Description of datasets

Name	$\|U\|$	$\|I\|$	$\|T\|$	#Ratings	Rating Density	#tags/ user	#tags/ item	Avg # users tagging/ rating an item	Avg # items tagged/ rated by user
Bibsonomy	116	361	75	2230	0.0533	15.1379	8.5208	6.1773	19.2241
Delicious	636	1027	45	4180	0.0064	9.7563	5.2775	4.0701	6.5723
Tripadvisor	1202	1890	5	4607	0.0020	1.7263	1.7339	2.4376	3.8328
MovieLens	704	3146	58	197025	0.0890	2.5739	2.2219	62.6271	279.8651
LibraryThing	1500	3500	21	102455	0.0195	8.7393	5.0626	29.2729	68.3033

7.4 Experimental Protocol

7.4.1 Datasets

Table 7.1 reports the key characteristics of datasets used in this study. We have two types of datasets: one type has only tags and the other has both the rating and tags. Bibsonomy[1] and Delicious[2] are social bookmarking datasets with tagging information only, and no item ratings. Tripadvisor[3] dataset contains tag as well as rating information for hotels reviewed by users. MovieLens is a popular movie rating dataset with tag information from users. LibraryThing[4] is an online catalog where users can tag and rate the books they have read.

I have pruned down the list of tags used in Bibsonomy dataset to 75, in order to rule out less frequent and uninformative tags. I also pruned the Delicious dataset, such that it contains 636 users, 1027 items and 45 most frequently used tags. The Tripadvisor dataset used here contains ratings given by 1202 users on 1890 hotels. The tags (one or more from the set {Solo, Family, Couples, Business, Friends}) assigned by an user to a hotel signifies the purpose for which the hotel might be suitable for that user. For MovieLens dataset, we selected 58 most frequent tags (e.g., "dystopia", "funny", "surreal", "satirical" are only a

[1] `http://www.kde.cs.uni-kassel.de/bibsonomy/dumps/`
[2] `http://grouplens.org/datasets/hetrec-2011/`
[3] `http://students.depaul.edu/~yzheng8/DataSets.html`
[4] `http://www.macle.nl/tud/LT/`

few to name) used by users on the movies. Unlike the other datasets, the movies (items) in MovieLens have genres associated with them which are considered as item attributes. There are 19 genres for 3146 movies rated by the selected 704 users who were frequent taggers. For LibraryThing dataset, I have selected 21 most frequent tags, 1500 most frequent taggers and 3500 most frequently tagged items for our study. All the datasets used in this paper have been made publicly available at `http://cs.gmu.edu/~mlbio/TagRecSys/`.

7.4.2 Comparative Approaches

Ground Truth Models

In order to assess the usefulness of the tag information, instead of using the predicted tags from collective classification I have used true tags (ground truth) in each of the proposed models. The performance of these models serve as an upper bound of the expected performance of the corresponding models that use predicted tag vectors.

- **Neighborhood-based approaches:** I represent the models that use ground truth tags in proposed neighborhood-based methods as **UT-KNN-G**, **IT-KNN-G** and **UIT-KNN-G**.
- **Latent factor based approaches:** I represent the models that use ground truth tags in proposed latent factor-based methods as **UT-FM-G**, **IT-FM-G** and **UIT-FM-G** to represent the true tag information added only for the user, for the item and for both, respectively.

Baseline Methods

I have used the following baseline approaches depending on whether rating information is available within the datasets or not.

- **User-CF-Pop** This is the standard user-user similarity based CF method [57] for selecting the "most similar" users with respect to the target user. It chooses the top-N most popular items out of all the items rated by the most similar users. This method is referred as **U-CF-Pop** that uses Equation (7.3) for the predicted ratings, using only the user-item rating matrix.

- **Item-CF**: This is the standard item-item similarity based CF method proposed by Sarwar *et al.* [50]. This method is referred as **I-CF** that uses Equation (7.4) for the predicted ratings, using only the user-item rating matrix.
- **Factorization Machines**: For latent factor-based methods, I have used the Factorization Machine [43] as the standard matrix factorization model (Equation (7.5)). The input rating matrix is converted into feature vector format for those datasets which have explicit real valued ratings available for items. I have denoted this baseline by **FM**.

7.4.3 Evaluation Strategy

I have validated the performance of these proposed models using top-N hit-rate metric [11] for datasets which only have tagging information. For each user, I have hidden $h\%$ of the items for validation purpose. If any of the top-N predicted items fall in the list of $h\%$ hidden items for an user, then I have considered it as a *hit*. The values of h used in our experiments will be discussed in detail in Section 7.4.5. For each user $u \in U$, if I_u^h are the true items that the user has rated, but were hidden from the learner during training, and if $\hat{I_u^h}$ were the predicted items for this user, then, Top-N *hit-rate* for the model is given by [11]:

$$\text{Top-N hit-rate} = \frac{1}{|U|} \sum_{u \in U} \mathbb{I}(\hat{I_u^h} \cap I_u^h \neq \emptyset) \tag{7.10}$$

where $\mathbb{I}(\cdot)$ is an indicator function that returns 1 if the arguement is true, 0 otherwise. For datasets where we have rating information available, I used the standard metrics mean absolute error (MAE) and root mean square error (RMSE) [57] to compare the results with other baseline methods.

7.4.4 Parameters

The neighborhood-based CF models proposed in this paper require two parameters: (i) the number of nearest neighbors (K) for a user and (ii) the number of top items (N)

to be recommended. Both these parameters were set as 10 in our experiments. Multi-label collective classification using wvRN classifier and relaxation labeling requires two parameters, α and β [33], which were set to 0.99 and 0.8, respectively. For each test node, I have predicted the tags with top probabilities based on the average number of tags present in the node's neighborhood. The parameter F, i.e., dimensionality of factorization used in FM based models, is set to 100 for our experiments. I have evaluated the performance of the FM-based models on a small validation set by varying the number of latent factors ($F \in \{5, 8, 10, 50, 80, 100, 150, 200\}$) and the value of 100 showed the best performance.

7.4.5 Cross Validation Setup

I have experimented with two setups for training and testing item set splits in for evaluating the different recommender systems: (i) **standard split**: for each user, 70% of the randomly chosen items were used for training and 30% ($h = 30$) were hidden for testing; (ii) **random split**: for each user I randomly chose either 70% or 50% or 30% items for training and the rest of the items for testing. Based on the percentage of items that were hidden during training (using either one of the two setups), parameter h is set and an updated user-item rating matrix is created. The user-user network G_U and item-item network G_I are constructed using this updated user-item rating matrix. These networks are used within the collective classification framework for predicting the tags for both the users and the items. For collective classification, the number of test users is $|U_{te}| = b \times |U|$, where $b = 0.2$ in our experiments unless stated otherwise. A total of 5 independent trials were conducted for both standard split and random split. For active learning experiments, I started with only 5% of randomly sampled nodes in the relational network as training samples, and at each round of active learning I have queried the tags of 5% of the *informative* users (or items for item-item network) chosen from the unlabeled pool. I have performed 15 rounds of active learning.

Table 7.2: Performance of latent factor based methods with respect to Mean Absolute Error (↓) for Tripadvisor/MovieLens/LibraryThing datasets with standard split and random split.

Tripadvisor							
Training split	FM	UT-FM	UT-FM-G	IT-FM	IT-FM-G	UIT-FM	UIT-FM-G
Standard	0.8704 ± 0.0077	0.8521 ± 0.0068	0.8490 ± 0.0042	0.8480 ± 0.0043	0.8310 ± 0.0047	**0.8429** ± 0.0044	0.8312 ± 0.0063
Random	0.8881 ± 0.0067	0.8718 ± 0.0088	0.8694 ± 0.0089	0.8650 ± 0.0061	0.8535 ± 0.0060	**0.8612** ± 0.0094	0.8523 ± 0.0073
MovieLens							
Standard	0.5741 ± 0.0013	0.5785 ± 0.0017	0.5789 ± 0.0016	**0.5706** ± 0.0016	0.5674 ± 0.0015	0.5751 ± 0.0019	0.5724 ± 0.0011
Random	0.5891 ± 0.0039	0.5938 ± 0.0040	0.5941 ± 0.0039	**0.5852** ± 0.0034	0.5818 ± 0.0036	0.5898 ± 0.0035	0.5872 ± 0.0031
LibraryThing							
Standard	0.6128 ± 0.0014	0.6165 ± 0.0015	0.6162 ± 0.0012	**0.6045** ± 0.0015	0.5959 ± 0.0012	0.6085 ± 0.0019	0.5985 ± 0.0012
Random	0.6207 ± 0.0014	0.6234 ± 0.0016	0.6228 ± 0.0015	**0.6144** ± 0.0019	0.6071 ± 0.0018	0.6188 ± 0.0018	0.6099 ± 0.0017

Table 7.3: Performance of latent factor based methods with respect to Root-Mean Square Error (↓) for Tripadvisor/MovieLens/LibraryThing datasets with standard split and random split.

Tripadvisor							
Training split	FM	UT-FM	UT-FM-G	IT-FM	IT-FM-G	UIT-FM	UIT-FM-G
Standard	1.0558 ± 0.0061	1.0503 ± 0.0034	1.0501 ± 0.0030	**1.0446** ± 0.0062	1.0349 ± 0.0075	1.0465 ± 0.0053	1.0382 ± 0.0065
Random	1.0751 ± 0.0063	1.0703 ± 0.0082	1.0690 ± 0.0072	**1.0635** ± 0.0067	1.0552 ± 0.0067	1.0667 ± 0.0077	1.059 ± 0.0062
MovieLens							
Standard	0.7624 ± 0.0020	0.7676 ± 0.0019	0.7679 ± 0.0025	**0.7575** ± 0.0023	0.7533 ± 0.0019	0.7626 ± 0.0026	0.7590 ± 0.0018
Random	0.7807 ± 0.0054	0.7862 ± 0.0057	0.7869 ± 0.0056	**0.7754** ± 0.0047	0.7708 ± 0.0051	0.7809 ± 0.0052	0.7771 ± 0.0041
LibraryThing							
Standard	0.7859 ± 0.0025	0.7909 ± 0.0026	0.7902 ± 0.0022	**0.7768** ± 0.0028	0.7674 ± 0.0025	0.7816 ± 0.0035	0.7704 ± 0.0025
Random	0.7930 ± 0.0021	0.7969 ± 0.0018	0.7963 ± 0.0019	**0.7860** ± 0.0021	0.7781 ± 0.0022	0.7915 ± 0.0020	0.7818 ± 0.0021

Table 7.4: Performance of neighborhood based methods with respect to Top-N Hit rate (↑) for Bibsonomy/Delicious datasets with standard split and random split.

Bibsonomy								
Training split	U-CF-Pop	I-CF	UT-KNN	UT-KNN-G	IT-KNN	IT-KNN-G	UIT-KNN	UIT-KNN-G
Standard	**0.3224** ± 0.0547	0.2207 ± 0.0438	0.2810 ± 0.0337	0.3569 ± 0.0283	0.1190 ± 0.0231	0.3207 ± 0.0387	0.1397 ± 0.0462	0.6069 ± 0.0320
Random	**0.3879** ± 0.0376	0.2759 ± 0.0528	0.3155 ± 0.0283	0.4707 ± 0.0248	0.2259 ± 0.0319	0.4569 ± 0.0122	0.2810 ± 0.0735	0.6931 ± 0.0302
Delicious								
Standard	**0.0739** ± 0.0038	0.0572 ± 0.0069	0.0440 ± 0.0179	0.1069 ± 0.0127	0.0302 ± 0.0115	0.0711 ± 0.0079	0.0336 ± 0.0158	0.1736 ± 0.0135
Random	**0.0884** ± 0.0086	0.0657 ± 0.0095	0.0629 ± 0.0110	0.1450 ± 0.0072	0.0346 ± 0.0084	0.1038 ± 0.0095	0.0733 ± 0.0149	0.2176 ± 0.0136

7.5 Results and Discussion

Tables 7.2, 7.3 and 7.4 show the performance of our proposed tag-based CF models on several real world datasets with respect to the baseline methods. Table 7.2 and 7.3 show the results for datasets with explicit rating information (Tripadvisor, MovieLens and LibraryThing), whereas Table 7.4 shows the results for datasets without explicit rating information (Bibsonomy and Delicious). The predicted tag-based FM methods (UIT-FM and IT-FM) perform better than the baseline FM for all the three datasets (Tables 7.2 and 7.3). However, as expected, the tag-based FM methods can not beat the ground truth models (UIT-FM-G, IT-FM-G and UT-FM-G). The best performance of the ground truth models in comparison to all other methods proves that the knowledge of true tags helps in tag-based item recommendation using FM. For predicted tag-based neighborhood methods we see a different behavior in Table 7.4. The baseline method U-CF-Pop performs better than UIT-KNN, UT-KNN and IT-KNN for both Delicious and Bibsonomy datasets. However, the ground truth models UIT-KNN-G is the best performer amongst all comparative methods. Looking carefully, it appears that item tag prediction (IT-KNN) is comparatively performing worse than user tag prediction (UT-KNN) for these two datasets, and that is why the overall

Table 7.5: p-value for tag-based latent factor methods vs Baselines with respect to MAE.

Dataset	UIT-FM vs FM	UIT-FM-G vs FM	UT-FM vs FM	IT-FM vs FM
Tripadvisor	$\approx$ **0**	$\approx$ **0**	**0.0113**	$\approx$ **0**
MovieLens	0.7645	0.4202	0.1034	0.1283
LibraryThing	0.1059	$\approx$ **0**	**0.0242**	$\approx$ **0**

Table 7.6: p-value for tag-based neighborhood methods vs Baselines with respect to Top-N Hit rate.

Dataset	UIT-KNN vs U-CF-Pop	UIT-KNN-G vs U-CF-Pop	UT-KNN vs U-CF-Pop	IT-KNN vs U-CF-Pop
Bibsonomy	**0.0201**	$\approx$ **0**	**0.0088**	$\approx$ **0**
Delicious	**0.0852**	$\approx$ **0**	**0.0036**	$\approx$ **0**

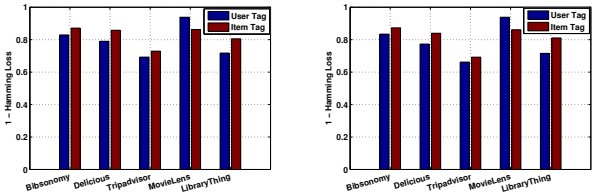
performance of UIT-KNN is getting affected.

7.5.1 Statistical Significance Tests

I have done paired t-tests with significance level = 0.10, between the performance outcomes of 5 independent runs of our tag-based methods and baselines. The p-values of these t-tests for our predicted tag-based CF models are listed in Tables 7.5 and 7.6. If the p-value ≤ 0.1 then we can say with 90% confidence that the results are statistically significant. Almost all the outcomes of our tag-based methods gave statistically significant scores (except MovieLens).

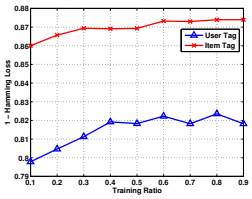
7.5.2 Performance of Collective Classification

In order to see how accurately the multi-label collective classification algorithm is predicting tags for users and items, I have shown the (1−Hamming Loss) metric in Figure 7.3 for all five datasets. It appears that the performance of the collective classifier for both the user and item tag prediction is good for all the datasets.

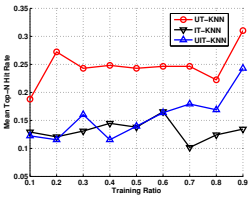


(a) Standard split (b) Random split

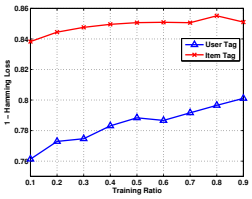
Figure 7.3: Performance of collective classification for user and item tag prediction with respect to (1-Hamming Loss) (↑) metric with standard split and random split (best viewed in color print).



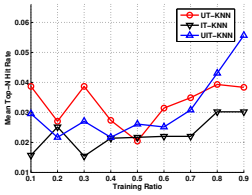
(a) Bibsonomy (multi-label accuracy↑)



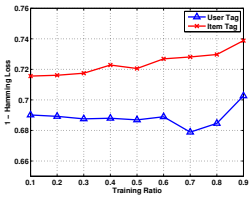
(b) Bibsonomy (Top-N hitrate↑)



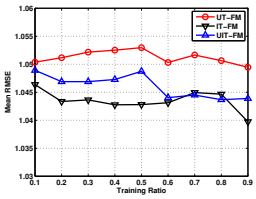
(c) Delicious (multi-label accuracy↑)



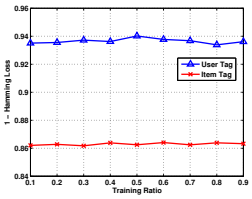
(d) Delicious (Top-N hitrate↑)



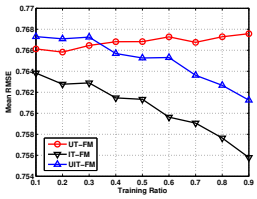
(e) Tripadvisor (multi-label accuracy↑)



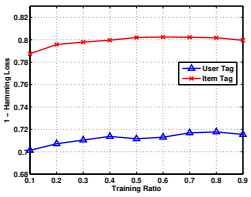
(f) Tripadvisor (mean RMSE↓)



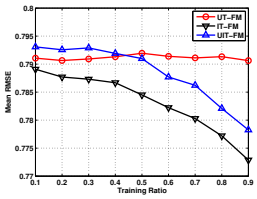
(g) MovieLens (multi-label accuracy↑)



(h) MovieLens (mean RMSE↓)

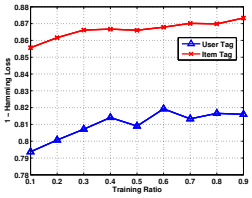


(i) LibraryThing (multi-label accuracy↑)

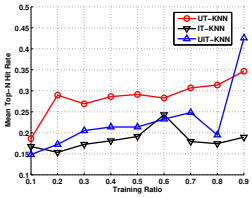


(j) LibraryThing (mean RMSE↓)

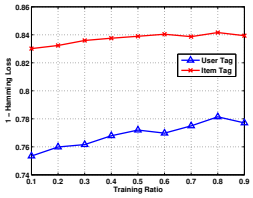
Figure 7.4: Tag Prediction Accuracy of Collective Classification and Performance of CF methods with respect to RMSE (↓) using standard split for varying training ratio in collective classification.



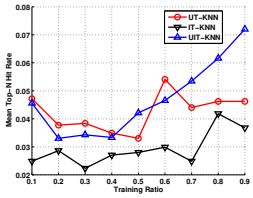
(a) Bibsonomy (multi-label accuracy↑)



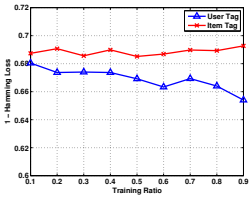
(b) Bibsonomy (Top-N hitrate↑)



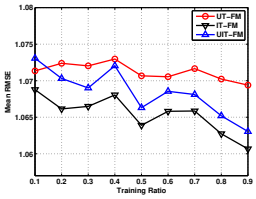
(c) Delicious (multi-label accuracy↑)



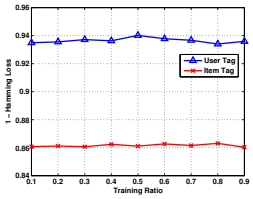
(d) Delicious (Top-N hitrate↑)



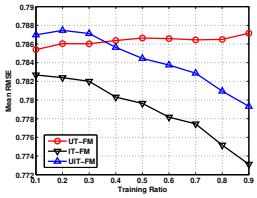
(e) Tripadvisor (multi-label accuracy↑)



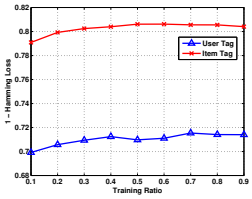
(f) Tripadvisor (mean RMSE↓)



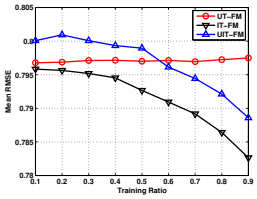
(g) MovieLens (multi-label accuracy↑)



(h) MovieLens (mean RMSE↓)



(i) LibraryThing (multi-label accuracy↑)



(j) LibraryThing (mean RMSE↓)

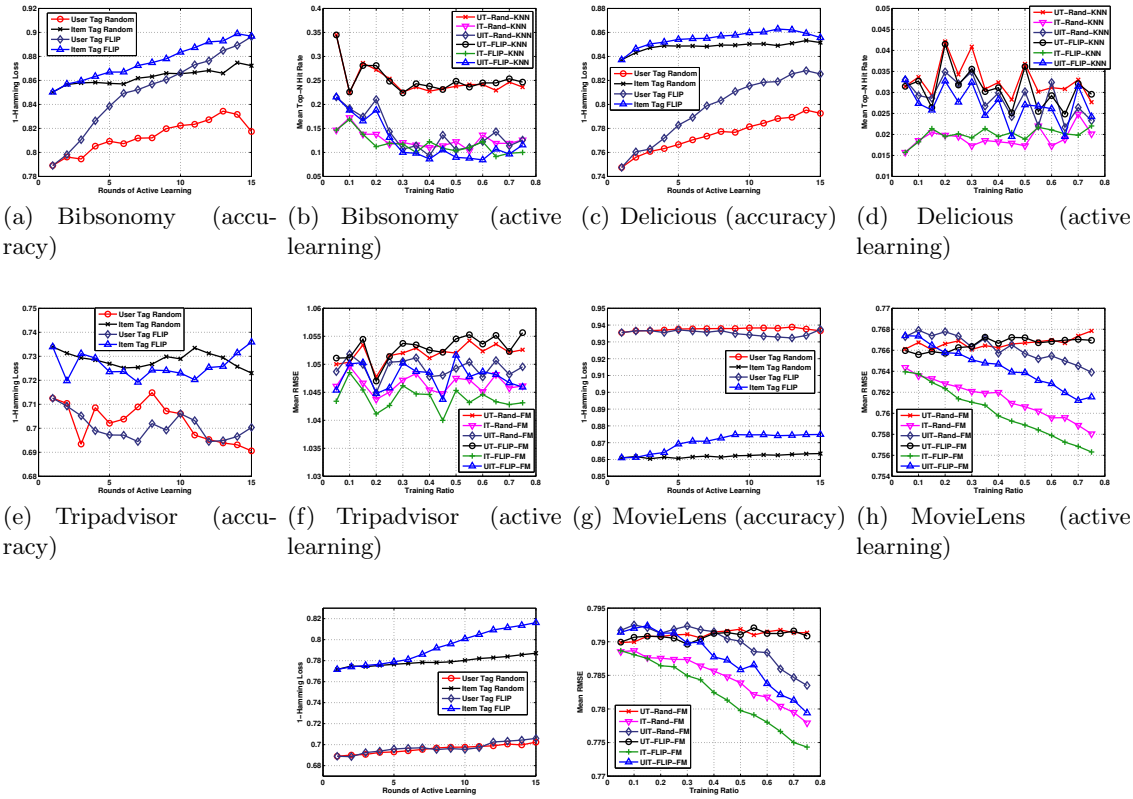
Figure 7.5: Tag Prediction Accuracy of Collective Classification and Performance of CF methods with respect to RMSE (↓) using random split for varying training ratio in collective classification.

7.5.3 Varying Training Ratio

For collective classification, I have assumed that we know the tags for 80% of the training users/items and tried to predict the tags of the rest 20% users/items (i.e., $b = 0.2$). To observe the effect of varying ratio of training users/items on the performance of UIT-KNN, UT-KNN, IT-KNN and UIT-FM, UT-FM, IT-FM, I have plotted the mean RMSE for Tripadvisor, MovieLens, LibraryThing and mean Top-N Hit rate for Bibsonomy and Delicious datasets. I have also plotted the multi-label accuracy metrics for tag prediction for users in all the three datasets. Figure 7.4(a)-(d) show that the multiple tag prediction accuracy of the collective classifier gradually increases with increasing training ratio varying from 0.1 through 0.9 with steps of 0.1, and so does the Top-N hit rate when we used standard split. However, for Tripadvisor data, since the multi-label accuracy stays the same irrespective of increasing training ratio, we do not see much variations in the RMSE scores (Figure 7.4(e)-(f)). Similarly, the RMSE scores decrease with inceasing training ratio for both MovieLens and LibraryThing datasets (Figure 7.4(h) and Figure 7.4(j)). Figure 7.5 shows the similar trends when we experimented with random split for train-test nodes.

7.5.4 Performance for Active Learning

Figure 7.6 and Figure 7.7 show the multi-label accuracy (1-Hamming Loss) for all the three datasets using (i) active learning with FLIP score criteria for predicting tags for users [47]; (ii) random sampling. In previous sections, I have shown that tags used as side information can definitely enhance the performance of recommendation system models. Figure 7.6 and Figure 7.7 show that using active learning is worthwhile in comparison to random sampling (using standard and random splits, respectively), in order to predict the tags accurately. The effect of FLIP based active learning is more prominent in reducing RMSE for MovieLens and LibraryThing datasets because these datasets have higher rating density than others, thereby, creating denser user-user and item-item implicit networks where collective classification is more effective. For example, considering the plot for LibraryThing dataset in Figure 7.6 and 7.7, we notice that with active learning in place only 65% of training data

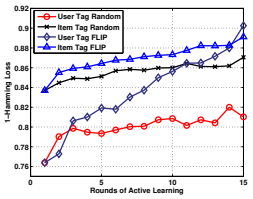


(a) Bibsonomy (accuracy) (b) Bibsonomy (active learning) (c) Delicious (accuracy) (d) Delicious (active learning)

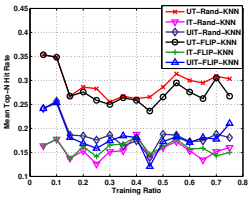
(e) Tripadvisor (accuracy) (f) Tripadvisor (active learning) (g) MovieLens (accuracy) (h) MovieLens (active learning)

(i) LibraryThing (accuracy) (j) LibraryThing (active learning)

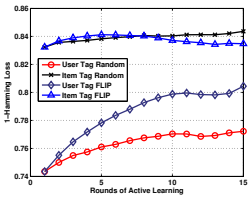
Figure 7.6: Tag Prediction Accuracy of Collective Classification w.r.t. 1-Hamming Loss (↑) with standard split using Active Learning with FLIP (best viewed in color print).



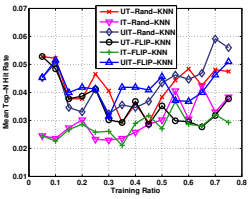
(a) Bibsonomy (accuracy)



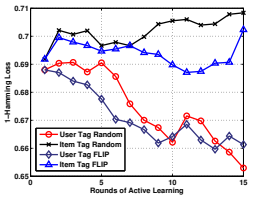
(b) Bibsonomy (active learning)



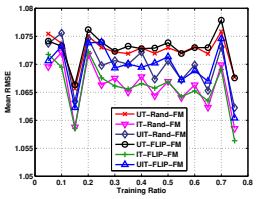
(c) Delicious (accuracy)



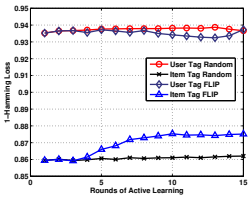
(d) Delicious (active learning)



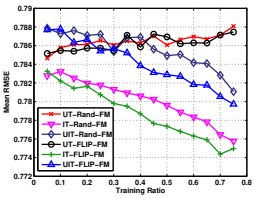
(e) Tripadvisor (accuracy)



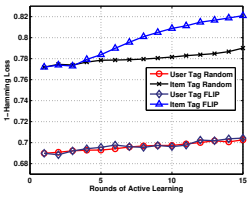
(f) Tripadvisor (active learning)



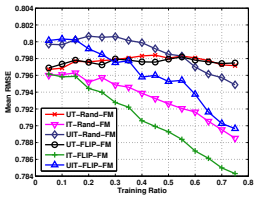
(g) MovieLens (accuracy)



(h) MovieLens (active learning)



(i) LibraryThing (accuracy)



(j) LibraryThing (active learning)

Figure 7.7: Tag Prediction Accuracy of Collective Classification w.r.t. 1-Hamming Loss (↑) with random split using Active Learning with FLIP (best viewed in color print).

was enough for collective classification to predict tags accurately that can compare with the RMSE scores achieved with 80% of the training data using no active learning.

7.5.5 Time Complexity

The time complexity of the multi-label collective classification algorithm for user and item tag predictions is given by $\mathcal{O}(m \times |P| \times maxiter) + \mathcal{O}(n \times |P| \times maxiter)$, where m is the total number of users, n is the total number of items in the system and $|P|$ is the number of tags used in the classification model and $maxiter$ is the number of iterations required for convergence of collective classification algorithms. According to Rendle [43], the time complexity of any tag-based latent factor model is $\mathcal{O}(F \times N_z(\mathbf{R}))$, where $N_z(\mathbf{R})$ gives the number of non-zero elements in the user-item rating matrix $\mathbf{R}$ and F is the dimensionality of factorization in FM. I have tracked the time consumption for the largest dataset we had, i.e., LibraryThing data and Table 7.7 gives the details on the timings for different algorithms. The time recorded here for the latent factor models is the combined time required for converting the $\langle u, i \rangle$ rating value into the feature vector $\mathbf{x}$ (Section 7.3.2 Equation (7.6)) and the prediction of ratings through matrix factorization using FM. As expected, the ground truth models take more time compared to their counterparts because the number of ground truth tags for the users/items are usually more than the predicted tags, thereby causing the feature vectors for the user-item pairs to have more non-zero values. Hence, this results in more time for learning the model parameters.

Table 7.7: Time consumption by different methods for LibraryThing dataset.

Method	**Time (seconds)**
UIT-FM	442.91412
UIT-FM-G	480.44012
UT-FM	304.48146
UT-FM-G	330.17284
IT-FM	253.8454
IT-FM-G	265.00208
FM	116.1661

Chapter 8: Conclusion and Future Work

8.1 Conclusion

In recent years, many real world datasets, especially social media datasets, have complex link structure which are not exploited by the traditional algorithms for network classification problems. To address this scenario, researchers have developed collective classification algorithms that leverage the topological structure of the datasets, and the complex correlations between the labels of samples and their neighbors. However, these methods relied heavily on the rich node features for training a classifier, which was further used to build a meta-classifier along with the relational features derived from the links. Hence, datasets without node features (e.g., social networks with no profile information from users) could not use these methods for node classification problems. Additionally, most of the collective classification algorithms work with single-labeled networks. However, given that most of the real world datasets are multi-labeled, there was a need for developing multi-label collective classification methods.

Extracting good samples for evaluating a collective classifier and acquiring training samples with fixed budget are two remotely linked topics. In the first case, we use the samples to evaluate the performance of a classifier, assuming that the collected samples are a good representation of the original data. Collecting these good samples is an important step. In the second case, a *fixed* number of samples are added to the training set at each step and a classifier is trained, thereby, making the classifier gradually stronger. The first procedure is sometimes referred as *selective sampling* or *active labeling* in literature, whereas, the second one is referred as *active learning* in general. Active labeling and active learning both are well-researched topics in machine learning. However, very little have been done

to bridge the gap between these two areas of work in the context of collective classification. Furthermore, research on active learning for networks has mostly dealt with datasets that have node features available during learning process, whereas, in real world there are plethora of networks for which node features are not accessible. In my thesis, I have aimed to develope an active learning strategy to work with a multi-label collective classifier that does not require node features during learning. I have also developed a variation of the forest fire sampling algorithm in order to improve the collective classification. These two projects helped to bridge the gap between active labeling and active learning that we have just discussed.

To extend the use of multi-label collective classifier and the active learning strategy, I have also developed a model in recommendation systems which can use tags (meta-labels predicted through collective classification, or available otherwise) as side information and improve the rating prediction performance for multiple real world datasets. This shows the broader spectrum to which my models can be applied in practice.

8.2 Contributions

Specifically, I have developed algorithms for improving the state-of-the-art multi-label collective classification algorithms to cope with more generalized version of the datasets available in real world. To evaluate the collective classifiers' generalization performance, I have proposed sampling techniques that are customized for network classification algorithms. I have also proposed multiple active learning strategies customized to work with multi-label collective classifiers. I have extended the application of active learning with multi-label collective classifier in the recommendation system framework. To summarize, my contributions are: (*i*) development of influence based multi-label collective classifier; (*ii*) development of sampling strategy for evaluating single-labeled collective classification; (*iii*) development of an active learning strategy on networks for spending the budget frugally `(FLIP-per-label)` when only a subset of all possible labels can be queried from an instance in multi-labeled

network; (*iii*) development of tag-based models for recommendation systems using multi-label collective classification and active learning. To the best of my knowledge, this thesis is the first to propose the use of multi-label collective classification in tag-based item recommendation systems. I have reported extensive experimental results on several real-world relational datasets for each of the developed models, demonstrating that these methods can give statistically significant performances.

8.3 Future Work

My work can be extended in several ways, as a part of future work in the following directions. The multi-label collective classifier used for active learning and for tag-based recommendation system is good enough to handle thousands of samples. But, when millions of samples are present then scalability becomes an issue. One major extension would be to parallelize the collective classification framework using Hadoop and Message Passing Interfaces (MPI). Secondly, the multi-label collective classifiers use one-vs-rest approach to train the meta-learner which involves considerable amount of time. In future, the one-vs-rest approach should be discarded to make the problem more scalable to solve. Thirdly, the sampling algorithms have been designed to treat single labeled network datasets only. It is worthwhile to design efficient sampling algorithms for multi-labeled network datasets as well. Networks often change over time, hence it makes sense to study collective classification in dynamic networks where labels of nodes change over time. A practical implementation of this scenario in tag-based recommender system would be the change in the choice of tags by an user over time. This calls for a study on temporal aspects of collective classification and tag-based item recommendation that is quite interesting. Finally, cold start problem is a well-known problem in recommender system. New user problem involves recommending products for a new user who has not rated any item before in the system. Similarly, new item problem involves situations when an item has not been rated by any user in the system. It is quite challenging to deal with cold-start situations in tag-based recommender systems, thus making it an interesting direction to pursue future work.

Bibliography

Bibliography

[1] N. Ahmed, F. Berchmans, J. Neville, and R. Kompella. Time-based sampling of social network activity graphs. In *Proceedings of the Eighth Workshop on MLG (2010)*, pages 1–9. ACM, 2010.

[2] N. Ahmed, J. Neville, and R. Kompella. Network sampling designs for relational classification. In *Sixth International AAAI Conference on Weblogs and Social Media*, 2012.

[3] M. Bilgic and L. Getoor. Effective label acquisition for collective classification. In *Proceeding of the 14th ACM SIGKDD*, pages 43–51. ACM, 2008.

[4] M. Bilgic, L. Mihalkova, and L. Getoor. Active learning for networked data. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[5] C. M. Bishop et al. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

[6] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on UAI*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

[7] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. In *ACM SIGMOD Record*, volume 27, pages 307–318. ACM, 1998.

[8] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[9] I. Dagan and S. P. Engelson. Committee-based sampling for training probabilistic classifiers. In *ICML*, volume 95, pages 150–157, 1995.

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[11] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, 22(1):143–177, 2004.

[12] M. Enrich, M. Braunhofer, and F. Ricci. Cold-start management with cross-domain collaborative filtering and tags. In *E-Commerce and Web Technologies*, pages 101–112. Springer, 2013.

[13] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer Series in Statistics, 2001.

[14] Y. Fu, B. Liu, Y. Ge, Z. Yao, and H. Xiong. User preference learning with multiple information fusion for restaurant recommendation.

[15] B. Gallagher and T. Eliassi-Rad. Leveraging label-independent features for classification in sparsely labeled networks: An empirical study. *Advances in Social Network Mining and Analysis*, pages 1–19, 2010.

[16] L. Getoor. Link-based classification. *Advanced methods for knowledge discovery from complex data*, pages 189–207, 2005.

[17] A. Goyal, F. Bonchi, and L. Lakshmanan. A data-based approach to social influence maximization. *Proceedings of the VLDB Endowment*, 5(1):73–84, 2011.

[18] A. S. Harpale and Y. Yang. Personalized active learning for collaborative filtering. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 91–98. ACM, 2008.

[19] T. Hofmann. Collaborative filtering via gaussian probabilistic latent semantic analysis. In *Proceedings of the 26th ACM SIGIR conference*, pages 259–266. ACM, 2003.

[20] D. Jensen, J. Neville, and B. Gallagher. Why collective inference improves relational classification. In *Proceedings of the tenth ACM SIGKDD*, pages 593–598. ACM, 2004.

[21] M. Ji and J. Han. A variance minimization criterion to active learning on graphs. AISTATS, 2012.

[22] M. Ji, J. Han, and M. Danilevsky. Ranking-based classification of heterogeneous information networks. In *Proceedings of the 17th ACM SIGKDD*, KDD '11, pages 1298–1306, New York, NY, USA, 2011. ACM.

[23] R. Jin and L. Si. A bayesian approach toward active learning for collaborative filtering. In *Proceedings of the 20th conference UAI*, pages 278–285. AUAI Press, 2004.

[24] X. Kong, X. Shi, and S. Philip. Multi-label collective classification. In *Proceedings of SIAM International Conference on Data Mining (SDM)*, pages 618–629, 2011.

[25] Y. Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 81, 2009.

[26] Y. Koren and R. Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*, pages 145–186. Springer, 2011.

[27] A. Kuwadekar and J. Neville. Relational active learning for joint collective classification models. In *Proceedings of the 28th (ICML-11), ICML*, volume 11, pages 385–392.

[28] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD*, pages 631–636. ACM, 2006.

[29] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD*, pages 177–187. ACM, 2005.

[30] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–12. Springer-Verlag New York, Inc., 1994.

[31] P. Lops, M. De Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[32] Q. Lu and L. Getoor. Link-based classification. In *Workshop at International Conference on Machine Learning (ICML)*, volume 20, page 496, 2003.

[33] S. Macskassy and F. Provost. Classification in networked data: A toolkit and a univariate case study. *The Journal of Machine Learning Research*, 8:935–983, 2007.

[34] S. A. Macskassy. Using graph-based metrics with empirical risk minimization to speed up active learning on networked data. In *Proceedings of 15th ACM SIGKDD.*, pages 597–606. ACM, 2009.

[35] A. McCallum, K. Nigam, et al. Employing em in pool-based active learning for text classification. In *Proceedings of ICML-98.*, pages 350–358, 1998.

[36] L. K. McDowell, K. M. Gupta, and D. W. Aha. Cautious inference in collective classification. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, page 596. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[37] A. Mnih and R. Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2007.

[38] J. Neville and D. Jensen. Iterative classification in relational data. In *Proc. AAAI-2000 Workshop*, pages 13–20, 2000.

[39] J. Neville and D. Jensen. Relational dependency networks. *The Journal of Machine Learning Research*, 8:653–692, 2007.

[40] J. Pfeiffer III, J. Neville, and P. Bennett. Active sampling of networks. In *10th International Workshop on Mining and Learning with Graphs*. http://www.cs.purdue.edu/homes/jpfeiff/pubs/ActiveSampling.pdf, 2012.

[41] C. Preisach, L. B. Marinho, and L. Schmidt-Thieme. Semi-supervised tag recommendation-using untagged resources to mitigate cold-start problems. In *Advances in Knowledge Discovery and Data Mining*, pages 348–357. Springer, 2010.

[42] A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl. Getting to know you: learning new user preferences in recommender systems. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 127–134. ACM, 2002.

[43] S. Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012.

[44] S. Rendle. Scaling factorization machines to relational data. In *Proceedings of the VLDB Endowment*, volume 6, pages 337–348. VLDB Endowment, 2013.

[45] T. Saha, H. Rangwala, and C. Domeniconi. Multi-label collective classification using adaptive neighborhoods. In *(ICMLA)*, volume 1, pages 427–432. IEEE, 2012.

[46] T. Saha, H. Rangwala, and C. Domeniconi. Sparsification and sampling of networks for collective classification. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, pages 293–302, 2013.

[47] T. Saha, H. Rangwala, and C. Domeniconi. Flip: Active learning for relational network classification. In *Machine Learning and Knowledge Discovery in Databases*, pages 1–18. Springer, 2014.

[48] T. Saha, H. Rangwala, and C. Domeniconi. Predicting preference tags to improve item recommendation. In *In submission*. SIAM SDM, 2015.

[49] R. Salakhutdinov and A. Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887. ACM, 2008.

[50] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[51] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th ACM SIGIR conference*, pages 253–260. ACM, 2002.

[52] P. Sen and L. Getoor. Link-based classification. *University of Maryland Technical Report CS-TR-4858*, 2007.

[53] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, 2008.

[54] B. Settles. Active learning literature survey. *University of Wisconsin, Madison*, 2010.

[55] L. Shi, Y. Zhao, and J. Tang. Batch mode active learning for networked data. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(2):33, 2012.

[56] Y. Shi, M. Larson, and A. Hanjalic. Tags as bridges between domains: Improving recommendation with tag-induced cross-domain collaborative filtering. In *User Modeling, Adaption and Personalization*, pages 305–316. Springer, 2011.

[57] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[58] L. Tang and H. Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD*, pages 817–826. ACM, 2009.

[59] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI02)*, pages 895–902. Citeseer, 2002.

[60] K. H. Tso-Sutter, L. B. Marinho, and L. Schmidt-Thieme. Tag-aware recommender systems by fusion of collaborative filtering algorithms. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1995–1999. ACM, 2008.

[61] G. Tsoumakas and I. Katakis. Multi-label classification. *International Journal of Data Warehousing & Mining*, 3(3):1–13, 2007.

[62] V. Vapnik. *The nature of statistical learning theory.* springer, 2000.

[63] X. Wang and G. Sukthankar. Multi-label relational neighbor classification using social context features. In *Proceedings of the 19th ACM SIGKDD*, pages 464–472, 2013.

[64] M. Zhang. Lift: Multi-label learning with label-specific features. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[65] M. Zhang and K. Zhang. Multi-label learning by exploiting label dependency. In *Proceedings of the 16th ACM SIGKDD*, pages 999–1008. ACM, 2010.

[66] M. Zhang and Z. Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, 2007.

[67] M. Zhang and Z. Zhou. A review on multi-label learning algorithms. 2013.

[68] Z.-K. Zhang, T. Zhou, and Y.-C. Zhang. Tag-aware recommender systems: a state-of-the-art survey. *Journal of computer science and technology*, 26(5):767–777, 2011.

[69] S. Zhao, N. Du, A. Nauerz, X. Zhang, Q. Yuan, and R. Fu. Improved recommendation based on collaborative tagging behaviors. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 413–416. ACM, 2008.

[70] Y. Zhen, W.-J. Li, and D.-Y. Yeung. Tagicofi: tag informed collaborative filtering. In *Proceedings of the third ACM conference on Recommender systems*, pages 69–76. ACM, 2009.

[71] X. Zhu, J. Lafferty, and Z. Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *ICML 2003 workshop*, pages 58–65, 2003.

Curriculum Vitae

Tanwistha Saha has spent most part of her life in the "City of Joy" Kolkata, in India. She received her Bachelor of Technology degree from Institute of Engineering and Management under University of Kalyani in India in Summer 2004. She has worked at the software firm Tata Consultancy Services (India) from Fall 2004 to Summer 2008. Since Fall 2008, she has been a PhD student at the Department of Computer Science in George Mason University, USA. She obtained her Master of Science degree in Computer Science from George Mason University in Spring 2011. Upon successful completion of her degree in December 2014, she will be joining Intel Corporation as Advanced Analytics Development and Deployment Engineer at their office in Hillsboro, in the suburbs of the beautiful "Rose City" Portland, Oregon, USA.