HIERARCHICAL LEARNING FOR LARGE MULTI-CLASS CLASSIFICATION IN NETWORK DATA

By

Lei Liu

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science - Doctor of Philosophy

2015

ABSTRACT

HIERARCHICAL LEARNING FOR LARGE MULTI-CLASS CLASSIFICATION IN NETWORK DATA

By

Lei Liu

Multi-class learning from network data is an important but challenging problem with many applications, including malware detection in computer networks, user modeling in social networks, and protein function prediction in biological networks. Despite the extensive research on large multi-class learning, there are still numerous issues that have not been sufficiently addressed, such as efficiency of model testing, interpretability of the induced models, as well as the ability to handle imbalanced classes. To overcome these challenges, there has been increasing interest in recent years to develop hierarchical learning methods for large multi-class problems. Unfortunately, none of them were designed for classification of network data. In addition, there are very few studies devoted to classification of heterogeneous networks, where the nodes may have different feature sets. This thesis aims to overcome these limitations with the following contributions.

First, as the number of classes in big data applications can be very large, ranging from thousands to possibly millions, two hierarchical learning schemes are proposed to deal with the so-called *extreme multi-class learning* problems [1]. The first approach, known as recursive non-negative matrix factorization (RNMF), is designed to achieve sublinear runtime in classifying test data. Although RNMF reduces the test time significantly, it may also assign the same class to multiple leaf nodes, which hampers the interpretability of the model as a concept hierarchy for the classes. Furthermore, since RNMF employs a greedy strategy to partition the classes, there is no theoretical guarantee that the partitions generated by the tree would lead to a globally optimal solution.

To address the limitations of RNMF, an alternative hierarchical learning method known as matrix factorization tree (`MF-Tree`) is proposed. Unlike RNMF, MF-tree is designed to optimize a global objective function while learning its taxonomy structure. A formal proof is provided to show the equivalence between the objective function of MF-tree and the Hilbert-Schmidt Independence Criterion (HSIC). Furthermore, to improve its training efficiency, a fast algorithm for inducing approximate MF-Tree is also developed.

Next, an extension of `MF-Tree` to network data is proposed. This approach can seamlessly integrate both the link structure and node attribute information into a unified learning framework. To the best of our knowledge, this is the first study that automatically constructs a taxonomy structure to predict large multi-class problems for network classification. Empirical results suggest that the approach can effectively capture the relationship between classes and generate class taxonomy structures that resemble those produced by human experts. The approach can also be easily parallelizable and has been implemented in a MapReduce framework.

Finally, we introduce a network learning task known as co-classification to classify heterogeneous nodes in multiple networks. Unlike existing node classification problems, the goal of co-classification is to learn the classifiers in multiple networks jointly, instead of learning to classify each network independently. The framework proposed in this thesis can utilize prior information about the relationship between classes in different networks to improve its prediction accuracy.

ACKNOWLEDGMENTS

The past five years PhD study at Michigan State University is a both painful and enjoyable life experience, just like climbing mountain, although it was a hard work for every step you move forward, I enjoy this journey and realized that I can not reach the mountain top enjoying the beautiful view without helps from my surrounding people. Though it is not enough to express my grateful in words to all of these people who helped me, I still would like to give them my most sincere thanks.

First and foremost, I would like to thank my PhD advisor, Dr. Pang-Ning Tan, who generously funded my PhD study with assistantships and internship opportunities. He offered me so much advices, patiently supervising me, and always guide me in the right and meaningful research direction. He has also provided many insightful discussions through our weekly individual meetings, group studies and data mining class. His guidance in both research and career have been priceless. Without his support and help, it is hard to image that I could finish this thesis. Besides research, his hard working nature and easy going personality have inspired me a lot, which I believe will continue benefit me throughout the rest life journey.

Besides my advisor, I would like to thank the rest of my PhD committee members, Prof. Rong Jin, Prof. Joyce Chai and Dr. Selin Aviyente for their support and guidance throughout my PhD programme. I would like to specially thank Dr. Jin for shaping my thinking during the Machine Learning class in Spring 2010. I also want to thank Prof. Joyce Chai and Dr. Selin Aviyente for providing brilliant comments and suggestions to my thesis. It was great honor to have worked with each of my PhD committee member and I sincerely thank them for all the help and guidance. I look forward to collaborating with each of my committee member in the future.

I want to thank present and past labmates in Michigan State University LINKS Lab, Prakash Mandayam Comar, Jianpeng Xu, Zubin Abraham, Courtland VanDam, Xi Liu, Shuai Yuan and Ding Wang. Especially thank Prakash Mandayam Comar for keeping my Lab and Intern hour interesting and insightful discussion of various research projects. My special thanks to our graduate director Eric Torng for his tremendous help and support over my time at MSU. I would like to thank our department secretaries Linda Moore and Norma Teague for all the administrative and conference travel works. I also want to give my sincere thanks to all my friends for keeping me happy at MSU.

Last but not least, special thanks to my family, words cannot express how grateful I am to my parents and sister. My hard working parents sacrifice their lives and support me and my sister for our PhD studies. Without their support, I would not have chance to study abroad and complete this thesis. At the end, I would like to express my appreciation to my beloved wife Sang Ni who has brought me fun and always cheer me up throughout my PhD study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1

Introduction

The rapid proliferation of network data in various domains such as computer science [2], computational biology [3] and social science [4], has attracted considerable interest in applying data mining techniques to study complex networks. Examples of network mining tasks include link prediction [5], community detection [6], and influence propagation [7]. The focus of this thesis is on predicting the class of each node in a network, a problem that is known as *node classification* [5], *link-based classification* [8], or *collective classification* [9] in the literature.

Node classification in a network is different than traditional classification, which assumes that the objects to be classified are independent and identically distributed (i.i.d). The i.i.d. assumption is inappropriate for network data as the class of a node depends not only on its attributes, but also on its relationship to other nodes in the network. Furthermore, the number of classes in a network can be extremely large, ranging from thousands to millions, a problem that is known as *extreme multi-class* learning [1]. Although there has been considerable research on extreme multi-class learning for i.i.d. data, none of them were specifically designed for network data. We termed the latter as *multi-class network learning* in this thesis.

1.1 Applications of Multi-class Network Learning

Below are four potential applications of multi-class learning in network data.

1.1.1 Prediction of User Interest in Social Networks

The growth of online social media has brought together billions of users who are actively engaged in daily conversations and information sharing. The widespread use of such websites is a marketing dream for advertisers who seek to predict the interest of users based on their demographic profile and friendship relations. Given the wide variety of user interests, this is a clear example of an extreme multi-class network learning problem.

1.1.2 Classification in Bibliographic Networks

With the massive number of scientific papers written in different research disciplines, finding relevant publications from the literature has become an increasingly challenging problem. An information system that can effectively organize the research papers is needed. Such a system must be capable of automatically classifying the millions of papers published annually into their respective categories based on the keywords used and citations among the papers.

1.1.3 Detecting Malware in Computer Network

Malicious software (or malware) automatically spreads from one device to another by covertly executing a software program without authorization from users. The infected devices can be used to steal sensitive information, send spam messages, launch denial-of-service attacks, and other malicious activities. Although many intrusion detection and prevention systems have been developed to monitor such activities, various techniques can be employed by hackers to evade detection. Despite such evasion techniques, one unavoidable aspect that is difficult to camouflage is that the infected devices must communicate back to their command and control servers. This has led to considerable interest in applying node classification techniques to detect infected devices in a networked en vironment. As there are many variants of computer viruses available, malware detection is another example of extreme multi-class network learning problem. In a previous study, I have developed a two-phase score propagation algorithm to predict the class label of each client IP in an HTTP communication network [2]. By jointly utilizing the IP-address connectivity graph ("who talks to whom") and flow level information, the proposed technique achieves a precision rate of about 90% when applied to a subnet of a large internet service provider.

1.1.4 Function Prediction in Protein Interaction Network

Proteins are the most essential macromolecule of life. Knowing the functionality of proteins is critical for developing new drugs, better crops, and the development of biofuels [10]. Since a protein usually interacts with other proteins in the biological system, one can model the relationship between proteins as an interaction network, where the nodes represent proteins and the links represent evidence of a shared function. Protein function prediction can thus be cast into a multi-class network learning problem.

1.2 Challenges of Multi-class Network Learning

This section presents some of the challenges in multi-class network learning that motivate the research directions of this thesis.

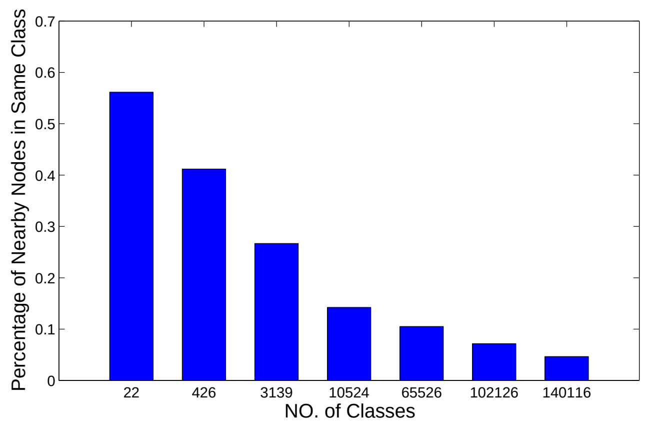


Figure 1.1 Fraction of neighboring nodes that belong to the same class as the the number of classes is varied.

1.2.1 Extreme Multi-Class Network Learning

Networks with large number of classes have become increasingly prevalent in recent years. For example, Wikipedia contains roughly 2.4 million articles from more than 600,000 classes. Building a classifier that can handle such a massive number of classes is challenging for the following reasons.

1.2.1.1 Training Accuracy

Learning a global classifier that accurately predicts all the classes is a challenging problem. Such a classifier must be extremely flexible and capable of fitting complex nonlinear decision surfaces, which in turn, makes it susceptible to the model overfitting problem. Furthermore, as the number of classes increases, the standard assumption that “nearby nodes in a network have similar classes” no longer holds, as illustrated in Figure 1.1. The figure was generated from the US Patent network data, where each node corresponds to a patent and each link corresponds to a citation between patents. The patents are assigned to classes, whose relationships are represented in a hierarchical structure. The number of classes can be varied by assigning the patents to classes at a particular depth in the hierarchy. The results suggest that the percentage of neighboring nodes that belong to the same class decreases from 56% to 4.6% as the number of classes increases from 22 to 140,116.

1.2.1.2 Testing Efficiency

Many traditional classifiers such as logistic regression, support vector machine, and boosting are primarily designed for binary class problems. A popular strategy to extend these methods to multi-class classification is to transform the data into multiple binary classification problems, using techniques such as one-versus-one and one-versus-all. In practice, these techniques are not scalable for handing extreme multi-class learning problems. For example, with the one-versus-one strategy, one has to apply $O(k^2)$ classifiers to predict the label of each test example, where k is the number of classes. The one-versus-one approach requires only $O(k)$ classifiers for testing, which is still expensive when the number of classes is extremely large. How to reduce the testing cost to sublinear time while maintaining good classification accuracy is one of the major challenges that must be addressed.

1.2.1.3 Imbalanced Class Distributions

Most classification methods work well when there are representative examples from each class in the training data. For imbalanced class problems, standard classifiers tend to be more biased toward accurate prediction of the larger classes instead of the rare classes, which are more interesting. Unfortunately, network data with imbalanced classes is very common in the real world. For example, the largest category in the U.S Patent citation network data contains 25,727 patents, while the smallest category has only 8 patents. How to deal with the highly imbalanced classes in network classification is another challenge that must be addressed.

1.2.1.4 Evolving Classes

Extreme multi-class learning problems typically arise in application domains where new classes are continuously generated as more data is collected (e.g., in malware detection or image classification). Unfortunately, most of the existing multi-class learning methods would assign their test examples to one of the existing classes in the training set. Developing multi-class learning methods capable of detecting new classes remains an open research problem.

1.2.1.5 Model Interpretability

Classification methods are often evaluated in terms of their model accuracies. For some applications, model interpretability is another important criteria, where it is insufficient to generate accurate models without providing an explanation of how the inference was derived. This is typically achieved by using sparse learning methods [11, 12] to identify the most relevant features characterizing the different classes. For extreme multi-class problems, many of the classes may not be independent of each other. In this case, another useful model interpretability criteria would be its ability to convey information about the relationships among the large number of classes. How to design an accurate classifier that produces such information is another key challenge that must be addressed.

1.2.2 Classification on Heterogeneous Networks

Most existing network classification tasks have focused on developing algorithms for a single network. The drawback of this approach is that information from a single network may not be sufficient to build an accurate classifier especially when the network has noisy attributes and incomplete link structure. This has led to the following question: Can we improve the performance of a classifier by integrating data from other related networks?

A previous study on spam detection in social media by Chen et al. [13] has shown that, instead of categorizing the social media postings alone (as spam or non-spam), the detection rate can be improved using information about the users who generated the posts. Another example is in collective classification for Wikipedia editor network [14], where the classification accuracy was found to improve when augmented with Wikipedia article network. Both of these applications are examples of heterogeneous network classification, where the nodes in the network have different types (e.g., humans and Web pages), each with their own set of attributes. The links of the networks also have different types, which makes the problem more challenging since each link type carries varying degree of information about the class similarity. Designing a technique that can handle different types of attributes and links is a challenge that must be addressed in heterogeneous network classification.

1.3 Thesis Contributions

This section summarizes the thesis contributions in terms of addressing the challenges due to the extreme multi-class and heterogeneous network learning problems described in the previous section.

First, two novel hierarchical learning methods were developed for extreme multi-class classification. Hierarchical learning [15, 16, 17, 18, 19] is an effective strategy to deal with large multi-class problems for several reasons. Not only does it yield a hierarchical structure that captures the relationships among the classes, it also helps to reduce testing time and alleviates the class imbalanced problem by allowing the rare classes to be combined with examples from other classes when building the classification model. Although there are some domains where the subclass-superclass relationships are known *a priori*, e.g., Wikipedia categorization, the class hierarchy is typically unavailable or incomplete. As a result, there have been growing interests in recent years to develop classification methods that automatically induce a hierarchy from data. For example, Platt et al. [15] employs a directed acyclic graph (dag) to combine multiple binary class classifiers into a multi-class classifier. Some methods such as filter tree [17] imposes an arbitrary tree structure on the classes and then learns a classifier to partition the training examples to follow their appropriate paths until they reach their assigned leaf nodes. Since the tree is arbitrarily constructed, they do not convey meaningful information about the relationship among the classes. Other methods such as top-down decision trees employ a greedy strategy to split the data into smaller subsets [20]. The models produced by such methods are also difficult to interpret because a class may be assigned to multiple leaf nodes in the hierarchy.

This thesis presents two novel hierarchical learning methods to address the challenges of extreme multi-class learning problems. The first method, called recursive non-negative matrix factorization (RNMF, in short) was designed to speed up the computation time for model testing. RNMF builds a structure known as a *label tree* by performing a soft partitioning of the classes in a way that minimizes an information theoretic loss function. A unique aspect of the proposed approach is that RNMF employs multiple one-class SVM models at its leaf nodes. This allows the classifier to detect new classes that are not present in its training data, a unique option that is not available in any existing hierarchical learning methods. By allowing the leaf node to contain more than one SVM models, this also helps to shorten the depth of the tree without significantly degrading its accuracy. The tree also allows the model to be applied in sublinear time.

One major limitation of RNMF is that there is no global objective function optimized by its tree-growing procedure. Instead, RNMF employs a greedy, divide-and-conquer approach to partition the classes based on a local objective function when constructing its tree. In addition, it allows a class to be split and assigned to multiple leaf nodes, which limits its interpretability as a class hierarchy. To address this issue, this thesis develops a novel method known as matrix factorization tree (MF-Tree, in short), which organizes the classes into a binary tree by solving a global objective function. A theoretical proof is given to demonstrate the equivalence between the objective function and a recently proposed criterion for taxonomy learning called Hilbert-Schmidt Independence Criterion (HSIC)[21][22][23][24]. This thesis demonstrates how HSIC can be modified to a supervised multi-class learning setting and shows the additive property of the function, which allows us to recursively decompose the optimization problem into multiple subproblems that can be organized into a binary tree structure. Despite these advantages, solving the subproblems at the top levels of the tree can be computationally prohibitive especially when the number of training examples and classes are extremely large. To address the scalability issue, an approximate MF-Tree induction algorithm is proposed, where the determination of class partitions at the higher levels of the tree is determined using a simpler approach instead of solving the more expensive matrix factorization problem. Experimental results suggested that the accuracy of the approximate MF-Tree approach is slightly lower than the original MF-Tree but it is between 20% - 40% faster (depending on how far down the tree the approximation scheme is adopted).

Building a hierarchical learning method for network data is challenging because the tree may partition the network into a large number of disjointed components. As the network becomes more disconnected, the link information is no longer useful for node classification. Another contribution of this thesis is to extend the MF-tree approach to handle network data. The proposed framework incorporates both the node attribute similarity and link connectivity into the matrix factorization framework. Although alternative network classification methods such as label propagation is available, these methods are ill-suited for extreme multi-class learning problems since the neighborhood assumption no longer holds for many of the nodes, as shown in Figure 1.1. The proposed MF-tree framework for network data is also trivially parallelizable. A MapReduce implementation of the framework was developed and tested on a large-scale Wikipedia network data that contains more than 600K classes. To the best of my knowledge, this is a data set with the largest number of classes that have ever been reported in the literature.

The final contribution of this thesis is in the development of a novel multi-class learning algorithm for heterogeneous networks. The approach jointly classifies a pair of networks with different types of attributes and links but share similar characteristics in terms of their class information. The effectiveness of the proposed co-classification method was demonstrated using Wikipedia article and editor network data. Empirical results showed that the proposed multi-class co-classification method is more effective than learning to classify the multiple classes in each network independently.

1.4 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents the background and related work of this thesis. Chapter 3 introduces the recursive non-negative matrix factorization (RNMF) approach for building a label tree in extreme multi-class learning problems [20]. Chapter 4 presents the matrix factorization tree (MF-Tree) approach and provides a theoretical proof to demonstrate the equivalence between the proposed objective function and the Hilbert-Schmidt Independence Criterion [21][22][23][24]. The MF-Tree approach is extended to network data in Chapter 5. Chapter 6 presents a novel multi-class co-classification framework for heterogeneous network data [14]. Finally, Chapter 7 summarizes the contributions of the thesis and presents the future directions of this research.

Chapter 2

Background & Related Work

This chapter introduces the research problem to be addressed in this thesis and presents the literature review on node classification, multi-class learning, and taxonomy learning.

2.1 Problem Definition

Consider a network, $\mathcal{N}$, which is a graph of interconnected nodes. Each node represents an entity that is characterized by a set of attributes (or features). Each node is also assumed to have a class label. Assuming the labels are known for a subset of nodes in the network, the node classification task is to assign a label to each of the remaining unlabeled nodes in the network. More formally, the problem can be defined as follows:

Definition 1 (Node Classification) *Let $\mathcal{X} \in \Re^d$ be the set of attribute values and $\mathcal{Y} = \{1, 2, \cdots, c\}$ be the set of discrete-valued class labels. Consider a network data $\mathcal{N} = (V, E, \mathcal{D})$, where $V = \{v_1, v_2, ...v_n\}$ is the set of nodes, $E \subseteq V \times V$ is set of links, and $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\} \subseteq (\mathcal{X} \times \mathcal{Y})^n$. Let $V_l \subset V$ be the set of nodes whose labels are known and $V_u = V - V_l$ be the set of unlabeled nodes. The goal of node classification is to infer a function $f : \mathcal{X} \to \mathcal{Y}$ that accurately assigns each node to its corresponding class label.*

When the number of classes, $c > 2$, this is also known as a *multi-class network learning* problem. We will use the terms multi-class node classification and multi-class network learning interchangeably throughout this thesis.

As the number of classes considered in this thesis can be very large, it would be useful to derive a hierarchical structure that captures the relationships among the classes.

Definition 2 (Hierarchical Multi-class Network Learning) *Let $\mathcal{Z} = \{1, 2, \cdots, p\}$ be the set of possible partitions of a class. Given a network data $\mathcal{G} = (V, E, \mathcal{D})$, where $\mathcal{D} \subseteq (\mathcal{X} \times \mathcal{Y})^n$, the goal of hierarchical multi-class network learning is to infer a set of classification functions $\Phi = \{f | f : \mathcal{X} \to \mathcal{Z}\}$, which are organized in a hierarchical structure induced from the given data set $\mathcal{D}$.*

The hierarchical structure is often represented in the form of a directed acylic graph or a rooted tree. This thesis focuses on a special type of hierarchical structure known as a *label tree* [18] in the literature.

Definition 3 (Label Tree) *A label tree [18] is a hierarchical structure $T =< \mathcal{V}, \mathcal{E}, \Phi, \Lambda >$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is the set of directed links connecting each node to its children, Φ is the set of classification functions, and Λ is the set of class labels.*

Let $v_0 \in V$ denote the root node, which has no incoming links, and $V_{\text{leaf}} \subset V$ denote the leaf nodes, which have no outgoing links. Furthermore, $V_{\text{int}} = \mathcal{V} - V_{\text{leaf}}$ are the internal nodes of the tree. Each internal node is associated with a classification function $f \in \Phi$ while each leaf node is associated with a class label $y \in \Lambda$. If $(v_i, v_j) \in \mathcal{E}$, then v_i is a superclass of v_j (or equivalently, v_j is a subclass of v_i).

2.2 Node Classification

Node classification in network data [25, 26, 27, 8] is an active research problem that has received considerable attention from both industry and academia. The problem is also known collective classification in the network mining literature [28]. Previous methods developed for node classification can be divided into 3 categories—attribute-based, link-based, and hybrid methods.

Attribute-based methods employ a set of attributes that characterize each node to learn their classification functions. The attributes are either explicit properties of the node themselves or derived from their neighborhood information. One of the earliest examples of using attribute-based methods is in Web page categorization, where the goal was to automatically classify Web pages based on the document content, including their HTML tags [29]. Furnkranz [30] also investigated the use of anchor text along with the words near the anchor text to predict the class label of web pages. Attribute-based methods are also popular for node classification in security related applications. For example, Ma et al. [31] combined lexical and host-level features to build statistical models for classifying malicious URLs whereas Le et al. [32] relied on the lexical features to detect phishing Web sites. One practical advantage of using attribute-based methods is that we can apply existing classification techniques, such as support vector machine (SVM), decision tree classifier and logistic regression, to the node attributes to determine their classes. The drawback of these methods is that the link structure is not explicitly included in the learning scheme.

Link-based methods, on the other hand, incorporates the link structure directly into the modeling approach. For example, the HITS algorithm [33] performs web page classification by exploiting the hyperlink topology. PageRank [34] based link analysis methods have also been widely used in web page classification and for malicious node detection [35]. Graph-based semi-supervised methods [36][37][38][35] is another popular link-based method for node classification using partially labeled data to label the remaining nodes. These methods are designed to optimize the following two criteria:

- The predicted class of labeled nodes in the network should agree with their true classes.
- Two nodes that are connected should share similar labels.

Zhu and Ghahramani [37] introduced the label propagation approach to transmit the labels from an initial set of seed nodes to other nodes in the graph. A link-based method has also been proposed for spam detection in [38], where an initial set of seed nodes were labeled as benign (non-malicious) and their labels are propagated to the rest of the graph until they converged. Nodes with low (benign) scores are then predicted as spam. A similar approach was developed in [38] to propagate the "malicious" scores from known malicious nodes to other nodes in the entire graph.

Finally, there have been extensive studies using a hybrid method of both attribute- and link-based information to improve node classification. For example, Yang et al. [39] showed that extracting meta data from related web sites is useful for improving the accuracy of hypertext classification. Zhu et al. [40] developed a matrix factorization algorithm that exploits both the content and linkage information to address the Web page classification problem. There have also been quite extensive works on using graph regularization methods to combine link structure with content information. For example, Zhang et al. [25] developed a risk minimization formulation for learning from both text and graph structures. In [41], Lu and Getoor designed a structured logistic regression model to capture both content and link information. [27] introduce a matrix alignment based approach to integrate the network node attributes and links information, which weights the attributes and links according to their influence on the classification decision. Gan and Suel [42] proposed a two-stage approach to improve a Web spam classifier's performance using the link structure.

2.3 Classification on Heterogeneous Networks

Although node classification is a well-studied problem, there has been very few research focusing on the classification of nodes in heterogeneous networks. The key assumption in heterogeneous network classification is that there are useful information in other related networks that may aid in the classification of nodes in a given network. For example, consider the problem of classifying Wikipedia articles. Although it is possible to develop attribute-based, link-based, and hybrid methods for classifying the articles based on the the Wikipedia article network alone, there are other related networks, e.g., Wikipedia editor network, that could be augmented to enhance the classification accuracy.

Current research on node classification in heterogeneous networks can be classified into two categories. The first category learns a unified classification function for the combined networks. For example, Ji et al. [43] employed a link-based approach to classify nodes in heterogeneous networks using a graph-based algorithm similar to [36]. Using network data from a bibliography network (DBLP), the authors showed that classification accuracy can be significantly if we incorporate data from the entire heterogeneous information network.

The second category learns a separate classification function for each underlying network but trains the multiple functions simultaneously. This strategy is equivalent to a multi-task learning approach for solving multiple related prediction problems [44]. For example, Chen et al. [13] presented a co-classification approach to simultaneously classify spammers in a user network and spam in a Web page graph. Their approach was inspired by the fact that both Web spam and spammer detection tasks are intuitively related, as spam content are more likely contributed by spammers than by non-spammers. Nevertheless, the method proposed in [13] is for binary class problems only. One of the contributions of this thesis is to extend the co-classification framework to multi-class network learning problems. The framework proposed in this thesis also allows the incorporation of side information (in the form of a class prior matrix) to bias the learning algorithm.

2.4 Multi-class Learning

Classification with large number of classes is an important but challenging problem, and has received increasing attention in recent years [45][46][47][19][48][49]. Existing methods typically employ a collection of binary classifiers, organized either in a flat (non-hierarchical) or a hierarchical (tree) structure.

2.4.1 Non-Hierarchical Methods

Traditional non-hierarchical learning methods, including one-versus-all [50], one-versus-one [51], and error-correcting output coding (ECOC) [52], require invocation of all the binary classifiers during test time in order to make their final prediction. Both one-versus-one and one-versus-all methods are inefficient when the number of classes is large due to the linear and quadratic number of binary classifiers that must be invoked during testing. The ECOC method assigns a unique m bit codeword to each class label (where $m > \log_2 c$) and trains a binary classifier to predict each bit of the codeword. Given a test instance, we must generate an m-bit vector by applying the m binary classifiers and then compute its distance to the codewords of all c classes, which is an expensive operation, to determine its predicted class. Compressed sensing based ECOC methods [53][54] have been proposed as an alternative. Such methods consider the sparsity of the output label space to ease the burden of large-scale multi-label learning. The authors concluded that, by incorporating compressed sensing into ECOC, the number of subproblems to be solved is logarithmic in the number of classes, thereby reducing the number of binary classifiers to be invoked during testing. Nonetheless, these methods are non-hierarchical as they do not generate a concept hierarchy to relate the classes. They also do not consider issues such as new class detection and the class imbalance problem, which are typically encountered in extreme multi-class learning problems.

2.4.2 Hierarchical Methods

Hierarchical methods [15, 16, 17, 18, 19] help to reduce the number of classifiers that need to be invoked during testing by organizing the classifiers in a rooted, hierarchical structure. Their runtime complexity for testing depends on the maximum depth of the hierarchy. Existing hierarchical methods can be divided into two categories, depending on how the hierarchical structure is created.

In the first category, a hierarchical structure is arbitrarily imposed on the classes. Each intermediate node in the hierarchy contains a binary classifier, which is trained to partition the classes in a way that is consistent with the imposed structure. One example is the filter tree method developed by Beygelzimer et al. [17], which learns the tree structure in a bottom-up fashion. This method was adapted in [55] to build a conditional probability tree for the labels. Another example is the Decision Directed Acyclic Graph (DDAG) [15] approach, which arranges a set of one-versus-one classifiers into a rooted binary DAG in order to reduce the test complexity from $O(c(c-1)/2)$ to $O(c)$. In all three cases, there is no hierarchical learning involved when the models are trained.

In the second category, the hierarchical structure is automatically inferred when building the multi-class models. For example, Bengio et al. [18] proposed a label embedding tree approach that learns c one-vs-all classifiers to obtain an initial confusion matrix, which was subsequently used as the affinity matrix for applying spectral clustering to partition the classes into smaller subgroups. Deng et al. [19] presented an approach that simultaneously learns the class partitioning and the weights of the linear classifiers by optimizing a joint objective function. However, the approach requires solving an integer programming problem, which is NP-hard. Instead, they relaxed the optimization problem into a linear program in order to find a polynomial time solution. Shi et al. [56] also introduced a relaxed hierarchical learning approach for large-scale visual recognition. At each node, the classes are colored as positive or negative by a binary classifier while a subset of the confusing classes are ignored. The coloring of classes and training of binary classifiers were performed simultaneously using a principled max-margin optimization method.

2.5 Taxonomy Learning

Taxonomy learning, which aims to encode the conceptual relationship among the classes using an unsupervised learning approach, is another related area of this thesis. A classical method for taxonomy learning is by using agglomerative hierarchical clustering methods such as single-link and complete-link to derive the hierarchical relationships. Another approach is to construct the taxonomy with a divisive (top-down) partitioning approach. For example, Kuang et al. [57] developed a hierarchical clustering method that recursively partitions the data into two subsets using a rank-2 nonnegative matrix factorization approach. Their proposed method also performs outlier detection to avoid generating too many small leaf nodes. Although the method resembles the matrix factorization based methods developed in this thesis, it was unsupervised and there is no global objective function solved by their algorithm. Hilbert-Schmidt Independence Criterion (HSIC) metric is another recently proposed criterion for unsupervised taxonomy learning [21][22][23][24]. Inspired by this metric, in this thesis, we demonstrate how HSIC can be applied to a supervised multi-class network learning problem.

2.6 Summary

This section presents a formal definition of the multi-class network learning problem and reviews some of the previous works related to this research. Although there has been quite an extensive study on node classification in networks, they were mostly focused on binary classification problems and were applied to a single network. The primary goal of this thesis is to address the existing limitations and expands the use of hierarchical multi-class learning methods to network data.

Chapter 3

Recursive NMF for Extreme Multi-Class Learning

As noted in Chapter 1, extreme multi-class learning is a challenging problem due to issues such as testing complexity, imbalanced class distributions, presence of new classes in the test set, and interpretability of the models. To overcome these challenges, this chapter describes a novel label tree learning algorithm called recursive non-negative matrix factorization (RN-MF) for extreme multi-class learning problems. Key advantages of RNMF include:

1. Its testing complexity is sublinear in the number of classes.

2. It can detect the presence of new classes in the test data.

RNMF is a label tree construction algorithm that recursively partitions the training data into smaller subsets based on their classes and attribute values until a stopping criterion is satisfied. At each recursive step, it learns a classifier to predict which partition to assign a given test example. RNMF allows for soft partitioning of the classes by minimizing an information-theoretic loss function, which can be cast into a regularized non-negative matrix factorization problem. The regularization term introduced by our framework is unique in that it minimizes the entropy of the partitions. Unlike other label tree learning methods, each leaf node at the bottom of the tree contains multiple one-class SVM models to distinguish instances from different classes. Experimental results suggest that the induced tree achieves accuracy comparable to more complex tree learning methods but is considerably shorter, thus reducing its overall testing time.

3.1 Label Tree Structure for RNMF

Let $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ denote a set of training examples sampled from a distribution P over $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X} = \mathcal{R}^d$ is the d-dimensional feature space and $\mathcal{Y} = \{1, 2, ...c\}$ is the set of classes. The goal of multi-class classification is to learn a function f that minimizes the classification error of any instance drawn from P, i.e., $\mathbf{Pr}_{(x,y)\sim P}[f(x) \neq y]$. A typical approach for multi-class learning is to reduce the problem into multiple binary classification tasks. However, when the number of classes is large, applying all the binary classifiers to determine the predicted class becomes computationally infeasible.

RNMF provides a systematic approach to construct the binary classifiers and arrange them in a label tree structure (see Definition 3). An illustration of the label tree structure is shown in Figure 3.1. The label tree obtained from RNMF differs from conventional decision trees in the following ways. First, conventional decision tree classifiers employ a decision stump (i.e., a classifier consists of only one splitting attribute) at each internal node of the tree whereas RNMF considers linear combination of the attributes to partition the input space. The weights of the attribute combination are obtained by solving a regularized non-negative matrix factorization problem. A regularization term is introduced to prevent the tree from distributing examples from one class to more than one children during the tree construction process. Second, each leaf node of conventional decision tree classifiers contains a class label whereas each leaf node of RNMF contains one or more 1-class SVM classifiers [58]. By allowing the leaf nodes to contain multiple classifiers, this helps to reduce the depth of the

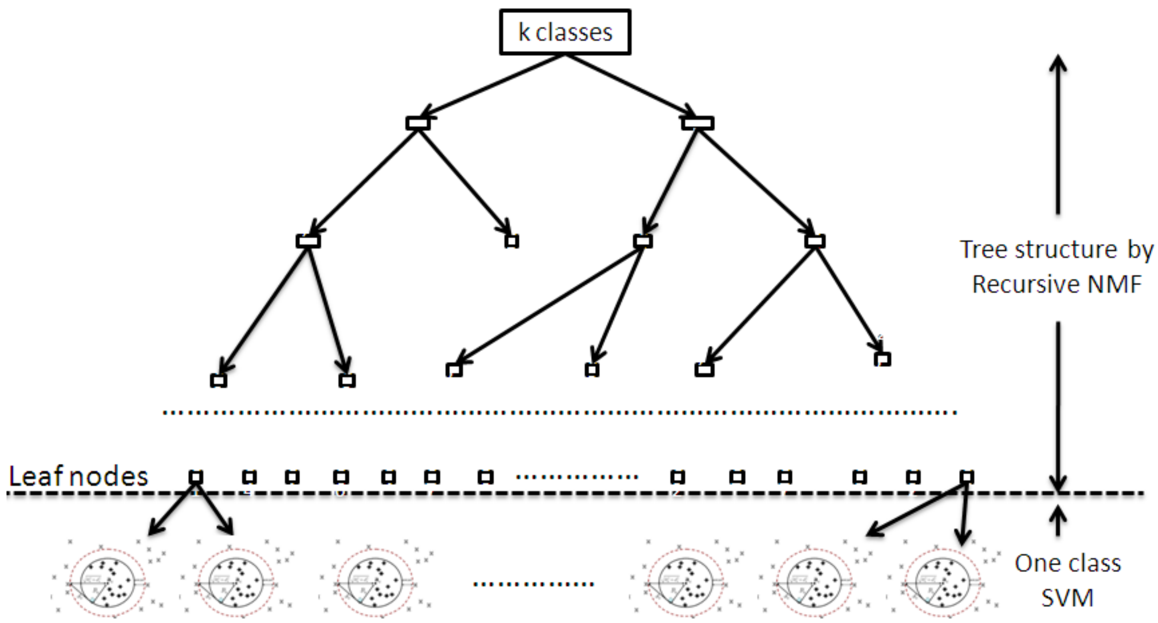


Figure 3.1 Proposed label tree structure using the RNMF algorithm.

tree without significantly degrading its accuracy. In 1-class SVM, the training examples from each class are encapsulated in a hypersphere constructed in an appropriate feature space. Once the enclosing hyperspheres for all the classes at a leaf node have been constructed, the test examples are classified based on their distance to the center of each hypersphere. Furthermore, the 1-class SVM models can be used to detect the presence of new classes.

The label tree structure generated by RNMF is parameterized by its branching factor (p) and maximum depth (d). As will be described in the next section, the two parameters can be varied by the user to alter the size of the label tree. For example, by choosing a branching factor larger than 2, this may result in purer splits among the children, thus reducing the maximum depth needed to achieve high accuracy. Decreasing the maximum depth parameter, on the other hand, may result in leaf nodes that contain examples that belong to a larger number of classes. This, in turn, may increase the number of one-class SVM models needed to achieve high accuracy.

3.2 RNMF Algorithm for Label Tree Construction

Let $\mathbf{X}$ be an $n \times d$ data matrix containing the feature vector of the training examples and $\mathbf{Y}$ be an $n \times c$ matrix that represents their corresponding class labels, i.e., $\mathbf{Y}_{ij} = 1$ if the i^{th} training example belongs to the j^{th} class, and zero otherwise. Similar to existing decision tree classifiers, RNMF seeks to recursively partition the training examples into purer subsets such that examples that belong to the same class should be assigned to the same partition. Specifically, at the each node v, our goal is to learn a partition matrix $\mathbf{L}$ of size $n \times p$, where n is the number of training examples assigned to the node and p is the branching factor, as well as the associated feature weight matrix $\mathbf{W}$, which is a $d \times p$ matrix that indicates the importance of each feature in partitioning the training examples. Each element L_{jq} represents the probability that the j^{th} training example should be assigned to the q^{th} partition at the given node v. To ensure $\mathbf{L}$ can be interpreted as probabilities, each column in $\mathbf{L}$ must sum up to 1. This can be achieved by adding the constraint $\mathbf{L}^T 1_n = 1_p$ into the objective function, where 1_p denote a p-dimensional column vector of all 1s. Furthermore, a regularization term is introduced to prevent training examples from the same class being assigned to different partitions. The partitions created at each node v are obtained by minimizing the following objective function:

$$\begin{aligned} \min_{\mathbf{L}\geq 0, \mathbf{W}\geq 0} \quad & D(\mathbf{X} \parallel \mathbf{L}\mathbf{W}^T) + \lambda H(\mathbf{L}^T\mathbf{Y}) \\ s.t. \quad & \mathbf{L}^T \mathbf{1}_n = \mathbf{1}_p \end{aligned} \tag{3.1}$$

where $D(\mathbf{X}||\mathbf{L}\mathbf{W}^T)$ refers to the Kullback-Leilber divergence between the input data matrix $\mathbf{X}$ and the product of its latent factors $\mathbf{L}\mathbf{W}^T$. The regularizer term $H(\mathbf{L}^T\mathbf{Y})$ corresponds to the entropy of the class distribution of each partition and is needed to avoid splitting the data from the same class to different partitions, i.e.,

$$H(\mathbf{L}^T\mathbf{Y}) = -\sum_{i=1}^{p} \frac{(\mathbf{L}^T\mathbf{Y})_{i+}}{(\mathbf{L}^T\mathbf{Y})_{++}} \sum_{j=1}^{c} \frac{(\mathbf{L}^T\mathbf{Y})_{ij}}{(\mathbf{L}^T\mathbf{Y})_{i+}} \log \frac{(\mathbf{L}^T\mathbf{Y})_{ij}}{(\mathbf{L}^T\mathbf{Y})_{i+}},$$

where $(\mathbf{L}^T\mathbf{Y})_{i+} = \sum_j (\mathbf{L}^T\mathbf{Y})_{ij}$ and $(\mathbf{L}^T\mathbf{Y})_{++} = \sum_{ij} (\mathbf{L}^T\mathbf{Y})_{ij}$. Note that entropy increases when the class distribution within each partition becomes more uniform and decreases when the distribution skews towards a small subset of the classes. Furthermore, since $\mathbf{Y}1_k = 1_n$, therefore, $\mathbf{L}^T\mathbf{Y}1_k = \mathbf{L}^T 1_n = 1_p$ (for a feasible solution that satisfies the constraint given in (3.1)). As a result, $\forall i : (\mathbf{L}^T\mathbf{Y})_{i+} = 1$ and $(\mathbf{L}^T\mathbf{Y})_{++} = p$. The entropy calculation can be simplified as follows:

$$H(\mathbf{L}^T\mathbf{Y}) = -\frac{1}{p}\sum_{i,j} (\mathbf{L}^T\mathbf{Y})_{ij} \log(\mathbf{L}^T\mathbf{Y})_{ij}$$

Note that λ is a user-specified parameter that controls the trade off between optimal partition of X and class purity of each partition[1]. The Lagrange formulation of (3.1) is given by:

$$\begin{aligned} \mathcal{L} &= \sum_{i,j} [X_{ij} \log \frac{X_{ij}}{(LW^T)_{ij}} - X_{ij} + (LW^T)_{ij}] \\ &- \lambda \sum_{ij} (L^T Y)_{ij} \log(L^T Y)_{ij} - \sum_s \mu_s (\sum_m L^T_{sm} - 1) \end{aligned}$$

The objective function is minimized by taking its partial derivative with respect to the

[1] λ is set to 1 in our experiments, but it can also be determined via cross-validation.

parameters W and L and setting them to zero:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_{pq}} &= -\sum_i \frac{X_{ip}L_{iq}}{(LW^T)_{ip}} + \sum_i L_{iq} = 0 && (3.2)\\ \frac{\partial \mathcal{L}}{\partial L_{pq}} &= -\sum_j \frac{X_{pj}W_{jq}}{(LW^T)_{pj}} + \sum_j W_{jq} - \lambda \sum_j Y_{pj}\log(L^TY)_{qj} - \lambda \sum_j Y_{pj} - \mu_q \\ &= 0 && (3.3)\end{aligned}$$

By summing up over the variable p in Equation (3.3), we have

$$\mu_q = - \frac{1}{n}\sum_{jp} \frac{X_{pj}W_{jq}}{(LW^T)_{pj}} + \sum_j W_{jq} - \lambda \sum_{jp} \frac{Y_{pj}}{n}\log(L^TY)_{qj} - \lambda \qquad (3.4)$$

Replacing (3.4) back into (3.3), the partial derivative can be rewritten as follow:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial L_{pq}} &= -\sum_j \frac{X_{pj}W_{jq}}{(LW^T)_{pj}} + \frac{1}{n}\sum_{jp} \frac{X_{pj}W_{jq}}{(LW^T)_{pj}} - \lambda \sum_j Y_{pj}\log(L^TY)_{qj} \\ &\quad + \lambda \sum_{jp} \frac{Y_{pj}}{n}\log(L^TY)_{qj} && (3.5)\end{aligned}$$

The objective function can be solved using a gradient descent approach. Following the approach used in [59], the gradient descent formula can be transformed into a multiplicative update formula as follows:

$$\begin{aligned}W_{pq} &= W_{pq}\frac{(X^TL)_{pq}}{(WL^TL)_{pq}} && (3.6)\\ L_{pq} &= L_{pq}\frac{\alpha}{\beta}, && (3.7)\end{aligned}$$

Algorithm 1 Recursive NMF at a node v

```
Input: Matrices $\mathbf{X}_v$, $\mathbf{Y}_v$, and $MaxIter$
Output: Matrices $\mathbf{W}_v$ and $\mathbf{L}_v$
Initialize: $\mathbf{W}_v^{old}$ and $\mathbf{L}_v^{old}$ to random matrices
for j=1 to $MaxIter$ do
    update $\mathbf{W}_v^{new}$ using (3.6)
    update $\mathbf{L}_v^{new}$ using (3.7)
    set $\mathbf{W}_v^{old} \leftarrow \mathbf{W}_v^{new}; \mathbf{L}_v^{old} \leftarrow \mathbf{L}_v^{new}$;
end for
```

where

$$\begin{aligned} \alpha &= \sum_j \frac{X_{pj} W_{jq}}{(LW^T)_{pj}} + \lambda \sum_j Y_{pj} \log(L^T Y)_{qj} \\ \beta &= \frac{1}{n} \sum_{jp} \frac{X_{pj} W_{jq}}{(LW^T)_{pj}} + \lambda \sum_{jp} \frac{Y_{pj}}{n} \log(L^T Y)_{qj} \end{aligned}$$

By initializing **W** and **L** to be non-negative matrices, the update formula ensures that the solution will also be non-negative.

Algorithm 1 summarizes the key steps of the proposed recursive non-negative matrix factorization (RNMF) algorithm for finding the partitions for any given node in the tree. The recursive partitioning step continues until one of the following stopping criteria is met: (1) stop if all the training examples in a partition belong to the same class, or (2) stop if the node contains less than $minleaf$ training examples, or (3) stop if there is only one class containing more than $minclass$ examples[2].

Once a leaf node is encountered, the algorithm will construct a one-class SVM model for each class i by learning a hypersphere to encapsulate all class i training examples that reach the particular leaf node. The hypersphere for class i is constructed by first labeling the

[2]We set $minleaf = 15$ and $minclass = 5$ for our experiments.

instances belonging to class i as $+1$ and those that belong to other classes as -1. We then construct two concentric hyperspheres such that the inner sphere encloses as many instances from class i as possible. The outer sphere is constructed in such a way that instances that do not belong to class i lie outside of it. The radial distance between the two hyperspheres is called the classifier's margin, which defines the objective function to be optimized by the 1-class SVM learning algorithm. For example, the hypersphere for predicting class k is obtained by solving the following [60]:

$$\begin{aligned}
\min_{R_k, a_k, d_k, \xi_i, \xi_l} \quad & R_k^2 - M d_k^2 + \frac{C}{N_k} \sum_{i:y_i=k} \xi_i + \frac{C}{N_{\bar{k}}} \sum_{l:y_l \neq k} \xi_l \\
\text{subject to} \quad & \| x_i - a_k \|^2 \leq R_k^2 + \xi_i, \quad \forall i : \ y_i = k \\
& \| x_l - a_k \|^2 \geq R_k^2 + d_k^2 - \xi_l, \quad \forall l : \ y_l \neq k \\
& \xi_i \geq 0, \quad \forall i : \ y_i = k \\
& \xi_l \geq 0, \quad \forall l : \ y_l \neq k
\end{aligned} \tag{3.8}$$

where N_k and $N_{\bar{k}}$ are the number of instances that belong to the k^{th} class and its complement, respectively. Here, the positive examples are enumerated by indices i and the negative examples by indices l. ξ_i and ξ_l are the slack variables while $C \geq 0$ controls the penalty for misclassification errors. The variables a_k and R_k represent the center and radius of the hypersphere constructed for the k^{th} class while $\sqrt{(R_k^2 + d_k^2)}$ defines the radius of the outer hypersphere. It can shown that the margin width is given by $\sqrt{(R_k^2 + d_k^2)} - R_k$. To maximize the margin, we need to maximize d_k^2 and minimize R_k^2 simultaneously, and the parameter $M \geq 0$ controls the trade-off between those two terms. The Wolfe dual form of the preceding optimization problem is given below:

$$
\begin{aligned}
\max_{\alpha} \quad & \sum_{i:y_i=k} \alpha_i K(x_i, x_i) - \sum_{l:y_l \neq k} \alpha_l K(x_l, x_l) \\
& - \sum_{i,j;y_i,y_j=k} \alpha_i \alpha_j K(x_i, x_j) \\
& - \sum_{l,m:y_l,y_m \neq k} \alpha_l \alpha_m K(x_l, x_m) \\
& + 2 \sum_{i,l:y_i=k,y_l \neq k} \alpha_i \alpha_l K(x_i, x_l) \\
\text{subject to} \quad & \sum_{i:y_i=k} \alpha_i = 1 + M, \quad 0 \leq \alpha_i \leq \frac{C}{N_k} \\
& \sum_{l:y_l \neq k} \alpha_l = M, \quad 0 \leq \alpha_l \leq \frac{C}{N_{\bar{k}}}
\end{aligned}
\tag{3.9}
$$

where $K(x, x')$ is a kernel function that measures the similarity between the pair of instances, x and x'. Among the widely used kernel functions include

$$
\begin{aligned}
\text{RBF Kernel:} \quad & K(x, x') = \exp\left[- \frac{\| x - x' \|^2}{2\sigma^2} \right] \\
\text{Polynomial Kernel:} \quad & K(x, x') = (1 + x^T x')^q
\end{aligned}
$$

The optimization problem given in (3.9) can be solved using quadratic programming.

After the label tree is constructed, we can apply the tree to classify unlabeled examples in the following way. Starting from the root node of the tree, we apply the feature weight matrix $\mathbf{W}$ to determine which partition the unlabeled example $\mathbf{x}_{\text{test}}$ should be sent. Specifically, we compute a partition membership vector $\boldsymbol{\pi}$ and assign the test example to the partition j

that has the largest magnitude, i.e.,

$$j = \arg\max_k \boldsymbol{\pi}_k, \quad \text{where} \quad \boldsymbol{\pi} = \mathbf{x}_{\text{test}}^T \mathbf{W}_v (\mathbf{W}_v^T \mathbf{W}_v)^{-1}$$

The procedure is repeated until the test example reaches one of the leaf nodes, where we can apply the 1-class SVM models associated with the leaf node to predict its label. Specifically, if the test example is encapsulated in the hypersphere for class i, then the test example is assigned to the class. However, if the test example is located within more than one hypersphere, we compute the distance between the test example to the hypersphere centers and assign it to the hypersphere with the smallest distance. Finally, if the test example is not encapsulated by any hypersphere, we declare it as belonging to a new class.

3.3 Experimental Evaluation

We conducted our experiments on the Wikipedia data dump generated on October 9, 2009. After preprocessing, there were about 3.6 million Wikipedia articles in the data, which were classified into 336K categories. We created two subsets of the data for our experiment, which are denoted as $\texttt{Wiki}_1$ and $\texttt{Wiki}_2$, respectively. $\texttt{Wiki}_1$ is a data set that contains 24,379 articles from the largest 214 categories whereas $\texttt{Wiki}_2$ contains 65,156 articles chosen from the largest 1,618 categories of the Wikipedia dump.

We compared the performance of RNMF against two baseline hierarchical learning algorithms proposed in the literature, namely, DAGSVM [15] and the confusion matrix approach (CM) described in [18]. For fair comparison, we set the branching factor (p) for our algorithm to be 2. The performance of the classifiers were evaluated according to their average

F1 scores for all classes. All the results reported in this study were based on 5-fold cross validation. The results for the two data sets are summarized in Tables 3.1 and 3.2.

Table 3.1 Comparison between RNMF and baseline algorithms on $\texttt{Wiki}_1$ data set.

	F1	Test time (s)	depth
CM [18]	0.4826 ± 0.0017	195.5 ± 9.2	10
DAGSVM [15]	0.5309 ± 0.002	942.9 ± 17.6	214
RNMF(p=2)	0.5116 ± 0.0012	132.2 ± 3.2	8

Table 3.2 Comparison between RNMF and baseline algorithms on $\texttt{Wiki}_2$ data set.

	F1	Test time (s)	depth
CM [18]	0.2518 ± 0.0025	336.2 ± 13.2	12
DAGSVM [15]	0.3012 ± 0.0019	1524.6 ± 16.5	1618
RNMF(p=2)	0.2892 ± 0.0036	289.3 ± 6.1	10

The results suggest that RNMF outperforms the CM approach both in terms of its F1 score and test efficiency. When compared against DAGSVM, which requires building a quadratic number of 1-vs-1 classifiers, the F1-score for RNMF is slightly lower but its test time is at least 7 times faster since the depth of RNMF's label tree is considerably shorter.

Note that the performance of RNMF depends on the branching factor (p) and the maximum depth (d) of the tree. We evaluated how the branching factor p and maximum depth of the tree affect the testing time and classification accuracy on the $\texttt{Wiki}_1$ data set. If the tree has only a single node, which is equivalent to a flat (non-hierarchical) model of k one-class SVM models, the average F1 score and test times of RNMF are comparable to DAGSVM (see Figures 3.2 and 3.3). As the depth of the tree increases, both the testing time and F1 score of RNMF would decrease. The decrease in testing time is because the number of 1-class SVM models that need to be invoked to predict the label of a test example at a given leaf node is smaller as the tree grows deeper. On the other hand, its F1-score is also lower for two reasons. First, as the tree is extended, the additional partitioning may divide

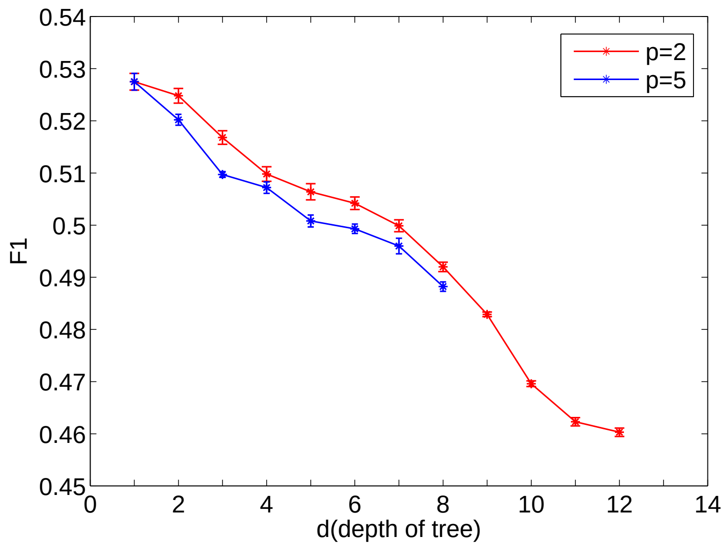


Figure 3.2 Average F1 score for RNMF as the branching factor (p) and maximum depth (d) are varied on $\texttt{Wiki}_1$ data set.

the training examples from the same class into multiple partitions. Second, there are less training examples available to build the 1-class SVM models at the leaf nodes as the tree grows longer, which makes it more susceptible to model overfitting.

Furthermore, if we compare their average F1 scores at the same maximum depth d, the tree with a branching factor equals to 2 is better than the tree with a branching factor equals to 5 (when $d > 1$). This is because the tree with smaller branching factor contains less number of nodes (since their maximum depths are the same), which means, its training examples will be split into less number of branches. This will result in a better classifier as more training examples become available at each leaf node to train the 1-class SVM models.

3.4 New Class Detection

As previously noted, one of the key advantages of RNMF is that it can detect new classes that are not present in the training set. To evaluate the performance of RNMF in terms of

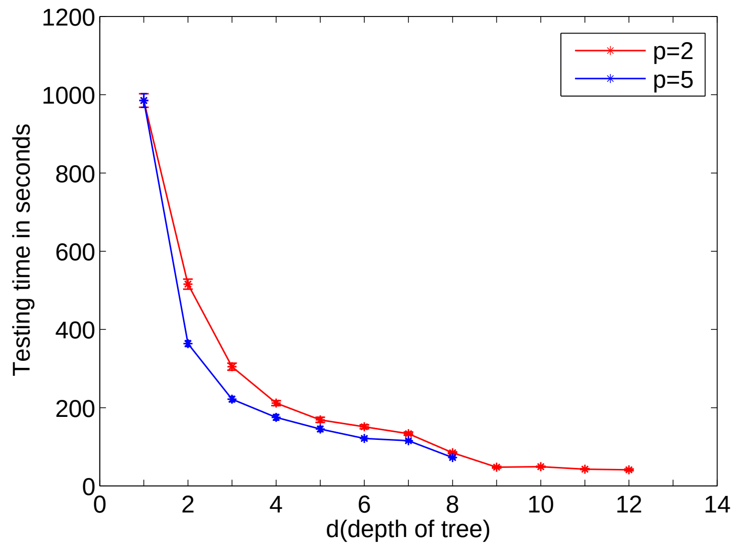


Figure 3.3 Test time for RNMF as the branching factor (p) and maximum depth (d) are varied on $\mathtt{Wiki}_1$ data set.

its ability for new class detection, we performed the following leave-one-class-out experiment on $Wiki_1$. Specifically, we eliminate one of the 214 classes from the training set and use the data instances belonging to the remaining 213 classes for training purposes. During testing, we apply the induced label tree on a test set that contains all 214 classes and assume the eliminated class to be the new class. We repeat the experiment 214 times, each time leaving out of the classes from the training set. The average F1 score of the new class is given in Table 3.3 (these results were reported after repeating the experiment 3 times). We compared the performance of the algorithm against DAGSVM.

DAGSVM [15] constructs a series of binary SVM classifiers and organizes them in a directed acyclic graph. An example of the DAGSVM structure for 5 classes is shown in Figure 3.4. Starting from the root node, DAGSVM eliminates one class at a time until it reaches a leaf node, which contains only a single class.

To extend DAGSVM to detect new class, we terminate the dag one level earlier, so that the leaf nodes contain training examples from two classes. A binary SVM classifier is then

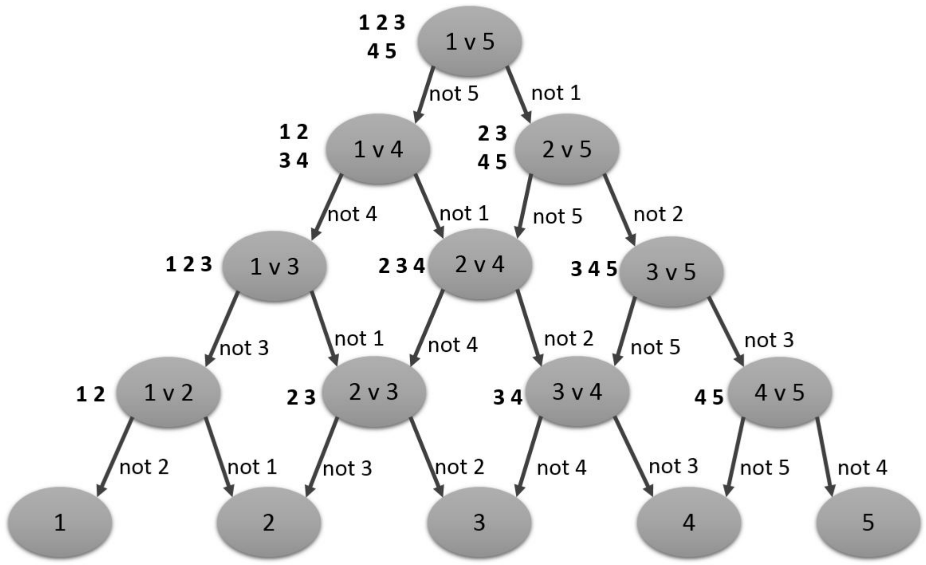


Figure 3.4 Example of DAGSVM Structure for 5 classes

trained to discriminate the classes. When a test example reaches the leaf node, we apply the binary classifier to predict its class. If the confidence of the SVM prediction is high, we assign the test example to its predicted class. However, if the confidence of the SVM prediction is low (i.e., when the test example is located within the geometric margin of the classifier), we predict it to be a new class.

Table 3.3 Average F1-score of the new class when applied to the `Wiki`$_1$ data set.

	# classes (Training)	# classes (Testing)	F1
RNMF	213	214	0.3192± 0.016
DAGSVM	213	214	0.2833 ± 0.012

Based on the results in Table 3.3, RNMF clearly outperforms DAGSVM in terms of its ability to detect new classes. We have also compared the F1-score of each class that was omitted from the training set in our new class detection experiment. The results are shown in Figure 3.5, where the horizontal axis corresponds to the F1 score of the DAGSVM algorithm whereas the vertical axis corresponds to the F1 score of RNMF. There are 214 data points in the plot, one for each class. If the point is located above the diagonal, this means the

F1-score for RNMF is better than DAGSVM for the particular omitted class. Since 82% of the points lie above the diagonal, this suggests that RNMF outperforms DAGSVM in terms of detecting the new class for the majority of the classes present in the $\texttt{Wiki}_1$ data.

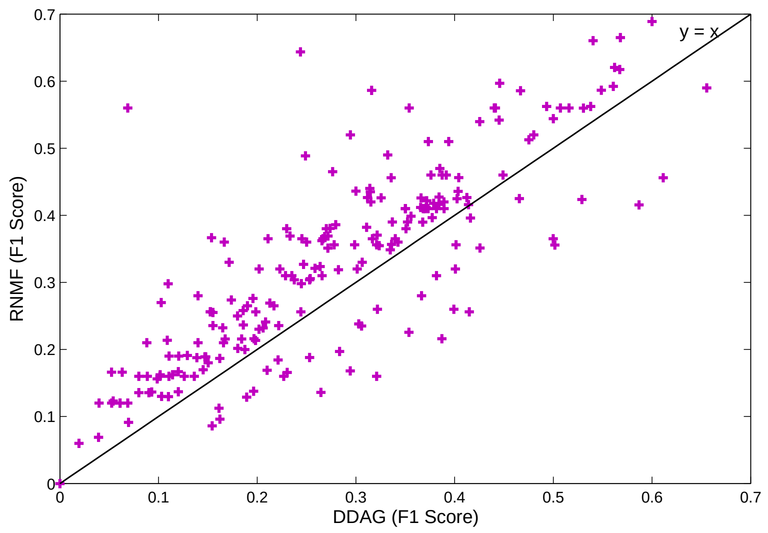


Figure 3.5 New Class Detection Performance for individual Class on $Wiki_1$

3.5 Conclusion

This chapter presents a novel label tree learning algorithm for extreme multi-class problem called recursive non-negative matrix factorization or RNMF. The algorithm creates soft partitions of the classes by solving a regularized non-negative matrix factorization problem and uses a set of 1-class SVM models to predict the classes at the leaf nodes of the tree. Experimental results on the Wikipedia data suggest that RNMF can achieve comparable accuracy as some of the state-of-the-art label tree algorithms but is considerably faster (since it produces shorter trees). In addition, it can detect new classes that are not present in the training data. However, RNMF has two major limitations. First, there is no global objective function optimized by the algorithm. Instead, it greedily partitions the training data to minimize a local objective function. Second, the classes can be fragmented to multiple leaf nodes in the tree. The next section presents an alternative method known as MF-tree that overcomes these limitations.

Chapter 4

Matrix Factorization Tree for Extreme Multi-Class Learning

Hierarchical learning methods [15, 16, 17, 18, 19] are designed to overcome the challenges of extreme multi-class classification by employing a set of binary classifiers and arranging them into a directed acyclic graph or rooted tree structure. These methods have several advantages. First, they can lead to sublinear test complexity since only the binary classifiers located along the path from the root node to leaf nodes are invoked when classifying a test example. Second, one could alleviate the class imbalanced problem particularly at the top levels of the hierarchy by grouping together the related classes[1]. Finally, the resulting concept hierarchy may offer interesting insights into the conceptual relationships among the large number of categories.

Over the years, there have been numerous hierarchical classification methods developed, from greedy top-down decision tree classifiers, which use simple attribute tests as internal nodes of the tree, to more advanced methods using binary classifiers as their internal nodes to distinguish the different classes. These methods have different pros and cons. While decision tree classifiers are easy to construct, they are susceptible to the class imbalanced and data fragmentation problems. In addition, each class can be distributed to multiple branches of

[1]Instead of learning to discriminate one class from another, some hierarchical methods are designed to distinguish one group of classes from another group at the top levels of the tree.

the tree, resulting in an unnecessarily large tree. This affects both the test efficiency and the ability of the classifier to generate a meaningful taxonomy about the conceptual relationships among the classes. Alternative methods, such as filter trees [17] and its variants, impose a random tree structure and learn an appropriate binary classifier at each internal node of the tree. Because of the arbitrariness in which the classes are assigned to the nodes, the resulting tree might be suboptimal and loses descriptive information about the class relationships. More importantly, both classes of methods were not designed to optimize a global objective function. To address these problems, we propose a novel method known as matrix factorization tree (MF-Tree, in short), which organizes the classes into a binary tree by solving a global objective function based on a squared loss error function.

We first provide a theoretical proof to demonstrate the equivalence between the proposed squared error loss function and the Hilbert-Schmidt Independence Criterion (HSIC) [21][22][23][24]. HSIC is a recently proposed criterion for taxonomy learning in an unsupervised learning setting. In this chapter, we demonstrate how HSIC can be applied to a supervised multi-class learning problem. We also showed the additive property of the objective function, thus allowing us to decompose the problem into a hierarchical binary classification task that can be organized in a binary tree structure. Nevertheless, for massive classification problems involving hundreds of thousands of classes or more, factorizing the matrices is computationally expensive especially at the top levels of the tree. To address this problem, we present an approximate algorithm for inducing MF-Tree, where the initial assignment of classes to the child nodes at the top level of the hierarchy is determined from the class sizes and distances between centroids of classes instead of solving the more expensive matrix factorization problem. The factorization can then be applied at lower levels of the tree when there are fewer number of classes and training examples to deal with. We showed that the performance of the approximate MF-Tree is comparable to that of MF-Tree, with a significant improvement in its training time.

In short, the main contributions of this chapter are as follows:

1. We introduce a novel hierarchical classification method for large multi-class learning problems known as MF-Tree. The tree is designed to optimize a global loss function.

2. We provide a theoretical proof showing the equivalence between optimizing the squared error loss function of a matrix factorization problem and the Hilbert-Schmidt Independence Criterion (HSIC) by setting appropriate constraints on the latent factors of the factorization.

3. We demonstrate the additive property of HSIC that enables the design of a hierarchical tree learning framework.

4. We present an approximate algorithm for learning MF-Tree, to speed up the tree induction process.

5. We provide extensive experiments to evaluate and compare the performance of the proposed method against several state-of-the-art baselines.

4.1 Preliminaries

This section presents a formal definition of the multi-class learning problem. We also introduce the concepts of Hilbert-Schmidt Independence Criterion and tree-structured covariance matrix. The connections between these concepts and the proposed MF-tree method will be shown in Section 4.2.

4.1.1 Problem Definition

Let $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ denote a set of training instances sampled from an unknown distribution P over $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X} \subseteq \mathcal{R}^d$ is the d-dimensional feature space and $\mathcal{Y} = \{1, 2, \cdots, c\}$ is the set of permissible categories. For large multi-class learning problems, we assume $c >> 2$. The goal of multi-class learning is to infer a target function f that minimizes the expected risk of misclassifying any instance drawn from P, i.e., $\min_f E_{(\mathbf{x},y)\in P}[\mathcal{L}(f(\mathbf{x}), y)]$, where $\mathbf{L}$ is known as the loss function.

A standard approach for multi-class learning is to reduce the problem into multiple binary classification tasks. However, when the number of classes c is too large, applying all the binary classifiers to predict the class of a test instance is computationally infeasible. Towards this end, there are various hierarchical labeled tree algorithms have been developed for large multi-class learning problems. A hierarchical labeled tree is a 4-tuple $T =< V, E, \Phi, \Lambda >$, where V is the set of nodes, E is the set of edges connecting parent nodes to their corresponding child nodes, $\Phi = \{f_v(\mathbf{x})|v \in V\}$ is the set of classifiers associated with the internal nodes of T, and $\Lambda = \{y_{\hat{v}}|y_{\hat{v}} \in \mathcal{Y}, \hat{v} \in V\}$ is the set of labels associated with the leaf nodes. In this chapter, we restrict our framework to binary rooted trees only, even though, in principle, it is applicable to rooted trees with a branching factor greater than 2. Our objective in this work is to learn an optimal tree T for large, multi-class learning problems.

4.1.2 Hilbert-Schmidt Independence Criterion (HSIC)

This section introduces the Hilbert-Schmidt Independence Criterion. Our presentation follows closely the definitions given in [21][23][22][24]. Let (X, Y) be a pair of random variables drawn from a joint distribution $Pr_{X,Y}$. The Hilbert-Schmidt Independence Criterion

(HSIC)[21] measures the dependence between X and Y by calculating the norm of its cross-covariance operator C_{xy} in Hilbert space. It has been shown that this norm vanishes if and only if X and Y are independent variables. A larger value of HSIC also indicates a stronger dependence between the two variables, with respect to the choice of kernels [23].

Let $\mathcal{F}$ be a Reproducing Kernel Hilbert Space (RKHS) of functions from $\mathcal{X}$ to $\mathcal{R}$ with a feature map ϕ, such that $\langle \phi(x), \phi(x^{'}) \rangle_{\mathcal{F}} = k_X(x, x^{'})$, where $k_X : \mathcal{X} \times \mathcal{X} \rightarrow \mathcal{R}$ is a positive definite kernel. Likewise, let $\mathcal{G}$ be the RKHS on $\mathcal{Y}$ with a feature map ψ, such that $\langle \psi(y), \psi(y^{'}) \rangle_{\mathcal{G}} = k_Y(y, y^{'})$, where $k_Y : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathcal{R}$ is the corresponding kernel for the metric space $\mathcal{Y}$. The covariance operator C_{xy} is defined as

$$C_{xy} = E_{x,y}[(\phi(x) - \mu_x) \otimes (\psi(y) - \mu_y)]^2_{HS}$$

where $\mu_x = E[\phi(x)]$, $\psi_y = E[\psi(y)]$ and $\otimes$ denote a tensor product. The Hilbert-Schmidt Independence Criterion is defined as the squared Hilbert-Schmidt norm of the cross-covariance operator C_{xy}, which can be written as follows:

$$\begin{aligned} \mathrm{HSIC}_{X,Y} &= \|C_{xy}\|^2_{HS} \\ &= E_{x,x^{'},y,y^{'}}[k_X(x, x^{'}) k_Y(y, y^{'})] \\ &\quad + E_{x,x^{'}}[k_X(x, x^{'})] E_{y,y^{'}}[k_Y(y, y^{'})] \\ &\quad - 2E_{x,y}[E_{x^{'}}[k_X(x, x^{'})] E_{y^{'}}[k_Y(y, y^{'})]] \qquad (4.1) \end{aligned}$$

Let $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_n, y_n)\}$ be the set of training instances drawn from Pr_{XY}. HSIC can be empirically estimated from the training data as follows [21, 22]:

$$\text{HSIC} = \frac{1}{(n-1)^2}\mathbf{tr}\Big[\mathbf{H}_n\mathbf{K}\mathbf{H}_n\mathbf{L}\Big] \tag{4.2}$$

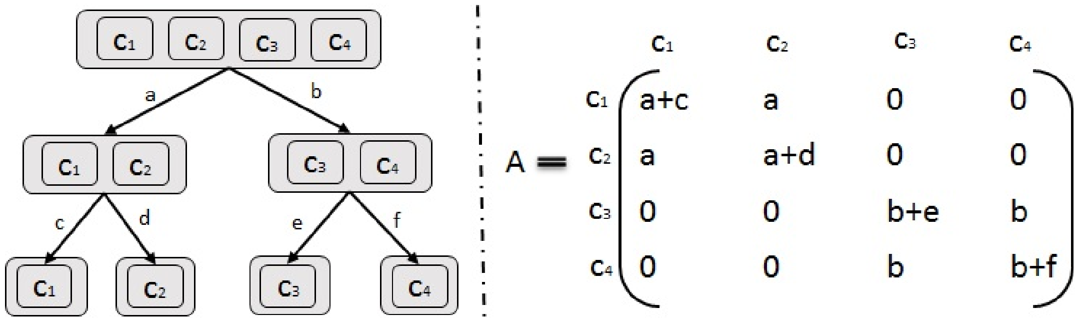
where $\mathbf{tr}[\cdot]$ denote the trace of a matrix, $\mathbf{H}_n = \mathbf{I} - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T$ is a centering matrix, $\mathbf{I}$ is the identity matrix, $\mathbf{1}_n$ is a column vector of all ones, $\mathbf{K}$ is an $n \times n$ kernel matrix for $\mathbf{x}$, and $\mathbf{L}$ is a $c \times c$ kernel matrix for the class labels y. The matrix product $\mathbf{H}_n\mathbf{K}\mathbf{H}_n$ corresponds to a centered kernel matrix. If $\hat{\mathbf{K}}$ denote the centered kernel matrix, then the computation of HSIC simplifies to $(n-1)^{-2}\mathbf{tr}[\hat{\mathbf{K}}\mathbf{L}]$. Note that the time complexity for learning HSIC from the data is $O(c^3 + nc^2 + n^2c)$, which is expensive when number of classes c is large.

4.1.3 Tree-Structured Covariance Matrix

Given a rooted binary tree T on the labeled set $\mathcal{Y}$, we can represent it as a tree-structured covariance matrix [22] as follows. **Definition**

Definition 4 *(Tree-Structured Covariance Matrix)*
A matrix $\mathbf{C} \in \mathcal{R}^{c \times c}$ is a covariance matrix associated with the label tree structure T, *where $\mathbf{C}_{ij}$ is a measure of similarity between the i-th and j-th classes computed based on the path length from the root node to the closest common ancestor between the leaf nodes for i and j in* T. *The larger the value of $\mathbf{C}_{ij}$, the closer is the relationship between the two classes.*

Figure 4.1 shows an example of a labeled tree structure for 4 classes, c_1, c_2, c_3, and c_4. Let a, b, c, d, e, f denote the edge weights between the nodes. Based on the structure, the tree-structured covariance matrix is a 4×4 matrix shown on the right hand side of the diagram. For example, $C_{1,2} = a$ since the closest common ancestor between the classes has a path length equals to a from the root node. Similarly, $C_{3,3} = b+e$, which is the path length



$$A = \begin{array}{c|cccc} & c_1 & c_2 & c_3 & c_4 \\ \hline c_1 & a+c & a & 0 & 0 \\ c_2 & a & a+d & 0 & 0 \\ c_3 & 0 & 0 & b+e & b \\ c_4 & 0 & 0 & b & b+f \end{array}$$

Figure 4.1 Example Tree Structure with 8 classes

from the root to the leaf node for c_3. Note that the edge weight a, b, c, d, e, f determines the similarity between the different categories. For brevity, we consider the edge weights to be 1 in the remainder of this chapter.

4.2 MF-Tree: Proposed Method

This section presents an overview of the proposed method. We first introduce the global objective function to be optimized by MF-Tree. We then show its connection to the HSIC measure described in the previous section. We also illustrate the additive property of HSIC, which allows us to develop a hierarchical tree-based induction algorithm.

4.2.1 Global Objective Function of MF-Tree

Let $\mathbf{X}$ denote an $n \times d$ centered design matrix, whose rows correspond to the data instances and columns correspond to the features. An uncentered design matrix $\tilde{\mathbf{X}}$ can always be centered by applying the following operation, $\mathbf{H}_n\tilde{\mathbf{X}}$, where $\mathbf{H}_n$ is the centering matrix defined in Section 4.1.2.

MF-Tree is designed to minimize the following regularized squared error loss function.

$$
\begin{aligned}
\mathcal{J} &= \|\mathbf{X} - \mathbf{Z}\mathbf{W}^T\|_F^2 + \lambda \mathbf{tr}\Big[\mathbf{W}\mathbf{A}\mathbf{W}^T\Big] \\
&= \mathbf{tr}\Big[\mathbf{X}\mathbf{X}^T - \mathbf{X}\mathbf{W}\mathbf{Z}^T - \mathbf{Z}\mathbf{W}^T\mathbf{X}^T + \mathbf{Z}\mathbf{W}^T\mathbf{W}\mathbf{Z}^T\Big] \\
&\quad + \lambda\ \mathbf{tr}\Big[\mathbf{W}\mathbf{A}\mathbf{W}^T\Big],
\end{aligned} \tag{4.3}
$$

where $\mathbf{Z}$ is an $n \times c$ matrix, which represents the class membership of each data instance, $\mathbf{W}$ is a $d \times c$ matrix, which represents the feature vector of each class, $\mathbf{A}$ is a symmetric, positive-definite, $c \times c$ class covariance matrix, and λ is the regularization parameter. $\|\cdot\|_F$ denote the Frobenius norm of a matrix and the superscript T denote the transpose operator. The intuition behind the objective function given in Equation (4.3) is to factorize the centered design matrix $\mathbf{X}$ into a product of latent factors $\mathbf{Z}$ and $\mathbf{W}$.

4.2.2 Relationship between Global Objective Function and HSIC

To demonstrate the relationship between the objective function and HSIC, let us take the partial derivative of $\mathcal{J}$ given in Equation (4.3) with respect to $\mathbf{W}$ and set it to zero:

$$
\begin{aligned}
\frac{\partial \mathcal{J}}{\partial \mathbf{W}} &= -2\mathbf{X}^T\mathbf{Z} + 2\mathbf{W}\mathbf{Z}^T\mathbf{Z} + 2\lambda\mathbf{W}\mathbf{A} = 0 \\
\mathbf{W} &= \mathbf{X}^T\mathbf{Z}\Big[\mathbf{Z}^T\mathbf{Z} + \lambda\mathbf{A}\Big]^{-1}
\end{aligned}
$$

Replacing this back into Equation (4.3) yields the following:

$$\begin{aligned}
\mathcal{J} &= \mathbf{tr}\Bigg\{\mathbf{X}\mathbf{X}^T - 2\mathbf{X}\mathbf{X}^T\mathbf{Z}\Big[\mathbf{Z}^T\mathbf{Z}+\lambda\mathbf{A}\Big]^{-1}\mathbf{Z}^T \\
&\quad + \mathbf{Z}\Big[\mathbf{Z}^T\mathbf{Z}+\lambda\mathbf{A}\Big]^{-1}\mathbf{Z}^T\mathbf{X}\mathbf{X}^T\mathbf{Z}\Big[\mathbf{Z}^T\mathbf{Z}+\lambda\mathbf{A}\Big]^{-1}\mathbf{Z}^T\Bigg\} \\
&\quad + \lambda\ \mathbf{tr}\Bigg\{\mathbf{X}^T\mathbf{Z}\Big[\mathbf{Z}^T\mathbf{Z}+\lambda\mathbf{A}\Big]^{-1}\mathbf{A}\Big[\mathbf{Z}^T\mathbf{Z}+\lambda\mathbf{A}\Big]^{-1}\mathbf{Z}^T\mathbf{X}\Bigg\} \\
&= \mathbf{tr}\Bigg\{\mathbf{X}\mathbf{X}^T - 2\mathbf{X}\mathbf{X}^T\mathbf{Z}\Big[\mathbf{Z}^T\mathbf{L}+\lambda\mathbf{A}\Big]^{-1}\mathbf{Z}^T \\
&\quad + \mathbf{X}^T\mathbf{Z}\Big[\mathbf{Z}^T\mathbf{Z}+\lambda\mathbf{A}\Big]^{-1}\Big[\mathbf{Z}^T\mathbf{Z}+\lambda\mathbf{A}\Big]\Big[\mathbf{Z}^T\mathbf{Z}+\lambda\mathbf{A}\Big]^{-1}\mathbf{Z}^T\mathbf{X}\Bigg\} \\
&= \mathbf{tr}\Bigg\{\mathbf{X}\mathbf{X}^T - \mathbf{X}\mathbf{X}^T\mathbf{Z}\Big[\mathbf{Z}^T\mathbf{Z}+\lambda\mathbf{A}\Big]^{-1}\mathbf{Z}^T\Bigg\}
\end{aligned}$$

Thus, finding a solution for $\mathbf{Z}$ that minimizes $\mathcal{J}$ is equivalent to finding one that maximizes the following objective function:

$$\max_{\mathbf{Z}} \mathbf{tr}\Bigg\{\mathbf{X}\mathbf{X}^T\mathbf{Z}\Big[\mathbf{Z}^T\mathbf{Z}+\lambda\mathbf{A}\Big]^{-1}\mathbf{Z}^T\Bigg\}$$

Let $\hat{\mathbf{K}} = \mathbf{X}\mathbf{X}^T$ be the centered kernel matrix associated with the training instances $\mathbf{X}$. During training, we enforce the following constraint:

$$\mathbf{Z}_{\text{train}}\Big[\mathbf{Z}_{\text{train}}^T\mathbf{Z}_{\text{train}}+\lambda\mathbf{A}\Big]^{-1}\mathbf{Z}_{\text{train}}^T = \mathbf{Y}\mathbf{C}\mathbf{Y}^T, \tag{4.4}$$

where $\mathbf{Y}$ is an $n \times c$ matrix of class labels and $\mathbf{C}$ is the tree-structured covariance matrix. This can be achieved by setting $\mathbf{Z}_{\text{train}} = \mathbf{Y}$ and $\mathbf{C} = \Big[\mathbf{Z}_{\text{train}}^T\mathbf{Z}_{\text{train}}+\lambda\mathbf{A}\Big]^{-1}$.

Theorem 1 *If $\mathbf{A}$ be a symmetric, positive-definite matrix, and $\lambda > 0$, then $\mathbf{Z}_{train}^T\mathbf{Z}_{train}+\lambda\mathbf{A}$ is also positive-definite.*

Proof 1 *Given any real vector* $\mathbf{v}$, $\mathbf{v}^T\left[\mathbf{Z}_{train}^T\mathbf{Z}_{train}+\lambda\mathbf{A}\right]\mathbf{v}=\mathbf{v}^T\mathbf{Z}_{train}^T\mathbf{Z}_{train}\mathbf{v}+\lambda\mathbf{v}^T\mathbf{A}\mathbf{v}=\|\mathbf{Z}_{train}\mathbf{v}\|^2+\lambda\mathbf{v}^T\mathbf{A}\mathbf{v}>0$ *since* $\mathbf{A}$ *is a positive definite matrix.*

Since $\mathbf{Z}_{\text{train}}^T\mathbf{Z}_{\text{train}}+\lambda\mathbf{A}$ is positive-definite, it is invertible. The equality constraint in Equation (4.4) ensures that an appropriate λ and $\mathbf{A}$ can be chosen to generate a viable tree, in which the entries of the matrix $\mathbf{C}$ must have a specific structure. Furthermore, let $\mathbf{L}=\mathbf{Y}\mathbf{C}\mathbf{Y}^T$ denote the kernel matrix associated with the class labels. The objective function can now be simplified to:

$$\max_{\mathbf{C}}\mathbf{tr}\left[\hat{\mathbf{K}}\mathbf{L}\right]\quad\text{s.t.}\quad\mathbf{L}=\mathbf{Y}\mathbf{C}\mathbf{Y}^T,\tag{4.5}$$

which is equivalent to the optimization of HSIC.

4.2.3 Additive Property of HSIC

In the previous subsection, we have shown that the global objective function for MF-Tree is equivalent to HSIC by setting appropriate constraints on $\mathbf{A}$ and λ. This enables us to define a kernel matrix for the class labels as $\mathbf{L}=\mathbf{Y}\mathbf{C}\mathbf{Y}^T$, where $\mathbf{C}$ is the tree-structured covariance matrix. In this section, we demonstrate the additive property of HSIC, which enables us to decompose the optimization problem into a hierarchical binary classification task.

First, we show that the tree-structured covariance matrix can be written as a sum of block diagonal matrices. We illustrate this with a simple example. Consider the following covariance matrix for a labeled tree that contains 8 leaf nodes (assuming the edge weights are equal to 1):

$$\mathbf{C}_8 = \begin{bmatrix} 3 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 3 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 3 & 2 & 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 2 & 1 & 1 \\ 0 & 0 & 0 & 0 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 3 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 2 & 3 \end{bmatrix} \tag{4.6}$$

We can rewrite the matrix as follows:

$$\mathbf{C}_8 \quad = \quad \mathbf{I}_2 \otimes \mathbf{1}_4\mathbf{1}_4^T + \mathbf{I}_4 \otimes \mathbf{1}_2\mathbf{1}_2^T + \mathbf{I}_8 \otimes \mathbf{1} \tag{4.7}$$

where $\otimes$ denote a Kronecker product and

$$\mathbf{I}_2 \otimes \mathbf{1}_4\mathbf{1}_4^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{I}_4 \otimes \mathbf{1}_2\mathbf{1}_2^T = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\mathbf{I}_8 \otimes \mathbf{1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The first term on the right hand side of Equation (4.7) simply partitions the first 4 rows and columns of the matrix into 1 cluster and the last 4 rows and columns into a second cluster. The second term on the right hand side partitions the first cluster into 2 smaller subclusters, while the third term puts each subcluster as a cluster itself (this is at the leaf node of the tree). So, we can simply write

$$\mathbf{C}_{2^m} = \mathbf{B}_2 + \mathbf{B}_4 + \cdots + \mathbf{B}_{2^m}, \tag{4.8}$$

where $\mathbf{B}_{2^j} = \mathbf{I}_{2^j} \otimes \mathbf{1}_{2^{m-j}}\mathbf{1}^T_{2^{m-j}}$, $\mathbf{1}_{2^0} = 1$, and m is the depth of the binary tree. We can further simplify the decomposition to the following:

$$\begin{aligned}
\mathbf{C}_8 &= \mathbf{B}_2 + \mathbf{B}_4 + \mathbf{B}_8 \\
&= \begin{bmatrix} \mathbf{B}_{2,1} \end{bmatrix} + \begin{bmatrix} \mathbf{B}_{4,1} & \mathbf{0}_{4\times4} \\ \mathbf{0}_{4\times4} & \mathbf{B}_{4,2} \end{bmatrix} + \begin{bmatrix} \mathbf{B}_{8,1} & \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} \\ \mathbf{0}_{2\times2} & \mathbf{B}_{8,2} & \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} \\ \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} & \mathbf{B}_{8,3} & \mathbf{0}_{2\times2} \\ \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} & \mathbf{0}_{2\times2} & \mathbf{B}_{8,4} \end{bmatrix}
\end{aligned}$$

where each diagonal block has the following form

$$\mathbf{B}_{2^i,j} = \begin{bmatrix} \mathbf{1}\mathbf{1}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{1}\mathbf{1}^T \end{bmatrix} \tag{4.9}$$

and the dimension for each of the submatrices of $\mathbf{0}$s and $\mathbf{1}\mathbf{1}^T$s in $\mathbf{B}_{2^i,j}$ is $2^{m-i} \times 2^{m-i}$. Furthermore, given a block matrix $\mathbf{B}_k$, we have

$$\mathbf{Y}\mathbf{B}_k\mathbf{Y}^T = \mathbf{Y}_1\mathbf{B}_{k,1}\mathbf{Y}_1^T + \mathbf{Y}_2\mathbf{B}_{k,2}\mathbf{Y}_2^T + \cdots + \mathbf{Y}_{k/2}\mathbf{B}_{k,k/2}\mathbf{Y}_{k/2}^T$$

where

$$\mathbf{Y} = \begin{bmatrix} \mathbf{Y}_1 \cdots \mathbf{Y}_j \cdots \mathbf{Y}_{k/2} \end{bmatrix} \tag{4.10}$$

$$\mathbf{B}_k = \begin{bmatrix} \mathbf{B}_{k,1} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_{k,2} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{B}_{k,j} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \mathbf{0} & \cdots & \cdots & \cdots & \cdots & \mathbf{B}_{k,k/2} \end{bmatrix} \tag{4.11}$$

$$\mathbf{Y}^T = \begin{bmatrix} \mathbf{Y}_1^T \\ \cdots \\ \mathbf{Y}_j^T \\ \cdots \\ \mathbf{Y}_{k/2}^T \end{bmatrix} \tag{4.12}$$

As each $\mathbf{B}_{i,j}$ corresponds to a block-diagonal matrix given by Equation (4.9), we can write

$$\text{HSIC} = \mathbf{YCY}^T = \sum_{i=1}^{\log_2 k} \mathbf{YB}_{2^i}\mathbf{Y}^T = \sum_{i=1}^{\log_2 k} \sum_{j=1}^{2^{i-1}} \mathbf{Y}_j \mathbf{B}_{2^i,j} \mathbf{Y}_j^T$$

This decomposition enables us to solve the optimization problem with a hierarchical tree-like decomposition. The outer sum corresponds to decomposition at each depth of the tree, while the inner sum corresponds to expanding the internal nodes associated with the particular depth. Instead of learning $\mathbf{C}$ directly by maximizing the trace of HSIC (see Equation (4.5)), we can learn the set of $\mathbf{B}_{2^i,j}$ iteratively, each of which is associated with an internal node in the tree structure.

4.2.4 MF-Tree construction

To construct the tree, we need to solve the objective function given in (4.5). Based on the discussion in the previous subsection, the objective function can be simplified as follows:

$$\max_{\{\mathbf{B}\}} \sum_{i,j} \mathbf{tr}\left[\mathbf{K}_{j,j}\mathbf{Y}_j\mathbf{B}_{i,j}\mathbf{Y}_j^T\right] \tag{4.13}$$

This allows us to solve the optimization problem one node at a time. Let $\mathbf{D}_j = \mathbf{Y}_j^T\mathbf{K}_{j,j}\mathbf{Y}_j$, which is simply a $c \times c$ similarity matrix of the classes, computed based on their attributes $\mathbf{X}$ and the known ground truth labels associated with the block $\mathbf{B}_{i,j}$. At each node of the tree construction process, we need to find a matrix $\mathbf{B}_{i,j}$ that maximizes its alignment with $\mathbf{D}_j$, i.e.:

$$\max_{\mathbf{B}_{i,j}} \mathbf{tr}\left[\mathbf{D}_j\mathbf{B}_{i,j}\right]$$

For notational convenience, we drop the subscripts i and j in the remainder of the discussion. Note that $\mathbf{B}$ is simply a co-association matrix, whose element

$$[\mathbf{B}]_{i,j} = \begin{cases} 1, & \text{if classes i and j are in the same partition;} \\ 0, & \text{otherwise.} \end{cases}$$

To create binary partitions, let $\mathbf{G}$ be a $c \times 2$ matrix[2], where

$$[\mathbf{G}]_{i,j} = \begin{cases} 1, & \text{if class i belongs to partition j;} \\ 0, & \text{otherwise.} \end{cases}$$

It is easy to show that $\mathbf{B} = \mathbf{G}\mathbf{G}^T$. Thus,

$$\mathbf{tr}\Big[\mathbf{D}\mathbf{B}\Big] = \mathbf{tr}\Big[\mathbf{D}\mathbf{G}\mathbf{G}^T\Big] = \mathbf{tr}\Big[\mathbf{G}^T\mathbf{D}\mathbf{G}\Big].$$

This objective function is equivalent to finding a clustering of the classes in a way that maximizes the within-cluster similarity.

4.2.5 Optimization

Note that the following optimization problem at each node

$$\max_{\hat{\mathbf{G}}} \mathbf{tr}\Big[\mathbf{G}^T\mathbf{D}\mathbf{G}\Big] \quad \text{s.t.} \quad \mathbf{G}^T\mathbf{1}_2 = \mathbf{1}_n$$

is not feasible unless we enforce a non-negative constraint $\mathbf{G} \succeq \mathbf{0}$ [61]. Furthermore, if we relax the condition that elements of $\mathbf{G}$ must be either 0 or 1 and replace it with a constraint that $\mathbf{G}$ must be an orthogonal matrix

$$\max_{\mathbf{G}} \mathbf{tr}\Big[\mathbf{G}^T\mathbf{D}\mathbf{G}\Big] \quad \text{s.t.} \quad \mathbf{G}^T\mathbf{G} = \mathbf{I}_2$$

[2]Here, we assume the number of classes associated with the given node is c. As the tree grows deeper, the number of classes also decreases, which in turn, reduces the number of rows in $\mathbf{G}$.

we can solve the problem easily by finding the first two eigenvectors corresponding to the largest eigenvalues of $\mathbf{D}$ [62, 63, 64]. Let $\mathbf{G\Lambda G}^T$ be the eigendecomposition of matrix $\mathbf{D}$.

By definition, $\mathbf{D} = \mathbf{Y}^T\mathbf{KY}$. Following the eigendecomposition of $\mathbf{D}$, we have

$$\mathbf{Y}^T\mathbf{KY} = \mathbf{G\Lambda G}^T$$

Since $\mathbf{G}^T\mathbf{G} = \mathbf{I}$:

$$\mathbf{G}^T\mathbf{Y}^T\mathbf{KYG} = \mathbf{\Lambda}$$

Let $\mathbf{P} = \mathbf{YG}$, which is an $n \times 2$ matrix that determines the assignment of each data instance to the corresponding child nodes. Thus,

$$\mathbf{P}^T\mathbf{KP} = \mathbf{\Lambda}$$

During the testing phase, let $\hat{\mathbf{k}}$ be an n-dimensional column vector that contains similarity between the test instance to all the training instances. We may compute the 2-dimensional partition vector $\hat{\mathbf{p}} \in \mathbf{R}^{2\times 1}$ as follows:

$$\hat{\mathbf{p}} = \mathbf{G}^T\mathbf{Y}^T\hat{\mathbf{k}} = \mathbf{P}^T\hat{\mathbf{k}}$$

After computing $\hat{\mathbf{p}}$, we assign the test instance to the partition with the larger value, i.e., $\arg\max_i(\mathbf{P}^T\hat{\mathbf{k}})_i$.

Algorithm 2 MF-Tree Construction

```
Input: $X_{train} \in \mathbf{R}^{n \times d}$, $Y_{train} \in \mathbf{R}^{n \times k}$
Output: Tree Structure, $T$
1. $root$ = TreeGrowth($X_{train}, Y_{train}$)
2. Insert $root$ into $T$
3. return $T$

function TreeGrowth($X$,$Y$)
1. $v$ = createNode()
2. $v.index$ = getIndex($X$)
3. if number of classes in $Y < 2$ then
4.     $v.class$ = unique($Y$)
5. else
6.     $(X^{(l)}, Y^{(l)}, X^{(r)}, Y^{(r)}, v.\mathbf{G})$ = Partition($X$,$Y$)
7.     $v.left$ = TreeGrowth($X^{(l)}, Y^{(l)}$)
8.     $v.right$ = TreeGrowth($X^{(r)}, Y^{(r)}$)
9. end if
10. return $v$

function Partition($X$,$Y$)
1. Compute the centered kernel $\mathbf{K}$ from $X$.
2. Compute $\mathbf{D} = Y^T \mathbf{K} Y$
3. Compute the first two eigenvectors $g^{(1)}, g^{(2)}$ of $\mathbf{D}$
4. Let $\mathbf{G} = [g^{(1)}, g^{(2)}]$
5. $(X^{(l)}, Y^{(l)}) = \{(\mathbf{x}_i, y_i) \big| y_i = k, \arg\max_j \mathbf{G}_{kj} = 1\}$
6. $(X^{(r)}, Y^{(r)}) = \{(\mathbf{x}_i, y_i) \big| y_i = k, \arg\max_j \mathbf{G}_{kj} = 2\}$
7. return $(X^{(l)}, Y^{(l)}, X^{(r)}, Y^{(r)}, \mathbf{G})$
```

4.2.6 MF-Tree Induction Algorithm

The pseudocode for constructing MF-Tree is summarized in Algorithm 2. Our algorithm recursively partitions each node by calling the `TreeGrowth` function. Each invocation of the function returns a root node for a subtree that partitions the classes and their associated training instances into 2 groups. Each node is assumed to have a complex structure, where $v.index$ contains the indices of the training instances assigned to the node, $v.class$ represents the class label if v is a leaf node, $v.\mathbf{G}$ is the discriminant function if v is an internal node, whereas $v.left$ and $v.right$ are its left and right child.

The `Partition` function decides how to partition each class and their training examples into different branches of the tree. The partitioning is done based on the magnitude of matrix **G** as described in the previous section. If a class k is assigned to the left child of a node, all the training instances that belong to class k will also be propagated to the left child. The algorithm continues to expand a node until all the training instances associated with the node belong to the same class.

To determine its time complexity, note that at each node v, we need to compute the kernel matrix $\mathbf{K}$, an operation that requires $O(n^2d)$ time. Computing $\mathbf{D}$ from the kernel matrix $\mathbf{K}$ and class label $\mathbf{Y}$ takes $O(n^2c+c^2n)$ time while finding its first two eigenvectors requires, in the worst-case, $O(c^3)$ time. Thus, the computational cost at each node is $O(n^2c+c^2n+c^3)$. The cost reduces significantly as the tree grows deeper since the size of matrices $\mathbf{K}$ and $\mathbf{D}$ decreases substantially. Once the tree structure is induced from the training set, we can predict the label for any test instance using the method presented in Algorithm 3.

Algorithm 3 Testing with MF-Tree

```
Input: $X_{test}$, $X_{train}$, $Y_{train}$, $T$
Output: $Y_{test}$
1. for each $\mathbf{x} \in X_{test}$ do
2.     $v = T.root$
3.     while $v.class = \emptyset$ do
4.         $\hat{k}$ = Similarity($\mathbf{x}, X_{train}, v.index$)
5.         $\hat{Y}$ = Subset($Y_{train}, v.index$)
6.         $\hat{G} = v.\mathbf{G}$
7.         $\hat{p} = \hat{G}^T\hat{Y}^T\hat{k}$
8.         $v$ = getChild($\hat{p}, v$)
9.     end while
9.     Insert $y = v.class$ into $Y_{test}$
10. end for
11. return $Y_{test}$
```

4.3 Approximate MF-Tree

As shown in the previous section, the time complexity for constructing MF-Tree can be quite expensive especially at the top levels of the hierarchy. This is due to the large size of matrices that must be constructed and factorized according to their largest two eigenvalues. However, as we traverse down the tree, the number of classes and training instances to be dealt with decreases considerably, which in turn, helps to speed up the computations. Thus, the bottleneck of our computation lies in the first few iterations of the MF-Tree algorithm. In this section, we present an approach to speed up the tree construction process by learning an approximation for the $\mathbf{G}$ matrix during the first few iterations. Once the matrices become small enough, we can proceed with applying the original `TreeGrowth` function shown in Algorithm 2.

Our proposed method attempts to balance the tradeoff between accuracy and efficiency of the labeled tree construction process. While efficiency is a concern at the top levels of the tree, accuracy is an issue at lower levels of the tree. Even if the class partitioning at the top levels is suboptimal, one might still be able to recover good partitions at subsequent iterations when refining the tree. Based on this rationale, we develop the following two-step approach to improve training efficiency. First, we apply a simple method to calculate $\mathbf{G}$ at the top levels of the tree. When the depth of the tree exceeds some threshold τ, we revert back to the original `TreeGrowth` function. We termed this approach as *Approximate MF-Tree.*

Consider the root node of the labeled tree. Let n be the number of training instances and c be the number of classes. We first sort the classes in decreasing order of their class size. We then assign the largest class (c_1) to the left node of the root and the second largest class

Algorithm 4 Approximate MF-Tree

```
Input: $X_{train} \in \mathbf{R}^{n\times d}$, $Y_{train} \in \mathbf{R}^{n\times k}$, $\tau$
Output: Tree structure $T$
1. Set depth, $d = 0$
2. $root$ = TreeGrowthApprox($X_{train}, Y_{train}, d, \tau$)
3. Insert $root$ into $T$
4. return $T$

function TreeGrowthApprox($X$,$Y$,$d$,$\tau$)
1. $v$ = createNode()
2. $v.index$ = getIndex($X$)
3. if $d < \tau$ and number of classes in $Y < 2$
4.     ($X^{(l)}, Y^{(l)}, X^{(r)}, Y^{(r)}$)= Partition2($X$,$Y$)
5.     $v.left$ = TreeGrowthApprox($X^{(l)}, Y^{(l)}, d+1$)
6.     $v.right$ = TreeGrowthApprox($X^{(r)}, Y^{(r)}, d+1$)
7. else
8.     $v$ = TreeGrowth($X, Y$)
9. end if
10. return $v$

function Partition2($X$,$Y$)
1. classes = Sort($Y$)
2. $(X^{(l)}, Y^{(l)}) = \{(\mathbf{x}_i, y_i)\big|y_i = c_1$ (largest class) $\}$
3. $(X^{(r)}, Y^{(r)}) = \{(\mathbf{x}_i, y_i)\big|y_i = c_2$ (second largest class) $\}$
4. meanvectors = Mean($X$, $Y$)
5. for each remaining class $c \in$ classes do
6.    $m_1$ = distance(meanvectors, $c$, $c_1$)
7.    $m_2$ = distance(meanvectors, $c$, $c_2$)
8.    if $m_1 \leq m_2$ then
9.       $(X^{(l)}, Y^{(l)}) = (X^{(l)}, Y^{(l)}) \cup \{(\mathbf{x}_i, y_i)\big|y_i = c\ \}$
10.   else
11.      $(X^{(r)}, Y^{(r)}) = (X^{(r)}, Y^{(r)}) \cup \{(\mathbf{x}_i, y_i)\big|y_i = c\ \}$
12.   end if
13. end for
14. return ($X^{(l)}, Y^{(l)}, X^{(r)}, Y^{(r)}$)
```

(c_2) to the right child. The sorting operations requires only $O(c \log c)$ computations. For the remaining classes, we compute their corresponding mean vectors, an operation that requires $O(nd)$ time. We then assign each class to the left or right child based on their distance to the mean vectors of the first two largest classes. For example, if the mean vector for c_3 is closer to the mean vector for c_1 than c_2, we assign c_3 to the left child of the root node. Otherwise, we assign it to the right child. This process is repeated until depth τ. This reduces the time complexity of each node from $O(c^3)$ to $O(c \log c + nd)$. As will be shown in our experimental results section, substantial improvement in training time can be achieved by setting τ to be 3 or 4 without losing significant accuracy.

4.4 Experimental Evaluation

In this section, we compared the performance of our proposed MF-Tree algorithm against several state-of-the-art baseline algorithms, including DAGSVM [15], Discriminative Relaxed Hierarchy Learning (DRHL) [56], Confusion matrix based label embedding tree learning approach (CMTL) [18] and Recursive Non-negative Matrix Factorization (RNMF) [20]. In addition to these methods, we have also implemented a recursive version of the HSIC clustering by dependence maximization (RDM) algorithm [23]. The latter would partition the data instances into p groups at each depth of the tree until a stopping criterion is met (i.e., when there are no other classes that have more than 5 instances at the leaf node). We construct a binary classifier at each internal node of the tree as well as 1-class SVM classifiers at the leaf nodes to assign the class labels.

4.4.1 Experimental Setup

To illustrate the effectiveness of the proposed method, our experiments were performed on two real-world data sets: (1) Caltech-256, [3], which is a standard multi-class classification data set with 256 categories containing 30,607 images and at least 80 images for each class, and (2) Wiki, a collection of Wikipedia article dump from October 9, 2009. We generated four subsets with varying number of classes for our experiments. The data sets are denoted as $Wiki_1$, $Wiki_2$, $Caltech_1$ and $Caltech_2$.

$Caltech_1$ and $Caltech_2$ are subsets of image data from the Caltech-256 collection. $Caltech_1$ is generated using the same approach as described in [56], in which we randomly sample 80 images from each class to create the data set. We use a repeated holdout method to create our training and test sets. Specifically, half of the sampled data were reserved for training while the remaining half for testing. We repeat the sampling process five times to create five different versions of the $Caltech_1$ data set. We then applied all the algorithms on the five data sets and compute their average F1-score. The F1-score is computed based on the harmonic mean of the precision (positive predictive value) and recall (sensitivity) for each class. Results are reported based on the average F1-score for all the classes over five versions of the data sets. In addition, we have generated a larger subset called $Caltech_2$ by randomly choosing 40 images from each class for training and all the remaining images in each class for testing. This process is again repeated five times to generate five versions of $Caltech_2$. For both $Caltech_1$ and $Caltech_2$, we have extracted SIFT (scale-invariant feature transform) features to represent each image.

$Wiki_1$ is a data set generated with the same setting as that described in [20]. Here, we choose the largest 214 categories of the Wikipedia articles. The resulting data set contains

[3]http://authors.library.caltech.edu/7694/

Table 4.1 Summary of data used for our experiments

Data set	# Classes	# data instances
$Caltech_1$	256	20480
$Caltech_2$	256	30607
$Wiki_1$	214	24378
$Wiki_2$	1618	65156

24,378 articles. In addition, we also created a larger sample data set from the Wikipedia dump, which covers the largest 1618 categories with 65,156 articles. Similar to $Caltech_1$, half of the sampled data were used for training and the remaining half for testing. The sampling process is also repeated five times to generate different versions of the data sets.

A summary of the data sets used for our experiments is given in table 4.1. Note that all the experiments were conducted on a single PC with Intel Core i7 CPU 2.6GHz and 8 GB memory, running Windows 7 operating system.

4.4.2 Experimental Results

This section presents summary results of our experiments. As previously noted, we use F1-score to measure the accuracy of the different approaches. In addition, we have compared the testing time for each method along with the size of the induced tree.

4.4.2.1 Overall Results Comparison

We first compare the performance of our MF-Tree algorithm against all the baseline methods. The results for data sets $Caltech_1$, $Caltech_2$, $Wiki_1$, and $Wiki_2$ are summarized in Table 4.2. The experimental results suggest the proposed MF-Tree algorithm consistently outperforms other state-of-the-art baselines in terms of their F1-score and testing efficiency on all 4 data sets evaluated in this study.

The size of the hierarchy generated by MF-Tree is comparable to some of the best algorithms. Even though RNMF generates a shorter than than MF-Tree, its test time is higher because the leaf nodes of RNMF may contain multiple classifiers that must be invoked in order to predict the final class. In contrast, the leaf nodes of MF-Tree has a single class label, which allows us to predict the class efficiently. Among all the competing methods, DAGSVM creates the largest tree. This is because it eliminates one class at a time at each level of the hierarchy. So the depth of the tree is equal to the number of classes. In the meantime, DRHL and RDM also create larger trees because they both allow a single class to reside in more than one leaf nodes. This affects the test time of the algorithms.

4.4.2.2 Effect of Parameter Tuning

Unlike MF-Tree, which is parameter-free, the performance of several baseline algorithms depends on the values of their parameters. For example, the branching factor p in RNMF method determines the structure and depth of the tree. If $p = 1$, this produces a decision stump consisting of c 1-class SVM models. This is equivalent to the one-versus-all approach. If $p = 2$, then it will produce a binary tree. The choice of p also affects the depth of the tree. As the depth increases, the test efficiency reduces significantly at the expense of decreasing F1 values. Similarly, tree construction parameters are also needed in other baseline methods. To provide a fair comparison, we vary the parameters for each method based on the suggestion of their original papers. Specifically, for RNMF [20], we set $p \in \{2, 3, 4, 5\})$, for RDM, we set $p = 2, 3, 4, 5$ and for $DRHL$[56], we set $\rho \in \{0.5, 0.6, 0.7, 0.8\})$.

We plotted the results in Figures 4.4, 4.5, 4.2 and 4.3. In each plot, the x-axis corresponds to the test time while the y-axis corresponds to the F1-score. Ideally, we seek for a classifier with lowest test time and highest F1-score. As can be seen from these plots, MF-Tree is

Table 4.2 Experimental Results

$Caltech_1$	F1	Test Time (s)	Depth of Tree
$DAGSVM$	0.369 ± 0.0019	436.8 ± 10.2	256
$RNMF$	0.360 ± 0.0022	91.2 ± 6.4	**9**
$DRHL$	0.375 ± 0.003	312.76 ± 9.6	196
$CMTL$	0.348 ± 0.006	122.18 ± 10.4	10
RDM	0.331 ± 0.0103	198.2 ± 11.2	18
$MF-Tree$	**0.3816 ± 0.03**	**77.5 ± 4.5**	10
$Caltech_2$	F1	Test Time (s)	Depth of Tree
$DAGSVM$	0.356 ± 0.0017	751.68 ± 11.8	256
$RNMF$	0.353 ± 0.0017	157.66 ± 7.9	**9**
$DRHL$	0.365 ± 0.0021	523.7 ± 12.2	196
$CMTL$	0.343 ± 0.005	192.8 ± 11.2	10
RDM	0.323 ± 0.0098	367.4 ± 10.5	18
$MF-Tree$	**0.376 ± 0.029**	**135.6 ± 5.6**	10
$Wiki_1$	F1	Test Time (s)	Depth of Tree
$DAGSVM$	0.5309± 0.002	942.9± 17.6	214
$RNMF$	0.5227 ± 0.0017	188.3± 9.3	**8**
$DRHL$	0.5217 ± 0.0019	457.5 ± 16.9	64
$CMTL$	0.4826 ± 0.0017	195.5 ± 9.2	10
RDM	0.4229 ± 0.0135	256.6 ± 14.7	33
$MF-Tree$	**0.5406 ± 0.0216**	**152.7 ± 4.2**	9
$Wiki_2$	F1	Test Time (s)	Depth of Tree
$DAGSVM$	0.3106± 0.0021	1618.6± 17.6	1618
$RNMF$	0.2998 ± 0.0017	351.8± 13.6	**10**
$DRHL$	0.3089 ± 0.0016	772.5 ± 16.9	116
$CMTL$	0.2459 ± 0.0029	312.2 ± 11.3	12
RDM	0.2296 ± 0.022	456.6 ± 12.5	52
$MF-Tree$	**0.3318 ± 0.016**	**267.7 ± 6.9**	11

consistently better than the baseline methods on all four data sets.

4.4.3 Comparison of Taxonomy Structure

In addition to their F1-scores, it is useful to compare the tree generated by the different methods against their ground truth structure. To do this, we need an evaluation measure for comparing the similarity between two trees. In this chapter, we apply the edit distance measure, which was originally used for string comparison, to compare the ordered labeled

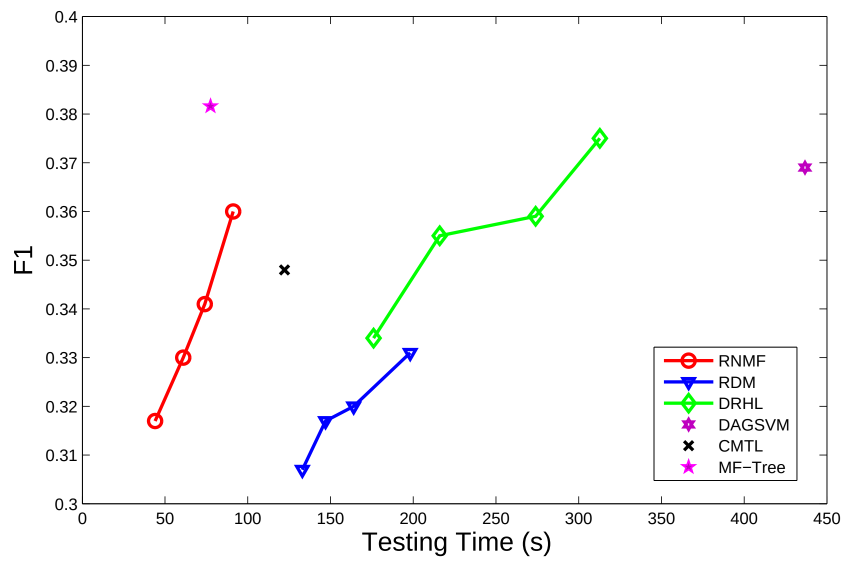


Figure 4.2 Performance Comparison on $Caltech_1$ Data

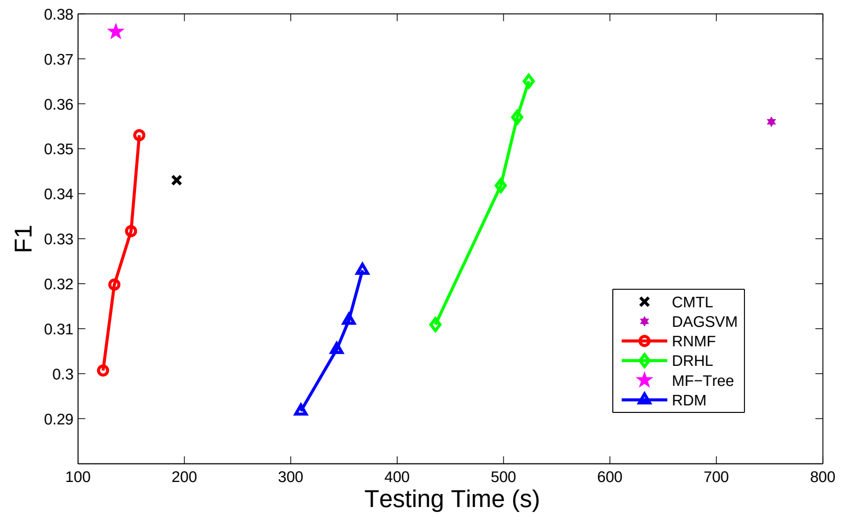


Figure 4.3 Performance Comparison on $Caltech_2$ Data

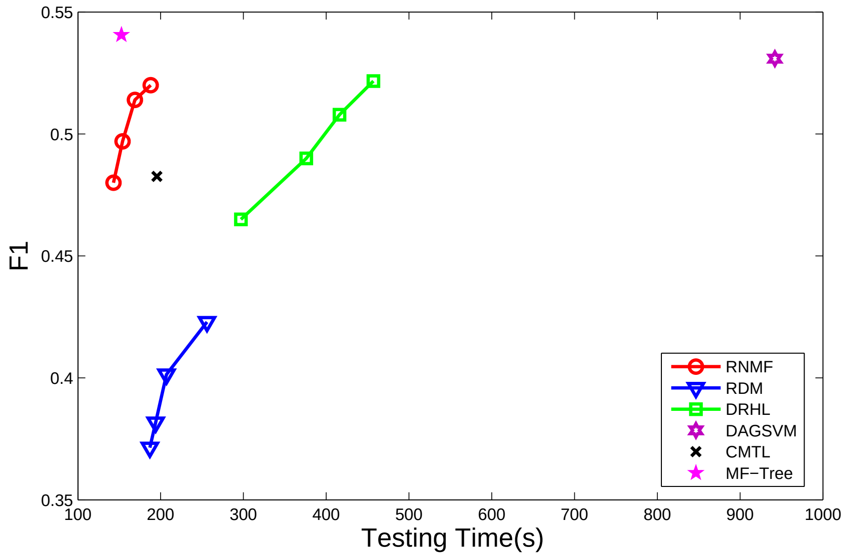


Figure 4.4 Performance Comparison on $Wiki_1$ Data

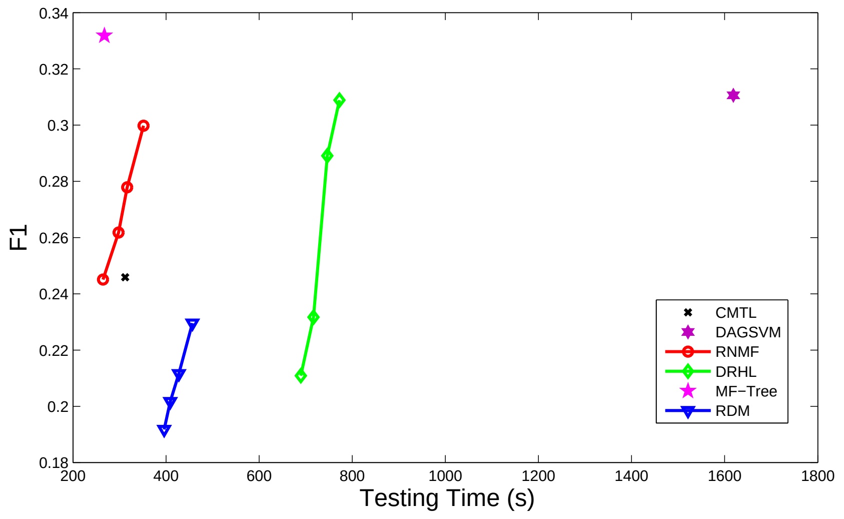


Figure 4.5 Performance Comparison on $Wiki_2$ Data

trees (see [65] for a review). Ordered labeled trees are trees in which the left-to-right order among siblings is significant. The distance between two trees is computed by considering the optimal mapping between the two trees. Specifically, the distance is given by the minimum cost of elementary operations to convert one tree into the other. An alternative way to map and edit the trees is by using tree alignment [66].

To quantify the difference between two trees based on their pairwise class ordering, we first construct an adjacency matrix A for a tree structure using the following formula:

$$A_{ij} = \frac{1}{num_{hops}(c_i \rightarrow c_j)}$$

where A_{ij} is the proximity between classes c_i and c_j and $num_{hops}(c_i \rightarrow c_j)$ is the number of hops to traverse from class c_i to c_j. To measure the difference between two tree structures represented by the matrices A and $\hat{A}$, we compute the mean squared error (MSE), which is defined as follows:

$$MSE(A, \hat{A}) = (\frac{1}{n} \sum_{i,j=1}^{n} (A_{ij} - \hat{A}_{ij})^2)$$

Figure 4.7 shows an example of using MSE to compare the tree structures. Let A be the adjacency matrix induced from the ground truth structure while $\hat{A}_{result1}$ and $\hat{A}_{result2}$ are the corresponding adjacency matrices for two competing tree structures, $result1$ and $result2$. Since $MSE(A, \hat{A}_{result1}) = 0.03125 > MSE(A, \hat{A}_{result2}) = 0.013825$, this suggests that the tree structure $result2$ is closer to the ground truth structure than $result1$. This result makes sense because the structure of $result2$ is more similar to the ground truth as only class c_2 is misplaced, whereas the classes c_2 and c_3 are misplaced in $result1$. For example, $A(c_3, c_4) = 0.5$ in the ground truth structure as the number of hops from category c_3 to c_4 is equal to two. However, $\hat{A}_{result1}(c_3, c_4) = 0.25$ for $result1$ since class c_3 is 4 hops

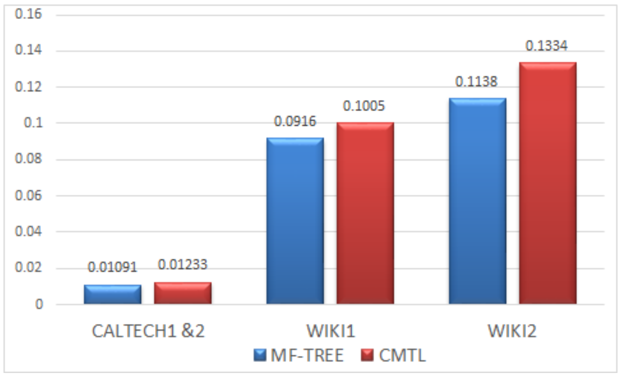
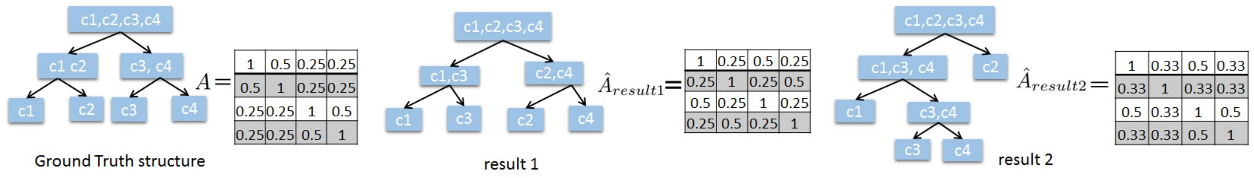


Figure 4.6 MSE comparison of the tree produced by MF-Tree against the one produced by CMTL



1	0.5	0.25	0.25
0.5	1	0.25	0.25
0.25	0.25	1	0.5
0.25	0.25	0.5	1

1	0.25	0.5	0.25
0.25	1	0.25	0.5
0.5	0.25	1	0.25
0.25	0.5	0.25	1

1	0.33	0.5	0.33
0.33	1	0.33	0.33
0.5	0.33	1	0.5
0.33	0.33	0.5	1

Figure 4.7 Example of Mapping Tree Structure to Adjacency Matrix

away from c_4. In contrast, $\hat{A}_{result2}(c_3, c_4) = 0.5$, which is the same as the distance according to the ground truth, which means, the tree structure of $result2$ was able to maintain the proximity between c_3 and c_4.

We compared the MSE score of MF-Tree against the confusion matrix based label embedding tree learning (CMTL) method. We choose this baseline method for two reasons. First, the size of the tree is comparable to MF-Tree. Second, similar to MF-Tree, this approach produces a tree in which each class can only reside in a single leaf node of the tree (whereas other methods allow a class to be assigned to multiple leaf nodes). Trees that restrict each class to a single leaf node are more interpretable as they may be used to define a concept hierarchy for the application domain. Our experimental results shown in Figure 4.6 suggest that MF-Tree produces a tree structure that is closer to the ground truth compared to CMTL. Note that $Caltech_1$ and $Caltech_2$ use the classes for their training data to construct the taxonomy, so we report them together as $Caltech1\&2$ in Figure 4.6.

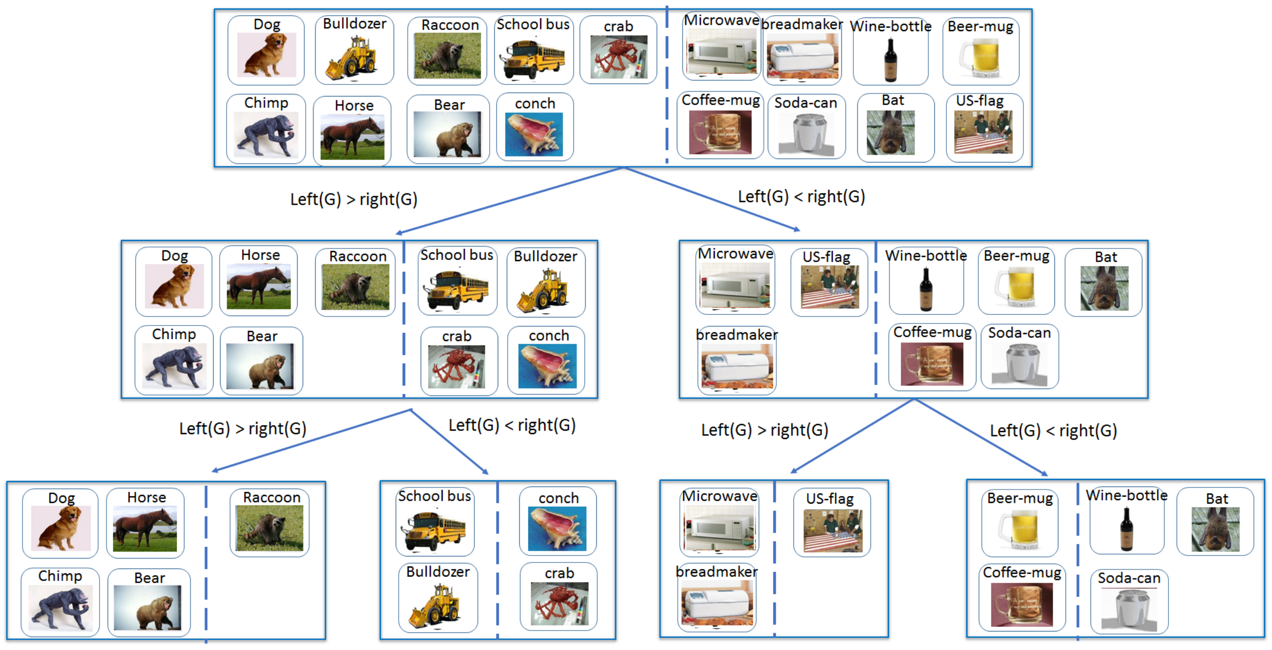


Figure 4.8 Illustration of tree structure generated

Figure 4.8 shows a subset of the tree structure generated by MF-Tree on the $Caltech_1$ data set. Observe that the tree was able to capture the relationships among many of the classes quite well. For example, some of the classes of animals (dog, horse, chimp, bear) was assigned to the same branch. Though there were exceptions and misclassifications (e.g., bats and crabs) but the related classes are often quite close together at the lower levels of the tree.

4.4.4 Results for Approximate MF-Tree

Despite its lower test time compared to other approaches, MF-Tree can still be expensive if the number of classes is too large. To overcome this problem, we propose an approximate MF-Tree learning algorithm. To justify the effectiveness of the approximate MF-Tree approach, we perform an experiment on the $Wiki_2$ data set. We choose this data set because it has the largest number of classes. As previously mentioned in Section 4.3, the approximate MF-Tree method uses an inexpensive partitioning method to assign classes to their child nodes at the top levels of the hierarchy. The algorithm requires a single parameter, τ, which determines the maximum depth at which to apply the inexpensive partitioning method.

The results comparing Approximate MF-Tree against MF-Tree for different thresholds of τ are summarized in Table 4.3. For this experiment, we vary τ from 3 to 6. We compare the average F1-score as well as the training time of the algorithms. Recall that the average F1-score for MF-Tree on the $Wiki_2$ data set is 0.3318. When $\tau = 3$, the F1-score reduces slightly to 0.3289. However, there was a 22.6% improvement in the training time of the algorithm. As τ increases, the F1-score gradually decreases while its training time continues to improve. Note that the size of the tree generated by the original MF-Tree algorithm has a depth equals to 11. When $\tau = 6$, the improvement in training time appears to taper off while its F1-score has decreased to 0.2868. This clearly shows a trade-off between accuracy and training time efficiency. In practice, we could set the τ threshold based on the specific needs of the problem. If our priority is lower training cost, we could set a higher τ value. τ can also be set based on the computational resources available. If there is limited memory, τ can be set to a threshold in such a way that the resulting data matrices can fit into the memory available.

Table 4.3 Approximate MF-Tree on $Wiki_2$

Threshold	F1	% improvement in training time
$\tau = 3$	0.3289	22.6%
$\tau = 4$	0.3169	32.3%
$\tau = 5$	0.3034	37.8%
$\tau = 6$	0.2868	39.5%

4.5 Conclusion

In this chapter, we proposed a novel hierarchical method for large-scale multi-class learning called MF-Tree. Our proposed algorithm is driven by a global objective function. We demonstrate the equivalence between the global objective function and the HSIC metric and show it has an additive property that can be exploited to design a hierarchical tree learning algorithm. Experimental results comparing the method against five baseline methods demonstrated the effectiveness of our method, in terms of its accuracy and test efficiency.

Chapter 5

Hierarchical Tree Learning for Large Multi-class Network

As noted in Chapter 1, classification of network data is an important problem across many disciplines, including computer science, biology, chemistry and social science. Previous work on network classification can be broadly categorized into three approaches: **Classification with node attributes only [30][31]**. The methods that belong to this category simply apply existing classifiers, such as support vector machine, rule-based classifiers, and logistic regression, to the attributes of each node in the network (e.g., the HTML tags in Web pages). Many efforts have been made to enhance the classification accuracy by constructing a better feature matrix for the nodes [30][31][67]. A major drawback of these methods is that they do not consider the link information when constructing the classifier. **Classification based on link analysis**. This includes PageRank based methods [34][35], HITS [33], local and global consistency[36], score propagation[37], hard label propagation [68], trust rank[38], anti-trust rank[35], minimum cut[69][70], spectral clustering[71],etc. This approach is based on the fundamental assumption that nodes that are connected should share similar labels. As a result, network classification task could be formulated as a regularized optimization problem. The drawback of link analysis based approaches is that the classification relies only on link structure is susceptible to missing link information. **Classification based on**

the use of both content and link information. This includes hybrid methods developed in [39][72][40][41][27][42][25]. Most of these research have shown that hybrid methods are generally better than using node attributes or link information alone.

Despite the extensive research on node classification for network data, most of the approaches were not designed for large multi-class learning problems. These approaches will encounter challenges such as class imbalance, high testing cost, and model interpretability problems when applied to networks with large number of classes. Furthermore, as noted in Chapter 1, the assumption that nearby nodes have similar classes is less likely to hold when the number of classes is large. The size of the network also increases considerably as the number of classes increases, all of which makes large multi-class network learning an extremely challenging problem.

As discussed in the previous two chapters, hierarchical learning methods have been developed [15, 16, 17, 18, 19] to overcome the challenges of extreme multi-class classification problems. However, as all of these methods were designed for non-network data, extending them to a hybrid network classification approach (using both node attributes and link structure information) is not an easy task. Furthermore, although open source tools are available for large scale network analysis, they are not designed for node classification tasks. Pegasus [1] [73], for example, was designed to analyze networks with billions of nodes and edges by providing functionalities to compute the diameter of the graph, the Pagerank scores of nodes, as well as to find connected components using techniques such as spectral clustering. The techniques developed in Pegasus rely on the link topology only, which overlooks the importance of node attributes in the analysis. As a result, there is an urgent need for new techniques to address the challenges in large multi-class network learning.

[1]http://www.cs.cmu.edu/ pegasus/index.htm

In the previous chapter, we have presented a framework for multi-class learning known as MF-Tree. This chapter presents an extension of the framework for large multi-class learning in network data. The modified framework systematically combines node attribute with link information to learn the hierarchical structure of the classes from network data. Specifically, we recursively optimize a global objective function to learn a binary discriminant function to partition the classes based on the node attributes and link structure until the partition contains only a single class. During testing, a test example is routed from the root to one of the leaf nodes in the tree structure, from which the label associated with the leaf node is assigned as the predicted class label.

To address the efficiency issue of large-scale network learning, our framework also has an approximate version, which allows the partitions at the top levels of the tree to be determine without requiring the expensive computation of matrix factorization. Similar to the approach described in the previous chapter, the framework allows us to choose the proper parameter setting (τ) based on the specific needs of the problem. If our priority is to achieve a high training efficiency, we could apply the approximate MF-Tree with larger τ. Otherwise, we could use regular MF-Tree to obtain the best classification accuracy. Second, the proposed approach is trivially parallelizable. A MapReduce implementation of the framework for network data has been developed to improve its scalability to larger networks. As proof of concept, the MapReduce framework was applied to the Wikipedia network data that contains more than 600K classes. To the best of our knowledge, this is the largest multi-class network learning problem that has ever been reported in the literature.

5.1 Methodology

Let $\mathcal{N} = (V, E, D)$ be a network of order n, where $V = \{v_1, v_2, \cdots, v_n\}$ is the set of nodes, $E \subseteq V \times V$ is the set of edges, and D is the set of data instances associated with V. Each instance in D is characterized by the tuple $(\mathbf{x}, y)$, where $\mathbf{x} \in \Re^d$ is a d-dimensional feature vector (known as the node attribute) and $y \in \{1, 2, \cdots, c\}$ is its class label. The goal of multi-class network learning is to induce a labeling function $f(V, E, D)$ that minimizes the expected misclassification rate for any instance drawn from network $\mathcal{N}$, i.e., $\min_f E_{(\mathbf{x},y)\sim D}[\ell(f(\mathbf{x}), y)]$, where ℓ is the loss function. The objective function used as an estimate of the regularized empirical loss function is described in the next subsection.

5.1.1 Global Objective Function

Let $\mathbf{X}$ be an $n \times d$ centered design matrix, whose rows correspond to the data instances and columns correspond to the node attributes. An uncentered design matrix $\hat{\mathbf{X}}$ can be easily centered by applying the matrix operation $\mathbf{H}_n\hat{\mathbf{X}}$, where $\mathbf{H}_n$ is the centering matrix defined as $\mathbf{H}_n = \mathbf{I} - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^T$. Let $\mathbf{A}$ denote the $n \times n$ adjacency matrix for the link structure of the network.

Our MF-Tree for network is then designed to optimize the following regularized loss function for an undirected graph:

$$\begin{aligned}
\mathcal{J} &= \|X - LW^T\|_F^2 + \lambda_1\|A - LPL^T\|_F^2 + \lambda_2\|W\|_F^2 + \lambda_3\|PL^T\|_F^2 \qquad (5.1)\\
&= \mathbf{tr}\Big[XX^T - XWL^T - LW^TX^T + LW^TWL^T\Big]\\
&\quad +\lambda_1\mathbf{tr}\Big[AA^T - ALP^TL^T - LPL^TA^T + LPL^TLP^TL^T\Big]\\
&\quad +\lambda_2\mathbf{tr}\Big[WW^T\Big] + \lambda_3\mathbf{tr}\Big[UU^T\Big]
\end{aligned}$$

where **L** is an $n \times c$ matrix, which indicates the class membership of each data instance, **W** is a $d \times c$ matrix, which represents the feature vector of each class, and **P** is a $c \times c$ is class covariance matrix, which represents the correlation between classes. λ_1, λ_2 and λ_3 are the regularization parameters. Intuitively, the objective function seeks to approximate the matrices X and A by their low rank approximation matrices given by the matrix products LW^T and LPL^T, respectively. The first two terms in the objective function aim to measure the error in the approximation while the last two terms are regularizers for the matrices W and PL^T.

The purpose of optimizing this objective function is to jointly factorize the centered data matrix **X** and adjacency matrix **A** to learn **L** and **W**. For directed graph, we can write $PL^T = U^T$ and reduce the objective function as follows:

$$\begin{aligned}
\mathcal{J} &= \|X - LW^T\|_F^2 + \lambda_1 \|A - LU^T\|_F^2 + \lambda_2 \|W\|_F^2 + \lambda_3 \|U\|_F^2 \qquad (5.2)\\
&= \mathbf{tr}\Big[XX^T - XWL^T - LW^TX^T + LW^TWL^T\Big]\\
&\quad + \lambda_1 \mathbf{tr}\Big[AA^T - AUL^T - LU^TA^T + LU^TUL^T\Big]\\
&\quad + \lambda_2 \mathbf{tr}\Big[WW^T\Big] + \lambda_3 \mathbf{tr}\Big[UU^T\Big]
\end{aligned}$$

5.1.2 Relationship between Proposed Method and HSIC

As we have discussed in the previous chapter, the Hilbert-Schmidt Independence Criterion (HSIC) [21][22][23][24] is a recently proposed unsupervised learning measure for taxonomy learning. We have provided a theoretical proof to demonstrate the equivalence between the objective function of MF-Tree and HSIC. Similarly, in this section, we first show that the network extension of MF-Tree is also equivalent to HSIC. Given the additive property of

HSIC, the task of large multi-class network learning can be decomposed into a set of binary classification tasks, which can be organized in a hierarchical tree structure.

To demonstrate the relationship between our global objective function and Hilbert-Schmidt Independence Criterion (HSIC), we first take the partial derivative of $\mathcal{J}$ with respect to **W** and **U** and set them to zero:

$$\begin{aligned}
\frac{\partial(\mathcal{J})}{\partial(W)} &= -2X^TL + 2WL^TL + 2\lambda_2 W = 0 \\
& W = X^TL[L^TL + \lambda_2 I]^{-1} \\
\frac{\partial(\mathcal{J})}{\partial(U)} &= \lambda_1(-2A^TL + 2UL^TL) + 2\lambda_3 U = 0 \\
&= U[\lambda_1 L^TL + \lambda_3 I] = \lambda_1 A^TL \\
& U = A^TL[L^TL + \frac{\lambda_3}{\lambda_1} I]^{-1}
\end{aligned}$$

Replacing these back into Equation (5.3), we have:

$$\begin{aligned}
\mathcal{J} \;=\; & \mathbf{tr}\Big\{XX^T - 2XX^TL\Big[L^TL + \lambda_2 I\Big]^{-1}L^T \\
& + L\Big[L^TL + \lambda_2 I\Big]^{-1}L^TXX^TL\Big[L^TL + \lambda_2 I\Big]^{-1}L^T\Big\} \\
& + \lambda_1 \mathbf{tr}\Big\{AA^T - 2AA^TL\Big[L^TL + \frac{\lambda_3}{\lambda_1}I\Big]^{-1}L^T \\
& + L\Big[L^TL + \frac{\lambda_3}{\lambda_1}I\Big]^{-1}L^TAA^TL\Big[L^TL + \frac{\lambda_3}{\lambda_1}I\Big]^{-1}L^T\Big\} \\
& + \lambda_2 \mathbf{tr}\Big\{\Big[L^TL + \lambda_2 I\Big]^{-1}L^TXX^TL\Big[L^TL + \lambda_2 I\Big]^{-1}\Big\} \\
& + \lambda_3 \mathbf{tr}\Big\{\Big[L^TL + \frac{\lambda_3}{\lambda_1}I\Big]^{-1}L^TAA^TL\Big[L^TL + \frac{\lambda_3}{\lambda_1}I\Big]^{-1}\Big\}
\end{aligned}$$

$$\begin{aligned}
&= \mathbf{tr}\Big\{XX^T - 2XX^TL\Big[L^TL+\lambda_2I\Big]^{-1}L^T \\
&\quad +\Big[L^TL+\lambda_2I\Big]^{-1}L^TXX^TL\Big[L^TL+\lambda_2I\Big]^{-1}L^TL\Big\} \\
&\quad +\lambda_1\mathbf{tr}\Big\{AA^T - 2AA^TL\Big[L^TL+\frac{\lambda_3}{\lambda_1}I\Big]^{-1}L^T \\
&\quad +\Big[L^TL+\frac{\lambda_3}{\lambda_1}I\Big]^{-1}L^TAA^TL\Big[L^TL+\frac{\lambda_3}{\lambda_1}I\Big]^{-1}L^TL\Big\} \\
&\quad +\lambda_2\mathbf{tr}\Big\{\Big[L^TL+\lambda_2I\Big]^{-1}L^TXX^TL\Big[L^TL+\lambda_2I\Big]^{-1}\Big\} \\
&\quad +\lambda_3\mathbf{tr}\Big\{\Big[L^TL+\frac{\lambda_3}{\lambda_1}I\Big]^{-1}L^TAA^TL\Big[L^TL+\frac{\lambda_3}{\lambda_1}I\Big]^{-1}\Big\}
\end{aligned}$$

Assuming $\lambda_1 = 1$ and $\lambda_2 = \lambda_3 = \lambda$, $\mathcal{J}$ can be further simplified as

$$\begin{aligned}
\mathcal{J} &= \mathbf{tr}\Big\{XX^T - 2XX^TL\Big[L^TL+\lambda I\Big]^{-1}L^T \\
&\quad +\Big[L^TL+\lambda I\Big]^{-1}L^TXX^TL\Big[L^TL+\lambda I\Big]^{-1}\Big[LL^T+\lambda I\Big]\Big\} \\
&\quad +\mathbf{tr}\Big\{AA^T - 2AA^TL\Big[L^TL+\lambda I\Big]^{-1}L^T \\
&\quad +\Big[L^TL+\lambda I\Big]^{-1}L^TXX^TL\Big[L^TL+\lambda I\Big]^{-1}\Big[LL^T+\lambda I\Big]\Big\} \\
&= \mathbf{tr}\Big\{XX^T - XX^TL\Big[L^TL+\lambda I\Big]^{-1}L^T\Big\} \\
&\quad +\mathbf{tr}\Big\{AA^T - AA^TL\Big[L^TL+\lambda I\Big]^{-1}L^T\Big\} \\
&= \mathbf{tr}\Big\{\Big[XX^T+AA^T\Big] - \Big[XX^TL[L^TL+\lambda I]^{-1}L^T + AA^TL[L^TL+\lambda I]^{-1}L^T\Big]\Big\} \\
&= \mathbf{tr}\Big\{\Big[XX^T+AA^T\Big] - \Big[XX^T+AA^T\Big]L[L^TL+\lambda I]^{-1}L^T\Big\}
\end{aligned}$$

The term $\Big[XX^T + AA^T\Big]$ can be regarded as the joint kernel matrix of both content and link structure, which already known given the design matrix and adjacency matrix. Thus, finding a solution that minimizes $\mathcal{J}$ is equivalent to solving the following trace form:

$$\max_{L} \mathbf{tr}\Big[XX^T + AA^T\Big] L[L^T L + \lambda I]^{-1} L^T \tag{5.3}$$

Let $K = XX^T + AA^T$ [2], we could enforce the following constraint during training:

$$L_{train}[L_{train}^T L_{train} + \lambda I]^{-1} L_{train}^T = YCY^T$$

where $Y \in \Re^{n\times c}$ is the class label matrix, $C \in \Re^{c\times c}$ is the tree structure covariance matrix. We could simplify the Equation (5.3) as follow:

$$\max_{C} \mathbf{tr}\Big[KYCY^T\Big] \tag{5.4}$$

where the covariance matrix C associated with the label tree structure is jointly estimated from the link structure and node attribute information of the network. C_{ij} is a measure of similarity between the i-th and j-th classes. The larger the value of C_{ij}, the closer is the relationship between two classes. Thus, we have shown that optimizing our regularized objective function for network learning is equivalent to the optimization of HSIC.

[2]To save the space and running time, we keep K as a sparse matrix. In our implementation, we only maintain the elements in XX^T with value larger than 0.3, this threshold value could be adjusted based on the specific needs of the problem

5.1.3 Additive Property of HSIC

As shown in previous chapter for MF-Tree, the tree-structure covariance matrix has an additive property, which can be formally written as follows:

$$\mathbf{C}_{2^m} = \mathbf{B}_2 + \mathbf{B}_4 + \cdots + \mathbf{B}_{2^m}, \tag{5.5}$$

where $\mathbf{B}_{2^j} = \mathbf{I}_{2^j} \otimes \mathbf{1}_{2^{m-j}} \mathbf{1}_{2^{m-j}}^T$, $\mathbf{1}_{2^0} = 1$, and m is the depth of the binary tree. Thus, we can write:

$$\mathbf{YCY}^T = \sum_{i=1}^{\log_2 k} \mathbf{YB}_{2^i}\mathbf{Y}^T = \sum_{i=1}^{\log_2 k} \sum_{j=1}^{2^{i-1}} \mathbf{Y}_j \mathbf{B}_{2^i,j} \mathbf{Y}_j^T$$

This additive property enables us to solve the optimization problem with a hierarchical tree-like decomposition. Instead of learning $\mathbf{C}$ directly by maximizing the objective function in (5.4), we can learn a set of block matrices $\mathbf{B}_{2^i,j}$ iteratively, each of which is associated with an internal node of the tree structure.

5.1.4 Hierarchical Tree Construction from Network Data

To construct the tree, we need to solve the objective function given in (5.4). Using the tree decomposition approach given in Equation (5.5), we simplify this to

$$\max_{\{\mathbf{B}\}} \sum_{i,j} \mathbf{tr}\left[\mathbf{K}_{j,j} \mathbf{Y}_j \mathbf{B}_{i,j} \mathbf{Y}_j^T\right] \tag{5.6}$$

where the optimization problem can be solved iteratively, one block matrix B at a time. Similar to MF-Tree, by setting $\mathbf{C}_j = \mathbf{Y}_j^T \mathbf{K}_{j,j} \mathbf{Y}_j$, our tree construction step is equivalent to finding a matrix $B_{i,j}$ that maximizes its alignment to C_j.

$$\max_{\{\mathbf{B}\}} \sum_{i,j} \mathbf{tr}\left[\mathbf{C}_j \mathbf{B}_{i,j}\right] \tag{5.7}$$

For brevity, we omit the subscript (i,j) in the rest of the sections. Since $\mathbf{B}$ is a symmetric matrix, let $\mathbf{B} = \mathbf{G}\mathbf{G}^T$, where $\mathbf{G} \in \Re^{c\times 2}$. It is easy to show that:

$$\mathbf{tr}\left[\mathbf{CB}\right] = \mathbf{tr}\left[\mathbf{CGG}^T\right] = \mathbf{tr}\left[\mathbf{G}^T\mathbf{CG}\right].$$

Following the same approach as MF-Tree, we can relax the condition that elements of $\mathbf{G}$ must be either 0 or 1 and add a condition that $\mathbf{G}$ is an orthogonal matrix:

$$\max_{\mathbf{G}} \mathbf{tr}\left[\mathbf{G}^T\mathbf{CG}\right] \quad \text{s.t.} \quad \mathbf{G}^T\mathbf{G} = \mathbf{I}_2$$

The optimization problem can be easily solved by finding the first two eigenvectors of $\mathbf{C}$. After performing eigenvalue decomposition on $\mathbf{C}$, we can write:

$$\mathbf{C} = \mathbf{G}\boldsymbol{\Lambda}\mathbf{G}^T \tag{5.8}$$

Once the matrix $\mathbf{G}$ is known, we can apply the following equation to determine the partition each test example show follow:

$$\hat{\mathbf{p}} = \mathbf{G}^T\mathbf{Y}^T\hat{\mathbf{k}},$$

where $\hat{\mathbf{k}}$ is the kernel similarity between the test example and the training examples. This is equivalent to the formula used for predicting the branch to follow in the MF-Tree framework described in the previous section. After computing $\hat{\mathbf{p}}$, we assign the test node to the partition with largest value. We summarized the hierarchical tree construction procedure in Algorithm 5 and the node testing procedure in Algorithm 6.

Algorithm 5 Hierarchical Tree Construction from Network Data

```
Input: $G = (V, E, D)$, $(X_{train}, Y_{train}) \subset D$, $X_{train} \in \mathbf{R}^{n\times d}$, $Y_{train} \in \mathbf{R}^{n\times k}$
Output: Tree Structure, $T$
1. construct $A$ from $G$
2. $root$ = TreeGrowth($X_{train}, Y_{train}, A$)
3. Insert $root$ into $T$
4. return $T$

function TreeGrowth($X$,$Y$,$A$)
1. $v$ = createNode()
2. $v.index$ = getIndex($X$)
3. if number of classes in $Y < 2$ then
4.     $v.class$ = unique($Y$)
5. else
6.     $(X^{(l)}, Y^{(l)}, X^{(r)}, Y^{(r)}, v.\mathbf{g})$ = Partition($X$,$Y$,$A$)
7.     $v.left$ = TreeGrowth($X^{(l)}$, $Y^{(l)}$,$A^{(l)}$)
8.     $v.right$ = TreeGrowth($X^{(r)}$, $Y^{(r)}$, $A^{(r)}$)
9. end if
10. return $v$

function Partition($X$,$Y$,$A$)
1. Compute the centered kernel $\mathbf{K}$ from $X$ and $A$.
2. Compute $\mathbf{C} = Y^T\mathbf{K}Y$
3. Compute the first two eigenvectors $g^{(1)}, g^{(2)}$ of $\mathbf{C}$
4. Let $\mathbf{g} = [g^{(1)}, g^{(2)}]$
5. $(X^{(l)}, Y^{(l)}) = \{(\mathbf{x}_i, y_i) | y_i = k, \arg\max_j \mathbf{g}_{kj} = 1\}$
6. $(X^{(r)}, Y^{(r)}) = \{(\mathbf{x}_i, y_i) | y_i = k, \arg\max_j \mathbf{g}_{kj} = 2\}$
7. return $(X^{(l)}, Y^{(l)}, X^{(r)}, Y^{(r)}, \mathbf{g})$
```

5.1.5 Parallelization of Hierarchical Tree Learning

Our framework can be easily parallelizable. As shown in Algorithm 5 and 6, the key step in the optimization of our objective function is to solve a matrix multiplication problem and to

Algorithm 6 Testing with Hierarchical Tree

```
Input: $X_{test}$, $X_{train}$, $Y_{train}$, $A$, $T$
Output:$Y_{test}$
1. for each $\mathbf{x} \in X_{test}$ do
2.     $v = T.root$
3.     while $v.class = \emptyset$ do
4.         $\hat{k} = \texttt{Similarity}(\mathbf{x}, X_{train}, A, v.index)$
5.         $\hat{Y} = \texttt{Subset}(Y_{train}, v.index)$
6.         $\hat{g} = v.\mathbf{g}$
7.         $\hat{p} = \hat{g}^T \hat{Y}^T \hat{k}$
8.         $v = \texttt{getChild}(\hat{p}, v)$
9.     end while
9.     Insert $y = v.class$ into $Y_{test}$
10. end for
11. return $Y_{test}$
```

find the eigenvectors corresponding to the two largest eigenvalues of the tree structure covariance matrix $\mathbf{C}$. To perform the matrix multiplication and extract the eigenvectors, we apply the HEIGEN algorithm [74], which provides an efficient matrix-matrix multiplication under the Hadoop/MapReduce distributed programming framework. HEIGEN has an eigensolver for billion-scale matrices to compute the top k eigenvectors using the Hadoop/MapReduce framework. We evaluated our parallelization approach for MF-Tree on a cluster of machines running MapReduce with 128 worker nodes have 8 cores and 32 GB RAM each. The results are reported in the next section.

5.2 Experimental Evaluation

This section presents the results of our experiments to evaluate the performance of the enhanced MF-Tree framework for network data.

5.2.1 Network Data

We evaluated the performance of the proposed framework on two real-world network data—U.S Patent and Wikipedia data sets. Details of the data sets are given below.

5.2.1.1 U.S. Patent Data

We have extracted the U.S. patents granted between January 1963 and December 1999 from a website called PatentGenius [3] along with the citations made to these patents between 1975 and 1999 (over 16 million). For each patent, we extracted the title, abstract, category, and citation information from each patent Web page. We ended up with 3,774,768 patents with 16,518,948 citation links. There are 22 general categories in the US Patent Data as shown in Table 5.1. Each general category has its own subcategories, which produces 426 classes under which a patent can be categorized. These classes are further divided into more subclasses, which in turn, produces a hierarchical structure between classes. For this experiment, we tested our algorithm using 3139 patent U.S. sub-categories. The class distribution of U.S. Patent Data is highly skewed, as shown in Figure 5.1, with the largest class containing 25,727 patents, and the smallest class has only 8 patents. A patent citation network is constructed from the data, where each patent is a node while its title and abstract represent the node attributes. The edge set contains citations between pairs of patents.

5.2.1.2 Wikipedia Data

We created network data from Wikipedia dump dated October 9, 2009, where each Wikipedia article is a node and its title and content form the node attributes. The link set contains citations between pair of articles. We generated two versions of the network data for evaluation

[3] http://www.patentgenius.com/

Table 5.1 22 General Categories of U.S. Patent Data

Class	No. of Patents
Business	26790
Chemistry	441687
Clothing, Fashion & Accessories	52895
Communications	281045
Construction	112621
Electrical & Energy	383168
Engineering	84506
Entertainment & Recreation	36342
Firearms & Weapons	5968
Food	20896
Health & Medicine	227264
Household	96166
Industrial	393949
Information Technology	479869
Material Science	418611
Optical & Optics	147182
Packaging & Containers	93128
Papers, Printing & Office Supplies	52285
Physics	159269
Plants, Animals & Agriculture	25913
Tools & Hardware	82137
Transportation & Automotive	153077

purposes.

1. **Wikipedia Data 1**, which contains articles from all the subclasses of the biology category. There are altogether 98,068 articles belonging to 2,924 classes with 331,916 hyperlinks between articles.

2. **Wikipedia Data 2**, which contains the entire 2,365,436 articles belonging to 614,428 classes.

Similar to the U.S. Patent data, the class distribution for Wikipedia data 1 is also skewed, as shown in Figure 5.1. Due to the imbalanced class distribution, accuracy is not a good metric for evaluating the performance of the framework. We will discuss the evaluation metric used

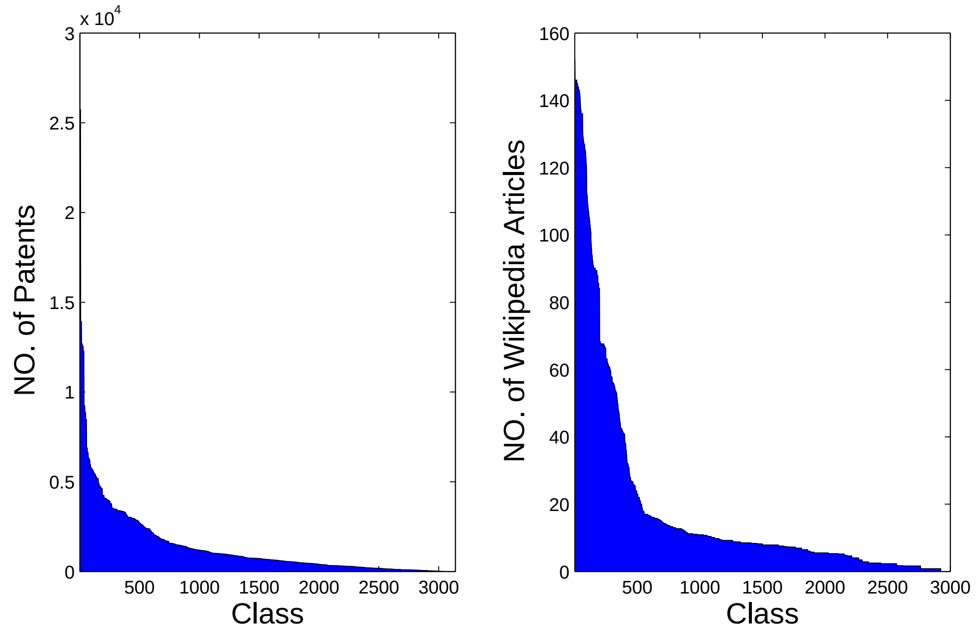


Figure 5.1 Class Distribution of U.S. Patent and Wikipedia Data 1

in the next section.

5.2.2 Evaluation Metric

Accuracy is a widely used evaluation metric for classification tasks. The metric can be computed from a confusion matrix that summarizes the total number of examples correctly and incorrectly predicted by a classification method. Figure 5.2 shows an example of a confusion matrix for a binary classification problem, where the columns are the predicted classes and the rows are the actual classes. The two classes are denoted as positive and negative class, respectively. In the confusion matrix, true positive (TP) is the number of positive examples correctly predicted, true negative (TN) is the number of negative examples correctly predicted, false positive (FP) is the number of negative examples incorrectly classified as positive, while false negative (FN) is the number of positive examples incorrectly classified

	Predicted Negative	**Predicted Positive**
Actual Negative	TN	FP
Actual Positive	FN	TP

Figure 5.2 Confusion Matrix

as negative. Then accuracy is formally defined as

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

.

Unfortunately, accuracy is not an appropriate metric for data sets with imbalanced classes. For example, if we have 99 instances belonging to the negative class and 1 instance from the positive class, then a classifier that predicts all 100 instances as negative class will have an accuracy of 99% even though it fails to detect any instance from the positive class. Instead of accuracy, we report the experimental results using both micro and macro-averaged F1 scores [75]. The metrics are formally defined as follows:

5.2.2.1 Micro-average F1

Let TP_k, FP_k, and FN_k be the true positive, false positive, and false negative of class $k \in C$. The micro-average F1 score is defined as:

$$\begin{aligned}
\text{Precision} &= \frac{\sum_{k \in C} TP_k}{\sum_{k \in C}(TP_k + FP_k)} \\
\text{Recall} &= \frac{\sum_{k \in C} TP_k}{\sum_{k \in C}(TP_k + FN_k)} \\
\text{Micro-average F1} &= \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision+Recall}}
\end{aligned}$$

Note that micro-average F1 is computed based on the average precision and recall values of all the test instances.

5.2.2.2 Macro-average F1

The macro-average F1 score is defined as:

$$\begin{aligned}
\text{Precision}_k &= \frac{TP_k}{TP_k + FP_k} \\
\text{Recall}_k &= \frac{TP_k}{TP_k + FN_k} \\
\text{Macro-average F1} &= \frac{1}{|C|} \sum_{k \in C} \frac{2 \times \text{Precision}_k \times \text{Recall}_k}{\text{Precision}_k + \text{Recall}_k}
\end{aligned}$$

The metric is computed by averaging the F1 scores for all the classes.

When the class distribution is skewed, the macro-average F1 score will be highly influenced by the F1-score of the rare classes. In contrast, the micro-average F1 score will be dominated by the performance of the classifier on instances from the larger categories.

5.2.3 Experimental Results

This section describes the results of our experiments on both U.S. Patent and Wikipedia network data sets. We compared the hierarchical tree learning framework (denoted as `HTL`) against a well-known label propagation algorithm [36] and the decision tree classifier (applied to node attributes only). For this experiment, we apply the holdout method, where half of the network data was used for training and the remaining half was reserved for testing purposes.

5.2.3.1 Results on U.S. Patent Data

Tables 5.2 and 5.3 showed the micro- and macro-average F1 scores for the various algorithms on the U.S. Patent Data. For this experiment, we vary the number of classes by mapping the patents to their corresponding classes at different levels of the class hierarchy. Recall that the top level of the class hierarchy contains 22 classes, while the second level contains 426 classes, and the subsequent level contains 3,139 classes.

Table 5.2 Micro-average F1 Comparison on U.S. Patent Data

No. Classes	Label Propagation (Link only)	Label Propagation (Link + Content)	Decision Tree	HTL
$k = 22$	0.582	0.592	0.569	**0.622**
$k = 426$	0.500	0.511	0.490	**0.512**
$k = 3139$	0.312	**0.330**	0.302	0.328

Table 5.3 Macro-average F1 Comparison on U.S. Patent Data

No. Classes	Label Propagation (Link only)	Label Propagation (Link + Content)	Decision Tree	HTL
$k = 22$	0.426	0.457	0.408	**0.466**
$k = 426$	0.338	0.3760	0.316	**0.396**
$k = 3139$	0.257	0.261	0.238	**0.276**

The following observations can be made based on the results shown in the tables:

1. Hybrid approaches are more effective than attribute-only and link-only node classification methods. For example, the performance of the proposed hierarchical tree learning (HTL) method is superior than simple decision tree classifier constructed on the node attributes only, both in terms of their micro- and macro-averaged F1 scores. In addition, a hybrid labeled propagation approach using both the link structure and Web content of the patents consistently outperforms label propagation with link-only information.

2. In terms of macro-average F1 scores, the proposed HTL method is better than the hybrid label propagation method for 3 all cases ($k = 22, 426$, and 3139). As noted earlier, macro-average F1 is mainly influenced by the performance of the rare classes, this result demonstrates the effectiveness of the proposed method for handling imbalanced class problems compared to the other baseline methods.

3. In terms of micro-average F1 scores, the proposed HTL method is better than the hybrid label propagation method on 2 out of 3 cases ($k = 22$ and 426). For $k = 3139$, our results are slightly worse. This suggests that the hybrid label propagation method is slightly better in terms of classifying instances from the larger categories.

To support our previous assertion that HTL is more effective at classifying the rare classes, consider the plot shown in Figure 5.3. The horizontal axis of the plot corresponds to the ID of individual classes in the U.S. Patent data, sorted by their increasing class size. The vertical axis, on the other hand, corresponds to the F1 score of the class. The results clearly showed that the F1 score of HTL for rare classes is higher than those obtained using link propagation and decision tree methods. HTL also outperforms decision tree classifier on all the classes as its curve lies above the curve for decision tree classifier. The figure also shows

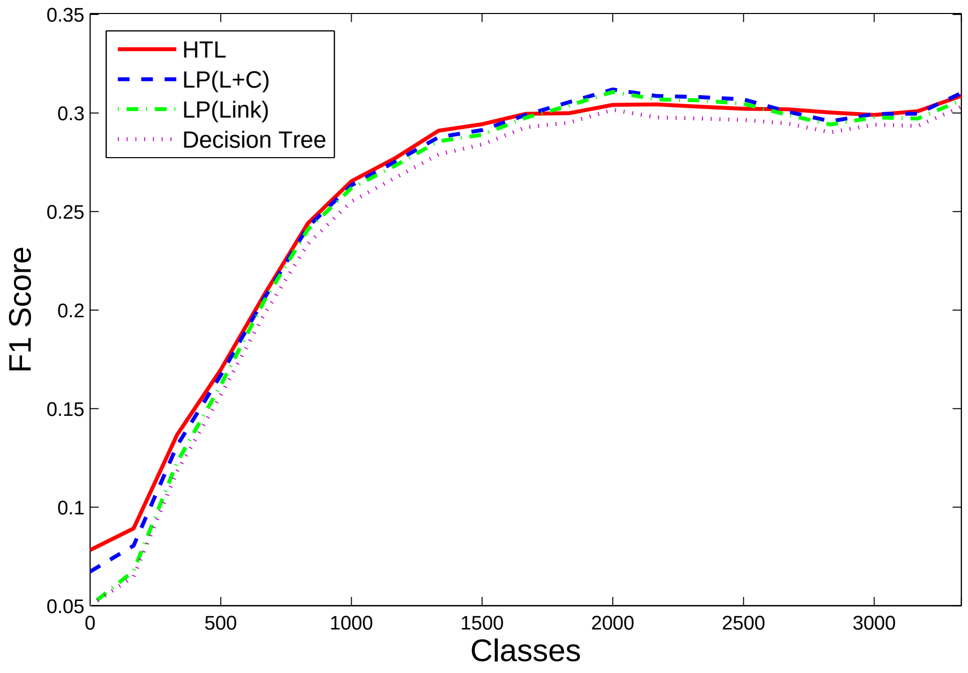


Figure 5.3 F1 Score for each individual class in the U.S. Patent Data (where the classes are sorted in increasing class size).

that the label propagation methods generally work better than HTL on the larger classes. This result is consistent with the micro- and macro-average F1 scores shown in Tables 5.2, 5.3, and 5.5.

Although the hybrid label propagation method is also quite effective, it does not produce a hierarchical tree structure that represents the conceptual relationship between classes. Decision tree classifiers, on the other hand, produces trees that (1) may not include all the classes in their leaf nodes, and (2) may contain classes that are split into multiple leaf nodes in the tree. In contrast, the binary tree generated by the proposed HTL method can be interpreted as a class hierarchy as it includes all the classes in the data and assigns each class to exactly one leaf node in its induced tree. Figure 5.4 provides a snippet of the tree structure obtained after applying HTL to the U.S. patent data, which contains 22 classes. The tree structure (up to depth 4) shows that we can capture the relationships among the classes very

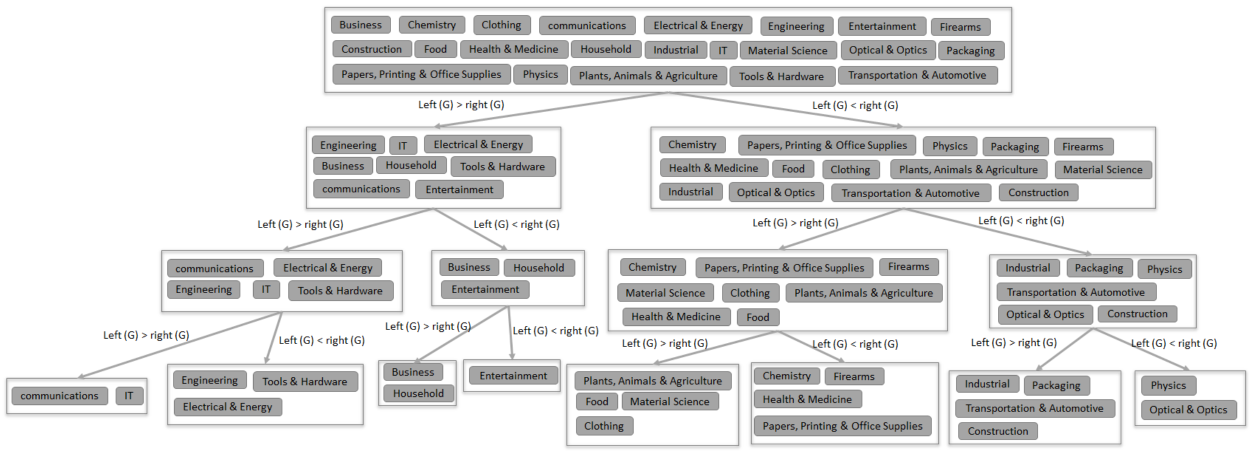


Figure 5.4 A snippet of the class hierarchy structure (up to depth 4) generated by the proposed method for the 22 classes of the U.S. Patent Data

well. For example, chemistry related classes (chemistry, firearms, health& medicine, papers, printing & office supplies) were assigned to the same branch. Although there were some uncertainty in the assignment (e.g., food with material science and clothing), in general, the classes located near the bottom levels of the tree are quite related to each other.

Testing Efficiency Comparison. To compare the test efficiency, we compared our method against decision tree classifier on the U.S. patent data and plotted their results in Figure 5.5. The horizontal axis in the figure corresponds the test time while the vertical axis corresponds to the macro-average or micro-average F1 score. We applied each method to the different variants of the U.S. Patent data (with $k = 22, 426$, and 3139). As we can seen from the figure, our method consistently achieves higher average F1 scores but is slower in terms of test efficiency even though the size of the tree produced by the HTL method is shorter than the size of a decision tree (see Table 5.4 for a comparison of the maximum depth of the trees). We believe this is because our method needs to perform more complex matrix computations at the internal nodes of the tree compared to a decision tree classifier, which uses simple attribute test conditions to determine which branch to follow.

Table 5.4 Tree Depth Comparison on U.S. Patent Data

	k=22	k=426	k=3139
Decision Tree	56	212	635
HTL	5	10	13

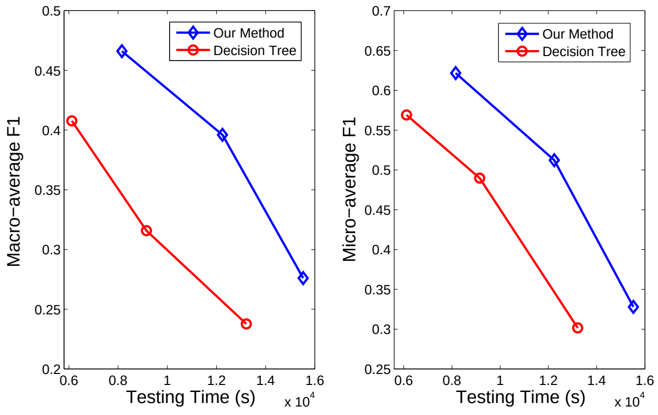


Figure 5.5 Testing Efficiency Comparison on U.S. Patent Data

5.2.3.2 Results on Wikipedia Data 1

In this section, we tested our algorithm on Wikipedia data 1, which contains 98,068 articles belonging to 2,924 subclasses of biology. Results in terms of both micro-average F1 and macro-average F1 are shown in Table 5.5.

Table 5.5 Performance Comparison on Wikipedia Data 1

	Label Propagation (Link only)	Label Propagation (Link + Content)	Decision Tree	HTL
Micro-average F1	0.413	0.425	0.416	**0.427**
Macro-average F1	0.351	0.365	0.349	**0.371**

The results suggest that the proposed method is slightly better than all 3 baselines in terms of both evaluation criteria. Since its macro-scale F1 score is slightly higher than that for hybrid label propagation, this agrees with our previous observation on the U.S. Patent data that the proposed method is better in terms of dealing with rare classes. This conclusion is supported when analyzing the F1 score of the individual classes. As shown in Figure 5.6,

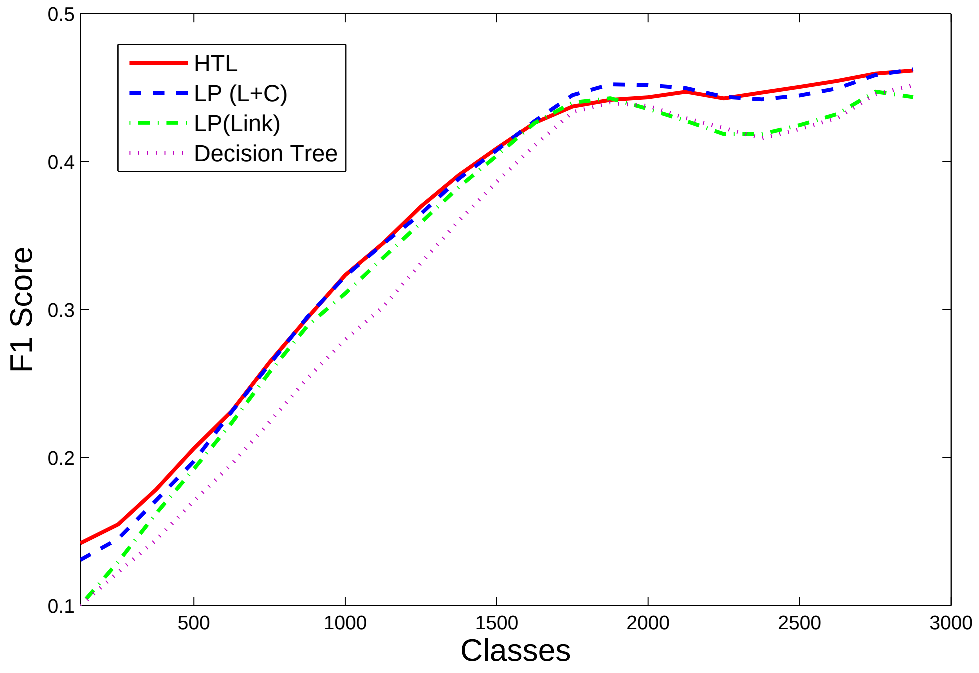


Figure 5.6 F1 Score for each individual class in Wikipedia Data 1 (where the classes are sorted in increasing class size).

the propose HTL method has highest F1 score when the class size is small. However, it is slightly worse than the label propagation methods for some of the larger classes.

Testing Efficiency Comparison. The test time for decision tree classifier on Wikipedia Data 1 is 427.9 seconds while the proposed HTL method takes 516.8 seconds. Although our method is slightly slower, it consistently outperforms the decision tree classifier in terms of both its micro-average and macro-average F1 scores. In addition, unlike decision trees, our method has the ability to generate meaningful taxonomy structure to reveal the class relationships. With the given 2,924 classes in Wikipedia data 1, the decision tree classifier generates a tree with maximum depth of 516, while the proposed HTL method generates a tree with maximum depth of 13.

5.2.3.3 Parallel Results on Wikipedia Data 2

To evaluate the performance of our parallel HTL method, we use the Wikipedia Data 2, which contains more than 600K classes and 2 million articles. The results are shown in Table 5.6. Since none of the competing methods are applicable to such a large size data set, we reported only the performance of our method in this section. Unlike Wikipedia data 1, the number of classes in this data set is 200 times larger, which explains its lower F1 scores. It is also computationally more expensive, requiring 27,153 seconds for testing.

Table 5.6 Results on Wikipedia Data 2 using the parallel HTL method.

Micro-average F1	0.3436
Macro-average F1	0.1738

5.3 Conclusion

In this chapter, we proposed an extension framework of our MF-Tree for large multi-class network classification. We demonstrate that our proposed method can systematically combine both the node attributes and link structure to effectively learn a hierarchical structure from large multi-class network learning problems. Experimental results confirmed the effectiveness of our method.

Chapter 6

Co-classification of Heterogeneous Networks

Most existing network learning techniques have focused on the analysis of a single network. One drawback of this type of methods is that information from a single network may not be sufficient to effectively learn an accurate classification model, especially when the network has noisy attributes or incomplete link structure. In this chapter, we introduce a framework that can jointly classify heterogeneous networks with different types of attributes and links but share similar characteristics in terms of their class information. The proposed framework is known as *co-classification*.

As an example of a co-classification task, consider the network of editors and articles in Wikipedia. Wikipedia has become the largest online knowledge repository with approximately 15 million articles written in more than 270 languages, among which more than 3 million of them are in English. These articles are written collaboratively by voluntary users around the world. Given its massive amount of information and the fact that most of its content can be edited freely by any users who have access to the Internet, it is not surprising Wikipedia has become a rich domain for a variety of interesting classification problems such as topic detection of Wikipedia articles, categorization of Wikipedia editors as humans or bots, and classification of Wikipedia content as featured articles, controversial/inflammatory, spam, or vandalism. These tasks can be cast into a network classification problem, in which the goal is to assign a class label to each node in the network (e.g., an article or a user) based on the node features and link information [76][77][78][40].

Most of the previous works have focused on classifying nodes in a single network. Since the Wikipedia articles and users are inextricably related, knowing the class label of an article often aids in the classification of users, and vice-versa. For example, in topic detection of Wikipedia articles, the category of an article provides evidence for the interest area of a user who edited the article's content. Similarly, in spam detection, we expect spam content to be more likely contributed by spammers than non-spammers. Assigning class labels to both articles and users in Wikipedia not only increases the amount of labeled data available, it is often a natural process during the creation of the training data itself (e.g., to label a user as a spammer or a vandal, one has to know whether the edited content should be considered a form of spam or vandalism). Despite the obvious relationship between their classes, none of the previous works on Wikipedia classification [79][80][81] were designed to co-classify the article and user networks.

A co-classification method was previously developed in [13] to detect spam and spammers in social bookmarking Web sites. The method is based on extending the least-square support vector machine (LS-SVM) formulation to network data, taking into account the correspondences between the classes in different networks. Nevertheless, the method assumes a binary class problem, where the nodes in each network are assigned to one of two possible classes (spam/non-spam or spammer/non-spammer). However, there are many Wikipedia classification problems in which the nodes in the user and article networks may have more than two classes. Though it may be possible to transform the multi-class problem into multiple binary classification problems (using the 1-vs-1 or 1-vs-rest strategies), this approach may not be effective because each binary classifier is trained independently, disregarding the effectiveness of other classifiers. It is also inapplicable when the number of classes in article and user networks are different.

In this chapter, we present an extension of the LS-SVM formulation for network data to a multi-class problem. Specifically, we formalize the joint classification tasks as a constrained optimization problem, in which the relationships between the classes of nodes in two different networks are modeled as graph regularization constraints. Unlike the previous binary class formulation in [13], the formulation proposed here allows us to incorporate prior knowledge about the potential relationships between classes in different networks to avoid overfitting.

6.1 Preliminaries

We begin with a brief discussion of the terminology and notations used. The terminology is given in the context of user and article networks in Wikipedia. **User:** A registered user is identified by a username and has an associated user page on Wikipedia. Unregistered users are also allowed to edit the Wikipedia pages. These users are identified by their corresponding IP address. In this study, we consider only registered Wikipedia users. Let U be the set of such users. **Article:** Articles in Wikipedia have a title and are organized by categories. Some articles are considered Wikipedia stubs, which has been created as a placeholder for a particular topic but does not contain any useful text information. In this study, we consider only non-stub Wikipedia articles. Let A be the set of such articles. **User-Article matrix:** The edits made by users on Wikipedia articles are represented in a user-article matrix M, where $M(i,j)$ is the total number of edits made by user i on article j. **Category relation matrix:** Let E_t be a category relation matrix estimated from the training data. The matrix represents the number of links between each user category and article category. For instance, $E_t(y^{(u)}, y^{(a)})$ is the total number of edits made by users from category $y^{(u)}$ on articles from category $y^{(a)}$, where $y^{(u)} \in \{1, 2, ...k_u\}$ and $y^{(a)} \in \{1, 2, ..., k_a\}$. k_u and k_a are the number of user and article categories, respectively.

Our proposed framework assumes that there is a correspondence between user and article categories (classes). For example, suppose we are interested in classifying the topic areas of Wikipedia users and articles. We expect users who are classified as "computer science" editors to edit more articles classified as "computer science" than other areas such as "biology". As will be shown in Section 6.2, this assumption can be enforced using a graph regularization constraint in our proposed co-classification framework.

Before describing our methodology, we first formalize the co-classification problem. Suppose we are given:

1. A set of l_u labeled users, $\mathcal{L}^{(u)} = \{(x_1^{(u)}, y_1^{(u)}), (x_2^{(u)}, y_2^{(u)}), \cdots, (x_{l_u}^{(u)}, y_{l_u}^{(u)})\}$ where $x_i^{(u)}$is a d_u-dimensional feature vector for the i-th user and $y_i^{(u)} \in \{1, 2...k_u\}$ is its class label.

2. A set of $n_u - l_u$ unlabeled users, $\mathcal{U}^{(u)} = \{x_{l_u+1}^{(u)}, x_{l_u+2}^{(u)}, \cdots, x_{n_u}^{(u)}\}$

3. A set of l_a labeled articles, $\mathcal{L}^{(a)} = \{(x_1^{(a)}, y_1^{(a)}), (x_2^{(a)}, y_2^{(a)}), ...(x_{l_a}^{(a)}, y_{l_a}^{(a)})\}$ where $x_j^{(a)}$is a d_a-dimensional feature vector for the j-th article and $y_j^{(a)} \in \{1, 2, \cdots, k_a\}$ is its class label.

4. A set of $n_a - l_a$ unlabeled articles, $\mathcal{U}^{(a)} = \{x_{l_a+1}^{(a)}, x_{l_a+2}^{(a)}, \cdots, x_{n_a}^{(a)}\}$

5. A set of user-article pairs, $\mathcal{M}$, where $(x_i^{(u)}, x_j^{(a)}) \in \mathcal{M}$ if user $x_i^{(u)}$ edits article $x_j^{(a)}$.

Definition 5 *(Co-Classification). Given $\mathcal{L}^{(u)}, \mathcal{L}^{(a)}$, and $\mathcal{M}$, the goal of co-classification is to learn a pair of classifiers: (1) $f_u : \Re^{du} \rightarrow \{1, 2, \cdots, k_u\}$ that accurately maps the feature* *vector of a user $x^{(u)}$ to its class label $y^{(u)}$ and (2) $f_a : \Re^{d_a} \rightarrow \{1, 2, \cdots, k_a\}$ that accurately maps the feature vector of an article $x^{(a)}$ to its class label $y^{(a)}$.*

6.2 Methodology

This section presents the detailed of our proposed co-classification framework. We first present a maximum margin classification approach to classify the Wikipedia users and articles independently. We then describe the proposed co-classification approach.

6.2.1 Independent Classification of Wikipedia Users and Articles

The framework developed in this study is an extension of the LS-SVM formulation for network data developed in [13]. Before describing the framework, we first consider the task of classifying Wikipedia users and articles independently. Given a set of labeled users $\mathcal{L}^{(u)}$, we may construct a LS-SVM classifier for users by minimizing the following objective function:

$$L_u(w, b, e, \alpha) = \frac{1}{2}\sum_{m=1}^{k_u}(w_m^{(u)T} w_m^{(u)}) + \gamma_1 \frac{1}{2}\sum_{i=1}^{l_u}\sum_{m \neq y_i} e_{im}^{(u)2}$$

$$-\sum_{i=1}^{l_u}\sum_{m \neq y_i} \alpha_{im}(w_{y_i}^{(u)} x_i^{(u)} + b_{y_i}^{(u)} - w_m x_i^{(u)} - b_m^{(u)} - 2 + e_{im}^{(u)}) \qquad (6.1)$$

where $\{\alpha_{im}\}$ is the set of Lagrange multipliers. Similarly, the corresponding Lagrangian formulation for classifying articles can be expressed as follow:

$$L_a(w, b, \varepsilon, \beta) = \frac{1}{2}\sum_{n=1}^{k_a}(w_n^{(a)T} w_n^{(a)}) + \gamma_2 \frac{1}{2}\sum_{j=1}^{l_a}\sum_{n \neq y_j} \varepsilon_{jn}^{(a)2}$$

$$-\sum_{j=1}^{l_a}\sum_{n\neq y_j}\beta_{jn}(w_{y_j}^{(a)}x_j^{(a)}+b_{y_j}^{(a)}-w_n^{(a)}x_j^{(a)}-b_n^{(a)}-2+\varepsilon_{jn}^{(a)}) \tag{6.2}$$

where $\{w_n^{(a)}\}$ is the set of parameters for the article classifier, γ_2 is a parameter that controls the penalty of misclassifying articles, and $\varepsilon_{jn}^{(a)}$ is the error of classifying the j-th article.

6.2.2 Co-Classification of Users and Articles

Instead of solving the optimization problems for classifying users and articles independently, we propose a co-classification algorithm that utilizes the link structure between users and articles from the two relation matrices E_t and M described in Section 6.1. The category relation matrix E_t is computed as follow:

$$E_t(y^{(u)},y^{(a)})=\sum_{\substack{i:y_i^{(u)}=y^{(u)}\\ j:y_j^{(a)}=y^{(a)}}} M(x_i^{(u)},x_j^{(a)})$$

where $x_i^{(u)}\in\mathcal{L}^{(u)}$, $y^{(u)}\in\{1,2,...k_u\}$, $x_j^{(a)}\in\mathcal{L}^{(a)}$, and $y^{(a)}\in\{1,2,...k_a\}$.

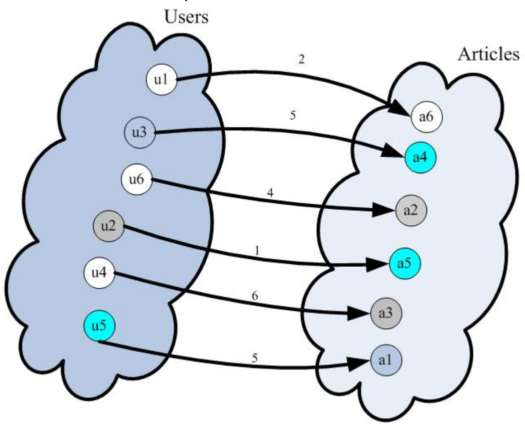


Figure 6.1 A Sample of User-Article network

Figure 6.1 shows an example of the links between Wikipedia users and articles. Suppose both the users and articles are divided into four categories, represented by nodes with the following colors: white, blue, gray, and cyan. The edge between the user and article corresponds to the number of edits, which is captured by the user-article matrix M. For example, $M(u_6, a_2) = 4$ means user u_6 have edited the article a_2 four times. Each entry in the category relation matrix $E_t(y^{(u)}, y^{(a)})$ corresponds to the sum of the weights of edges between users from category $y^{(u)}$ and articles from category $y^{(a)}$. For example, E_t(white, gray) $= M(u_6, a_2) + M(u_4, a_3) = 10$.

One limitation of using E_t is that if the number of labeled users connected to the labeled articles is small, then the relationship between the user and article classes may not be accurately represented by E_t. Furthermore, the presence of noisy links in Wikipedia data also have an adverse affect that may degrade classification performance. In many cases, we may have some prior knowledge about the probability of a link between nodes in two classes in the user and article networks. This information can be encoded using a prior matrix E_p to avoid overfitting the model to the observed relationships between classes in the different networks. Based on these two considerations, we construct an adjusted category relation matrix as follows:

$$E(y^{(u)}, y^{(a)}) = \alpha E_t(y^{(u)}, y^{(a)}) + (1 - \alpha) E_p(y^{(u)}, y^{(a)}), \tag{6.3}$$

where $0 \leq \alpha \leq 1$ is a tuning parameter that controls the tradeoff between E_t and E_p. Its value can be determined via cross-validation or set to a constant, say 0.5, for equal weighting. We use the following prior matrix in our experiments: where the rows and columns are the user and article categories, respectively.

The values in E_t should be normalized for two reasons. First, the values in E_t and E_p do not have the same scale. The larger values in E_t tend to dominate the overall value of E, thus

Table 6.1 Prior category relation Matrix $E_p(k_u, k_a)$

	C_{a1}	C_{a2}	C_{a3}	C_{a4}
C_{u1}	1	-1	-1	-1
C_{u2}	-1	1	-1	-1
C_{u3}	-1	-1	1	-1
C_{u4}	-1	-1	-1	1

losing information about our prior knowledge. Second, it is useful to normalize the values in E_t so that it ranges between [-1,1]. As will be seen shortly, this helps to enforce a penalty or reward scheme for our regularization constraint. Based on these two considerations, we normalize E_t as follows:

$$E_t''(y_i^{(u)}, y_j^{(a)}) = E_t(y_i^{(u)}, y_j^{(a)}) / \sum_{k_a} E_t(y_i^{(u)}, y_j^{(a)})$$

$$E_t'(y_i^{(u)}, y_j^{(a)}) = E_t''(y_i^{(u)}, y_j^{(a)}) - \frac{\sum_{k_a} E_t''(y_i^{(u)}, y_j^{(a)})}{k_a}$$

After normalization, if $y_i^{(u)}$ and $y_j^{(a)}$ are related categories, then $E_t'(y_i^{(u)}, y_j^{(a)})$ is positive-valued; otherwise, $E_t'(y_i^{(u)}, y_j^{(a)})$ is negative-valued. The normalized value for E_t' will be used to estimate E in (6.3).

The category relation matrix will be used to enforce a constraint between the user and article classifiers. Specifically, the assumption described in Section 6.1 states that Wikipedia users will not likely edit articles in their non-related categories. The following graph regularization constraint is therefore introduced to penalize models in which users edit articles that belong to non-related categories:

$$-\gamma_3 \sum_{i=1}^{l_u} \sum_{j=1}^{l_a} \sum_{M(x_i^{(u)}, x_j^{(a)}) \neq 0} E'(y_i^{(u)}, y_j^{(a)})(w_{y_i^{(u)}} x_i^{(u)} + b_{y_i^{(u)}} - w_{y_j^{(a)}} x_j^{(a)} - b_{y_j^{(a)}})$$

The overall objective function for our co-classification framework of users and articles can be written as follows:

$$\begin{aligned}
L &= \frac{1}{2}\sum_{m=1}^{k_u}(w_m^{(u)T}w_m^{(u)}) + \gamma_1\frac{1}{2}\sum_{i=1}^{l_u}\sum_{m\neq y_i} e_{im}^{(u)2} \\
&+ \frac{1}{2}\sum_{n=1}^{k_a}(w_n^{(a)T}w_n^{(a)}) + \gamma_2\frac{1}{2}\sum_{j=1}^{l_a}\sum_{n\neq y_j} \xi_{jn}^{(a)2} \\
&- \sum_{i=1}^{l_u}\sum_{m\neq y_i}\alpha_{im}(w_{y_i}^{(u)}x_i^{(u)} + b_{y_i}^{(u)} - w_m^{(u)}x_i^{(u)} - b_m^{(u)} - 2 + e_{im}^{(u)}) \\
&- \sum_{j=1}^{l_a}\sum_{n\neq y_j}\beta_{jn}(w_{y_j}^{(a)}x_j^{(a)} + b_{y_j}^{(a)} - w_n^{(a)}x_j^{(a)} - b_n^{(a)} - 2 + \varepsilon_{jn}^{(a)}) \\
&- \gamma_3\sum_{i,j}\sum_{M(x_i^{(u)},x_j^{(a)})\neq 0} E^{'}(y_i^{(u)},y_j^{(a)})(w_{y_i}^{(u)}x_i^{(u)} + b_{y_i}^{(u)} \\
&- w_{y_j}^{(a)}x_j^{(a)} - b_{y_j}^{(a)})
\end{aligned} \tag{6.4}$$

where $(w^{(u)}, w^{(a)}, e, \xi, \alpha, \beta)$ are the model parameters to be estimated from training data and $(\gamma_1, \gamma_2, \gamma_3)$ are user-specified parameters. For our experiments, we set $\gamma_1 = \gamma_2 = 1$ whereas γ_3 is estimated from the data via cross validation.

The objective function is solved by taking the derivative of L with respect to each of the model parameters and setting them to zero.

This leads to the following set of linear equations, and could be expressed in matrix notation as follows:

$$\begin{bmatrix} \mathbf{C^{(u)}} & \mathbf{0} \\ \mathbf{0} & \mathbf{C^{(a)}} \end{bmatrix} \begin{bmatrix} \chi^{(\mathbf{u})} \\ \chi^{(\mathbf{a})} \end{bmatrix} = \begin{bmatrix} \mathbf{b^{(u)}} \\ \mathbf{b^{(a)}} \end{bmatrix} \tag{6.5}$$

where $C^{(u)}$ is a matrix that contains $x_i^{(u)}$ and $E^{'}(m, y_j^{(a)})$, the vector $\chi^{(u)} = (w_1^{(u)}, ..., w_{k_u}^{(u)}, b_1^{(u)}, ..., b_{k_u}^{(u)}, e, \alpha)^T$, the vector $b^{(u)} = (p_1, q_1, 0_{l_u \times k_u}, 2_{l_u \times k_u})^T$, $0_{l_u \times k_u}$ is a $l_u \times k_u$-dimensional vector of 0s, $2_{l_u \times k_u}$ is a $l_u \times k_u$-dimensional vector of 2s. Analogously, $C^{(a)}$, $\chi^{(a)}$, and $b^{(a)} = (p_2, q_2, 0_{l_a \times k_a}, 2_{l_a \times k_a})^T$ are the corresponding matrix and vectors for solving the parameters of the article classifier, where:

$$\mathbf{p_1(m)} = \gamma_3 \sum_{i,j} \sum_{\substack{y_i^{(u)}=m \\ M(x_i^{(u)}, x_j^{(a)}) \neq 0}} E^{'}(m, y_j^{(a)}) \mathbf{x_i^{(u)}}$$

$$q_1(m) = \gamma_3 \sum_{i,j} \sum_{\substack{y_i^{(u)}=m \\ M(x_i^{(u)}, x_j^{(a)}) \neq 0}} E^{'}(m, y_j^{(a)})$$

$$\mathbf{p_2(n)} = \gamma_3 \sum_{i,j} \sum_{\substack{y_j^{(a)}=n \\ M(x_i^{(u)}, x_j^{(a)}) \neq 0}} E^{'}(y_i^{(u)}, n) \mathbf{x_j^{(a)}}$$

$$q_2(n) = \gamma_3 \sum_{i,j} \sum_{\substack{y_j^{(a)}=n \\ M(x_i^{(u)}, x_j^{(a)}) \neq 0}} E^{'}(y_i^{(u)}, n)$$

The block structure of the matrix equation in Equation (6.5) suggests that the system of linear equations can be decoupled into two subproblems, one for learning the parameters of the user classifier and the other for article classifier.

$$\mathbf{C^{(u)}} \chi^{(\mathbf{u})} = \mathbf{b^{(u)}} \tag{6.6}$$

Algorithm 7 Co-Classification Algorithm

```
Input: $\mathcal{U}^{(a)}$, $\mathcal{U}^{(u)}$, $\mathcal{L}^{(a)}$,$\mathcal{L}^{(u)}$, $M$, $E^{'}$ and $\gamma$
Output: $(y_i^{(u)}|i = l_u + 1, ...n_u)$, $(y_j^{(a)}|j = l_a + 1, ...n_a)$
Initialize: 1. $(\alpha, \beta, b^{(u)}, b^{(a)}) \leftarrow$ linearsolver($\mathcal{L}^{(a)}$,$\mathcal{L}^{(u)}$, $M$, $E^{'}$, $\gamma$)
2. $y^{(a)} \leftarrow$ Classify($\mathcal{U}^{(a)}, \alpha, b^{(a)}$)
3. $y^{(u)} \leftarrow$ Classify($\mathcal{U}^{(u)}, \beta, b^{(u)}$)
```

$$\mathbf{C^{(a)}}\chi^{\mathbf{(a)}} = \mathbf{b^{(a)}} \tag{6.7}$$

It is worth noting that the solutions for both equations are not entirely independent because the terms p_1, p_2, q_1 and q_2 depend on the link structure of M and E. It is sufficient to solve Equations (6.6) and (6.7) for $(\alpha, b_m^{(u)}, \beta, b_n^{(a)})$ in order to classify the unlabeled users and articles.

Algorithm 7 summarizes the high-level overview of our co-classification algorithm. The `linearsolver` function solves the set of linear equations to obtain the model parameters. The `classify` function takes the models parameters and unlabeled data as input to generate their predicted class labels as output.

6.3 Experimental Evaluation

This section presents the experiments performed to evaluate the performance of the proposed co-classification algorithm.

6.3.1 Data Set

To evaluate the performance of our algorithm, we downloaded the Wikipedia dump dated Oct-09-2009. After preprocessing, we chose the following four topics from each network as our ground truth classes ($k_u = k_a = 4$): biology, natural science, computer science and political science. The label for each user is assigned based on label for the majority class of articles written by the user. We created a sample that contains 5361 users and 6403 articles for our experiments.

The experimental results reported in this section is obtained using 5-fold cross validation. For each run, we use 4 of the folds for training and the remaining fold as test data. We repeated the experiment ten times, each time using a random division of the data into five folds. Table 6.2 shows the category relation matrix $E_t^{'}$ obtained from the training data.

Table 6.2 Category relation Matrix from training data $E_t^{'}$

	C_{a1}	C_{a2}	C_{a3}	C_{a4}
C_{u1}	0.6658	-0.2225	-0.2274	-0.2159
C_{u2}	-0.2186	0.6983	-0.2385	-0.2412
C_{u3}	-0.2342	-0.2271	0.5430	-0.0816
C_{u4}	-0.2423	-0.2462	-0.2398	0.7283

Our experimental results is summarized in Table 6.3. The table shows a comparison between the performance of our proposed multi-class LS-SVM framework against applying the LS-SVM algorithm independently on the user and article networks. For the latter case, we employ the 1-vs-rest strategy to train the independent LS-SVM models. Note that *Pre* means the average precision for all four categories, *Rec* means the average recall for all four categories, and F_1 is the harmonic mean of precision and recall. Clearly, the F_1 measure for our proposed framework is higher than that for LS-SVM.

A more detailed analysis can be seen in Figures 6.2 and Figure 6.3, where we have shown the F_1 values for each class in the article and user networks. Notice that the improvement in F_1 is observed across all the classes. These results suggest that our multi-class graph regularization framework along with the addition of prior matrix enables our algorithm to

Table 6.3 Four categories average results

	Article			User		
5-fold cross validation	*Pre*	*Rec*	F_1	*Pre*	*Rec*	F_1
LS-SVM	48.8169	49.6420	49.2260	41.5724	40.6410	41.1014
Co-LS-SVM	50.1656	53.5612	51.8078	42.8800	46.2185	44.4867

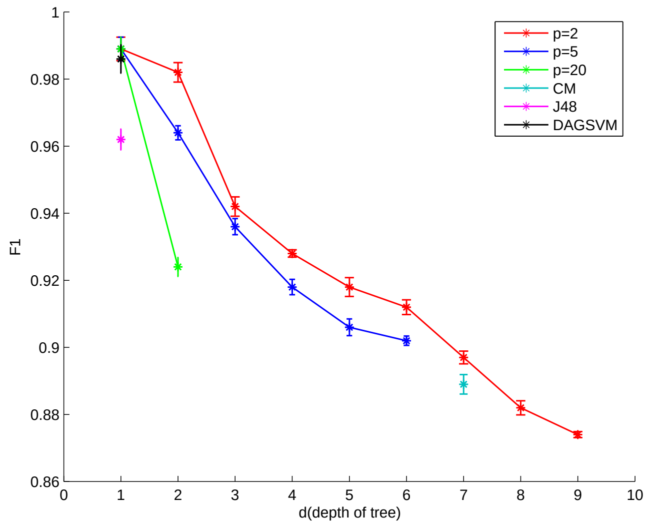


Figure 6.2 Comparison of F_1 measure for each Article category between LS-SVM and Co-LS-SVM algorithms

outperform LS-SVM models that are trained independently.

Despite the observed improvement, our overall F_1 measure is around 50%, which is not very high. This probably is because the presence of noisy links in Wikipedia that adversely affects our classification results. Furthermore, we use the content of the user page in Wikipedia to represent the feature vector of each user. Since some of the user pages contain only a few uninformative words or even empty, their feature vector may not be effectively used for classification. As a next step in our work, we may consider removing the noisy links and exploring additional features to represent the users.

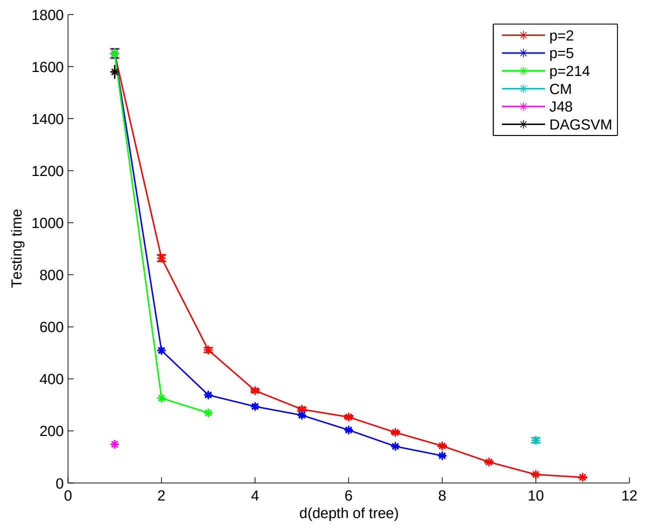


Figure 6.3 Comparison of F_1 measure for each User category between LS-SVM and Co-LS-SVM algorithms

6.4 Conclusion

In this chapter, we proposed a co-classification framework for Wikipedia user and article networks, which utilizes relational information between these two networks and learn the classifier models simultaneously. Experiment results have justified the effectiveness of our method.

Chapter 7

Future Work

In this thesis, I have presented my research to address some of the key challenges of large multi-class network learning. First, a hierarchical learning technique known as recursive non-negative matrix factorization was developed to deal with the test efficiency of large multi-class problems. Another unique advantage of the approach is that it can accommodate the detection of new classes in the test data. Second, I present another approach known as matrix factorization tree (MF-Tree), which was designed to optimize a global objective function. A proof is also given to demonstrate the equivalence between the proposed regularized loss function and the Hilbert Schmidt Information Criterion. We also show the additive property of the objective function, which allows us to decompose the large multi-class learning problem into a series of binary classification tasks that can be organized in a binary tree structure. Third, we extended the MF-Tree approach to deal with network data. The enhanced MF-Tree approach systematically combines both the node attributes and link structure information to perform large multi-class node classification and class hierarchy generation from the network data. Finally, we introduce an approach to jointly classify heterogeneous nodes in multiple networks in order to improve classification accuracy.

For future work, we plan to investigate the following two problems as extensions of this thesis:

7.1 Joint Label Tree Construction and Link Prediction for Network with Large Number of Classes

Since the nodes and link structure are interleaved with each other in a network, node classification and link prediction are two related learning tasks. This is because, in most networks, more links are exist between nodes of the same class than between nodes of different classes. Thus, we conjecture that the learning of one task could aid the learning of the other. The hierarchical network learning method proposed in Chapter 5 builds a hierarchical structure by optimizing an objective function to partition the data into subgroups at each internal node. The partitioning may break links between pair of nodes that should linked together, which may degrade classification performance. Designing an effective hierarchical learning method that can jointly perform node classification and link prediction would be an interesting subject for future research.

7.2 Classification of Documents into Reading Positions

From an application perspective, hierarchical learning methods can potentially be used to provide navigational interface of content, i.e., a way to access content of a document collection. Currently, a popular way to access a large collection of documents is by using a search interface that allows users to submit a query and view a dynamically generated ranked list of relevant documents. Such an approach may work fine when a user is trying to locate a specific document in the collection but is insufficient when users need to access the pertinent documents in some logical order, e.g., for learning, research or editorial purposes.

Search engines hide document relationships while navigational interfaces tend to capture only fixed (static) relationships that do not adapt to a user's specific need. In both cases, to determine which resources to read and in what order, users must manually sift through the documents returned by the system, which is a tedious process. Worse still, an individual may read a significant number of documents in the wrong order, until he/she understood how they relate to each other. Consequently, the user may have to re-read the documents again in the right order to fully grasp their contents.

I have previously developed a novel algorithm to generate reading orders over set of documents (from general to more specific) in [82]. Such type of reading order is extremely useful in several contexts and applications. For instance, it can benefit researchers looking for papers on a particular topic, editors selecting articles to publish on a web site, or IT personnel trying to figure out which documentation they should read first. My previous solution is based on an unsupervised learning method to generate reading orders by analyzing the conceptual relations among the documents. However it does not consider the document link relations such as citations between documents, their common authors, etc. An interesting research question would be to map the reading order generation as a hierarchical network learning task, where each document is a node and the links represent the document link relations. The goal here is to learn a classification or ranking model that accurately assigns a given document to its proper reading position. Note that a ranking problem can be converted into a series of classification problems. For example, to determine the rank among documents d_1, d_2, and d_3, we can transform it into classifying whether d_1 should precede d_2, d_1 should precede d_3 and d_2 should precede d_3.

REFERENCES

REFERENCES

[1] A. Choromanska, A. Agrawal, and J. Langford, “Extreme multi class classification,” in *Neural Information Processing Systems Conference (NIPS) Workshop: Extreme Classification: Multi-Class & Multi-Label Learning with Millions of Categories*, (Lake Tahoe, NV), 2013.

[2] L. Liu, S. Saha, R. Torres, J. Xu, P.-N. Tan, A. Nucci, and M. Mellia, “Detecting malicious clients in isp networks using http connectivity graph and flow information,” in *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*, pp. 150–157, 2014.

[3] P. Zhao and J. Han, “On graph query optimization in large networks,” *Proc. VLDB Endow.*, vol. 3, pp. 340–351, Sept. 2010.

[4] M. Adedoyin-Olowe, M. M. Gaber, and F. Stahl, “A survey of data mining techniques for social media analysis,” *CoRR*, 2013.

[5] J. Scripps, R. Nussbaum, P. Tan, and A. Esfahanian, “Link-based network mining,” in *Structural Analysis of Complex Networks* (M. Dehmer, ed.), Birkhauser, 2010.

[6] P. Mandayam-Comar, P.-N. Tan, and A. Jain, “Identifying cohesive subgroups and their correspondences in multiple related networks,” in *Proc of the 2010 IEEE/WIC/ACM Int’l Conf on Web Intelligence (WI-2010)*, (Toronto, Canada), 2010.

[7] D. Kempe, J. Kleinberg, and E. Tardos, “Maximizing the spread of influence through a social network,” in *Proc of the ACM International Conference on Knowledge Discovery and Data Mining*, 2003.

[8] Q. Lu and L. Getoor, “Link-based classification,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2003.

[9] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, “Collective classification in network data,” *AI Magazine*, vol. 29, no. 3, pp. 93–106, 2008.

[10] G. Pandey, V. Kumar, and M. Steinbach, “Computational approaches for protein function prediction: A survey,” Tech. Rep. 06-028, Department of Computer Science and Engineering, University of Minnesota, Twin Cities, 2006.

[11] M. E. Tipping, "Sparse bayesian learning and the relevance vector machine," *Journal of Machine Learning Research*, vol. 1, pp. 211–244, 2001.

[12] D. Donoho, "Compressed sensing," *IEEE Transactions on Information Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[13] F. Chen, P. Tan, and A. Jain, "A co-classification framework for detecting web spam and spammers in social media web sites," in *Proceeding Of ACM CIKM International Conference On Information And Knowledge Management*, (Hong Kong,China), pp. 1807–1810, 2009.

[14] L. Liu and P.-N. Tan, "A framework for co-classification of articles and users in wikipedia," in *Web Intelligence'10*, pp. 212–215, 2010.

[15] J. C. Platt, N. Cristianini, and J. Shawe-taylor, "Large margin dags for multiclass classification," in *Advances in Neural Information Processing Systems*, pp. 547–553, MIT Press, 2000.

[16] V. Vural and J. G. Dy, "A hierarchical method for multi-class support vector machines," in *Proceedings of the twenty-first international conference on Machine learning*, pp. 105–112, 2004.

[17] A. Beygelzimer, J. Langford, and P. Ravikumar, "Multiclass classification with filter trees," in *Information Theory and Application Workshop*, 2007.

[18] S. Bengio, J. Weston, and D. Grangier, "Label embedding trees for large multi-class tasks.," in *Neural Information Processing Systems*, pp. 163–171, 2010.

[19] J. Deng, S. Satheesh, A. C. Berg, and F. F. F. Li, "Fast and balanced: Efficient label tree learning for large scale object recognition," in *Advances in Neural Information Processing Systems 24* (J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, eds.), pp. 567–575, 2011.

[20] L. Liu, P. M. Comar, S. Saha, P.-N. Tan, and A. Nucci, "Recursive nmf: Efficient label tree learning for large multi-class problems," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR'2012)*, pp. 2148–2151, 2012.

[21] A. Gretton, O. Bousquet, A. Smola, and B. Schölkopf, "Measuring statistical dependence with hilbert-schmidt norms," in *Proceedings of the 16th International Conference on Algorithmic Learning Theory (ALT'2005)*, pp. 63–77, 2005.

[22] M. B. Blaschko and A. Gretton, "Learning taxonomies by dependence maximization," in *Proceedings of the Advances in Neural Information (NIPS'2008)*, pp. 153–160, 2008.

[23] L. Song, A. J. Smola, A. Gretton, and K. M. Borgwardt, "A dependence maximization view of clustering," in *Proceedings of the 24th International Conference on Machine Learning (ICML'2007)*, pp. 815–822, 2007.

[24] M. B. Blaschko, W. Zaremba, and A. Gretton, "Taxonomic prediction with tree-structured covariances," in *Machine Learning and Knowledge Discovery in Databases-European Conference (ECML/PKDD'2013)*, pp. 304–319, 2013.

[25] T. Zhang, A. Popescul, and B. Dom, "Linear prediction models with graph regularization for web-page categorization," in *Proceeding of ACM SIGKDD International Conf on Data Mining*, (Philadelphia,PA), pp. 812–826, 2006.

[26] R. Heatherly, M. Kantarcioglu, and B. Thuraisingham, "Social Network Classification Incorporating Link Type Values," in *IEEE Intelligence and Security Informatics*, (, Dallas, Texas,), June 2009.

[27] J. Scripps, P.-N. Tan, F. Chen, and A.-H. Esfahanian, "A matrix alignment approach for collective classification," in *ASONAM*, pp. 155–159, 2009.

[28] J. Scripps, P.-N. Tan, F. Chen, and A.-H. Esfahanian, "A matrix alignment approach for link prediction," in *ICPR*, pp. 1–4, 2008.

[29] O.-W. Kwon and J.-H. Lee, "Web page classification based on k-nearest neighbor approach," in *Proceedings of the 5th International Workshop on Information Retrieval with Asian Languages (IRAL)*, pp. 9–15, 2000.

[30] J. Fürnkranz, "Exploiting structural information for text classification on the www," in *Proceedings of the Third International Symposium on Advances in Intelligent Data Analysis*, IDA '99, pp. 487–498, 1999.

[31] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious urls," in *KDD*, 2009.

[32] A. Le, A. Markopoulou, and M. Faloutsos, "Phishdef: Url names say it all," *INFOCOM*, 2011.

[33] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *J. ACM*, vol. 46, pp. 604–632, Sept. 1999.

[34] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," technical report, 1999.

[35] V. K. Stanford and V. Krishnan, "Web spam detection with anti-trust rank," 2006.

[36] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Scholkopf, "Learning with local and global consistency," in *Proc. of Advances in Neural Information Processing Systems*, pp. 321–328, 2003.

[37] X. Zhu and Z. Ghahramani, "Learning from labeled and unlabeled data with label propagation," tech. rep., 2002.

[38] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen, "Combating web spam with trustrank," in *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pp. 576–587, 2004.

[39] Y. Yang, S. Slattery, and R. Ghani, "A study of approaches to hypertext categorization," *Journal of Intelligent Information Systems*, vol. 18, pp. 219–241, 2002.

[40] S. Zhu, K. Yu, Y. Chi, and Y. Gong, "Combining content and link for classification using matrix factorization," in *Proceeding Of ACM SIGIR*, (Netherlands), pp. 487–494, 2007.

[41] Q.Lu and L.Getoor, "Link-based classification," in *Proceeding of the twentieth International Conference on Machine Learning(ICML'2003)*, (Washion DC), pp. 496–503, 2003.

[42] Q.Gan and T.Suel, "Improving web spam classifiers using link structure," in *AIRWeb '07: Proceeding of the 3rd international workshop on Adversarial information retrieval on the web*, (New York,USA), pp. 17–20, 2007.

[43] M. Ji, Y. Sun, M. Danilevsky, J. Han, and J. Gao, "Graph regularized transductive classification on heterogeneous information networks," in *Proceedings of ECML/PKDD*, pp. 570–586, 2010.

[44] R. Caruana, "Multitask learning," *Machine Learning*, vol. 1, no. 28, pp. 41–75, 1997.

[45] P. M. Comar, L. Liu, S. Saha, A. Nucci, and P.-N. Tan, "Weighted linear kernel with tree transformed features for malware detection," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM'2012)*, p-p. 2287–2290, 2012.

[46] P. M. Comar, L. Liu, S. Saha, P.-N. Tan, and A. Nucci, "Combining supervised and unsupervised learning for zero-day malware detection.," in *Proceedings of the 33rd IEEE International Conference on Computer Communications (INFOCOM'2013)*, pp. 2022–2030, 2013.

[47] L. Liu, P. M. Comar, A. Nucci, S. Saha, and P.-N. Tan, "Missing or inapplicable: Treatment of incomplete continuous-valued features in supervised learning," in *Proceedings of SIAM International Conference on Data Mining (SDM'2013)*, pp. 46–54, 2013.

[48] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of International Computer Vision and Pattern Recognition (CVPR 2014)*, 2014.

[49] R. Ghani, "Using error-correcting codes for efficient text classification with a large number of categories," masters thesis, Carnegie Mellon University, 2001.

[50] R. Ryan and K. Aldebaro, "In defense of one-vs-all classification," *Journal of Machine Learning Research*, vol. 5, pp. 101–141, December 2004.

[51] T. Hastie and R. Tibshirani, "Classification by pairwise coupling," *The Annals of Statistics*, vol. 26(2), pp. 451–471, 2001.

[52] J. Langford and A. Beygelzimer, "Sensitive error correcting output codes," in *Conference on Learning Theory*, pp. 158–172, 2005.

[53] D. Hsu, S. M. Kakade, J. Langford, and T. Zhang, "Multi-label prediction via compressed sensing," *CoRR*, vol. abs/0902.1284, 2009.

[54] J. Huang, T. Zhang, and D. N. Metaxas, "Learning with structured sparsity," in *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, pp. 417–424, 2009.

[55] A. Beygelzimer, J. Langford, Y. Lifshits, G. Sorkin, and A. Strehl, "Conditional probability tree estimation analysis and algorithms," in *Proceedings of UAI*, pp. 51–58, 2009.

[56] T. Gao and D. Koller, "Discriminative learning of relaxed hierarchy for large-scale visual recognition," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV'2011)*, pp. 2072–2079, 2011.

[57] D. Kuang and H. Park, "Fast rank-2 nonnegative matrix factorization for hierarchical document clustering," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'2013)*, pp. 739–747, 2013.

[58] P.-Y. Hao, J.-H. Chiang, and Y.-H. Lin, “A new maximal margin spherical structured multi-class support vector machine,” in *Applied Intelligence*, 2009.

[59] D.Lee and H.Seung, “Algorithm for non-negative matrix factorization,” in *Neural Information Processing Systems*, pp. 556–562, 2001.

[60] H. Pei-Yi, C. Jung-Hsien, and L. Yen-Hsiu, “A new maximal-margin spherical-structured multi-class support vector machine,” *Applied Intelligence*, vol. 30, pp. 98–111, 2009.

[61] D. Chen and R. J. Plemmons, “Nonnegativity constraints in numerical analysis,” in *Symposium on the Birth of Numerical Analysis* (A. Bultheel and R. Cools, eds.), World Scientific Publishing, 2010.

[62] T. Ngo, M. Bellalij, and Y. Saad, “The trace ratio optimization problem,” *SIAM Review*, vol. 54, no. 3, pp. 545–569, 2012.

[63] J. Wenhao and C. Fu-lai, “A trace ratio maximization approach to multiple kernel-based dimensionality reduction,” *Neural Networks*, vol. 49, pp. 96–106, 2014.

[64] D. Boley, “Principal direction divisive partitioning,” *Data Mining and Knowledge Discovery*, vol. 2, no. 4, pp. 325–344, 1998.

[65] E. Tanaka and K. Tanaka, “The tree-to-tree editing problem,” *International Journal Pattern Recognition and Artificial Intelligence*, vol. 2, no. 2, pp. 221–224, 1988.

[66] T. Jiang, L. Wang, and K. Zhang, “5th annual symposium on combinatorial pattern matching,” in *Advances in Neural Information Processing Systems*, pp. 75–86, 1994.

[67] P. Mandayam Comar, L. Liu, S. Saha, A. Nucci, and P.-N. Tan, “Weighted linear kernel with tree transformed features for malware detection,” in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM ’12, pp. 2287–2290, 2012.

[68] U. Raghavan, R. Albert, and S. Kumara, “Near linear time algorithm to detect community structures in large-scale networks.,” *Phys Rev E Stat Nonlin Soft Matter Phys*, vol. 76, no. 3 Pt 2, p. 036106, 2007.

[69] L. Hagen and A. B. Kahng, “New spectral methods for ratio cut partitioning and clustering,” *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 11, pp. 1074–1085, Nov. 2006.

[70] A. Blum and S. Chawla, "Learning from labeled and unlabeled data using graph mincuts," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, (San Francisco, CA, USA), pp. 19–26, 2001.

[71] U. von Luxburg, "A tutorial on spectral clustering," *CoRR*, vol. abs/0711.0189, 2007.

[72] D. Achlioptas, A. Fiat, A. Karlin, and F. McSherry, "Web search via hub synthesis," in *Proceedings of the 42Nd IEEE Symposium on Foundations of Computer Science*, FOCS '01, 2001.

[73] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ICDM '09, pp. 229–238, 2009.

[74] U. Kang, B. Meeder, E. E. Papalexakis, and C. Faloutsos, "Heigen: Spectral analysis for billion-scale graphs," *IEEE Trans. Knowl. Data Eng.*, pp. 350–362, 2014.

[75] Y. Yang, "An evaluation of statistical approaches to text categorization," *Inf. Retr.*, vol. 1, no. 1-2, pp. 69–90, 1999.

[76] J. Abernethy, O. Chapelle, and C. Castillo, "Web spam identification through content and hyperlinks," in *Proceeding Of The Sigir Workshop On Adverasrial Information Retrieval On Web (Airweb'08)*, (Beijing,China), pp. 41–44, 2008.

[77] G. Attardi, A. Gull, and F. Sebastiani, "Automatic web page categorization by link and context analysis," in *Proceedings of THAI'99, First European Symposium on Telematics, Hypermedia and Artificial Intelligence, Varese, IT*, pp. 205–119, 1999.

[78] D. A. Cohn and T. Hofmann, "The missing link - a probabilistic model of document content and hypertext connectivity," in *NIPS*, pp. 430–436, 2000.

[79] K. Smets, B. Goethals, and B. Verdonk, "Automatic vandalism detection in wikipedia: Towards a machine learning approach," in *Proceedings of the Association for the Advancement of Artificial Intelligence(AAAI)*, pp. 43–48, AAAI Press, 2008.

[80] M. Potthast, B. Stein, and R. Gerling, "Automatic vandalism detection in wikipedia," *Advances in Information Retrieval*, pp. 663–668, 2008.

[81] G. Druck, G. Miklau, and A. McCallum, "Learning to predict the quality of contributions to wikipedia," in *Proceedings of the AAAI Workshop on Wikipedia and Artificial Intelligence (WIKIAI 08)*, pp. 7–12, 2008.

[82] L. Liu, G. Koutrika, and S. Simske, "Generating reading orders over document collections," in *Proceedings of the 31st IEEE International Conference on Data Engineering*, ICDE'15, 2015.