Investigating Label Hierarchies for Multi-Label Classification

A B. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Aneesh Barthakur
(140101009)

under the guidance of

Dr Prithwijit Guha

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Investigating Label Hierarchies for Multi-Label Classification**" is a bona fide work of **Aneesh Barthakur** (**Roll No. 140101009**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr Prithwijit Guha**

Assistant Professor,

November, 2018

Department of Electronics & Electrical Engineering,

Guwahati.

Indian Institute of Technology Guwahati, Assam.

Contents

Abstract

Multi-Label Learning (MLL) is a supervised learning problem that considers the setting where given an object, multiple categories are relevant to it. This is opposed to single label problems, where the target prediction is a single label. Multi-Label Learning deals with a number of challenging sub-problems such as considering the relationships between labels, high computational complexity and a large output space. MLL has a wide range of applications such as document classification, video and audio annotation and recently, product recommendations. In this work, we extend the HOMER algorithm which decomposes the MLL task for a large number of labels, to smaller MLL tasks, by building a hierarchy over the label space. We replace the default partitioning procedure with a graph partitioning approach, where the graph in question is based on the co-occurrences of labels. We choose the Kernighan-Lin algorithm for graph partitioning and evaluate on two benchmark datasets - delicious and mediamill. We also compare our variation, named HOMER-G, with two previously proposed variants HOMER-k and HOMER-R. We find that HOMER-G generally outperforms the other variants with respect to its recall, but lags when it comes to precision and Hamming Loss. We also investigated an ensemble of HOMER-R trees which show promising results.

Chapter 1

Introduction

Multi-Label Learning (MLL) is a supervised learning problem that considers the setting where given an object, multiple categories are relevant to it. This is opposed to single label problems, where the target prediction is a single label. Multi-label Learning deals with a number of challenging sub-problems such as considering the relationships between labels, high computational complexity in terms of models as well as query time, large number of classes or categories, imbalanced classes and high dimensional data . However, along with the challenges also comes a wide range of applications such as document classification, video and audio annotation , predictions of gene and protein functions and recently, product recommendations.

In Chapter 2 we first introduce the problem of Multi-Label Learning formally, and review some of the important metrics associated with it. Thereafter we review a representative subset of the literature that seeks to solve this problem, and also include the results of a comparative study [MKGD12] for the benefit of the reader.

In Chapter 3, we present our proposed method for the problem of multi-label classification, which is an extension of the HOMER algorithm [TKV08]. The HOMER algorithm was one of the first MLL methods specifically designed to tackle large label sets. Further,

the study by [MKGD12] concluded that HOMER is one of the best performers across a range of metrics and MLL methods. HOMER employs a divide and conquer approach and builds a tree-hierarchy over the label space. Thereafter, simpler MLL methods are used for the reduced subproblems at the nodes. We extend the HOMER algorithm by replacing the default partitioning procedure with a graph partitioning approach, where the graph in question is based on the co-occurrences of labels. We hypothesize that this better captures the correlations between labels and these changes will yield partitions with better intra-cluster similarity. We call this method HOMER-G.

Although initially, we tried various graph clustering algorithms for the graph partitioning algorithms, we settled on the Kernighan Lin algorithm [KL70], and conducted three experiments on two benchmark datasets - *mediamill* and *delicious*. We compare HOMER-G with two other variants proposed by the authors of HOMER, based on partitioning by K-Means and random partitioning. We find that HOMER-G generally outperforms the other methods with respect to its recall, but lags when it comes to precision and Hamming Loss. Overall, HOMER-G outperforms the other variants on the delicious dataset with respect to the F1 score. Besides HOMER-G, we also investigate an ensemble of HOMER-R trees, where the partitioning method is random. The ensemble method also shows promising results.

In Chapter 4 we present our conclusion and list a few directions for future research.

Chapter 2

Review of Prior Work

The Multi Label Learning (MLL) problem can be defined as follows [GV]

Given $\mathcal{X} = X_1 \times \cdots \times X_D$, a D-dimensional input space (of numeric or categorical features), and an output space $L = \{\lambda_1, \lambda_2, \cdots, \lambda_Q\}$ of Q labels, a multi label pattern is a pair $(\mathbf{x}, Y)$, where $\mathbf{x} \in \mathcal{X}$ is the *instance vector* and $Y \subseteq L$ is called the *label set*.

Y can be represented *sparsely* as a set $\{\lambda_1, \lambda_2, \cdots, \lambda_k\}$, or *densely* as a binary vector $\mathrm{y} \in \{0,1\}^Q$.

Within this framework, we can define three related tasks included in MLL [GV]

1. **Label Ranking** (LR) : The LR problem consists of learning a real valued function $f : \mathcal{X} \times L \to \mathbb{R}$, which induces an ordering in L corresponding to the relevance of each label for an instance $\mathbf{x}$. In other words, LR consists of learning a function f that satisfies the following property for $\forall \mathbf{x} \in \mathcal{X}, \lambda_i, \lambda_j \in L$

$$f(\mathbf{x}, \lambda_i) < f(\mathbf{x}, \lambda_j) \qquad \textit{If } \lambda_i \text{ is more relevant to } \mathbf{x} \text{ than } \lambda_j$$

2. **Multi-Label Classification**(MLC) : Given an instance $\mathbf{x}$, the task of an MLC algorithm is to predict a label set $Y \subseteq L$ whose elements are relevant to $\mathbf{x}$. Alternatively, an MLC algorithm produces a bipartition $(Y, \overline{Y})$ such that Y is relevant to $\mathbf{x}$. *Multi-*

class classification(MCC) and *binary classification*(BC) are special cases of MLC, where the label set Y is of size 1. An MLC classifier can be derived from an LR model using thresholding techniques.

3. **Multi-Label Ranking**(MLR) : The Multi-Label Ranking problem is a generalization of LR and MLC, where the task is both to provide a ranking f of the label space, as well as a bipartition $(Y, \overline{Y})$. For a consistent ranking, labels in Y must be ranked higher than all labels in $\overline{Y}$.

A host of approaches exist that tackle the MLL problem. In this chapter, we review some of these methods which, in our opinion, are representative of MLL methods in general. We utilize the taxonomy proposed in [TKV10] for this purpose. The remainder of this chapter is organized as follows. Section 2.1 introduces some of the important metrics for MLL. Section 2.2 reviews the current literature surrounding Multi Label Learning. Section 2.3 provides a brief summary of an extensive comparative study of MLL methods conducted by [MKGD12]. Finally, Section 2.4 summarizes the topics covered in this chapter.

2.1 Metrics

Before introducing multi-label metrics, let us first briefly review some metrics for binary classification

- The *precision* ($\frac{\text{tp}}{\text{tp+fp}}$) of an algorithm is the fraction of positive predictions which it made correctly.
- The *recall* ($\frac{\text{tp}}{\text{tp+fn}}$) of an algorithm is the fraction of positive examples which have been correctly predicted.
- The *F1 Score* ($\frac{2\times\text{precision}\times\text{recall})}{\text{precision+recall}}$), is the *harmonic mean* of the precision and recall.

[TKV10] classified MLL metrics into two types, *label-based* metrics and *example-based* metrics. Label-based metrics first compute the true positives (tp), true negatives (tn), false

positives (tn) and false negatives (fn) for *each label*. Thereafter, given any *binary* metric B, the final calculation of the corresponding ML metric can follow two approaches

1. *macro-approach*: In this approach, the metric B is first calculated for each label $\lambda \in L$, and then averaged across all labels. That is,

$$B_{\text{macro}} = \frac{1}{Q} \sum_{i=1}^{Q} B(\text{tp}_i, \text{fp}_i, \text{tn}_i, \text{fn}_i) \tag{2.1}$$

2. *micro-approach*: In this approach, tp, tn, fp, fn are first aggregated across all labels, and the aggregated measures are used to calculate B_{micro}. That is,

$$B_{\text{micro}} = B(\frac{1}{Q} \sum_{i=1}^{Q} \text{tp}_i, \frac{1}{Q} \sum_{i=1}^{Q} \text{fp}_i, \frac{1}{Q} \sum_{i=1}^{Q} \text{tn}_i, \frac{1}{Q} \sum_{i=1}^{Q} \text{fn}_i) \tag{2.2}$$

Thus, we can have micro and macro version of precision and recall. The micro and macro versions of the F1 score are calculated from the micro and macro versions of precision and recall.

On the other hand, example-based metrics are first calculated for *each instance*, and then averaged across all instances. Let T be the number of test instances, and Z, Y denote the set of predicted and true labels for a sample . Then, some example based metrics are

1. $\text{precision}_E = \frac{1}{T} \sum_{i=1}^{T} \frac{|Z_i \cap Y_i|}{|Z_i|}$, similar to before.

2. $\text{recall}_E = \frac{1}{T} \sum_{i=1}^{T} \frac{|Z_i \cap Y_i|}{|Y_i|}$, similar to before.

3. Hamming Loss $= \frac{1}{T} \sum_{i=1}^{T} \frac{1}{Q} |Z_i \Delta Y_i|$, where Δ is the symmetric difference between sets. Thus, Hamming loss is nothing but the fraction of labels *incorrectly predicted*.

4. F1 score $= \frac{1}{T} \sum_{i=1}^{T} \frac{2|Z_i \cap Y_i|}{|Z_i| + |Y_i|}$, similar to before.

2.2 MLL Methods

In this section, we review some of the methods used to solve the MLL problem. We organize the methods using the taxonomy presented in [TKV10] and also used in [GV]. This taxonomy mainly clubs MLL methods into two types, *problem transformation methods* and *algorithm adaptation methods*. Besides this, [GV] considers a third type of *ensemble methods*, which utilize ensembles of MLL methods. These are dealt with in Section 2.2.1, Section 2.2.2 and Section 2.2.3 respectively.

This review is by no means comprehensive - the literature on MLL methods is vast. Instead it covers a few representative methods to give the reader a sense of the ideas prevalent in the literature. For a more complete (and excellent) treatment of MLL methods, see [GV] and references therein.

2.2.1 Problem Transformation Methods

Problem transformation methods are those which transform the problem of multilabel classification into single-label classification tasks such as binary classification. Thereafter any method for single-label classification can be used for the actual learning. Problem transformation methods are independent of the algorithm used for prediction.

Binary Methods

Binary Relevance [TKV10] takes an OVA (*one vs all*) approach and decomposes the MLC task with Q labels to Q binary classification tasks corresponding to each label. The training dataset for the binary classifier for label λ_k is formed by considering all training examples annotated with λ_k as positive, and all others as negative. During prediction, each classifier predicts a score for the corresponding label, which can be thresholded to output the set of predicted labels. Binary Relevance is simple to implement and its computational complexity at prediction is $O(Q)$. It can also be parallelized in training and testing phases[RPHF11]. However, it suffers from drawbacks such as

1. It implicitly assumes label independence by ignoring the relationship between labels. However, labels do have strong correlations which need to be exploited. Moreover, by ignoring the relationships between labels, it may fail in predicting label combinations and label ranking [TDS+09].

2. If the label space L is large, the binary classifiers may suffer from *class imbalance* due to a very large negative class [ACMM12].

The **Classifier Chains** model also creates Q binary classifiers for the multi-label classification task. However, the classifiers are linked in a randomly ordered chain, such that the each classifier utilizes *the labels predicted by its predecessors* in addition to the instance $\mathbf{x}$ for prediction. Thus, CC considers the relationships between labels, albeit in a random manner. The complexity of CC is $O(Q)$.

As the ordering of the chain itself can influence performance, [RPHF11] also proposed an *Ensemble of CC classifiers* (ECC), consisting of a set of randomly ordered Classifier Chains, trained on randomly drawn (with replacement) subsets of the training set.

Finally, the **BRPlus**(BR+) [ACMM12] method seeks to incorporate label dependencies by augmenting the feature space of the i^{th} binary classifier in a BR model with a $(Q-1)$ dimensional binary vector indicating the presence of the remaining labels in $L \setminus \lambda_i$. During prediction, unlabeled instances were first annotated using the prediction of an additional helper BR model.

Pairwise Methods

Ranking by Pairwise Comparision (RPC) [HFCB08] follows an OVO (*one vs one*) approach and trains $Q(Q-1)/2$ classifiers, one for each pair of labels. The training set $T_{i,j}$ for the classifier $\mathcal{C}_{i,j}$ corresponding to (λ_i, λ_j) is constructed as follows. Given an instance $\mathbf{x}$ in the original training set

- If λ_i is preferred to λ_j, $\mathbf{x}$ is added to $T_{i,j}$ with target label set to 1. If λ_j is preferred

to λ_i, the target label is set to 0.

- If nothing is known about the preference between λ_i and λ_j, $\mathbf{x}$ is not added to $T_{i,j}$.

If the label preferences are not known and only the target labelset $Y \subseteq L$ is known, it is trivial to construct a preference where labels present in Y are preferred over labels which are absent, and there is no preference among present labels and absent labels.

At prediction time, results of each classifier are combined by counting votes from each classifier. That is, the rank of a label λ_i is equal to the number of classifiers that predicted λ_i.

Calibrated Label Ranking(CLR) [FHLB08] [PF07] produces a label ranking as well as a partition by the introduction of a *calibration label* λ_0. Thereafter, an RPC classifier is learnt over the augmented label space $L_c = L \cup \{\lambda_0\}$. Label preferences with the calibration label λ_0 for an instance $(\mathbf{x}, Y)$ is defined as follows

- If $\lambda_i \in Y$, then λ_i is preferred to λ_0 for $\mathbf{x}$
- Otherwise, λ_0 is preferred to λ_i.

The motivation behind adding a calibrating label is that it acts as a splitting point between relevant and irrelevant labels, as all relevant labels are preferred over λ_0 and λ_0 is preferred over all irrelevant labels. Thus, during prediction when the CLR (which is simply the augmented RPC) model outputs a label ranking

$$\lambda_{i_1} \succ \cdots \succ \lambda_{i_j} \succ \lambda_0 \succ \lambda_{i_{j+1}} \succ \cdots \lambda_{i_Q} \tag{2.3}$$

From this, both a label ranking (by removing λ_0 from Eqn 2.3) and a bipartition $(Y, \overline{Y}) = (\{\lambda_{i_1}, \cdots, \lambda_{i_j}\}, \{\lambda_{i_{j+1}}, \cdots, \lambda_{i_Q}\})$ can be inferred.

[FHLB08] also prove that the classifiers $\mathcal{C}_{0,j}$, $j = 1, \cdots, Q$ are equivalent to a BR model for a symmetric base learner. Thus, the CLR method can be thought of as an ensemble pooling all of the RPC and BR base classifiers.

Pairwise methods perform better than BR in some domains [GV], however they suffer from $O(Q^2)$ complexity at prediction time as well as high space complexity.

Quick Weighted CLR (QWCLR) [LMPF10] reduces the time complexity by adapting the *Quick Weighted voting* algorithm (for multi-class classification) for MLL. The Quick Weighted (QW) voting algorithm computes the class with the *highest accumulated voting mass* (i.e. the highest number of votes) without evaluating all pair-wise classifiers. The key idea behind Quick Weighted (QW) voting, is that even early on in the evaluation process, it is possible to exclude some of the classes from the *top classes list*, because they have already lost too many votes. The Quick Weighted voting algorithm can be described as follows

1. Order all labels in order of increasing *voting loss*. The voting loss for λ_j is simply the number of classifiers $\mathcal{C}_{i,j}$ evaluated which did not vote in favour of λ_j. That is, it is the number of votes lost by λ_j.

2. The class with the current *minimum voting loss* is the current candidate for the *top class*.

3. If all the classifiers corresponding to the top class *candidate* λ_{cand} have already been evaluated, output λ_{cand} as the top class.

4. Otherwise find the class λ_j with the minimum voting loss value, which is not λ_{cand} and for whom classifier $\mathcal{C}_{\text{cand},j}$ has not yet been evaluated. Evaluate $\mathcal{C}_{\text{cand},j}$ and return to Step 1.

The key idea to understanding the Step 3 is that the voting loss is non decreasing w.r.t. the number of classifiers evaluated. Hence if all the classifiers corresponding to the candidate have been evaluated, its voting loss cannot decrease further. And hence it must have the minimum voting loss value, or in other words, the *maximum votes*. So we can safely output the candidate as the top class.

The QW voting algorithm is applied to MLL by QWCLR [LMPF10] by repeatedly applying it to a CLR model. After each iteration, the top label is removed from the CLR model. Thus the order of labels returned by each iteration is the *label ranking*. Once the *calibrating label* λ_0 is returned, all remaining classes can be considered irrelevant, and a *bipartition* is also obtained.

QWCLR effectively reduces the prediction complexity from $O(Q^2)$ evaluations to $O(Q \log Q)$, making it close to the $O(Q)$ complexity of BR and CC.

Label Combination Methods

The **Label Powerset** (LP)[TKV10] approach considers each element of the powerset of L, $\mathcal{P}(L)$, as a separate class, and thereafter utilizes a multi-class algorithm on the transformed problem. Thus, given a new instance, LP predicts a single class - which actually represents a set of labels $Y \subseteq L$. Although this method quite explicitly considers the correlation between labels, the power set of L grows exponentially leading to high time and space complexity.

RAndom k-labEL sets(RAkEL)[TV07] is an ensemble approach based on random projections of the label space. For each member of the ensemble, RAkEL considers a small random subset of L of size k, and thereafter learns an LP (Label Powerset) classifier for it. [TV07] hypothesizes that by considering an adequate number of small random subsets of L, RAkEL is able to effectively model the correlations between labels. During prediction, the ranking of a label λ_i is calculated by averaging the votes of ensemble members whose label-sets contain λ_i.

[RI10] improves RAkEL by formulating the subset selection problem as an *unweighted Set Cover Problem* (SCP)[1]. The particular formulation can be described as follows

- A set of k-labelsets *guarantees a coverage* of size r, if each of $\binom{|L|}{r}$ subsets of L of size r are covered at least once.

[1]Given a universe $\mathcal{U}$ and a family of its subsets $\mathcal{S}$, a *cover* is a subfamily $\mathcal{C} \subseteq \mathcal{S}$ such that its union is $\mathcal{U}$. The unweighted Set Cover Problem (SCP) is the problem of selecting the smallest cover $\mathcal{C}$.

- Then, [RI10] selects the minimum number of k-labelsets which guarantee a coverage of size r to build the ensemble.

- This can be formulated as an unweighted SCP where

  - $\mathcal{U} = \{s | s \subseteq L, |s| = r\}$, the set of all r-subsets.

  - $\mathcal{S} = \{s | x \in s \Rightarrow |x| = r, |\cup_{x \in s}| = k\}$, i.e. each element in $\mathcal{S}$ is a set of r-subsets of $|L|$, whose union is a k-subset of $|L|$.

The search version of SCP is known to be NP-Hard. [RI10] utilize a greedy approximation algorithm to derive the k-labelsets.

Hierarchy Of Multi-label classifiERs (HOMER) [TKV08] was one of the first MLL methods to tackle the problem of datasets with a *large* label space by building a tree-hierarchy over the label space. At each node, starting with the root, similar labels are clustered into a *meta-label*, which form the child nodes. Each child node filters the data of the parent, by only keeping the instances that are annotated with at least one of the labels present in its meta-label. Single labels form leaves of the tree.

At each internal node, a multi-label classifier is learnt to predict the meta-labels of its children. During prediction, an instance $\mathbf{x}$ is forwarded down the tree till it reaches the leaves. The predicted labels are the labels whose corresponding leaves are reached by $\mathbf{x}$.

Thus, HOMER breaks the MLC problem for a large number of classes, into several MLC problems dealing with a smaller number of classes. By partitioning the labels space based on similarity, fewer subtrees are activated, resulting in logarithmic prediction time. [TKV08] proposes a *balanced k-means* clustering algorithm for partitioning, which performs better compared to vanilla k-means [TKV08]. A more detailed treatment of HOMER is presented in Section 3.1.

Experimental results reported by the authors showed that HOMER outperformed BR, both in terms of prediction time and performance. A more extensive study conducted in 2012 [MKGD12], found HOMER (along with RF-PCT) outperformed most standard MLL

methods.

2.2.2 Algorithm Adaptation Methods

This section reviews the family of algorithm adaptation methods. Algorithm adaptation methods are those which extend single label algorithms to deal with multi-label data. We review two subfamilies of algorithm adaptation methods - tree based methods and *instance based methods*.

Tree Based Methods

Multi-Label-C4.5 (MLC4.5) [CK01] is an adaptation of the popular C4.5 decision tree algorithm for the MLC setting. C4.5 trees recursively split the training set by optimizing the *information gain* of a split. Given a set of training examples S (with categorical features[2]), and a (categorical) attribute A being considered, the information gain is the difference between the entropy of the whole set S and the weighted sum of the entropy of the subsets caused by the split.

$$IG(S, A) = \text{entropy}(S) - \sum_{v \in A} \frac{|S_v|}{|S|} \text{entropy}(S_v) \tag{2.4}$$

The entropy of a set S is defined as

$$\text{entropy}(S) = -\sum_{i=1}^{C} p(c_i) \log p(c_i) \tag{2.5}$$

where $p(c_i)$ is the probability (or relative frequency) of class c_i in S.

[2]In case of numeric features, C4.5 trees consider splits of the type $x_j > t$, where t is a real value. That is, C4.5 splits the data along orthogonal hyperplanes in the feature space. In case of categorical features, C4.5 splits the data according to the value of the attribute.

[CK01] extends C4.5 to MLL, by first redefining the entropy of a set as

$$\text{entropy}(S) = -\sum_{i=1}^{Q} p(\lambda_i)\log p(\lambda_i) + (1 - p(\lambda_i))\log(1 - p(\lambda_i)) \tag{2.6}$$

The rationale behind this can be understood with a brief discussion of information theory

- The entropy of an event is the expected amount of information gained, by observing that event. The information gained from an event E is $I(E) = -\log p(E)$. Thus the expectation of information gained is $-\sum_{\forall E} p(E)\log p(E)$. The information gained from observing two independent events E_1 and E_2 is the sum of the individual information gains. That is, $I(E_1, E_2) = I(E_1) + I(E_2)$.(Clearly, Shannon's solution of a logarithmic information gain follows this property)

- In a multi-class setting, the event in question is the answer to the question *which class does this instance belong to?*. As such, the definition of entropy given in Eqn 2.5 is straight forward.

- In a multi-class setting, the event in question is the answer to the question *is this label relevant to this instance?*. This, has only two outcomes : Yes or No. Thus the entropy of this event for label λ_i is given by $p(\lambda_i)\log p(\lambda_i) + (1 - p(\lambda_i))\log(1 - p(\lambda_i)$.

- Now if we assume that label occurrences are independent events, we must sum this value across all Q labels, arriving at Eqn 2.6.

Further, the calculation of $|S|$ in Eqn 2.4 must count an instance multiple times if it has multiple labels, to remain a weighted sum.

Predictive Clustering Trees(PCT) [BRR00] is a generic framework for prediction. Given a learning problem, PCT requires the definition of a metric to compute "distances" between instances and the definition of a "prototype" of a given set of instances. Thereafter, PCT utilizes the metric to recursively partition the data by clustering, in a way that minimizes the intra-class variation. Leaves are defined using some termination condition.

Thereafter the "prototype" function is calculated for each leaf. During prediction, the instance is propagated down the tree, and the prototype of the leaf reached is returned. In case of multi-label learning, this is the predicted label set (or probabilities of each label). [KVSD13] defines the parameters for PCT for the MLL setting.

Instance Based Methods

Multi-Label K-Nearest Neighbours (MLKNN) was one of the first *lazy*[3] MLL methods. Given an instance $\mathbf{x}$, MLKNN first finds the K-Nearest Neighbours $N_K(\mathbf{x})$ and computes the membership counts $\vec{C}_x$ for each label among $N_K(\mathbf{x})$. Thereafter, the label-set of the test instance $\mathbf{x}$ is determined through the *maximum a posteriori* (MAP) principle

$$\vec{y}(l) = \arg\max_{b \in 0,1} Pr(H_b^l | E^l_{\vec{C}_x(l)}) \quad l \in L \tag{2.7}$$

where,

- H_1^l is the event that label l is relevant to $\mathbf{x}$ and H_0^l is the event that it is not.
- E_j^l is the event that the label l appears *exactly* j times in the KNNs of $\mathbf{x}$.

Using Bayes Rule, Eqn 2.7 can be rewritten as

$$\begin{aligned} \vec{y}(l) &= \arg\max_{b \in 0,1} \frac{Pr(H_b^l) Pr(E^l_{\vec{C}_x(l)} | H_b^l)}{Pr(E^l_{\vec{C}_x(l)})} \\ &= \arg\max_{b \in 0,1} Pr(H_b^l) Pr(E^l_{\vec{C}_x(l)} | H_b^l) \end{aligned} \tag{2.8}$$

To evaluate Eqn 2.7, we need the prior probabilities $Pr(H_b^l)$ and the posterior probabilities $Pr(E_j^l | H_b^l)$. [ZZ07] estimates these probabilities from the data based on frequency counting at the global and local (KNN) level respectively.

[3]Lazy learning is a paradigm in machine learning, where generalization of the training data is delayed till query time, as opposed to *eager* approaches which seek to generalize the model at training time itself.

2.2.3 Ensemble Methods

Random Forests of PCT (RFPCT) [KVSD13] and **Random Forests of MLC45** (RFMLC45) [MKGD12] are examples of ensembles of multi-label methods which utilize PCTs and MLC4.5 trees as base classifiers respectively. The diversity among base classifiers is achieved by bagging and by considering only a random subset of attributes for splitting at a node. **Ensemble of CCs** [RPHF11] (ECC), treated in Section 2.2.1 is another example of an ensemble method.

2.3 Comparison

In 2012, Madjarov et al.[MKGD12] conducted an extensive study of 12 MLL methods over 11 benchmark datasets, and reported 16 evaluation measures for the results. Tables 2.2, 2.3, 2.4 and 2.5 present the results reported by [MKGD12] for the hamming loss, precision, recall, and F1 score measures respectively for a subset of the datasets. A summary of the datasets is provided in Table 2.1. Methods that did not finish within 1 week under the available resources are marked "DNF" in the corresponding table cell.

Table 2.1: Summary of various multi-label datasets

name	domain	instances	nominal	numeric	labels	cardinality	density	distinct
yeast	biology	2417	0	103	14	4.237	0.303	198
enron	text	1702	1001	0	53	3.378	0.064	753
corel5k	images	5000	499	0	374	3.522	0.009	3175
delicious	text (web)	16105	500	0	983	19.020	0.019	15806
mediamill	video	43907	0	120	101	4.376	0.043	6555

Table 2.2: Performance of MLL approaches with regards to the *Hamming Loss* measure.

Dataset	BR	CC	CLR	QWML	HOMER	ML-C4.5	PCT	ML-kNN	RAkEL	ECC	RFML-C4.5	RF-PCT
yeast	0.190	0.193	0.190	0.191	0.207	0.234	0.219	0.198	0.192	0.207	0.205	0.197
enron	0.045	0.064	0.048	0.048	0.051	0.053	0.058	0.051	0.045	0.049	0.047	0.046
corel5k	0.017	0.017	0.012	0.012	0.012	0.010	0.009	0.009	0.009	0.009	0.009	0.009
mediamill	0.032	0.032	0.043	0.043	0.038	0.044	0.034	0.031	0.035	0.035	0.030	0.029
delicious	0.018	0.018	DNF	DNF	0.022	0.019	0.019	0.018	DNF	DNF	0.018	0.018

Table 2.3: Performance of MLL approaches with regards to the *Precision*(sample-based) measure.

Dataset	BR	CC	CLR	QWML	HOMER	ML-C4.5	PCT	ML-kNN	RAkEL	ECC	RFML-C4.5	RF-PCT
yeast	0.722	0.727	0.719	0.718	0.663	0.620	0.705	0.732	0.715	0.667	0.738	0.744
enron	0.703	0.464	0.650	0.624	0.616	0.623	0.415	0.587	0.708	0.652	0.690	0.709
corel5k	0.042	0.042	0.329	0.326	0.317	0.005	0.000	0.035	0.000	0.002	0.018	0.030
mediamill	0.731	0.741	0.201	0.203	0.597	0.056	0.694	0.724	0.705	0.690	0.765	0.772
delicious	0.443	0.399	DNF	DNF	0.369	0.001	0.001	0.424	DNF	DNF	0.472	0.512

Table 2.4: Performance of MLL approaches with regards to the *Recall* (sample-based) measure.

Dataset	BR	CC	CLR	QWML	HOMER	ML-C4.5	PCT	ML-kNN	RAkEL	ECC	RFML-C4.5	RF-PCT
yeast	0.591	0.600	0.601	0.600	0.714	0.608	0.490	0.549	0.615	0.673	0.491	0.523
enron	0.497	0.507	0.557	0.453	0.610	0.487	0.229	0.358	0.469	0.560	0.398	0.452
corel5k	0.055	0.056	0.264	0.264	0.250	0.002	0.000	0.014	0.000	0.001	0.005	0.009
mediamill	0.450	0.424	0.101	0.101	0.563	0.052	0.379	0.470	0.353	0.372	0.456	0.476
delicious	0.155	0.157	DNF	DNF	0.303	0.001	0.001	0.112	DNF	DNF	0.176	0.160

Table 2.5: Performance of MLL approaches with regards to the *F1 Score (micro)* (sample-based) measure.

Dataset	BR	CC	CLR	QWML	HOMER	ML-C4.5	PCT	ML-kNN	RAkEL	ECC	RFML-C4.5	RF-PCT
yeast	0.650	0.657	0.655	0.654	0.687	0.614	0.578	0.628	0.661	0.670	0.589	0.614
enron	0.582	0.484	0.600	0.525	0.613	0.546	0.295	0.445	0.564	0.602	0.505	0.552
corel5k	0.047	0.048	0.293	0.292	0.280	0.003	0.000	0.021	0.000	0.001	0.008	0.014
mediamill	0.557	0.539	0.134	0.135	0.579	0.054	0.490	0.570	0.471	0.483	0.572	0.589
delicious	0.230	0.225	DNF	DNF	0.343	0.001	0.001	0.017	DNF	DNF	0.256	0.244

A brief summary of the experimental setup in [MKGD12] is given below

- All experiments were conducted on a server with Intel Xeon processor at 2.50 Ghz with 64 GB RAM. The operating system used was Fedora 14.
- SVMs were used as base classifiers where relevant. Hyperparameters for the SVMs were set using 10-fold cross validation (with the exception of RAkEL, where 5-fold cross validation was used.)
- Methods were trained on all available training samples, and evaluated on all available test samples.
- For methods which required parameters, for instance k in HOMER, parameters were

instantiated using the recommendations in the literature. See [MKGD12] for full details. In particular, for HOMER, 5 different values of k were used, and the best results were reported.

Some of the conclusions that were drawn by [MKGD12] are

- HOMER has the best *recall*, while RF-PCT has the best *precision*.
- The best performing methods according to quite a few of the evaluation metrics and most datasets is either HOMER or RF-PCT. In terms of efficiency as well, they performed the best.

The final recommendation by [MKGD12] was that HOMER, CC, BR and RF-PCT should be used as benchmarks for MLL methods.

2.4 Summary

In this chapter we reviewed the problem of multi-label learning and its associated metrics, as well as some of its representative algorithms. Multi-Label Learning is a supervised learning problem that considers the setting where given an object, multiple categories are relevant to it. MLL consists of three related tasks - Label Ranking , which requires learning a model that outputs real valued scores for each label, Multi-Label Classification, which bipartitions the label space into relevant and irrelevant labels, and Multi-Label Ranking, which does both. Multi-label metrics are classified into label-based and example-based metrics. The former first aggregates true positives, false positives etc for each label across the test set and then aggregates it across labels, while the latter calculates a metric for each sample and averages it across the test set. Approaches for multi-label learning fall loosely into problem transformation methods, which transform the problem into simpler binary classification tasks, and algorithm adaptation methods, which extend a single-label classification algorithm to the multi-label setting. A third class of ensemble methods,

which use MLL methods as base classifiers can also be considered. Some examples of MLL methods are Binary Relevance, Classifier Chains, Calibrated Label Ranking, Random k-Labelsets, HOMER, Multi-Label C4.5, Predictive Clustering Trees, Multi-Label k-NN and Random Forest of PCTs. Finally, a comparative study done by [MKGD12] found that HOMER and RF-PCT performed the best across several metrics and datasets, and recommended that BR, CC, HOMER and RF-PCT be used as benchmark methods for MLL tasks.

Chapter 3

Proposed Method

In this chapter, we present our proposed method for the problem of multi-label classification, and present experimental results that give us insight into its working and efficacy. For the MLC problem we extend the HOMER algorithm. The HOMER algorithm was one of the first MLL methods specifically designed to tackle large label sets. Further, an extensive study of MLL methods conducted by [MKGD12] concluded that HOMER is one of the best performers across a range of metrics and MLL methods. It goes further to recommend HOMER as a benchmark method, along with BR, CC and RF-PCT.

We extend the HOMER algorithm by replacing the default partitioning procedure for the label space, with a graph partitioning approach, where the graph in question is based on the co-occurrences of labels. We hypothesize that this better captures the correlations between labels and these changes will yield partitions with better intra-cluster similarity.

In the next section Section 3.1 we review the details of the HOMER [TKV08] algorithm, and properly describe its framework, which we utilize when we describe our proposed approach in Section 3.2. But first, let us also review the definition of a multi-label pattern.

Given $\mathcal{X} = X_1 \times \cdots \times X_D$, a D-dimensional input space (of numeric or categorical variables), and an output space $L = \{\lambda_1, \lambda_2, \cdots, \lambda_Q\}$ of Q labels, a multi label pattern is a pair $(\mathbf{x}, Y)$, where $\mathbf{x} \in \mathcal{X}$ is the *instance vector* and $Y \subseteq L$ is called the *label set*.

In Section 3.2 we present our proposed changes to the HOMER algorithm, and explain our motivation. In Section 3.4 we present our experimental analysis of the same. Finally, Section 3.5 summarizes the topics covered in this chapter.

3.1 HOMER

Hierarchy Of Multi-label classifiERs (HOMER) was one of the first MLL methods to tackle the problem of datasets with a *large* label space (with 100s or 1000s of labels). HOMER does this by employing a divide and conquer approach and building a tree-hierarchy over the label space.

Given a training dataset D, the HOMER tree can be described as follows

1. Each node n has a labelset $L_n \subset L$ and dataset D_n satisfying

$$\text{For } \forall\ (\mathbf{x}, Y) \in D_n, \qquad Y \subseteq L_n \tag{3.1}$$

2. The labelset L_n corresponds to the *meta label* μ_n of the node n.

3. Each node n is either a leaf node or an internal node. If it is an internal node, it satisfies the property $\cup_{c \in \text{children}(n)} L_c = L_n$ and $L_i \cap L_j = \phi$, $\forall i, j \in \text{children}(n)$. If it is a leaf node, $|L_n| = 1$.

4. Each *internal* node contains a multi-label classifier $\mathcal{C}_n$, whose task is to predict the meta-labels μ_c of its child nodes.

3.1.1 Building the Hierarchy

Given the above formulation, the induction of the tree hierarchy is straightforward.

procedure CREATEHIERARCHY(LabelSet L, Dataset D, Partitioner MakePartition)

Initialize Tree

Initialize *root* with (L, D)

DivideNode(*root*)

function DIVIDENODE(Node n)

if $|L_n| = 1$ **then**

return

PARTITION $\leftarrow$ MakePartition(L_n)

for $L_c \in$ PARTITION **do**

$D_c \leftarrow \phi$

for $(\mathbf{x}, Y) \in D_n$ **do** ▷ Filter the data of the parent node

$Y_c \leftarrow Y \cap L_c$

if $Y_c \neq \phi$ **then**

$D_c \leftarrow D_c \cup (\mathbf{x}, Y_c)$

Initialize c with (L_c, D_c) and add to T

DivideNode(c)

return

Thus, building the hierarchy requires the definition of a procedure to *partition* L. [TKV08] suggests that partitioning of the labels space should be done based on similarity, so that fewer subtrees are activated, resulting in logarithmic prediction time. In the most optimistic case, the predicted labels form the full set of leaves of a single subtree, in which case the test instance would follow only a single path till the root of the sub tree.

With respect to the partitioning algorithm, [TKV08] introduced three variants of HOMER

1. HOMER-k : Labels are clustered by the vanilla k-means algorithm. Labels are represented as a vector of their occurrences in the training data set, i.e. given a label λ_i

its vector representation $\vec{\lambda}_i$ is given by

$$\vec{\lambda}_i(j) = \begin{cases} 1 & \text{if } \lambda_i \in Y_j \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

2. HOMER-B : Labels were clustered using a variant of k-means proposed by [TKV08] called Balanced K-Means, which outputs clusters with equal number of points.

3. HOMER-R : Labels were randomly partitioned into k equal sized clusters. This variant was mainly introduced as an ablation study, i.e. to answer the question *"Does the partitioning method matter?"*

3.1.2 Training & Prediction

Training

To train classifier $\mathcal{C}_n$ at node n, we first process each instance $(\mathbf{x}, Y)$ in dataset D_n to produce the *meta*-dataset $Z_n = \{(\mathbf{x}, M)\}$, where M is the set of meta-labels relevant to $(\mathbf{x}, Y)$. A ML pattern $(\mathbf{x}, Y)$ can be considered annotated with μ_n, if $L_n \cap Y \neq \phi$, i.e. Y contains at least one of the labels in L_n. Classifier $\mathcal{C}_n$ is thereafter trained on Z_n.

Prediction

Given an unseen instance $\mathbf{x}$, predicting its labels can be described as follows

1. Starting with the root, for a node n recursively forward $\mathbf{x}$ to child node c, if μ_c is predicted by $\mathcal{C}_n$.

2. If a leaf is reached, add it to the set of predicted labels.

3.1.3 Complexity

The time complexity for building the tree can be described by the recurrence

$$T(Q) = 2T(Q/2) + f(Q) \tag{3.3}$$

where $f(Q)$ is the time taken to partition the labelset at a node. Since typically, $f(Q) = \omega(Q)$, and the regularity condition ($2f(Q/2) = kf(Q)$ where $k < 1$) can also be considered to hold for any partitioning algorithm, the "master theorem" yields the solution to this recurrence as $T(Q) = \Theta(f(Q))$.

Prediction time in the best case scenario (for good partitions) is logarithmic in the size of the label space. Training time can be thought of as roughly linear in the number of labels, as it depends on the number of internal nodes - which is equal to $|L| - 1$.

3.2 Proposed Method

As mentioned in the previous section, HOMER provides a general framework for hierarchical multi-label classification, where the hierarchy can be derived by different methods. Experimental results in [TKV08] also indicate that hierarchies created by grouping *similar* labels together perform better than *random* hierarchies.

To that end, in this work we hypothesize that the similarities between labels are *better modeled as graphs* and by applying graph partitioning/clustering algorithms on these graphs, we can achieve superior partitions in terms of inter-cluster similarity. Moreover, similar graphical analysis has been conducted in [SKK16] to find partitions for RAkEL, and the results were promising.

In this method, we model the interrelations between labels as a co-occurrence graph defined as follows

Definition 3.2.1. Co-occurrence Graph Given a training dataset $D = \{(\mathbf{x}, Y)|\mathbf{x} \in$

$\mathcal{X}, Y \subseteq L\}$ of multi-label patterns, the co-occurrence graph constructed from D is a weighted undirected graph G_{CO}, where each vertex $v \in V$ corresponds to a label $\lambda \in L$ and the weight of an edge $(v_i, v_j) \in E$ is the number of patterns in D that are annotated with both λ_i and λ_j. That is,

$$w_{i,j} = |\{(\mathbf{x}, Y) \in D | \lambda_i \in Y, \lambda_j \in Y\}| \tag{3.4}$$

We investigate partitioning the co-occurrence graph G_{CO} using three algorithms - the Kernighan-Lin algorithm, Girvan-Newman and another algorithm based on minimum spanning trees. We review these algorithms in Section 3.3. Analogous to previous names, we call this method **HOMER-*G***.

In our variation of HOMER, we also change the definition of a leaf node from a node n, whose label set is of size one, to a label set of size *threshold*, where *threshold* is a user-specified parameter. Then, at each leaf node n we train a MLC classifier $\mathcal{C}_n$ on the dataset D_n, with output label space L_n. During prediction, we consider the predictions of all leaf nodes which have been reached, as scores for the corresponding labels, and set to 0 the labels whose leaf nodes have not been reached. The final score array is thresholded at 0.5 to yield the final predicted labels. This change is motivated by the idea that the most difficult predictions are between similar labels and so it is better to train a single classifier at that level that can utilize all the data available, to properly learn the discriminative function.

We also investigate the utility of using an *ensemble* of HOMER-R trees. Each HOMER-R tree is trained on the full training set and label space, but with different random hierarchies. The prediction of the ensemble is made by averaging the scores for each label and thresholding at 0.5. Scores for labels which are not reached in a hierarchy are set to 0.

3.3 Graph Clustering Algorithms

The graph clustering problem is the problem of finding clusters within a graph. A graph cluster is ill-defined and different algorithms define it differently, but intuitively it is a subgraph which has high connectivity within itself, and low connectivity with other clusters. We focus on clustering algorithms which are capable of hierarchical clustering, by recursive application, as this is in line with the requirements of our method. Before proceeding further, we define a few preliminaries

- A cut (S, T) of G is a partition of V. That is, $S \cup T = V$ and $S \cap T = \phi$. The cut value of this partition $c(S, T)$ is the sum of weights of edges that cross the partition,i.e $\sum_{u \in S, v \in V} w(u, v)$.
- An $s - t$ cut of G, is a cut (S, T) of V such that $s \in S$ and $t \in T$. A minimum $s - t$ cut of G, is the $s - t$ cut with the minimum cut value.

In Sections 3.3.1, 3.3.2, 3.3.3 we review the Kernighan-Lin algorithm, the clustering algorithm based on min-cut trees proposed in [FTT04] and the Girvan-Newman (edge centrality) algorithm respectively.

3.3.1 Kernighan-Lin Algorithm

The Kernighan-Lin algorithm [KL70] partitions the graph into equal sized clusters A, B while minimizing

$$\text{Cost} = \sum_{u \in A, v \in B} w(u, v) - \left(\sum_{u,v \in A} w(u, v) + \sum_{u,v \in B} w(u, v) \right) \tag{3.5}$$

The first term in the cost is *external cost*, the sum of the weights of inter-cluster edges (which is nothing but the cut value), while the second term is the *internal cost*, the sum of intra-cluster edges. The algorithm begins with a equal sized 2-way random partition (or can alternatively take a partition as input) and uses a greedy algorithm to sequentially pair

vertices between A and B whose interchange yields the *maximum gain* in cost. Particularly, it starts with A and B, and follows the steps

1. Computes the pair (x, y) such that $x \in A, y \in B$, whose interchange causes maximum decrease in cost.

2. Removes the pair (x, y) from A and B, and repeat this till all vertices are exhausted.

3. Choose a k such that interchanging the first k pairs results in maximum decrease of cost.

4. Interchange the k pairs, and restart from Step 1 with the new partition (A', B') and repeat all steps, till cost does not decrease any more, i.e. a local minimum is reached.

[KL70] also suggest choosing bad swaps to "perturb" the solution once a local minimum is reached. The time complexity of the Kernighan-Lin algorithm is $O(|V|^2 \log |V|)$.

3.3.2 Clustering with Min Cut Trees

In 2004, [FTT04] proposed a hierarchical graph clustering algorithm to find web communities in web graphs using *minimum cut trees*. Minimum cut trees, also known as Gomory-Hu trees, were first presented in [GH61] and allow us to efficiently compute all $s-t$ min cuts for an undirected weighted graph. Given an undirected weighted graph G, the minimum cut between any two vertices $s, v \in V$ is the same as the cut corresponding to the removal of the minimum weighted edge on the path between s and t in the corresponding Gomory-Hu tree T_G. Moreover, the minimum $s-t$ cut value, is the weight of the aforementioned edge. A minimum cut tree can be constructed in $O(|V| - 1)$ maximum flow computations. A maximum flow can be calculated using Karger's [KS96] Las Vegas algorithm in $O(T|E|)$ time, where T is the number of times the algorithm is repeated. Thus the minimum-cut tree can be computed in $O(T|E||V|)$ time.

The algorithm proposed by Flake et al. [FTT04] can be described as follows

1. Expand graph G to G' by adding an artificial sink t, connected to all vertices by an edge of weight α.

2. Construct the Gomory-Hu tree $T_{G'}$ corresponding to G'.

3. Remove the artificial sink t from $T_{G'}$, The resultant connected components of $T_{G'}$ correspond to web communities/clusters.

By defining communities (or clusters) in this manner, [GH61] are able to prove that α is an upper bound on the *inter-community* cut value, and a lower bound on the *intra-community* cut value. Thus, α defines the "distance", so to speak, between clusters. As α goes to infinity, the min-cut tree will obtain a star topology with the artificial sink at its center, and Flake's algorithm gives us $|V|$ trivial clusters. As α goes to 0 however, t must be a leaf node of the tree, because for any $s \in |V|$, the minimum $s-t$ cut will obviously be the trivial cut $(\{t\}, V)$, with cut value as 0. Clearly, α has a monotonic relation with the number of clusters. Thus an appropriate value of α can be found using a binary search-like procedure.

Calculation of connected components at Step 3 can be done using graph traversal in $O(|V| + |E|) = O(|V|)$, for a tree. Together with the computation of the Gomory-Hu tree, this gives an overall time complexity of $O(|V| + T|V||E|) = O(T|V||E|)$.

3.3.3 Girvan-Newman Algorithm

Girvan and Newman proposed an algorithm for bi-partitioning a graph in [GN02][NG04] by defining the notion of *edge betweenness*. The edge betweenness of an edge is equal to the number of *shortest paths* that contain that edge. The intuition behind this is that clusters are connected by few edges, and since shortest paths between vertices of different clusters must pass through one of them, their edge centrality values will be large. In short, edges with high edge betweenness act as "highways" between clusters. Thus, removal of such edges will disconnect the graph and uncover the clusters.

The edge betweenness is a specific realization of *edge centrality*, which is a property of an edge that measures how important an edge is, w.r.t some process. The same argument holds for the maximum central edges being inter-cluster edges, when edge centrality is defined as the *random-walk edge betweenness* or *current-flow edge betweenness*. Briefly, random-walk betweenness is the probability of crossing an edge on a random walk, between two random vertices. The current-flow betweenness is defined by considering the graph to be a circuit, with edges having unit resistance (for unweighted graphs). The current-flow betweenness is the average value of current carried by an edge when a voltage difference is applied across some pair of vertices.

The steps of the Girvan-Newman algorithm are

1. Compute edge centralities for each edge.
2. Remove edge with maximum centrality (ties are broken at random).
3. *Re-calculate* edge centralities.
4. Repeat Steps 1-3 till the graph is disconnected. The connected components are the required clusters.

In practice *edge betweenness* performs better (when used with the Girvan-Newman algorithm) than *random-walk edge betweenness* or *current-flow betweenness* and can also be computed faster ($O(|E||V|$ vs $O(|E||V|^2)$). It turns out, re-computation of the edge centralities is essential to finding good clusters, thus making for an overall of $O(|E| \times |E||V|) = O(|E|^2|V|)$ complexity to find the set of clusters.

3.4 Results and Discussion

In this section we report the experimental results that we have obtained, and offer our analysis for the same. First in Section 3.4.1 we describe the datasets that we run our

experiments on, and in Section 3.4.2 we describe the programming environment we use. Finally, in Section 3.4.3 we present our experiments with analysis.

3.4.1 Datasets and Metrics

We conduct our experiments on two multi label datasets - *mediamill* and *delicious*. Summaries for both datasets can be found in Table 2.1.

Mediamill [SWvG+06] belongs to the multimedia domain, and originates from the 2005 NIST TRECVID challenge dataset, which contains data about annotated videos. The target labels are 101 manually annotated "concepts" such as vehicle, truck, smoke, crowd, sky etc. Delicious contains textual data of web pages annotated with their tags. Delicious was created by [TKV08] and is named because of its origin from the `del.icio.us` website. `del.icio.us` is a social networking website for sharing and storing bookmarks. Bookmarks (which correspond to a webpage) are annotated with tags, a subset of which (after some filtering) form the target label space.

We utilize the training-test splits provided in [PV14], which have been processed from the original sources to include as many training labels as possible in the test set. [PV14] provides 5 splits, all experiments are run on the first split.

We report the example-based metrics *Hamming Loss, precision, recall* and *F1 score*, reviewed in 2.1. Where we report times (for training, testing or partitioning) we report it in turns of CPU times (total time spent in processing) in seconds. This is measured using the `clock()` function in the Python `time` module.

3.4.2 Environment

Our implementation of HOMER-G and all experiments is in `Python 3.6`. All experiments were run on single cores, with no parallelization.

- We use the `networkx` library for implementation of the Kernighan-Lin, Girvan-Newman algorithms, as well as the implementation of the Gomory-Hu Tree.

- We use `scikit-learn` for implementation of Gaussian Naive Bayes, Bernoulli Naive Bayes and Random Forest classifiers, which are used as base classifiers for BR models. Moreover, we use it's implementation of K-Means to write our own version of HOMER-k in `Python`.
- We use a `scikit-learn` compatible library, `scikit-multilearn` for its implementation of BR.

3.4.3 Experiments

In this section, we present three experiments to expose the inner workings of the HOMER algorithm and our proposed variant. For the graph clustering algorithm, we use the Kernighan-Lin algorithm reviewed in Section 3.3.1. Experiments show that the Girvan-Newman algorithm always produces the trivial partition $(\{v\}, V \setminus v)$, while Flake et al.'s algorithm either produces too many clusters (nearly one for each vertex) or no clusters for various values of α. Thus, we utilize the Kernighan-Lin algorithm because it gives a balanced partition. We henceforth call this variant HOMER-G.

In Experiment A, we compare HOMER-G, HOMER-R and HOMER-k and measure the trends with increasing size of leaves. This is motivated by the idea that increasing the leaf size allows classifiers at the leaf to learn the correlations between more number of classes directly, rather than if it was replaced by a subtree of smaller leaf size. In Experiment B we consider an ensemble of HOMER-R trees and investigate its performance with increase in the ensemble size. In Experiment C we replace the base classifiers, which were earlier Naive Bayes classifiers, with Random Forests of increasing size and compare the performance of HOMER-G, HOMER-R and HOMER-k.

A. Comparision of different partitioners at different leaf sizes

In this experiment, we measure the performance in terms of prediction as well as efficiency (in terms of training, testing and partitioning times) of HOMER with respect to three

different partitioning algorithms for the label space - *K-means* (as described in 3.1), random partitioning and graph partitioning using the Kernighan-Lin algorithm. We only consider binary partitions. Moreover we plot the variation of metrics with respect to the maximum size of the leaf nodes. Leaf sizes are varied from 1 to 10. Results reported are the means of 10 runs for each combination of partitioner, dataset and leaf size. Note that this is necessary, due to the random initialization of K-means and the Kernighan-Lin algorithm, and the random nature of the random partitioner.

Fig 3.1 and Fig 3.2 plot the variation of predictive performance and efficiencies (w.r.t. CPU times) for the *delicious* dataset, while Fig 3.3 and Fig 3.4 report the same for the *mediamill* dataset. The ML classifiers used at each node (internal *and* leaf) are BR models with base classifiers as Bernoulli and Gaussian Naive Bayes classifiers for *delicious* and *mediamill* respectively.

Our observations and analysis are as follows

- On both datasets, HOMER-k clearly dominates in terms of precision and HOMER-G dominates in terms of recall. HOMER-k has the worst recall on *delicious*, and recall in between HOMER-R and HOMER-G on *mediamill*. The precision of HOMER-G is almost the same as HOMER-R on both datasets. This seems to imply that although HOMER-k returns fewer incorrect labels, it fails to return all relevant labels. On the other hand, HOMER-G seems to be returning a lot of irrelevant labels, but however more of the relevant ones.

- HOMER-k has the least hamming loss, while HOMER-G has similar Hamming Loss to HOMER-G on *mediamill*, and slightly worse Hamming Loss on *delicious*.

- Overall, precision and recall performances interact so that HOMER-k dominates on *mediamill* w.r.t. the F1 score, and HOMER-G beats HOMER-k with a good margin on *delicious*. HOMER-R performs the worst on both datasets w.r.t. F1 score.

- Precision and recall both improve (more or less) on *delicious* with leaf size, leading to

a better F1 score. Hamming loss also improves. This seems to imply that for delicious, it is beneficial to learn a classifier that directly learns the correlations between a subset of classes (rather than have a subtree for that subset) . Possibly this will drop after leaf size increases beyond a threshold where the classes per leaf become too many. On *mediamill*, precision increases while recall decreases, leading to a more or less same F1 score. Moreover, hamming loss increases. One explanation for this could be that as the leaf size increases, many classes are reached that would not have been reached otherwise. This leads to more labels retrieved, both relevant and irrelevant.

- Surprisingly, partitioning time remains more or less constant as leaf size increases, whereas the expectation would be a *decrease*. Possibly, the largest chunk of the partitioning time occurs at the higher levels of the nodes. HOMER-k takes the longest on *mediamill*, while HOMER-G takes the longest on *delicious*. The second result is expected as Kernighan-Lin scales cubicly with the number of labels.

- As expected training time decreases with increase in leaf size, due to fewer nodes. HOMER-G takes the least time, which is expected due to the balanced nature of the Kernighan-Lin algorithm. However, we would expect that HOMER-R, has similar training time to HOMER-G since it is balanced as well. However, this is not the case. This could be an indicator that HOMER-G's base classifiers converge faster, possibly due to better partitioning of data (compared to HOMER). However, more experiments need to be conducted to confirm. HOMER-k has the highest training time, possibly because it produces unbalanced partitions, and hence more number of nodes.

- Testing times decrease with increase in leaf size for mediamill, which is expected due to decrease in tree depth. HOMER-k has the least testing time which seems to imply that fewer subtrees are activated - indicating better partitions. HOMER-R performs the worst, which is expected of a random partition.

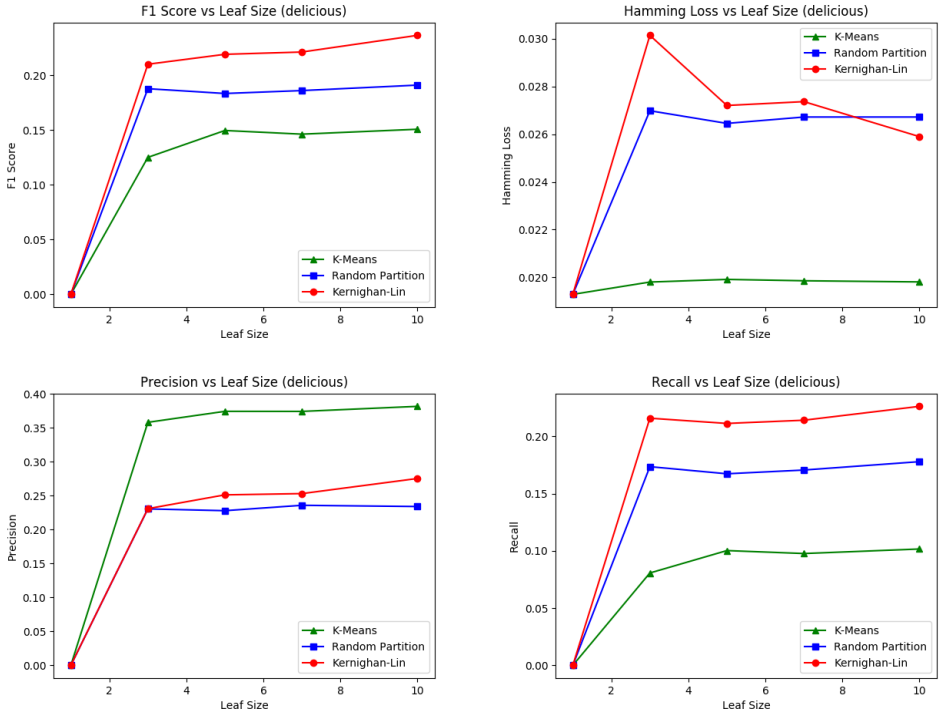

Fig. 3.1: Performance of HOMER with various partitioning schemes on *delicious* at different leaf sizes, with respect to the sample based metrics F1 score, precision, recall and Hamming Loss. Base classifier for trees is the Naive Bayes classifier.

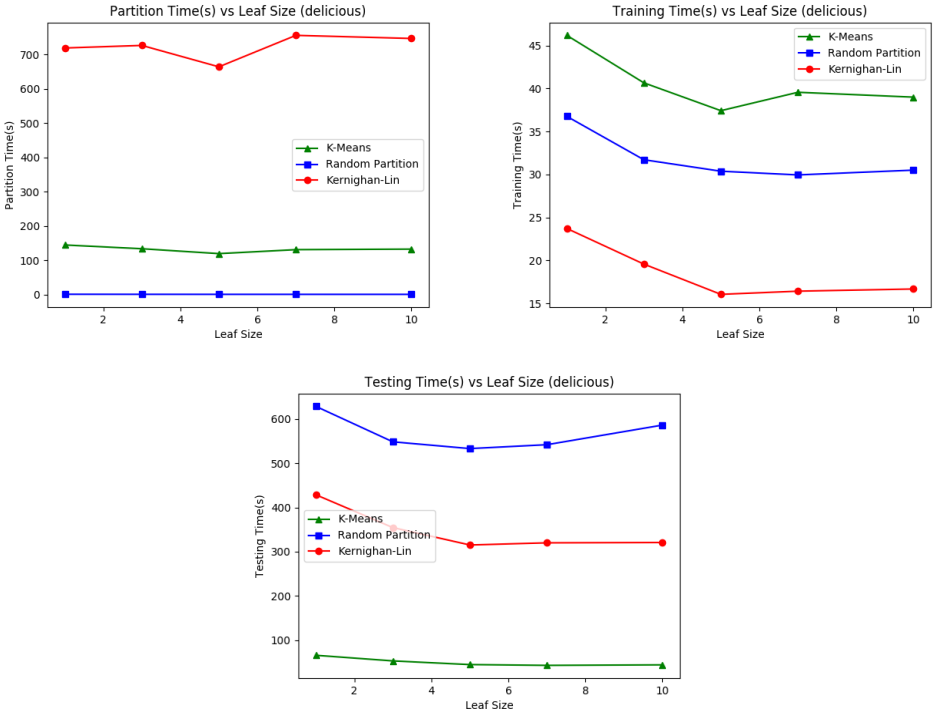

Fig. 3.2: Times taken to partition,train and test HOMER with various partitioning schemes on *delicious* at different leaf sizes. Times reported are CPU times, in seconds. Base classifier for trees is the Naive Bayes classifier.

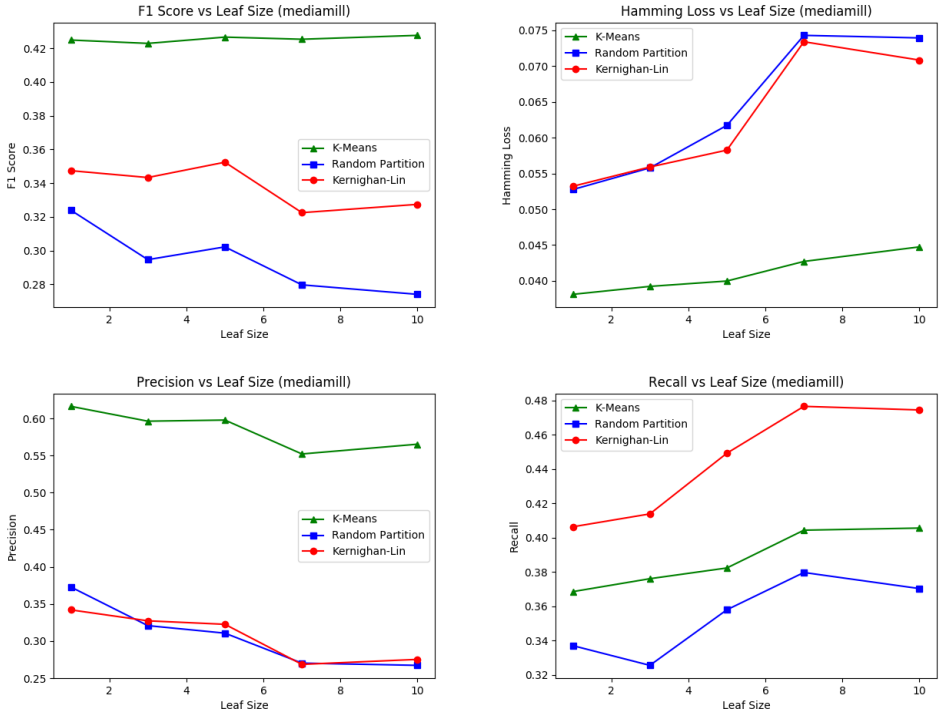

Fig. 3.3: Performance of HOMER with various partitioning schemes on *mediamill* at different leaf sizes, with respect to the sample based metrics F1 score, precision, recall and Hamming Loss. Base classifier for trees is the Naive Bayes classifier.

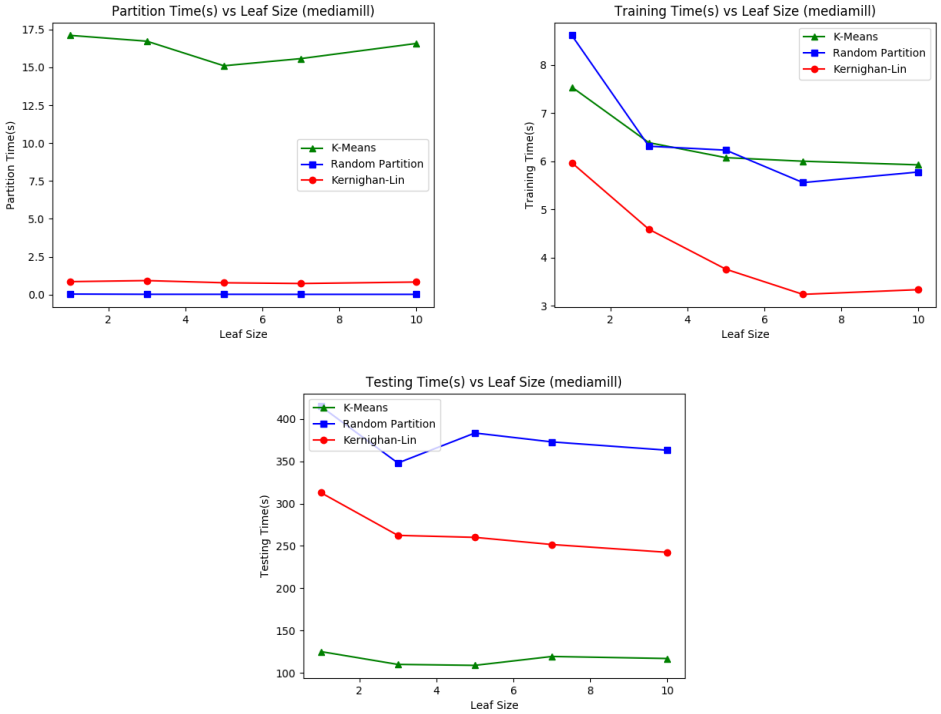

Fig. 3.4: Times taken to partition,train and test HOMER with various partitioning schemes on *mediamill* at different leaf sizes. Times reported are CPU times, in seconds. Base classifier for trees is the Naive Bayes classifier.

B. Performance of ensemble of HOMER-R trees

In this experiment, we measure the performance (w.r.t prediction and efficiency) of an ensemble of HOMER models which have been created using the random partitioning method. For each tree, the ML classifiers used at each node (internal *and* leaf) are a BR model with base classifiers as Bernoulli and Gaussian Naive Bayes classifiers for delicious and mediamill respectively. Prediction of individual members of the ensemble (HOMER-R trees) are combined by averaging the predicted probabilities and thresholding the resultant matrix.

For each dataset, we create 25 random trees and from this pool consider an ensemble of n random trees, where n varies from 1 to 25 (adding one tree *sequentially* at each step), and evaluate the ensemble as described earlier. We do this for 4 runs (for each dataset) and report the mean of the results.

Fig 3.5 and Fig 3.6 plot the variation of predictive performance with respect to the number of trees for *delicious* and *mediamill* respectively.

Our observations and analysis are as follows

- As the ensemble size increases, we observe an increase in precision and a decrease in recall for both datasets. This results in an increasing F1 score for *mediamill*, and a decrease for *delicious*. Moreover, the F1 score trend closely mirrors the trend in precision and recall for *mediamill* and *delicious* respectively.

- The Hamming loss for both datasets shows a decreasing trend.

- All prediction metrics seem to be convergent. Thus, increasing the number of trees beyond a point will not likely yield improvements.

- The training time, testing time and partitioning time show a perfectly linear trend which is obvious - anything else would be very surprising - since the ensemble is just a simple averaging of individual members, and there is no other overhead. We have not plotted these results.

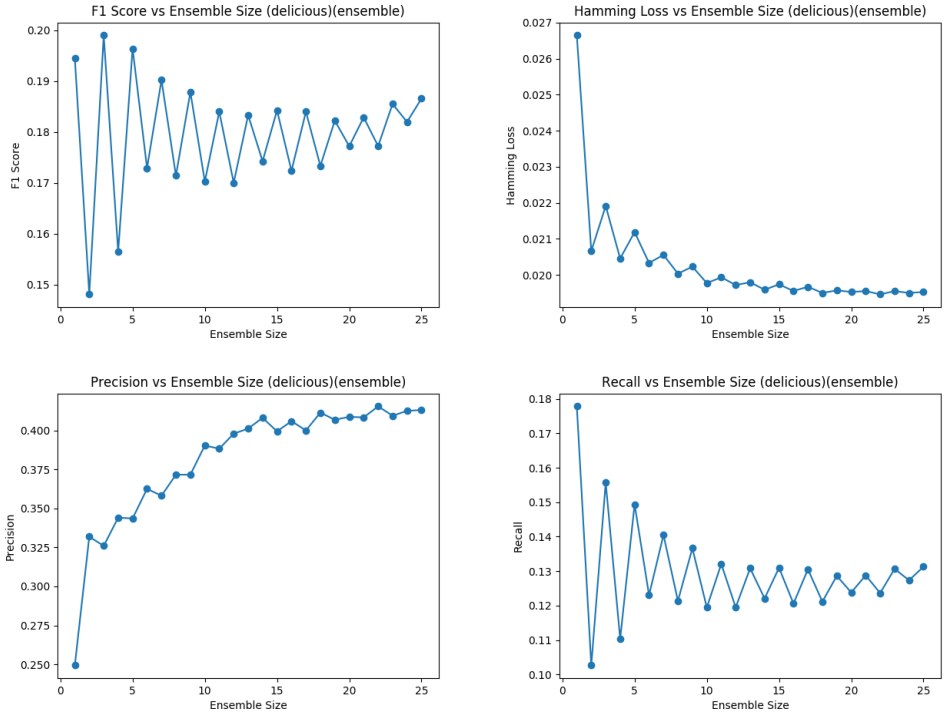

Fig. 3.5: Performance of an ensemble of HOMER-R trees on *delicious* at different sizes of ensemble, with respect to the sample based metrics F1 score, precision, recall and Hamming Loss. Base

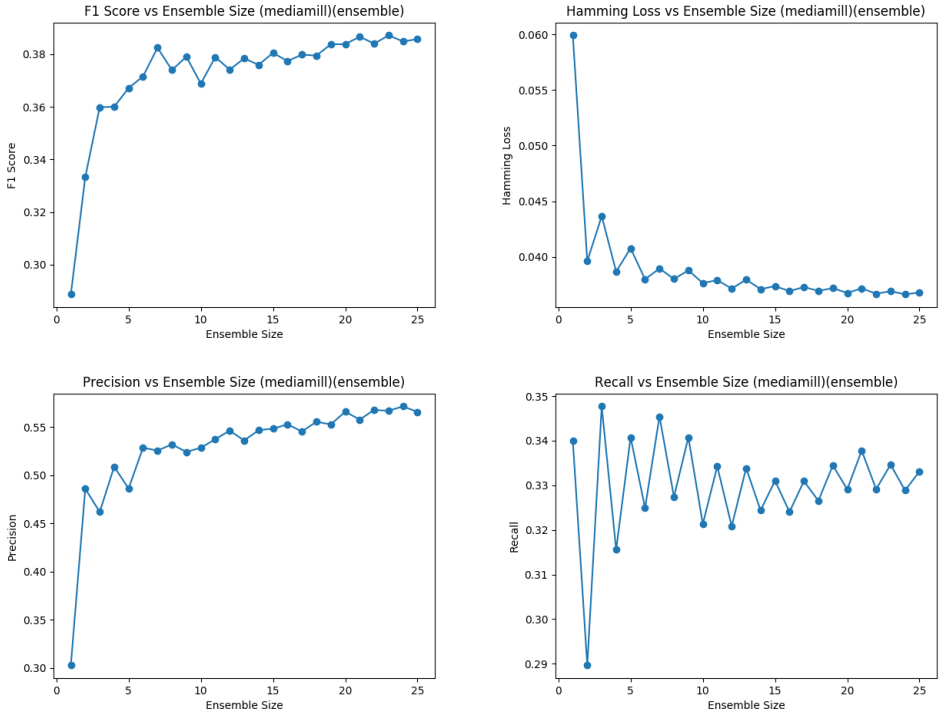

Fig. 3.6: Performance of an ensemble of HOMER-R trees on *mediamill* at different sizes of ensemble, with respect to the sample based metrics F1 score, precision, recall and Hamming Loss.

C. Comparision of partitioners with Random Forest as base classifier

Similar to Experiment A, in this experiment we compare the performances of HOMER with K-Means, Kernighan Lin and random partitioner as the partitioning methods. However, this time we keep the leaf size *fixed at 5*, and use a Random Forest of C4.5 trees as the base classifier model in BR (which is used as the ML classifier at nodes). We measure the performances at various *sizes of the Random Forest*, which we vary from 5 to 25 C4.5 trees. This is to answer the question : *how does the predictive performance of HOMER change with successively stronger base classifiers?*.

Similar to Experiment A, we only consider binary partitions. This time the results reported were the means of only 5 runs, due to the higher times required in training *multiple* large random forests for *each* node in the HOMER tree.

Fig 3.7 and Fig 3.8 plot the variation of predictive performances w.r.t. the Random Forest size for *delicious* and *mediamill* respectively.

Our observations and analysis are as follows

- There seems to be an overall increase in prediction performance with increase in the Random Forest size. Both precision and recall increase, resulting also in an increasing F1 score. Hamming loss is decreasing in *mediamill* and shows a slight increase, then plateau in *delicious*. Thus overall, we conclude that using better base classifiers, does increase the overall predictive performance.

- However, different algorithms benefit differently from the switch to Random Forests. On *delicious*, HOMER-k has the highest recall and lowest precision, reversing the trend observed in Section 3.4.3. Overall, this results in HOMER-k performing nearly as well as HOMER-G on delicious, where HOMER-G earlier dominated due to high recall in Experiment A. On the other hand, on *mediamill*, where HOMER-k already dominated in Section 3.4.3, here it absolutely outclasses the other methods in terms of F1 score. This can be explained in the increase in recall which is nearly the same

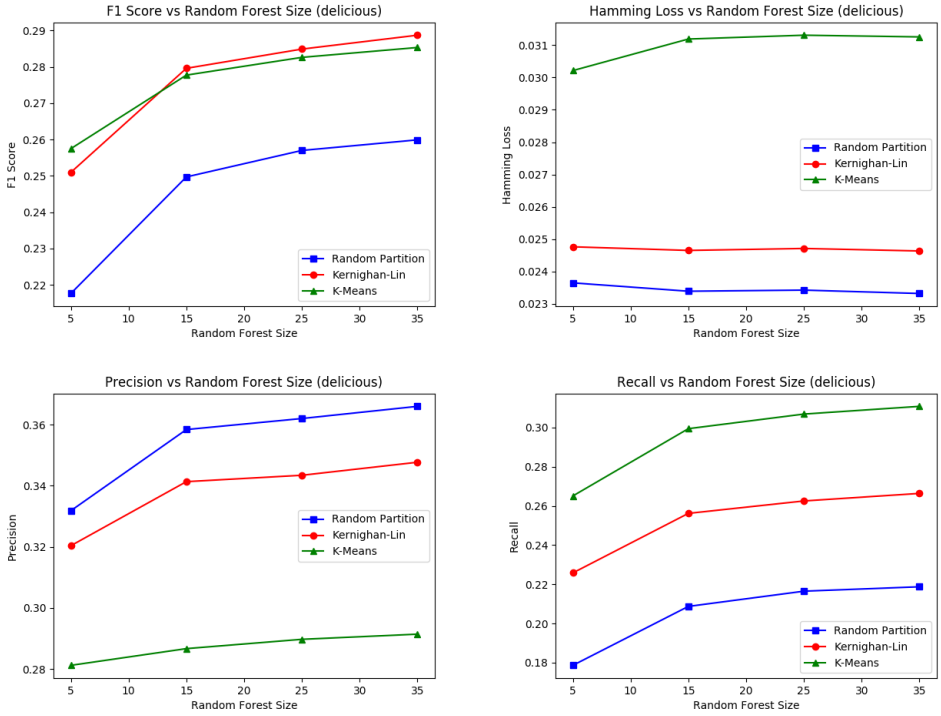

Fig. 3.7: Performance of an ensemble of HOMER-R trees on *delicious* at different sizes of Random Forest (used as base classifier), with respect to the sample based metrics F1 score, precision, recall and Hamming Loss. Leaf size is set at 5.

as HOMER-G, while still maintaining the highest precision.

- Naturally, prediction time is independent of the classifiers, so remains the same. The training and test times show a linear increase. The training times follow the order HOMER-k > HOMER-R > HOMER-G. This can be explained in the same way as the first experiment. The testing times for *mediamill* follow the order HOMER-R > HOMER-G > HOMER-k, which were the same trends observed in the first experiment. However, for *delicious*, the order is HOMER-k > HOMER-R > HOMER-G. The increase in testing time of HOMER-k in *delicious*, along with the higher recall and lower precision seems to indicate that HOMER-k is now returning a lot more labels, relevant and irrelevant, with the use of Random Forests as the base classifier.

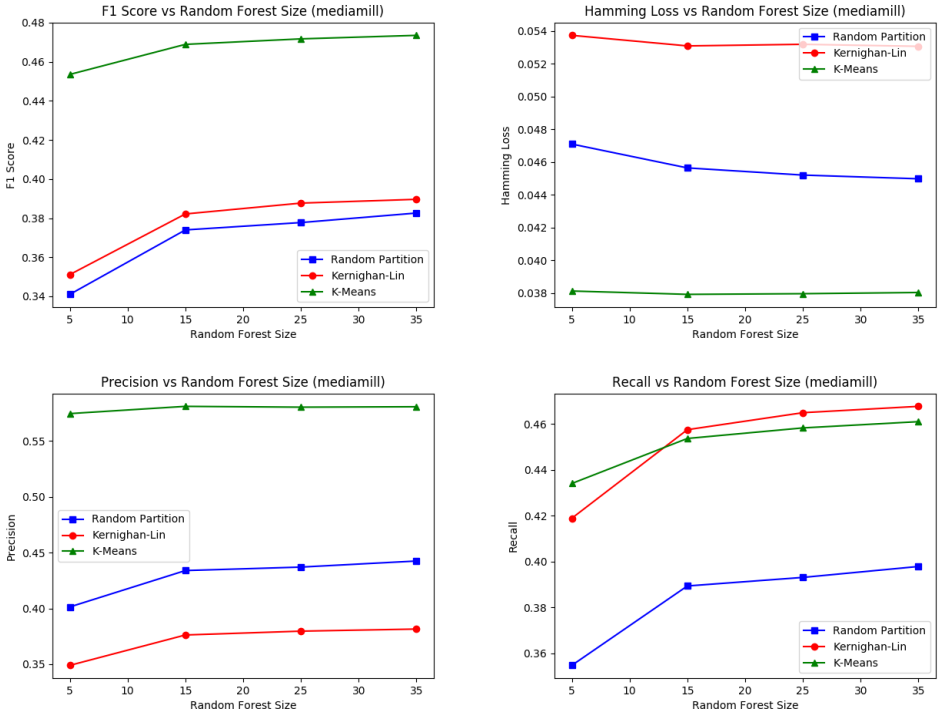

Fig. 3.8: Performance of an ensemble of HOMER-R trees on *mediamill* at different sizes of Random Forest (used as base classifier), with respect to the sample based metrics F1 score, precision, recall and Hamming Loss. Leaf size is set at 5.

3.5 Summary

In this chapter, we presented our proposed changes to the HOMER algorithm to tackle the problem of multi-label classification. We extend the HOMER algorithm by replacing the default partitioning procedure with a graph partitioning approach, where the graph in question is based on the co-occurrences of labels. We hypothesize that this better captures the correlations between labels and these changes will yield partitions with better intra-cluster similarity.

Thereafter we conduct three experiments on two datasets - *mediamill* and *delicious* to validate our hypothesis. We compare our variation HOMER-G, with two previously proposed variants HOMER-k and HOMER-R. The results are not entirely in our favor, but not completely bleak either. We find that HOMER-G performs consistently outperforms the other methods with respect to its recall, except for the *delicious* dataset when Random Forests are used as the base classifier, but lags when it comes to precision and Hamming Loss. Overall, it outperforms the other methods on the *delicious* dataset with respect to the F1 score. We also investigate an ensemble of HOMER-R trees which show promising results.

Chapter 4

Conclusion & Future Work

In this work, we extended the HOMER algorithm to tackle the problem of multi-label classification by replacing the default partitioning procedure with a graph partitioning approach, where the graph in question is based on the co-occurrences of labels. We call this approach HOMER-G. We hypothesize that this better captures the correlations between labels and these changes will yield partitions with better intra-cluster similarity. In particular, we believe that the co-occurrence data is a rich source of information that has the potential to yield better partitions. However, we find that our formulation of the co-occurrence graph is not particularly amenable to graph clustering, which is why we were unable to evaluate HOMER-G for other algorithms besides the Kernighan-Lin algorithm. It is quite possible that we have not represented the data in the right form. Whether the Kernighan-Lin algorithm managed to vindicate our hypothesis, is a question whose answer is still unclear.

We conducted three experiments on two datasets - *mediamill* and *delicious* to validate our hypothesis. We compared our variation HOMER-G, with two previously proposed variants HOMER-k and HOMER-R. The results are not entirely in our favor, but not completely bleak either. We find that HOMER-G performs consistently outperforms the other methods with respect to its recall, except for the *delicious* dataset when Random Forests are used as the base classifier, but lags when it comes to precision and Hamming

Loss. Overall, it outperformed the other methods on the *delicious* dataset with respect to the F1 score. We also investigated an ensemble of HOMER-R trees which show promising results.

We conclude that this preliminary work, although not a complete proof of our hypothesis, is sufficiently promising to merit further research. In particular, it opens up a lot of future research directions that we can consider

1. On one hand, graph partitioning algorithms did not perform as well as expected on the given problem. Particularly, they are prone to trivial partitions, except the *Kernighan-Lin* algorithm, which is designed to give balanced partitions. On the other hand, the co-occurrence matrix is a rich source of information about label correlations. It is however possible that the co-occurrence graph in its present form is not very amenable to present graph partitioning algorithms. In this direction we seek to address the following points

    - The co-occurrence graph can be thought of as the superimposition of *cliques*, corresponding to each training instance. The min-cut of a clique is always the trivial cut $(v, V \setminus v)$. This could explain why several of the graph clustering algorithms tested are prone to trivial cuts. Is it possible to mathematically analyze this hypothesis?

    - Instead of the weights of the graph, it is sometimes recommended to partition using another measure of similarity between vertices such as the *Jaccard similarity* which is based on the neighbors of a vertex in a graph. For a co-occurrence graph, which is usually complete, this is meaningless. However, we can first prune the graph, and then compute the Jaccard similarities. Moreover, pruning the graph may also break the symmetry that *could* be causing the trivial cut scenario in the first place.

    - Or perhaps, we have just not found all the right clustering algorithms. There is

still a large body of graph clustering algorithms to look through. One promising candidate is the use of DBScan, with co-occurrence values as similarities.

2. Ensembles of HOMER-R trees show promising results. But in its current form, they are still too slow. Since prediction time is approximately logarithmic in the number of labels, one possible solution could be to train HOMER-R trees on subsets of the label space, similar to RAkEL [TV07]. This could also add more diversity to the ensemble and improve classification performance. Another direction to increase diversity is to train the trees by bagging the training data.

   Moreover, the ensemble combination method currently being utilized is a simplistic averaging scheme. Other strategies can be investigated. One particular strategy can be to weight the trees based on their performance on a held-out subset of the training data.

3. Further experimentation should be done with different base classifiers and datasets. Particularly, [MKGD12] reports results with SVMs as the base classifier. In the future, we should also train HOMER-G with SVMs to see how it compares to benchmark algorithms for MLL. Moreover, in this work, we only compared HOMER-G with HOMER-k, however [TKV08] reported overall better results (in terms of both performance and efficiency) with the balanced clustering variant HOMER-B. Thus, further experiments should include comparison with HOMER-B as well.

4. Further, we should try to answer the question - *why does HOMER-G have high recall but low precision?*

References

[ACMM12] Everton Alvares-Cherman, Jean Metz, and Maria Carolina Monard. Incorporating label dependency into the binary relevance framework for multi-label classification. *Expert Systems with Applications*, 39(2):1647 – 1655, 2012.

[BRR00] Hendrik Blockeel, Luc De Raedt, and Jan Ramon. Top-down induction of clustering trees. *CoRR*, cs.LG/0011032, 2000.

[CK01] Amanda Clare and Ross D King. Knowledge discovery in multi-label phenotype data. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 42–53. Springer, 2001.

[FHLB08] Johannes Fürnkranz, Eyke Hüllermeier, Eneldo LozaMencía, and Klaus Brinker. Multilabel classification via calibrated label ranking. *Machine Learning*, 73(2):133–153, Nov 2008.

[FTT04] Gary William Flake, Robert E Tarjan, and Kostas Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1(4):385–408, 2004.

[GH61] Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[GN02] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[GV] Eva Gibaja and Sebastin Ventura. Multilabel learning: a review of the state of the art and ongoing research. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(6):411–444.

[HFCB08] Eyke Hllermeier, Johannes Frnkranz, Weiwei Cheng, and Klaus Brinker. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16):1897 – 1916, 2008.

[KL70] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[KS96] David R Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM (JACM)*, 43(4):601–640, 1996.

[KVSD13] Dragi Kocev, Celine Vens, Jan Struyf, and Sao Deroski. Tree ensembles for predicting structured outputs. *Pattern Recognition*, 46(3):817 – 833, 2013.

[LMPF10] Eneldo Loza Mencía, Sang-Hyeun Park, and Johannes Fürnkranz. Efficient voting prediction for pairwise multilabel classification. *Neurocomput.*, 73(7-9):1164–1176, March 2010.

[MKGD12] Gjorgji Madjarov, Dragi Kocev, Dejan Gjorgjevikj, and Sašo Džeroski. An extensive experimental comparison of methods for multi-label learning. *Pattern recognition*, 45(9):3084–3104, 2012.

[NG04] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[PF07] Sang-Hyeun Park and Johannes Fürnkranz. Efficient pairwise classification. In Joost N. Kok, Jacek Koronacki, Raomon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Machine Learning: ECML 2007*, pages 658–665, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[PV14] Yashoteja Prabhu and Manik Varma. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 263–272, New York, NY, USA, 2014. ACM.

[RI10] Lior Rokach and Ehud Itach. An ensemble method for multi-label classification using an approximation algorithm for the set covering problem. In *Proceedings of the 2nd International Workshop on Learning from Multilabel Data (MLD)*, pages 37–44, 2010.

[RPHF11] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333, Jun 2011.

[SKK16] Piotr Szymanski, Tomasz Kajdanowicz, and Kristian Kersting. How is a data-driven approach better than random choice in label space division for multi-label classification? *CoRR*, abs/1606.02346, 2016.

[SWvG+06] Cees G. M. Snoek, Marcel Worring, Jan C. van Gemert, Jan-Mark Geusebroek, and Arnold W. M. Smeulders. The challenge problem for automated detection of 101 semantic concepts in multimedia. In *Proceedings of the 14th ACM International Conference on Multimedia*, MM '06, pages 421–430, New York, NY, USA, 2006. ACM.

[TDS+09] Grigorios Tsoumakas, Anastasios Dimou, Eleftherios Spyromitros, Vasileios Mezaris, Ioannis Kompatsiaris, and Ioannis Vlahavas. Correlation-based pruning of stacked binary relevance models for multi-label learning. In *Proceedings of the 1st international workshop on learning from multi-label data*, pages 101–116, 2009.

[TKV08] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *Proc.*

ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD08), volume 21, pages 53–59. sn, 2008.

[TKV10] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. *Mining Multi-label Data*, pages 667–685. Springer US, Boston, MA, 2010.

[TV07] Grigorios Tsoumakas and Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. In *European conference on machine learning*, pages 406–417. Springer, 2007.

[ZZ07] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038 – 2048, 2007.