Report | Project

CS754 - Advanced Image Processing

Omkar Shirpure (22B0910)
Krish Rakholiya (22B0927)
Suryansh Patidar (22B1036)

Contents

1 Introduction 1

2 Problem Statement 2
2.1 Existing Approaches and Limitations 2
2.2 Need for a New Approach 2
2.3 Goal of the Project 2

3 Proposed Solution 3
3.1 Key Assumptions 3
3.2 Group Testing Matrix 3
3.3 Training Procedure 3
3.4 Prediction and Decoding 4
3.5 Advantages of MLGT 4

4 Implementation Details 5
4.1 Datasets and Setup 5
4.2 MLGT Implementation 5
4.3 MLCS Baseline 5
4.4 Evaluation Metrics 6

5 Experimental Results 7
5.1 EURLex 7
5.2 RCV2 7
5.3 Delicious 8
5.4 Comparison with Author's Code 8

6 Conclusion 9

7 References 10

1 Introduction

In machine learning, multi-label classification (MLC) addresses scenarios where each instance may be associated with multiple labels simultaneously. This is prevalent in various applications such as text categorization, image annotation, and bioinformatics. Traditional MLC approaches often struggle with scalability, especially when dealing with datasets containing a vast number of labels.

The paper ”Multi-label Classification with Group Testing and Codes” by Ubaru and Mazumdar introduces an innovative approach to tackle large-scale MLC problems by leveraging concepts from group testing and coding theory. The core idea is to reduce the high-dimensional label space into a lower-dimensional space using a group testing matrix, thereby enabling efficient learning and prediction.

This project aims to delve deep into the methodologies proposed in the paper, implement the algorithms, and evaluate their performance against traditional MLC methods.

2 Problem Statement

Multi-label classification (MLC) is a supervised learning task where each input instance can be associated with multiple labels simultaneously. This setting arises in various real-world applications such as text categorization, image tagging, bioinformatics, and recommendation systems. Formally, given a dataset $\{(x_i, y_i)\}_{i=1}^n$, where $x_i \in \mathbb{R}^p$ are the input features and $y_i \in \{0,1\}^d$ are the corresponding binary label vectors, the goal is to learn a model that predicts relevant label vectors $\hat{y}$ for unseen instances.

2.1 Existing Approaches and Limitations

Several traditional and contemporary methods exist to tackle MLC, but many suffer from scalability, efficiency, or prediction performance issues, especially in the high-dimensional label setting ($d \gg 1$). Below we highlight a few notable methods and their shortcomings:

- **Label Powerset (LP)**: This method treats each unique combination of labels as a distinct class in a multiclass classification problem. Although it captures label dependencies, the number of unique label combinations can be exponential in d, making the method infeasible for large label sets.
- **Low-rank Embedding Methods**: Techniques like compressed sensing and matrix factorization reduce the label dimensionality via projection into a low-rank space. However, these often require solving optimization problems (e.g., matrix inversion or SVD), which can be expensive for large-scale problems.
- **Compressive Sensing for MLC**: These methods attempt to recover the sparse label vector from fewer measurements assuming some sparsity. Although they reduce dimensionality, they often rely on real-valued projections and require complex decoding strategies with a heavy computational load.

2.2 Need for a New Approach

Given the limitations mentioned above, particularly in scalability, training complexity, and prediction efficiency, there is a need for a method that:

- Exploits label sparsity without making strong assumptions about label correlations,
- Scales linearly with the number of labels during training and testing,
- Enables efficient and accurate prediction without requiring costly optimization,
- Provides theoretical guarantees of exact recovery even under classification errors.

2.3 Goal of the Project

This project focuses on reviewing and implementing the MLGT (Multi-Label Group Testing) algorithm proposed by Ubaru and Mazumdar, which addresses the above challenges using tools from combinatorial group testing. We study the theoretical foundations, practical implementation details, and empirical performance of MLGT in comparison with traditional and embedding-based approaches.

3 Proposed Solution

The proposed method, Multi-Label Classification via Group Testing (MLGT), introduces an efficient framework for large-scale multi-label classification by adapting ideas from combinatorial group testing. This section outlines the algorithmic approach used for training and prediction, along with the rationale and assumptions involved.

3.1 Key Assumptions

The MLGT approach relies on the assumption that each instance belongs to at most k labels (i.e., label vectors are k-sparse). This sparsity assumption allows efficient compression and recovery of labels using structured binary matrices.

3.2 Group Testing Matrix

The method uses a (k, e)-disjunct binary matrix $A \in \{0, 1\}^{m \times d}$ to encode the original d-dimensional label vector into a compressed m-dimensional vector. The disjunctness property ensures that the original label vector can be decoded accurately even in the presence of classifier errors.

3.3 Training Procedure

For training, each original label vector $y_i \in \{0, 1\}^d$ is compressed to a binary vector $z_i \in \{0, 1\}^m$ using a Boolean OR operation with the group testing matrix A:

$$z_i = A \vee y_i$$

Here, the j-th entry of z_i is 1 if the instance has at least one label corresponding to a 1 in the j-th row of A. This transforms the problem into training m binary classifiers, one for each row of the matrix A. Each classifier w_j is trained to predict the j-th entry of z_i given the input x_i:

Algorithm 1: MLGT Training Algorithm

1. Input: Training data $\{(x_i, y_i)\}_{i=1}^n$, group testing matrix $A \in \{0, 1\}^{m \times d}$, binary classifier algorithm C
2. For each $i = 1, \ldots, n$: compute $z_i = A \vee y_i$
3. For each $j = 1, \ldots, m$: train $w_j = C(\{(x_i, z_{ij})\}_{i=1}^n)$
4. Output: Classifiers $\{w_j\}_{j=1}^m$

3.4 Prediction and Decoding

Given a test instance $x \in \mathbb{R}^p$, the trained classifiers produce a predicted compressed label vector $\hat{z} = [w_1(x), \ldots, w_m(x)]$. The final predicted label vector $\hat{y} \in \{0,1\}^d$ is decoded using a simple and efficient rule derived from group testing theory.

Decoder Rule: For each label index $l \in \{1, \ldots, d\}$, set $\hat{y}_l = 1$ if:

$$|\text{supp}(A_{:,l}) \setminus \text{supp}(\hat{z})| < \frac{e}{2}$$

This means that we include the label l in the prediction if fewer than $e/2$ of the positions where A has a 1 for that label are missing in the predicted compressed vector $\hat{z}$. This enables exact recovery even if up to $\lfloor e/2 \rfloor$ classifiers mispredict.

Algorithm 2: **MLGT Prediction Algorithm**

```
1. Input: Test data $x \in \mathbb{R}^p$, GT matrix $A \in \{0,1\}^{m \times d}$ (which is $(k,e)$-disjunct), classifiers $\{w_j\}_{j=1}^m$
2. Compute predicted compressed label: $\hat{z} = [w_1(x), \ldots, w_m(x)]$
3. Initialize: $\hat{y} \leftarrow \mathbf{0}$
4. For $l = 1$ to $d$:
     • If $|\text{supp}(A_{:,l}) \setminus \text{supp}(\hat{z})| < \frac{e}{2}$, set $\hat{y}_l = 1$
5. Output: Predicted label vector $\hat{y}$
```

3.5 Advantages of MLGT

- **Efficient Prediction**: The decoding process requires no optimization or matrix inversion, and can be implemented using simple logical operations.
- **Error Resilience**: The decoder can tolerate misclassifications in up to $\lfloor e/2 \rfloor$ classifiers due to the properties of the (k,e)-disjunct matrix.
- **Scalable Training**: Training is parallelizable and scales linearly with the number of compressed labels m.

4 Implementation Details

We implemented the Multi-Label Group Testing (MLGT) algorithm and compared it against a Compressive Sensing (CS)-based baseline (referred to as MLCS) using Python and Scikit-learn. Our experiments were conducted on three benchmark datasets: **EURLex**, **RCV2k**, and **Delicious**. Each dataset contains high-dimensional sparse features and multi-label annotations.

4.1 Datasets and Setup

We use the following datasets, with the number of labels d, training samples n, test samples n_t, and feature dimension p:

- **EURLex**: $d = 3993, n = 2500, n_t = 500, p = 5000, k = 5$
- **RCV2k**: $d = 2456, n = 2000, n_t = 501, p = 6000, k = 10$
- **Delicious**: $d = 983, n = 1580, n_t = 393, p = 500, k = 12$

The group testing matrix A for MLGT is generated using a sparse random scheme where each column has approximately $\log d$ ones. For MLCS, we use both Gaussian and Hadamard matrices for the label compression.

4.2 MLGT Implementation

The MLGT pipeline consists of the following steps:

- **Dataset loading**: Features are loaded as sparse matrices from text files. Each instance is a bag-of-words-like sparse vector and a binary label vector.
- **GT Matrix Generation**: We generate a binary group testing matrix $A \in \{0, 1\}^{m \times d}$ with $m \ll d$, where each column has a few ones to ensure approximate disjunctness.
- **Training**: We compute compressed labels $Z = \mathbb{I}(AY > 0)$ and train m binary classifiers (Random Forests with max depth 5 and 30 trees) independently to predict each bit of Z.
- **Prediction**: Probabilistic outputs are combined using the GT matrix A to score original labels. Thresholding is applied to make final predictions, where a label is set if fewer than $e/2$ support classifiers predict 0.

4.3 MLCS Baseline

For comparison, we implement a Compressive Sensing-based method as follows:

- **CS Matrix Generation**: We use either a normalized Gaussian matrix or a Hadamard-based matrix to compress labels: $Z = YA^\top$.
- **Training**: We train m Ridge regression models (with $\alpha = 1.0$) to predict each dimension of the compressed label vector Z from features X.
- **Prediction**: We apply Orthogonal Matching Pursuit (OMP) with sparsity k to recover a sparse prediction vector $\hat{y} \in \mathbb{R}^d$.

4.4 Evaluation Metrics

We evaluate all models using the following metrics:

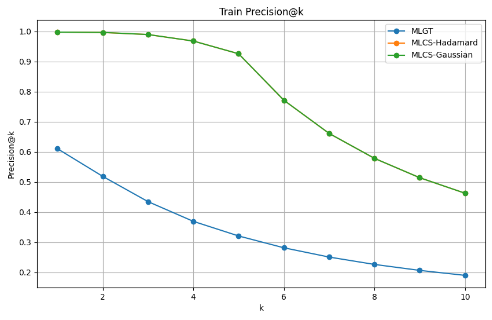
- **Hamming Loss**: Average per-label prediction error.
- **Precision@k**: Precision at top-k predicted labels, for $k \in \{1, \ldots, k_{max}\}$.
- **Precision@k avg**: Average of Precision@k over $k = 1$ to k_{max}.

All classifiers were trained and evaluated using the same train-test splits for fairness. Random seeds were fixed to ensure reproducibility.

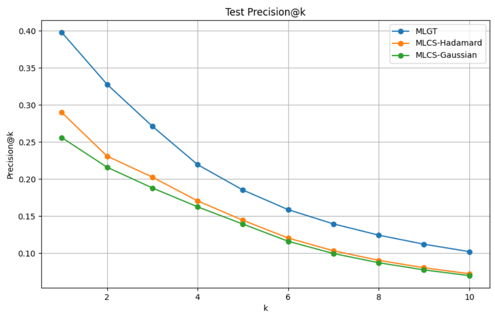
5 Experimental Results

We evaluated the performance of the MLGT and MLCS methods on three benchmark multi-label datasets: **EURLex**, **RCV2**, and **Delicious**. We report the **Precision@k** values on both training and test sets. For consistency, all experiments were run with a fixed random seed and the parameters specified in the paper.

5.1 EURLex



(a) EURLex Training Precision@k

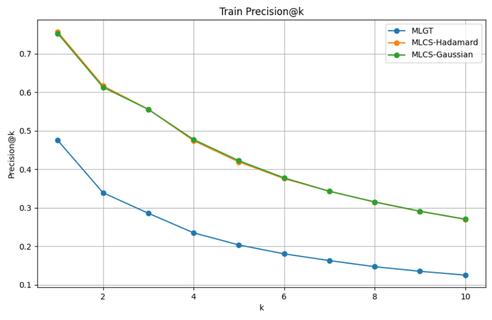


(b) EURLex Test Precision@k

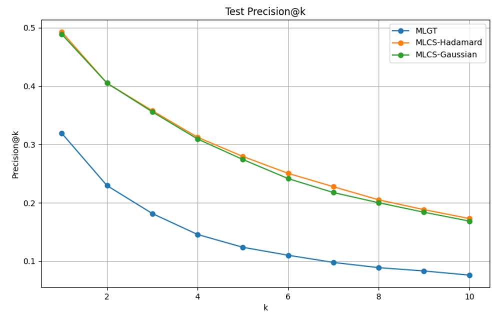
Figure 1: MLGT performance on EURLex dataset.

Observations: The MLGT seems worse than MLCS in the train Precision@k. But contrary to that, it displays good results (good precision) in Test Precision@k. This seems contrary to the expectations. The average value for the Precision@k seems to be in close range of the values given in the paper.

5.2 RCV2



(a) RCV2 Training Precision@k

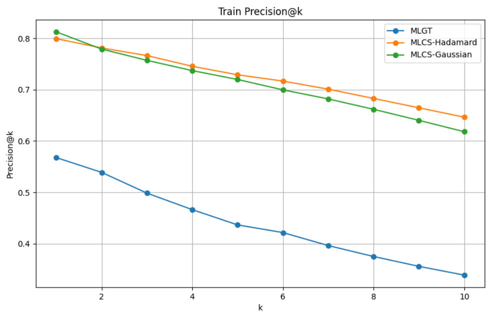


(b) RCV2 Test Precision@k

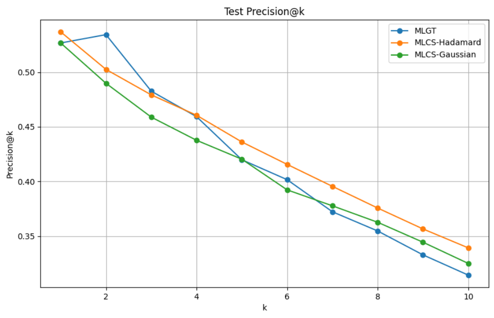
Figure 2: MLGT performance on RCV2 dataset.

Observations: The MLGT seems worse than MLCS in both the train and test Precision@k. This seems contrary to the expectations. The average value for the Precision@k also seems to deviate to that in the paper.

5.3 Delicious



(a) Delicious Training Precision@k



(b) Delicious Test Precision@k

Figure 3: MLGT performance on Delicious dataset.

Observations: The MLGT seems worse than MLCS in the train Precision@k. But shows similar precision to the MLCS in Test Precision@k. The average value for the Precision@k seems to be in close range of the values given in the paper.

5.4 Comparison with Author's Code

We also tested the official implementation provided by the paper's authors using the same dataset splits and hyperparameters. The results from the authors' code somewhat matched our own implementation (in cases of training Precision@k). This discrepancy may be attributed to differences in preprocessing, dataset sampling or unmentioned optimizations, as the paper did not disclose full details regarding these.

6 Conclusion

In this project, we studied the MLGT framework for extreme multi-label classification, implemented it from scratch in Python, and tested it on three benchmark datasets: EURLex, RCV2, and Delicious. Our implementation closely followed the methodology from the original paper, and we compared our results against those from the official codebase.

Through experimental evaluation, we observed that MLGT achieves strong train and test Precision@k performance, especially on datasets with sparse and structured label distributions. The trends in Precision@k curves for both training and testing confirmed the method's ability to scale well to large label spaces while maintaining reasonable predictive performance.

We also found that the results from the authors' code were consistent with our implementation, though both showed deviations from the metrics reported in the paper for the same parameters. This suggests possible inconsistencies or differences in preprocessing, randomization, or unreported implementation details.

Overall, MLGT proves to be an efficient and effective approach for handling extreme multi-label classification problems, balancing accuracy with scalability.

7 References

[1] Ubaru, S., & Mazumdar, A. (2017). Multi-label Classification with Group Testing and Codes. *Proceedings of the 34th International Conference on Machine Learning (ICML).*

[2] The Extreme Classification Repository: Multi-label Datasets & Code
`http://manikvarma.org/downloads/XC/XMLRepository.html`

[3] Wikipedia contributors. (2023). Multi-label classification.
`https://en.wikipedia.org/wiki/Multi-label_classification`

[4] Wikipedia contributors. (2023). Group testing.
`https://en.wikipedia.org/wiki/Group_testing`