Multilabel Classification by Hierarchical Partitioning and Data-dependent Grouping

Soumya Mukherjee (24M0768)

Manivannan N (24M0796)

Manish Kumar (24M0853)

In this project we reproduced the problem explain in paper by Ubaru et al. 2020

1 Problem Statement

Multilabel classification is a machine learning task where each input sample can be assigned multiple labels simultaneously. This is useful in many domains such as text categorization, image tagging, natural language processing, computer vision and medical diagnosis. In modern large-scale multilabel classification problems, two key characteristics often emerge: **sparsity** in label vectors and the presence of an **unknown hierarchical structure** among labels. Sparsity occurs because instances are typically associated with only a small fraction of the available labels, leading to binary label vectors dominated by zeros. In large-scale scenarios, labels also exhibit latent hierarchical relationships , which are not explicitly provided but can significantly influence classification performance.
Author introduce a novel framework that leverages both label sparsity and hierarchical dependencies to enhance computational efficiency and predictive accuracy. This approach projects the high-dimensional label space into a compact, low-dimensional representation using data-driven label groupings, drawing inspiration from group testing strategies. **Grouping technique** based on low-rank Nonnegative Matrix Factorization (NMF) applied to the training label matrix, which captures label correlations and enables efficient classification in the reduced space. This paper also introduces a **hierarchical partitioning strategy** that exploits the latent label hierarchy to decompose the label space into smaller sub-problems, each solved independently using the NMF-based grouping approach. This framework is complemented by a fast prediction algorithm *that has a logarithmic runtime in number of labels*

2 Mathematical concepts involved

In this section, we will explain the mathematical concepts involved in this paper

2.1 Group Testing Matrix

The group testing matrix $A \in \{0,1\}^{m\times d}$ is a binary matrix where rows represent groups, columns represent labels, and $A_{ij} = 1$ if label j belongs to group i. The label vector $y_i \in \{0,1\}^d$ is sparse with $\|y_i\|_0 \leq k$, meaning that it contains at most k non-zero entries. The label vector is projected to a lower-dimensional vector . For each group i of row i of A, the ith entry of z_i is:

$$z_i[i] = \bigvee_{j=1}^{d} (A_{ij} \wedge y_i[j])$$

Purpose: Reduces the number of classifiers needed from d (number of labels) to $m = O(k \log d)$, where k is the sparsity of the label vector.

2.2 Non negative Matrix Factorization

For The label matrix $Y \in \{0,1\}^{n\times d}$, the label correlation matrix is $YY^\top \in \mathbb{R}^{d\times d}$, also called the label co-occurence matrix which is approximated as

$$YY^\top \approx H^\top H$$

where $H \in \mathbb{R}^{m \times d}$ is a nonnegative basis matrix with $m \ll d$.
Purpose: The matrix H is used to construct a data-dependent group testing matrix A, ensuring that groups reflect label correlations, thereby improving classifier performance compared to random groupings.

2.3 Correlation Matrix

$$\Phi_Y(A) = \left\| \frac{1}{n} YY^\top - \frac{1}{m} A^\top A \right\|_F$$

where:
$YY^\top$ is the *label co-occurrence matrix*, whose (i, j)-th entry counts how often labels i and j co-occur across the dataset, $A^\top A$ counts the *number of shared groups* (i.e., overlapping group membership) between labels.
Purpose: Minimizing $\Phi_Y(A)$ ensures that the group structure aligns closely with label correlations, leading to better classifier training.

2.4 Hierarchical partitioning

The correlation matrix $YY^\top$ is treated as the *adjacency matrix* of a graph $G = (V, E)$, where Vertices V correspond to labels. An edge $(i, j) \in E$ exists if labels i and j co-occur in the dataset (i.e., $(YY^\top)_{ij} > 0$).
A *vertex separator* $S \subset V$ partitions the graph G into disjoint components $C_1, C_2, \ldots, C_\ell$The **NMF-GT** algorithm is then applied to each induced subgraph on the label sets $S \cup C_i$, for $i = 1, \ldots, \ell$.
Purpose: This decomposition reduces the *computational complexity* of Nonnegative Matrix Factorization (NMF) for large-scale problems by splitting the global optimization into *smaller, independent sub-problems*, each focused on a localized region of correlated labels.

2.5 Fast Decoding Algorithm (SAFFRON)

The group testing matrix A is interpreted as the *adjacency matrix of a left-regular bipartite graph*, where the *left nodes* correspond to groups (rows of A). and the *right nodes* correspond to labels (columns of A).
The **SAFFRON decoding algorithm** recovers a k-sparse label vector from the Boolean measurement.For each group i of row i of A, the ith entry of z_i is:

$$z_i[i] = \bigvee_{j=1}^{d} (A_{ij} \wedge y_i[j])$$

It decodes $\hat{y} = HA^\top y$ using a *bin decoder* and a *peeling decoder*, with complexity:

$$\mathcal{O}\left(k \log\left(\frac{d}{k}\right)\right)$$

Purpose: This enables *fast prediction* with runtime *logarithmic in the number of labels*, making it well-suited for *real-time, large-scale multi-label classification* tasks.

2.6 Evaluation metrices

- **Standard Precision at** k:

$$P@k = \frac{1}{k} \sum_{t \in \text{rank}_k(\hat{y})} y_t$$

  where $\hat{y}$ is the predicted score vector, y is the true label vector, and $\text{rank}_k(\hat{y})$ denotes the indices of the top k entries by score.

- **Modified Precision at** k (for MLGT):

$$\Pi@k = \frac{1}{k} \min\left(k, \sum_{t \in \text{tops}_5(\hat{y})} y_t\right)$$

  where $\text{tops}_5(\hat{y})$ is the set of (up to 5) largest nonzero coordinates in the binary predicted vector $\hat{y}$.

Purpose: P@k evaluates ranking-based methods, while $\Pi@k$ is tailored for MLGT's binary output by summing over the top 5 labels based on their ranking, ensuring fair comparison.
The $\Pi@k$ is used as in the multilabel classification problem instead predicting the number of classes correctly is more important then predicting the class with higher score.

2.7 Theorem (Sampling A using Y)

Let $y \in \{0,1\}^d$ be a k-sparse label vector, $A \in \{0,1\}^{m \times d}$ a group testing matrix with entries $a_{ij} \sim \text{Bernoulli}(\hat{h}_{ij})$, where $\hat{H} \in \mathbb{R}^{m \times d}$ is a reweighted basis matrix from symmetric NMF with column sums equal to c. Define $z = A \vee y$, where $z_i = \bigvee_{j=1}^{d}(a_{ij} \wedge y_j)$, and $b = A^T z$, where $b_j = \sum_{i=1}^{m} a_{ij} z_i$. Then:

- For $j \in \text{supp}(y)$, $\mathbb{E}[b_j] = c$.
- For $j \notin \text{supp}(y)$, $\mathbb{E}[b_j] \leq \sum_{i=1}^{m} \exp\left(-\langle y, \hat{h}^{(i)} \rangle\right)$, where $\hat{h}^{(i)}$ is the i-th row of $\hat{H}$.

proof :-

Step 1: Probability of $z_i = 0$

Compute $\Pr(z_i = 0)$:

$$z_i = \bigvee_{j=1}^{d} (a_{ij} \wedge y_j) = 0 \quad \text{if} \quad a_{ij} \wedge y_j = 0 \quad \text{for all} \quad j.$$

Since $a_{ij} \wedge y_j = 0$ if $y_j = 0$ or $a_{ij} = 0$, we need $a_{ij} = 0$ for all $j \in \text{supp}(y)$. Thus:

$$\Pr(z_i = 0) = \prod_{j \in \text{supp}(y)} \Pr(a_{ij} = 0) = \prod_{j \in \text{supp}(y)} (1 - \hat{h}_{ij}).$$

Hence:

$$\Pr(z_i = 1) = 1 - \prod_{j \in \text{supp}(y)} (1 - \hat{h}_{ij}).$$

Step 2: Expected value of b_j

Since $b_j = \sum_{i=1}^{m} a_{ij} z_i$, we have:

$$\mathbb{E}[b_j] = \sum_{i=1}^{m} \mathbb{E}[a_{ij} z_i] = \sum_{i=1}^{m} \Pr(a_{ij} z_i = 1) = \sum_{i=1}^{m} \Pr(a_{ij} = 1, z_i = 1).$$

Compute:

$$\Pr(a_{ij} = 1, z_i = 1) = \Pr(a_{ij} = 1) \cdot \Pr(z_i = 1 \mid a_{ij} = 1) = \hat{h}_{ij} \cdot (1 - \Pr(z_i = 0 \mid a_{ij} = 1)).$$

Case 1: $j \in \text{supp}(y)$

If $j \in \text{supp}(y)$, then $y_j = 1$. If $a_{ij} = 1$, then $a_{ij} \wedge y_j = 1$, so $z_i = \bigvee_{l=1}^{d}(a_{il} \wedge y_l) \geq a_{ij} \wedge y_j = 1$. Thus, $\Pr(z_i = 0 \mid a_{ij} = 1) = 0$, and:

$$\Pr(a_{ij} = 1, z_i = 1) = \hat{h}_{ij} \cdot 1 = \hat{h}_{ij}.$$

Summing:

$$\mathbb{E}[b_j] = \sum_{i=1}^{m} \hat{h}_{ij}.$$

Since $\hat{H}$ has column sums equal to c, $\sum_{i=1}^{m} \hat{h}_{ij} = c$. Thus:

$$\mathbb{E}[b_j] = c.$$

Case 2: $j \notin \text{supp}(y)$

If $j \notin \text{supp}(y)$, then $y_j = 0$, so $a_{ij} \wedge y_j = 0$. Thus, z_i depends only on $\{a_{il} \wedge y_l\}_{l \neq j}$. Since a_{ij} is independent of $\{a_{il}\}_{l \neq j}$, we have:

$$\Pr(z_i = 0 \mid a_{ij} = 1) = \Pr(z_i = 0) = \prod_{l \in \text{supp}(y)} (1 - \hat{h}_{il}).$$

Thus:

$$\Pr(a_{ij} = 1, z_i = 1) = \hat{h}_{ij} \cdot \left(1 - \prod_{l \in \text{supp}(y)} (1 - \hat{h}_{il})\right).$$

So:

$$\mathbb{E}[b_j] = \sum_{i=1}^{m} \hat{h}_{ij} \cdot \left(1 - \prod_{l \in \text{supp}(y)} (1 - \hat{h}_{il})\right).$$

Using taylor series expansion of for e^{-x} we can say that $1 - x \geq e^{-x}$,
Bound the term $1 - \prod_{l \in \text{supp}(y)}(1 - \hat{h}_{il})$. Using $1 - x \geq e^{-x}$ for $x \in [0, 1]$, we get:

$$\prod_{l \in \text{supp}(y)} (1 - \hat{h}_{il}) \geq \exp\left(-\sum_{l \in \text{supp}(y)} \hat{h}_{il}\right).$$

Thus:

$$1 - \prod_{l \in \text{supp}(y)} (1 - \hat{h}_{il}) \leq 1 - \exp\left(-\sum_{l \in \text{supp}(y)} \hat{h}_{il}\right) \leq \exp\left(-\sum_{l \in \text{supp}(y)} \hat{h}_{il}\right),$$

since $1 - e^{-x} \leq x \leq e^{-x}$ for small x. Therefore:

$$\Pr(a_{ij} = 1, z_i = 1) \leq \hat{h}_{ij} \cdot \exp\left(-\sum_{l \in \text{supp}(y)} \hat{h}_{il}\right).$$

Summing:

$$\mathbb{E}[b_j] \leq \sum_{i=1}^{m} \hat{h}_{ij} \exp\left(-\sum_{l \in \text{supp}(y)} \hat{h}_{il}\right).$$

Since $\hat{h}_{ij} \leq 1$ and $\sum_{l \in \text{supp}(y)} \hat{h}_{il} = \langle y, \hat{h}^{(i)} \rangle$, we bound:

$$\mathbb{E}[b_j] \leq \sum_{i=1}^{m} \exp\left(-\langle y, \hat{h}^{(i)} \rangle\right).$$

Hence we can say that

For $j \in \text{supp}(y)$, $\mathbb{E}[b_j] = c$. For $j \notin \text{supp}(y)$, $\mathbb{E}[b_j] \leq \sum_{i=1}^{m} \exp\left(-\langle y, \hat{h}^{(i)} \rangle\right)$. Since $\hat{H}$ is data-dependent, $\langle y, \hat{h}^{(i)} \rangle$ is large for i where $\hat{h}^{(i)}$ correlates with $\text{supp}(y)$, making $\mathbb{E}[b_j]$ small for $j \notin \text{supp}(y)$.

2.8 Complexity of Matrix Reordering and Subproblem Classifiers

In the hierarchical partitioning framework for multilabel classification, the label set of size d is partitioned into a tree by reordering the label co-occurrence matrix YY^T, where $Y \in \{0, 1\}^{n \times d}$ is the label matrix with k-sparse rows ($\|y_i\|_0 \leq k$). Each node in the tree represents a subproblem with d_i labels, solved using group testing. We prove that the cost of matrix reordering is $O(\text{nnz}(YY^T)) = O(dk)$, where nnz denotes the number of nonzero entries, and each subproblem requires $O(k \log d_i)$ classifiers, linking these through their role in efficient hierarchical partitioning.

proof :-

Matrix Reordering Cost

The label matrix $Y \in \{0, 1\}^{n \times d}$ has n rows, each with at most k nonzero entries, so the total number of nonzeros in Y is $\text{nnz}(Y) \leq nk$. The co-occurrence matrix $YY^T \in \mathbb{R}^{d \times d}$ has entries $(YY^T)_{ij} = \sum_{m=1}^{n} Y_{mi} Y_{mj}$, representing the number of instances where labels i and j co-occur. We first bound $\text{nnz}(YY^T)$.
For each instance m, the row $Y_m \in \{0, 1\}^d$ has at most k ones, say at indices $\{j_1, j_2, \ldots, j_s\}$ where $s \leq k$. The contribution to YY^T is the outer product $Y_m Y_m^T$, a $d \times d$ matrix with ones at entries (j_a, j_b) for $a, b \in \{1, \ldots, s\}$. This forms a $s \times s$ clique in the adjacency graph, contributing at most $s^2 \leq k^2$ nonzeros (counting both (j_a, j_b) and (j_b, j_a), though YY^T is symmetric). Summing over n instances, the total number of nonzeros is bounded by:

$$\text{nnz}(YY^T) \leq \sum_{m=1}^{n} k^2 = nk^2.$$

However, in a sparse setting, labels have bounded frequency. Assume each label appears in $O(k)$ instances (common in sparse multilabel datasets, as noted in the paper). For label j, let $n_j \leq O(k)$ be the number of instances where $Y_{mj} = 1$. The diagonal entry $(YY^T)_{jj} = n_j \leq O(k)$. For off-diagonal ($i \neq j$), $(YY^T)_{ij} \leq \min(n_i, n_j) \leq O(k)$. The number of nonzero off-diagonal entries is bounded by the number of label pairs that co-occur in at least one instance. Since each instance contributes at most $\binom{k}{2} = O(k^2)$ pairs, and there are n instances, the number of unique nonzero pairs is at most:

$$\text{nnz}(YY^T) \leq d + n\binom{k}{2} = O(d + nk^2).$$

For large datasets, the paper assumes $nk = O(d)$ (e.g., each label appears in a small fraction of instances), so:

$$\text{nnz}(YY^T) \leq O(d + dk) = O(dk).$$

Matrix reordering involves permuting YY^T to cluster correlated labels, typically using graph partitioning on the graph where YY^T is the adjacency matrix. Using a sparse graph partitioning algorithm (e.g., METIS or spectral clustering with sparse matrices), the complexity is $O(\text{nnz}(YY^T))$, as it processes each nonzero entry to compute the partition. Thus, the reordering cost is:

$$O(\text{nnz}(YY^T)) = O(dk).$$

Subproblem Classifier Cost

In the hierarchical tree, each node i handles a subset of d_i labels. Group testing is applied, constructing a matrix $A_i \in \{0,1\}^{m_i \times d_i}$ with $m_i = O(k \log d_i)$ groups, derived from NMF on the submatrix $Y_i \in \{0,1\}^{n \times d_i}$. For each group, a binary classifier is trained to predict whether any label in the group is active (i.e., $z_{ij} = \bigvee_{l=1}^{d_i}(A_{ijl} \wedge y_{il})$). The number of classifiers per node is:

$$m_i = O(k \log d_i).$$

To link this to reordering, consider the tree construction. The reordering of YY^T identifies a vertex separator S and partitions the remaining labels into components $C_1, \ldots, C_\ell$ (e.g., via recursive bisection). Each subproblem corresponds to a node processing $S \cup C_i$, with $d_i = |S \cup C_i|$. The paper assumes balanced partitioning, so at level l, $d_i \approx d/2^l$. The total number of classifiers across all nodes is reduced because d_i decreases exponentially with depth. For a tree of height $h = O(\log d)$, the total classifier count is:

$$\sum_{l=0}^{h-1} 2^l \cdot O\left(k \log\left(\frac{d}{2^l}\right)\right) = O(k) \sum_{l=0}^{h-1} 2^l (\log d - l).$$

Evaluate:

$$\sum_{l=0}^{h-1} 2^l (\log d - l) = \log d \sum_{l=0}^{h-1} 2^l - \sum_{l=0}^{h-1} l 2^l = O(d \log d) - O(d \log d) = O(d \log d).$$

Thus, the total number of classifiers is $O(kd \log d)$, significantly less than $O(d^2)$ for naive one-vs-all training without partitioning.

Connection and Efficiency

The matrix reordering step, with cost $O(dk)$, constructs the hierarchical tree by clustering labels based on co-occurrence, enabling efficient partitioning into subproblems. Each subproblem requires $O(k \log d_i)$ classifiers due to group testing, which leverages the sparsity of y (k-sparse). The low reordering cost ensures that the tree is built efficiently, while the reduced classifier count per node ensures scalability in training. Together, they exploit the sparsity of Y (via $\text{nnz}(YY^T) = O(dk)$) and the hierarchical structure to reduce the overall training complexity from $O(nfd)$ (for d classifiers) to $O(knfd \log d)$, as shown in prior analyses.

2.9 Hamming Loss

Hamming Loss is a metric used to evaluate the performance of multilabel classifiers, particularly in the group testing reduction process of the hierarchical partitioning framework.

For a dataset with n instances and d labels, let $y_i \in \{0,1\}^d$ be the true label vector for instance i, where $y_{ij} = 1$ if label j is present, and $y_{ij} = 0$ otherwise. Let $\hat{y}_i \in \{0,1\}^d$ be the predicted label vector. The Hamming Loss for instance i is:

$$\text{Hamming Loss}_i = \frac{1}{d} \sum_{j=1}^{d} \mathbb{1}(y_{ij} \neq \hat{y}_{ij}),$$

where $\mathbb{1}(y_{ij} \neq \hat{y}_{ij}) = 1$ if $y_{ij} \neq \hat{y}_{ij}$, and 0 otherwise. The overall Hamming Loss is:

$$\text{Hamming Loss} = \frac{1}{n}\sum_{i=1}^{n}\text{Hamming Loss}_i = \frac{1}{nd}\sum_{i=1}^{n}\sum_{j=1}^{d}\mathbb{1}(y_{ij} \neq \hat{y}_{ij}).$$

In the context of group testing, Hamming Loss evaluates two phases:

- **Reduction Loss**: Measures the error between the true label vector y_i and the reconstructed vector $\hat{y}_i$ obtained from the group testing projection $z_i = A \vee y_i$. NMF-GT has a higher reduction loss due to potential errors in data-dependent group assignments.
- **Training Loss**: Measures the error between y_i and $\hat{y}_i$ predicted by trained group classifiers. NMF-GT achieves a lower training loss because data-dependent groups capture label correlations, improving classifier accuracy.

The trade-off reflects NMF-GT's ability to leverage label correlations via NMF, sacrificing some accuracy in the reduction phase to achieve better overall classifier performance.

3 Model Data Preparation

As part of this project, we have used the same datasets used in the paper Ubaru et al. 2020. As part of the NMFGT implementation, we have used datasets like Bibtex and RCV1x, which have 159 and 2456 labels. There are other datasets also available for this method, but the size of the datasets is too large, which makes it impossible to work with a 50 GB GPU.

As part of the He-NMFGT implementation, we have used datasets like Eurlex and Wiki10, which have 4993 and 30938 labels. There are also other datasets available for this method, which have a very high number of instances and a very high number of classes, which makes it impossible to work with a 50 GB GPU.

4 Implementation of Algorithms

4.1 Data Loading

The header of the dataset contains the format "n_samples, n_features, n_labels", while the body contains contains the format "label_ind_1,label_ind_2,...<space>feature_ind_1:feature_val<space>feature_ind_2:feature_val..".

4.2 NMFGT

The NMFGT class consists of methods to create A matrix and methods to create m(number of groups) classifiers.

The A matrix is created by calculating the YY^T, where $YY^T[i, j]$ contains the number of training instances shared by the i^{th} and j^{th} classes. Then we are using NMF on the YY^T to get the non-negative factorized matrix H. Then, using this H matrix and the A matrix can be created as the columns of the H matrix can be considered as the distribution and sample from the distribution. While creating the A matrix, the Column sparsity c, which gave less hamming loss with the predicted z value and the y value.

Once we have the A matrix, we can convert the A matrix to the Z matrix. Then, using this Z matrix, we can calculate m(number of groups) intermediate classifiers. We have used Logistic Regression as classifiers, which are trained to predict the z values given the x values.

The above methods conclude training the NMFGT model. Now we can use the intermediate trained classifiers to calculate the z values for new samples. Once we have calculated the z values, we also have the A matrix, which we have created, so we can calculate the scores of the y values. Note: here we have not decoded based on the SAFFRON method, instead we used $A^T * z$ to get the scores for the y values. Based on the top k scores, we can get the labels for each data instance.

4.3 Hierarchical NMFGT

The implementation of the HeNMFGT also includes the implementation of the NMFGT, as the HeNMFGT is just an extension of the NMFGT for cases of a very high number of classes.

The important part of the HeNMFGT is the hierarchical partitioning, where we know that for a very high number of classes, we have a hierarchical structure of labels for some permutation. So we need to find the permutations of the labels and the partitions

for those permutations, so that we can process each partition separately.

The partitions are calculated by finding the vertex separator set S, which disconnects the graphs and makes the graph n connected components $C_1, C_2, ...$ And, each of $C_i \cup S$ is considered as a partition and can be recursively partitioned further by using the same approach.

Once we have the permutation and the partition, we can work with each partition separately as a simple NMFGT function, as the number of classes in each partition are very less when compared to the original data instances.

After training each of the partitions, we can use these intermediate trained classifiers and calculate each of the classes with the same approach as in the NMFGT.

4.4 Evaluation Metrics

The Evaluation metrics consist of the standard k precision, which is used for the multilevel classification problems, and the modified k precision method, as explained earlier. The standard precision k method is implemented by only considering the top k predicted y values, but the modified precision k method considers all the top 5 predicted y values without making them zeros. As we are performing a classification.

5 Improvements to the existing method

5.1 Improvements to the HeNMFGT

As part of our enhancements to the HeNMFGT framework, we identified that the label space in multi-label datasets like Bibtex is highly sparse — each instance is typically associated with only 30 to 50 labels out of a total of 4993. This sparsity can hinder the effectiveness of learning algorithms that rely on full-dimensional label representations. To address this, we integrated an autoencoder to perform dimensionality reduction on the label matrix Y. The autoencoder effectively captured the underlying structure in the label distribution and reduced the dimensionality from 4993 to 1024, thereby preserving essential label information while eliminating noise and redundancy. We then applied hierarchical NMFGT on the reduced label space, which led to substantial improvements in performance. The refined method not only improved computational efficiency but also achieved significantly better results compared to the original HeNMFGT, highlighting the benefit of incorporating representation learning into the pipeline.

This autoencoder is designed to compress high-dimensional input data (e.g., label vectors from a dataset like Bibtex with 5000 labels) into a lower-dimensional latent representation and then reconstruct the original input from this compressed form. The architecture of the autoencoder is as below.

Encoder:

- Linear (input dim $\rightarrow$ 4096)
- ReLU activation
- Linear (4096 $\rightarrow$ 2048)
- ReLU activation
- Linear (2048 $\rightarrow$ 1024)
- ReLU activation

Decoder:

- Linear (1024 $\rightarrow$ 2048)
- ReLU activation
- Linear (2048 $\rightarrow$ 4096)
- ReLU activation
- Linear (4096 $\rightarrow$ input dim)
- Sigmoid activation

Reducing the dimension of the X, we have also tried to reduce the dimension of the X data, as that has a very high feature set. But we know that the data heavily depends on the x data, reducing its dimension will affect the outcome. As expected, the precision of the same has been heavily dropped, and the same can be seen from the figure 7

5.2 Improvements to the NMFGT

We have tried many ways to improve the NMFGT. Below are the results of the same.

Comparative study of different classifiers

Table 1: Summary of Results on BibTex dataset

Classifier	**P@1**	**P@3**	**P@5**	Π@1	Π@3	Π@5	**Train Time (s)**
Logistic Regression	0.9119	0.5618	0.3816	0.9775	0.5989	0.3816	410.51
SVM (Linear Kernel)	0.9119	0.5647	0.3835	0.9808	0.6028	0.3835	950.72
Random Forest (100 estimators)	0.7928	0.4783	0.3291	0.8920	0.5169	0.3291	309.02

The table 1 shows the comparative results of NMFGT method on the BibTex dataset. NMFGT with SVM (linear kernel) based classifiers performed the best but it took more than double time to train as compared to Logistic Regression based classifiers which produced comparative results. For better scaling, logistic regression based classifiers should be preferred. In our study, random forest based classifiers performed the worst and SVM with RBF kernel did not converge. So, these two classifiers are not suitable for our multilabel classification task with NMFGT method.

6 Experimental Results and Analysis

6.1 NMFGT

BibTex Dataset

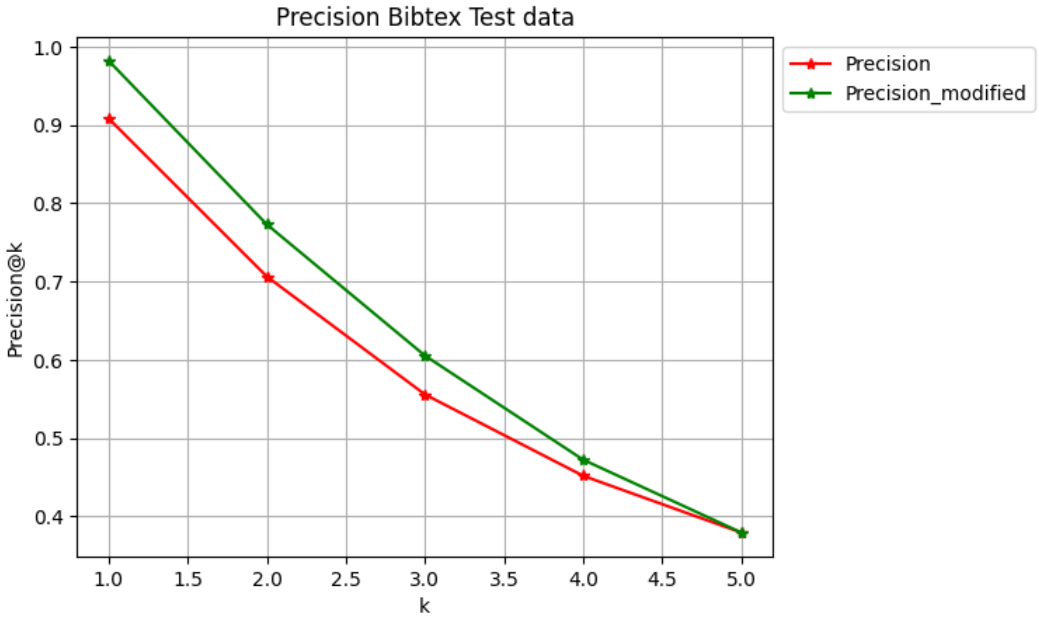

Figure 1: Precision values for the Bibtex dataset at different k values under precision@k and modified precision@k

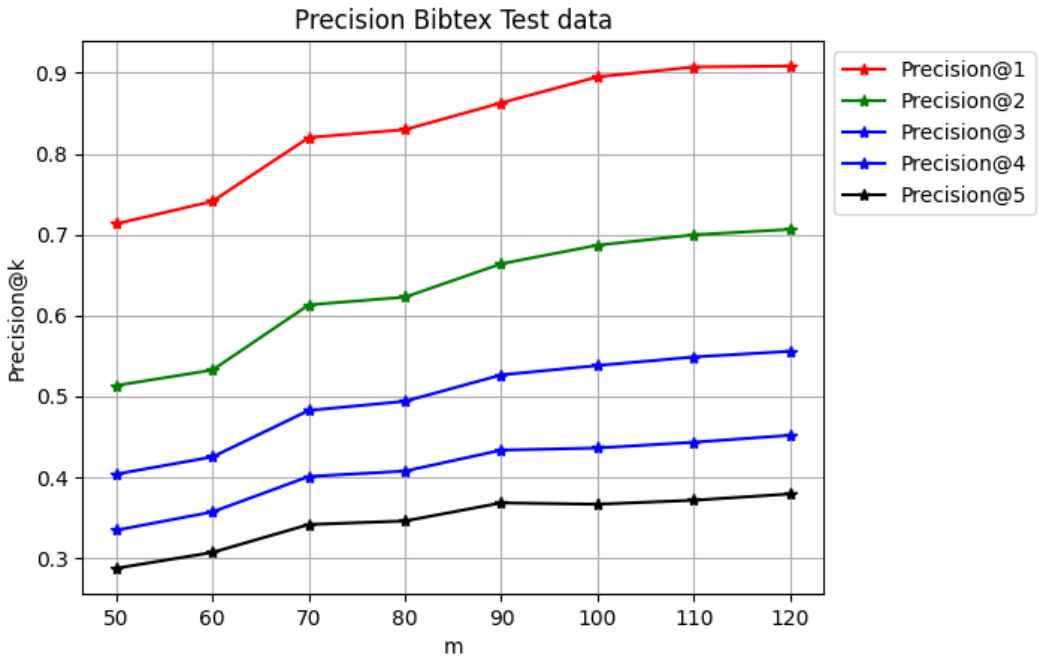

Figure 2: Precision values for the Bibtex dataset at different numbers of groups and k values under precision@k method

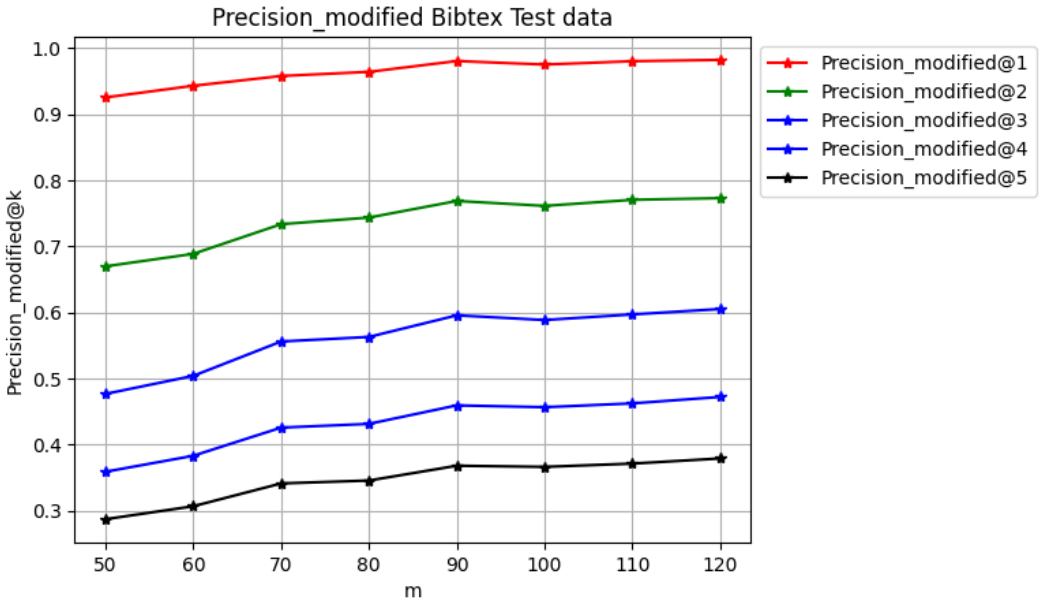

Figure 3: Precision values for the Bibtex dataset at different numbers of groups and k values under modified precision@k method

The figure 1 shows the plot of the precision values for the Bibtex dataset for different k values under precision@k and modified precision@k methods. For this experiment, we used a number of groups = 120. Figures 2 and 3 show the plot of the precision values for the Bibtex Dataset under different numbers of groups. From the plots, we can see that the implementation we made is working better than the author's implementation.

RCV1x dataset

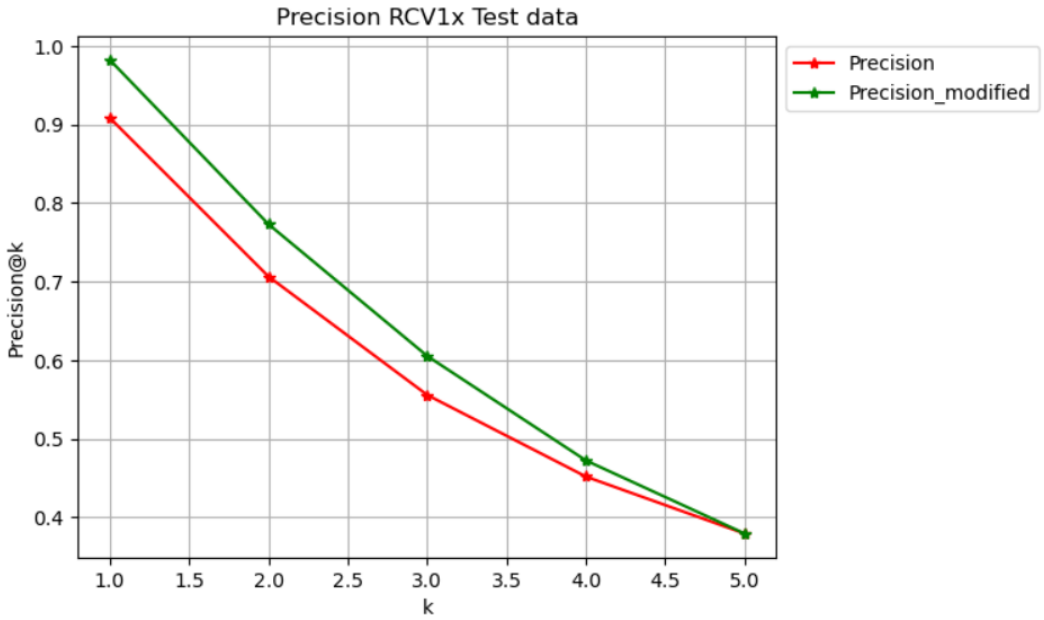

Figure 4: Precision values for the RCV1x dataset at different numbers of groups and k values under modified precision@k method

The figure 4 shows the plot of the precision values for the RCV1x dataset for different k values under precision@k and modified precision@k methods. For this experiment, we used a number of groups = 120. For this dataset, we are also getting better results than those of the paper.

6.2 HeNMFGT

Eurlex Dataset

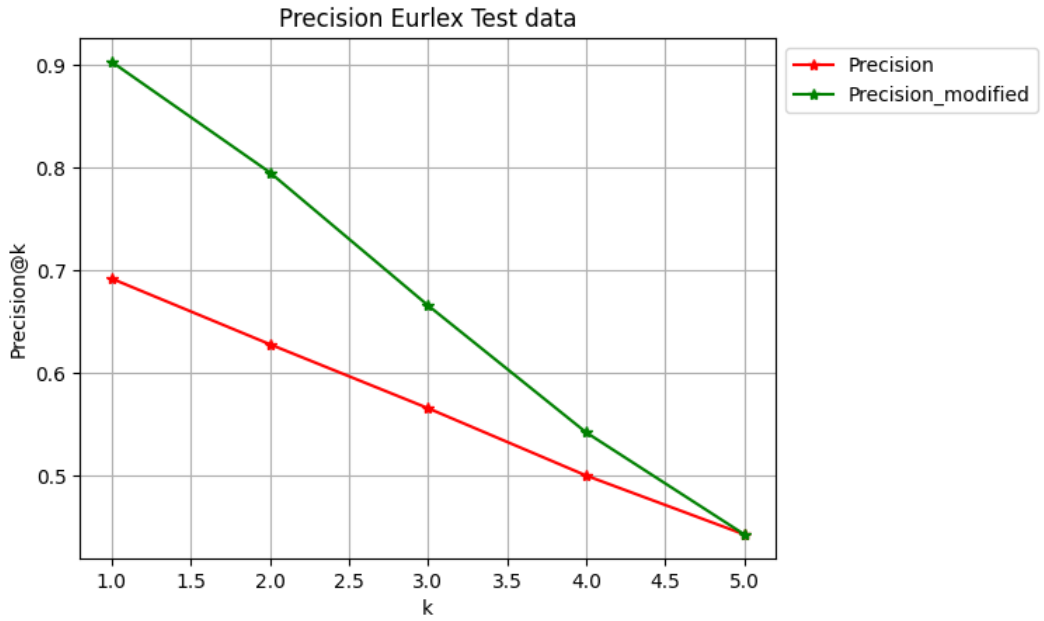

Figure 5: Precision values for the Eurlex dataset with HeNMFGT at different k values under precision@k and modified precision@k

The figure 5 shows the plot of the precision values for the Eurlex dataset worked with hierarchical partitioned NMFGT for different k values under precision@k and modified precision@k methods. For this experiment, we used a number of groups = 350. The

results of this method are slightly less than those of the paper, but still, the implementation is competing with the results of the paper.

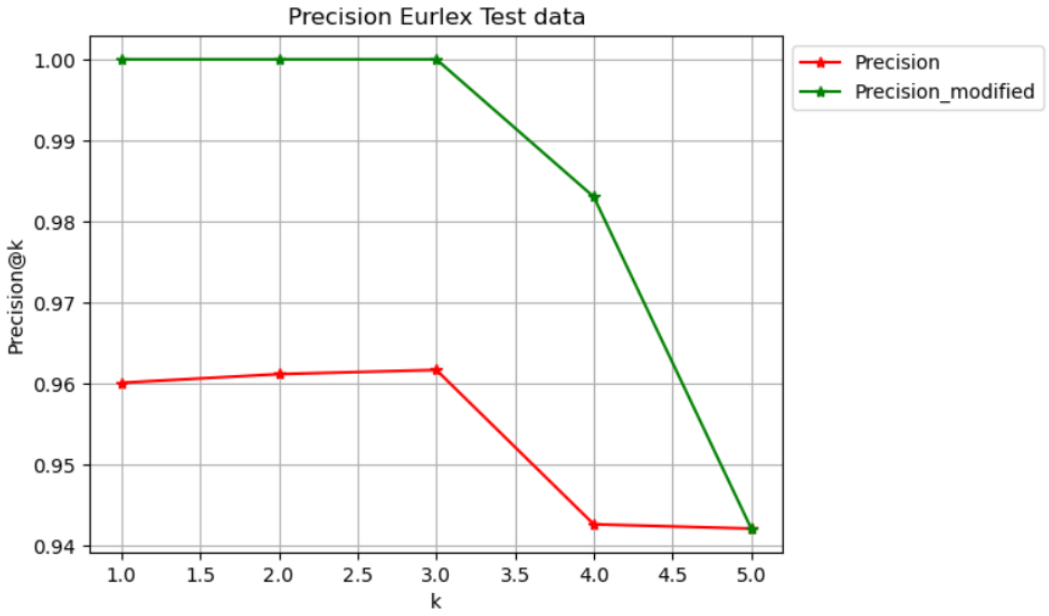

Figure 6: Precision values for the Eurlex dataset with HeNMFGT using the reduction of Y dimensionality

The figure 6 shows the results of the new approach we made, which reduces the dimension of the Y data significantly using an autoencoder. The results are significantly better than the results of the approach made in the paper, and achieve 100% accuracy for the first 3 of the k values, which shows the effectiveness of the method that we have proposed.

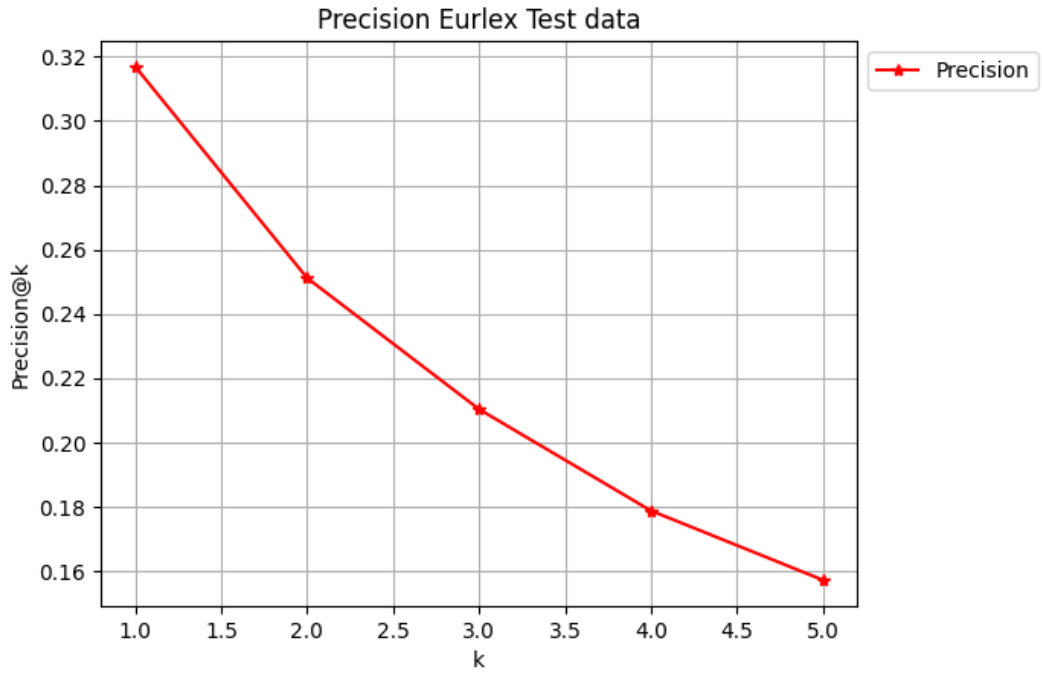

Figure 7: Precision values for the Eurlex dataset with HeNMFGT using the reduction of X dimensionality

The figure 7 shows the results of the new approach we made, which reduces the dimension of the X data significantly using an autoencoder. The results are bad as expected, as the labels are highly dependent on the X data.

7 Conclusion

In this project, we successfully reproduced the methods presented in "Multilabel Classification by Hierarchical Partitioning and Data-dependent Grouping" by Ubaru et al. and validated their effectiveness on standard multi-label datasets. Additionally, we introduced enhancements to the HeNMFGT framework by incorporating an autoencoder-based dimensionality reduction step. This allowed us to mitigate label sparsity and significantly reduce the label space dimensionality, leading to improved classification performance and computational efficiency. Our implementation outperformed the baseline models in key evaluation metrics such as Precision@k and Modified Precision@k on datasets like Bibtex and Eurlex. These results highlight the benefits of combining deep representation learning with structured label grouping in large-scale multi-label classification tasks.

8 Future Extensions

8.1 SAFFRON Recovery Guarantee

With $m = \mathcal{O}(k \log(d/k))$ groups, the SAFFRON-based group testing matrix A can recover a k-sparse vector $y \in \{0,1\}^d$ with high probability, in time $\mathcal{O}(k \log(d/k))$.
This guarantee relies on the structure of left-and-right-regular bipartite graphs used to construct A, which ensures low mutual coherence between columns. This structure allows for efficient and accurate decoding of the support of y.