MLBD Final Project Report

Hriday Kondru (B20CS021)
kondru.1@iitj.ac.in
Indian Institute of Technology
Jodhpur, Rajasthan, India

Ruthvik K (B20AI037)
ruthvik.2@iitj.ac.in
Indian Institute of Technology
Jodhpur, Rajasthan, India

ABSTRACT

Extreme multilabel classification (XMLC) is a classification task where each instance has to be assigned to a subset of a large number of possible labels, which is prevalent in various applications such as text classification and recommendation systems. This project aims to tackle the challenging problem of XMLC on the LF-AmazonTitles-131K BoW dataset.

KEYWORDS

XML Classification, FastXML, Dimensionality Reduction, BoW Dataset

ACM Reference Format:
Hriday Kondru (B20CS021) and Ruthvik K (B20AI037). 2023. MLBD Final Project Report. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Extreme multi-label classification (XMLC) is a type of classification problem where each instance, such as a text document or a user's past behavior on a website, needs to be assigned to a subset of a very large number of possible labels. This is different from traditional classification problems where each instance is assigned to a single label or multi-label classification problems where each instance needs to be assigned to a subset of a small number of possible labels. In XMLC, the number of possible labels can be very large, ranging from millions to billions.

The task of XMLC is challenging because the label space is very high-dimensional, meaning that there are a large number of possible labels that an instance can be assigned to. Additionally, the label assignments tend to be very sparse, meaning that for a given instance, only a small subset of labels are relevant. For example, a news article might be relevant to only a few topics out of a large number of possible topics. Finally, processing XMLC data requires a lot of computational resources, which can be a bottleneck for some applications.

XMLC is often used in applications like text classification and recommendation systems. In text classification, an article or document needs to be assigned to a set of relevant topics or categories.

The project involved performing XMLC on the LF-AmazonTitles-131K BoW dataset using techniques such as FastXML and dimensionality reduction. A comparison study was conducted to evaluate the results obtained using different techniques, which will be discussed in the report.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RUNNING THE CODE

2.1 Project Files

- **Click to Download Project Folder**
- **Click to view Poster**

This folder contains all the files necessary to run the project on Google Colab including the B20CS021_B20AI037_Project.ipynb notebook the train.txt and test.txt files.

2.2 Setting up the project

(1) Download and unzip the .zip project folder
(2) Upload the B20CS021_B20AI037_Project.ipynb notebook to any folder in Google Drive
(3) Upload the train and test .txt files to the same folder
(4) Open the .ipynb notebook in Google Colab
(5) Update the path of the train and test .txt files in the "Loading train and test data" section of the notebook to the path where the train and test .txt files were uploaded
(6) Connect to a run-time and execute all the cells

3 DATA SET EXPLANATION

3.1 Labels

Upon examining the `label_raw_ids.txt` file, it becomes apparent that the labels are assigned a numerical value ranging from 0 to 131072, indicating that there are a total of 131073 labels.

3.2 Train and Test Data

The Test and Train csv files for the BoW dataset have a specific format, with the first line indicating the number of Train/Test points, the BoW Feature Dimensionality, and the number of Labels. Starting from the second line, the data is listed in the following format: the labels for each point (separated by commas), followed by the feature indices and corresponding values (separated by colons and spaces).

3.3 Data Characteristics

- Number of train points: 294805
- Number of test points: 134835
- Number of features: 40000
- Number of labels: 131073

The number of features in the data-set is quite large (40000), which may have an effect on the time required for training and predictions. To mitigate this issue, the project will examine the concept of dimensionality reduction, which will be discussed in greater detail later on.

4 APPROACH FOLLOWED

4.1 Loading the data

Since the data is present in a sparse format in the .txt files as discussed in Section 3.2, a function `convert_to_csr` was defined. It loops through each row of the data, extracts the labels and feature index-value pairs, and sets the corresponding entries in LIL (List of Lists) matrices, which are more efficient for indexing. Finally, the function returns the CSR matrices which are obtained by converting the LIL matrices.

4.2 Evaluation Metrics

4.2.1 ***Precision@k.***

$$P@k := \frac{1}{k} \sum_{l \in rank_k(\hat{y})} y_l$$

$\hat{y}$ is the predicted vector of relevance scores, $\text{rank}_k(\hat{y})$ is the set of indices of the top k elements of $\hat{y}$ in descending order, and y_l is the binary indicator of whether item l is relevant or not. [6]

This was implemented in the function `precision_at_k` which iterates over each pair of true and predicted labels, and for each pair, it calculates the number of correctly predicted labels in the top-k predicted labels, and updates the precision score for each value of k. The precision_at_k function in the Omikuji [5] repository was used as a reference

Finally, the function calculates the average precision across all values of k and returns it as the output.

4.2.2 ***Hamming Loss.*** In multi-label classifications, the hamming loss is defined as the fraction of the labels that are incorrectly predicted, i.e., the fraction of the labels that are either predicted but not present in the true labels or present in the true labels but not predicted.[1]

4.3 FastXML

FastXML[3] is a fast and accurate algorithm for multi-label classification tasks, where each instance can be associated with multiple labels simultaneously.

FastXML uses a tree-based data structure to efficiently handle high-dimensional feature spaces and a scalable learning approach that can handle large datasets.

FastXML learns a hierarchy over the feature space rather than the label space, based on the observation that only a small number of labels are active in each region of the feature space. It uses this hierarchy to efficiently predict the set of relevant labels for a given input instance by traversing the feature space hierarchy and focusing exclusively on the set of labels active in the region where the instance belongs. This approach allows FastXML to handle very large label spaces and to achieve high accuracy while remaining computationally efficient.

The FastXML implementation by Refefer [4] was used in the code.

5 RESULTS AND ANALYSIS

5.1 NOTE

All the precision@k and Hamming loss results are compiled in separate tables, **Table 4**, **Table 5**, **Table 6**, **Table 7** where each table corresponds to a `n_trees` parameter value

5.2 FastXML on Original Data Set

FastXML models were trained on the original data-set with varying the `n_trees` parameter between values 1, 4, 8 and 16. The results are as given in **Table 1**.

Table 1: FastXML Results on Original Data

n_trees	Training Time	Prediction Time	Model Size
1	4 min	1 min	60 MB
4	19 min	2 min	238 MB
8	39 min	3 min	475 MB
16	1 hr	6 min	949 MB

5.3 FastXML on Dimensionally Reduced Data

5.3.1 ***Introduction.*** Singular Value Decomposition (SVD) is a mathematical technique for decomposing a matrix into three simpler matrices. Truncated SVD[2] is a variant of SVD that only keeps a subset of the most important information. It is often used for dimensionality reduction in large and sparse datasets.

Because the original data consists of 40000 features (dimensions), which is a quite large, dimensionality reduction was utilized to assess its impact on the results.

5.3.2 ***Reducing Dimensions to 100.*** The time taken for training the *FastXML* model in this case was considerably lesser than when trained on the original data. But, the precision drops by a large amount. Further, it was observed that the size of the models decreased when compared to the models trained on the original data.

Results are as shown in **Table 2**

Table 2: FastXML Results on Data with 100 features

n_trees	Training Time	Prediction Time	Model Size
1	2 min	30 sec	37 MB
4	8 min	30-40 sec	147 MB
8	17 min	50 sec	293 MB
16	35 min	1 min	585 MB

5.3.3 ***Reducing Dimensions to 200.*** Similar to the previous case with 100 dimensions (features), the time taken for training the *FastXML* model in this case was considerably lesser than when trained on the original data and, the precision drops by a large amount. Further, it was observed that unlike in the previous case, the size of the models actually increase when compared to the models trained on the original data.

Results are as shown in **Table 3**

Table 3: FastXML Results on Data with 200 features

n_trees	Training Time	Prediction Time	Model Size
1	2 min	30 sec	69 MB
4	9 min	40-50 sec	274 MB
8	19 min	1 min	594 MB
16	40 min	1 min	1.1 GB

5.3.4 ***Reducing Dimensions to 500.*** The time taken for training the *FastXML* model in this case was marginally lesser than when trained on the original data. But, the precision drops by a large amount. Further, it was observed that the size of the models increase considerably when compared to the models trained on the original data. Where the original model took 60 MB, this model took 165 MB which is more than double the size and the *precision@1* score dropped from 10.14% to 3.39% (in the case of n_trees=1)

5.3.5 ***Reducing Dimensions to 1000.*** Training the *FastXML* model with n_trees=1 took much longer (>30 min) than when trained on the original training data of 40000 features (4 min)

5.3.6 ***Reducing Dimensions to 10000.*** *TruncatedSVD fit* method on the training data exceeded the RAM limits of the Google Colab run-time which is 12 GB

5.4 Precision@k and Hamming Loss results

Results are as shown in **Table 4**, **Table 5**, **Table 6**, **Table 7**

Table 4: Precision@k (P@k) and Hamming Loss results with `n_trees=1`

Number of Features	P@1	P@3	P@5	Hamming Loss
40000	10.14%	8.4%	6.86%	0.0000231
200	2.47%	2.02%	1.55%	0.0000247
100	1.87%	1.6%	1.24%	0.0000249

Table 5: Precision@k (P@k) and Hamming Loss results with `n_trees=4`

Number of Features	P@1	P@3	P@5	Hamming Loss
40000	13.55%	11.35%	9.78%	0.0000156
200	2.92%	2.51%	2.14%	0.0000160
100	2.12%	1.89%	1.62%	0.0000160

5.5 Analysis

- Based on the given observations, it can be concluded that the original data had many 0 or empty values, making it "*sparse*". When training the *FastXML* model on this original data, the model's running time and size were significantly less than when using *TruncatedSVD* with either *n_components*=10000 or *n_components*=1000 to reduce the data's dimensionality. This is because *TruncatedSVD* causes the data to lose its sparsity, leading to an increase in data size and subsequently increasing the model's training time and size. The observations when *n_components* was set to 500 can be explained with the same reasons.
- When using *TruncatedSVD* with *n_components*=200 during the training of the *FastXML* model, the peak RAM usage and model size are higher compared to when the original data is used. This is due the same reason given in the previous point. The use of *TruncatedSVD* results in data becoming dense, causing an increase in data size and ultimately leading to an increase in the model's size. Moreover, the precision is significantly reduced, indicating that a substantial amount of information is lost when the dimensions are reduced.
- By applying *TruncatedSVD* with *n_components*=100 on the data during training, the training time and model size were reduced compared to using the original data. This is because, despite the loss of sparsity caused by *TruncatedSVD*, with *n_components*=100, the data size decreased enough to simplify the training process. However, the precision was significantly reduced, indicating that the simplified data also lost a considerable amount of information due to dimensional reduction.

Table 6: Precision@k (P@k) and Hamming Loss results with `n_trees=8`

Number of Features	P@1	P@3	P@5	Hamming Loss
40000	16.65%	13.84%	11.85%	0.0000155
200	3.31%	2.81%	2.39%	0.0000159
100	2.35%	2.09%	1.8%	0.0000160

Table 7: Precision@k (P@k) and Hamming Loss results with `n_trees=16`

Number of Features	P@1	P@3	P@5	Hamming Loss
40000	19.78%	16.5%	14.14%	0.0000155
200	3.64%	3.11%	2.65%	0.0000159
100	2.72%	2.35%	2.01%	0.0000160

6 METHODS ATTEMPTED

6.1 ECLARE

Extreme Classification with Label Graph Correlations The ECLARE method can be used on the LF-AmazonTitles-131K BoW dataset for organizing and managing the large collection of text documents in the dataset. The dataset contains over 131,000 titles from the Amazon online store, each represented as a bag-of-words (BoW) document.

Using the ECLARE method, the documents in the dataset can be classified and labeled based on their content, structure, and metadata. This would allow for easier management and retrieval of documents from the dataset.

For example, the ECLARE method could be used to classify the documents based on the categories or topics they belong to, such as electronics, books, or home and garden. Labels could then be assigned to each document based on its classification, allowing for efficient retrieval of documents belonging to a specific category.

6.2 GalaXC

While the GalaXC method is not specifically designed for managing the LF-AmazonTitles-131K BoW dataset, it is possible to apply the method to the dataset by treating each title in the dataset as an individual object and using the GalaXC method to classify and categorize the objects based on their properties.

Once the objects have been cross-identified, they can be classified based on their properties, such as the keywords or phrases used in the title, the length of the title, or the overall sentiment or tone of the title. This hierarchical classification scheme would allow for easy categorization of titles into different groups or classes.

6.3 DeepXML

The deep neural network used in the DeepXML method typically consists of several layers of interconnected nodes, including input, hidden, and output layers. The input layer takes the numerical features as input, and the hidden layers learn to extract high-level features and patterns from the data. The output layer produces a set of scores or probabilities for each label, indicating the likelihood that the document belongs to each label.

During the training process, the DeepXML method optimizes the weights and biases of the network to minimize the difference between the predicted labels and the ground-truth labels for a subset of the data. Once the network has been trained, it can be used to predict the labels of unseen documents in the LF-AmazonTitles-131K BoW dataset.

These methods attempted are also shown in the colab file shared but the only issue i faced was to after creating the environment in colab to activate the environment the code was not working for these methods. So finally i considered using the FastXML method which was working fine.

7 INCREASING PREDICTION ACCURACY FOR RARE LABELS

7.1 Class Weighting

The class weighting method used in this code implemented aims to address the issue of imbalanced class distribution in the training data, particularly for rare labels. In traditional classification algorithms, the objective function minimizes the overall error rate, which may lead to poor performance on the minority class. By assigning higher weights to rare labels during training, the algorithm will pay more attention to correctly predicting these labels, even if it means sacrificing some accuracy on the majority class.

The results obtained are show in Table 8, Table 9

Table 8: Results after performing Class Weighting

n_trees	Training Time	Prediction Time	Model Size
1	4 min	1 min	60 MB
4	20 min	2 min	240 MB
8	40 min	3 min	480 MB
16	1 hr	6 min	950 MB

Table 9: Precision@k (P@k) results with Class Weighting

n_tree	P@1	P@3	P@5
1	11.15%	9.23%	7.54%
4	14.90%	12.48%	10.75%
8	18.31%	15.22%	13.03%
16	21.75%	18.14%	15.55%

From the results we can see there is a slight increase of prediction accuracy after performing Class Weighting.

7.2 Label Smoothing

Label smoothing is a regularization technique used to prevent the model from being overconfident in its predictions. It smooths the label distribution by spreading out the probability mass from the ground-truth label to other labels. For this we implement convert_to_csr_with_label_smoothing() function which applies label smoothing to the training labels. Specifically, for each training label row, it converts the label to a one-hot encoded vector and then applies the smoothing factor α to each element of the vector, proportional to the sum of the elements in the vector.

By smoothing the label distribution in this way, the model is encouraged to assign some probability mass to rare labels, which would otherwise be assigned zero probability. This can help improve the prediction accuracy of rare labels, which are often difficult to predict accurately due to the scarcity of training examples. Results obtained are shown in Table 10,11

Table 10: Results after performing Label Smoothing

n_trees	Training Time	Prediction Time	Model Size
1	3 min	1 min	49 MB
4	15 min	2 min	224 MB
8	30 min	3 min	464 MB
16	45 min	5 min	920 MB

Table 11: Precision@k (P@k) results with Label Smoothing

n_tree	P@1	P@3	P@5
1	18.25%	15.11%	12.34%
4	24.84%	20.43%	17.60%
8	28.30%	23.53%	20.11%
16	33.54%	28.04%	24.03%

We can clearly see the considerable increase in the prediction accuracy.Label Smoothing provides a more fine-grained regularization method that takes into account the sparsity and distribution of label occurrences, which has better captured the complexity of our dataset.

7.3 Ensemble Learning

The Ensemble Learning method helps increase the prediction accuracy for rare labels we implement a code where we combine multiple weak classifiers (base classifiers) to form a strong classifier. In this case, a Random Forest Classifier is used as the base classifier. The OneVsRestClassifier wrapper is used to extend the binary classification algorithm to multi-class classification, the use of multiple weak classifiers helps to capture the rare label patterns in the data, making them less likely to be overlooked. Additionally, since the random forest classifier tends to handle imbalanced datasets well, this method is likely to be especially effective on datasets with rare labels.

The results obtained are shown below
P@1 -> 17.61%
P@3 -> 14.75%
P@5 -> 12.71%

7.4 Transfer Learning

The use of transfer learning with BERT in this code helps in improving the prediction accuracy for rare labels in our dataset in the following ways:

- Better handling of rare words and phrases:
  BERT is pre-trained on a large corpus of text data and has learned to encode the meaning of words and phrases in a way that captures their context in sentences. This enables BERT to handle rare words and phrases more effectively compared to traditional machine learning models, which may not have seen such rare words during training.
- Better handling of rare words and phrases:
  BERT is pre-trained on a large corpus of text data and has learned to encode the meaning of words and phrases in a way that captures their context in sentences. This enables BERT to handle rare words and phrases more effectively compared to traditional machine learning models, which may not have seen such rare words during training.
- Improved feature representation:
  By using BERT's pre-trained representations as input features to a downstream task, such as multi-label classification of product titles, the model can benefit from the rich semantic information captured by BERT. This can lead to more effective feature representations for the task, which can in turn improve prediction accuracy.
- Fine-tuning on the downstream task:
  The BERT model is further fine-tuned on our specific downstream task of multi-label classification of product titles. This helps the model to adapt to the specific characteristics of our dataset, such as the distribution of labels and the relationships between words and phrases in the titles. Fine-tuning also allows the model to adjust the weights of its parameters to better fit the data and improve prediction accuracy.

Overall, the use of transfer learning with BERT can help in improving prediction accuracy for rare labels in our dataset by enabling better handling of rare words and phrases, improving feature representations, and fine-tuning the model on the downstream task.
P@1 -> 34.21%
P@3 -> 33.51%
P@5 -> 30.14%

8 ANALYSIS

Based on the results obtained, the methods performed in the following order:

- Label Smoothing
- Transfer Learning
- Class Weighting
- Ensemble Learning

Label Smoothing works by modifying the loss function during training to reduce overconfidence in the model's predictions. This helps to prevent the model from making overly confident but incorrect predictions, which can be particularly important for rare labels. As a result, this method performed the best among the four methods tested.

Transfer Learning, on the other hand, works by leveraging pre-trained models, such as BERT, to improve the accuracy of the model on a new task. This can be particularly useful for tasks with limited training data, such as in the case of the LF-AmazonTitles 131K dataset. By leveraging the pre-trained knowledge of the BERT model, the Transfer Learning method was able to improve the accuracy of the model on the rare labels, resulting in the second-best performance among the four methods.

Class Weighting and Ensemble Learning both aim to address the issue of imbalanced data, but they were not as effective as the other two methods in this case. Class Weighting assigns higher weights to the minority class during training to increase their importance, while Ensemble Learning combines the predictions of multiple models to improve the accuracy. However, in this case, these methods did not provide as significant improvements in accuracy as Label Smoothing and Transfer Learning.

REFERENCES

[1] 2021. Model Evaluation: Quantifying the Quality of Predictions. https://scikit-learn.org/stable/modules/model_evaluation.html#hamming-loss.
[2] 2021. Truncated Singular Value Decomposition and Latent Semantic Analysis. https://scikit-learn.org/stable/modules/decomposition.html#truncated-singular-value-decomposition-and-latent-semantic-analysis.
[3] Yashoteja Prabhu and Manik Varma. 2014. 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* (New York, New York, USA) *(KDD '14)*. Association for Computing Machinery, New York, NY, USA, 263–272. https://doi.org/10.1145/2623330.2623651
[4] Refefer. 2021. FastXML: Extreme Multi-label Classification. https://github.com/Refefer/fastxml. GitHub repository.
[5] Tom Tong. 2018. Evaluation metrics for machine learning models. https://github.com/tomtung/omikuji/blob/master/src/model/eval.rs.
[6] Manik Varma. 2006. XML repository of multi-label datasets. http://manikvarma.org/downloads/XC/XMLRepository.html.