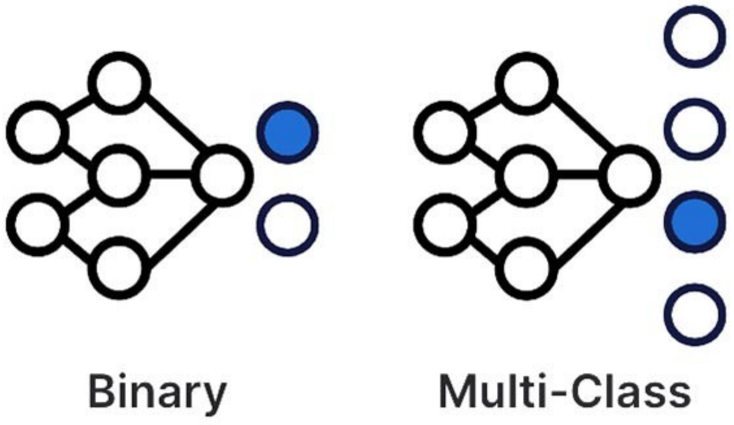
ARNULPHY Mathis

Ma412

Multi-Label Classification of Scientific Literature Using the NASA SciX Corpus



20 Juin 2025

M. Atilla Kaan Alkan

Table des matières

Introduction

In the era of scientific information overload, the ability to automatically annotate research papers with relevant keywords is crucial for enhancing information retrieval, indexing, and semantic search. This project focuses on building a machine learning system capable of predicting relevant keywords from scientific documents using only their titles and abstracts. The keywords correspond to scientific concepts or topics such as *"solar wind"*, *"lunar composition"*, and others that help describe the content of the paper.

This task is formulated as a multi-label text classification problem, where each document may be associated with multiple labels simultaneously. Unlike single-label classification, where each instance is assigned to one and only one class, multi-label classification allows the model to capture the multi-faceted nature of scientific documents, which often touch on several interrelated topics.

The dataset used for this project is the SciX corpus, curated by the NASA ADS/SciX team and publicly available on HuggingFace. It is made of scientific papers including astronomy and planetary science. The corpus we used is split into two subsets : training (18,677 documents) and val (3,025 documents combined). Our goal is to train a model that, given a paper's title and abstract, can predict the most appropriate set of tags for this article.

I Data Exploration and Preprocessing

1 Dataset Merging

The SciX dataset, as provided on HuggingFace, is divided into two subsets : train and val. However, for preprocessing purposes, we made the deliberate choice to merge these subsets into a single corpus prior to any cleaning or transformation. This choice was motivated by two key reasons : First, applying text normalization and label filtering operations only once avoids inconsistencies that could arise if preprocessing were done separately on each subset. Secondly, the dataset does not provide documentation on how the original splits were constructed. By recombining and reshuffling the entire dataset ourselves, we ensure a fair and randomized split between training and testing.

2 Column Selection and Text Construction

The raw dataset contains several columns, including `title`, `abstract`, `bibcode`, `keywords`, and `verified_uat_ids`. After inspection, we determined that only the title and abstract fields are necessary for building our model input. The `bibcode` (a unique document identifier) and `verified_uat_ids` (an auxiliary label field) were removed to reduce memory usage and processing complexity.

Next, we concatenated the title and abstract into a new column called `text`, which will serve as the sole input to the machine learning model. This unified representation ensures that the model captures both the succinct description (title) and the detailed context (abstract) of each paper.

3 Label Filtering and Balancing

The original dataset contains 1,864 unique UAT labels. However, a large portion of these labels are extremely rare, appearing only a handful of times across the dataset. To improve model robustness and reduce the number of output classes, we decided to filter out labels that occur fewer than 20 times across the entire corpus of more than 20,000 articles.

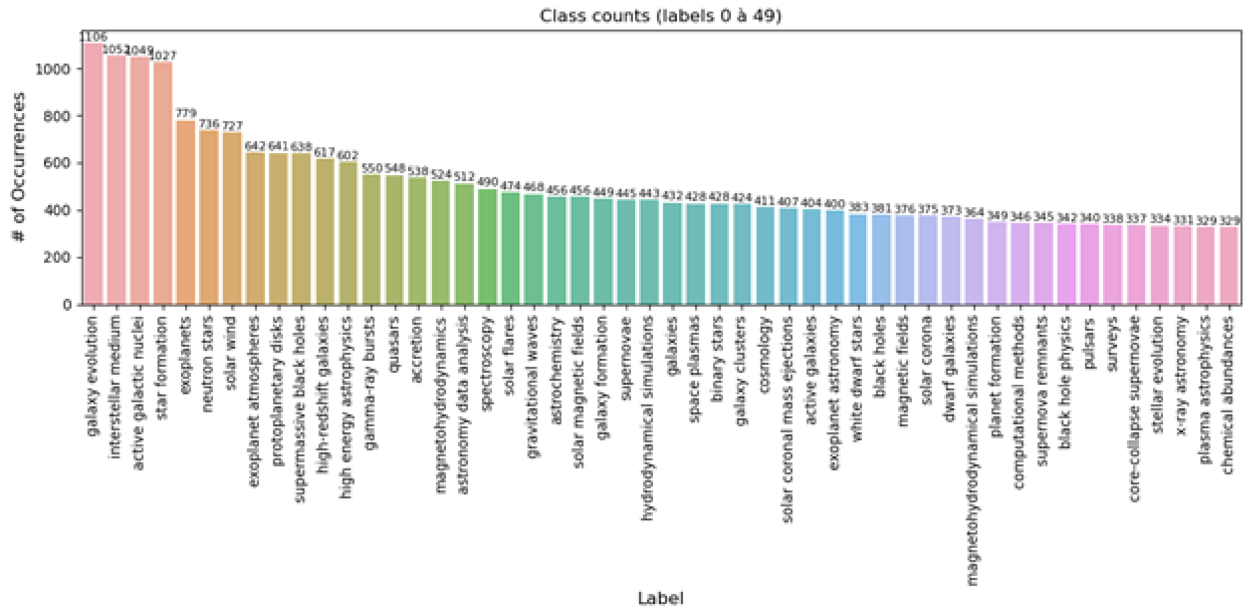


FIGURE 1 – Article distribution for labels

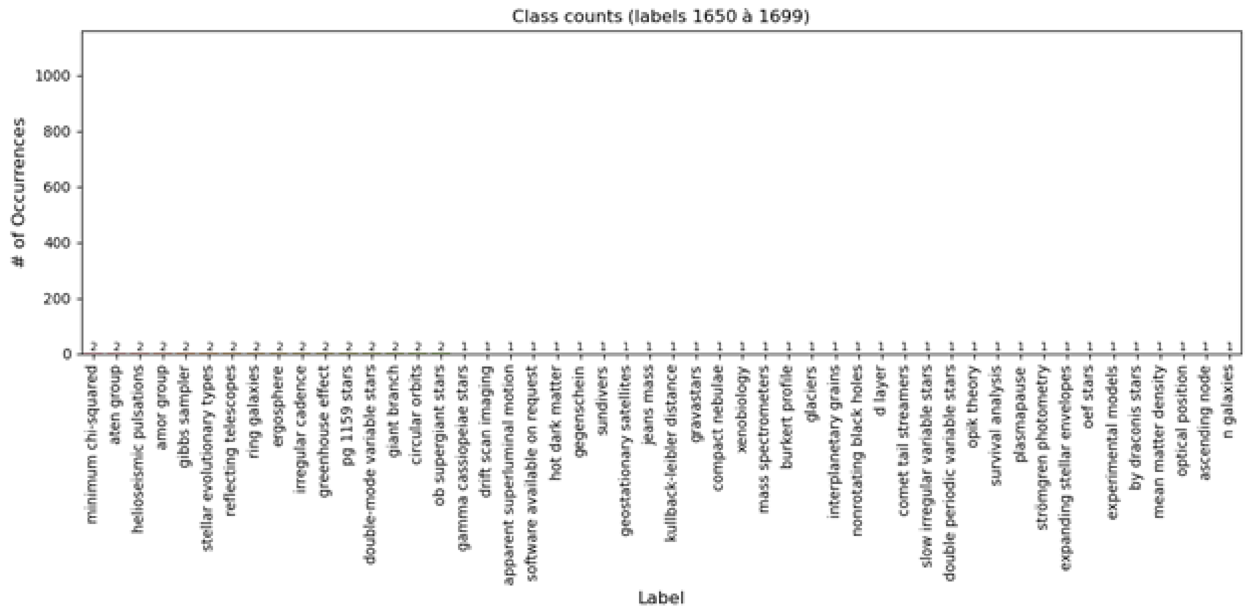
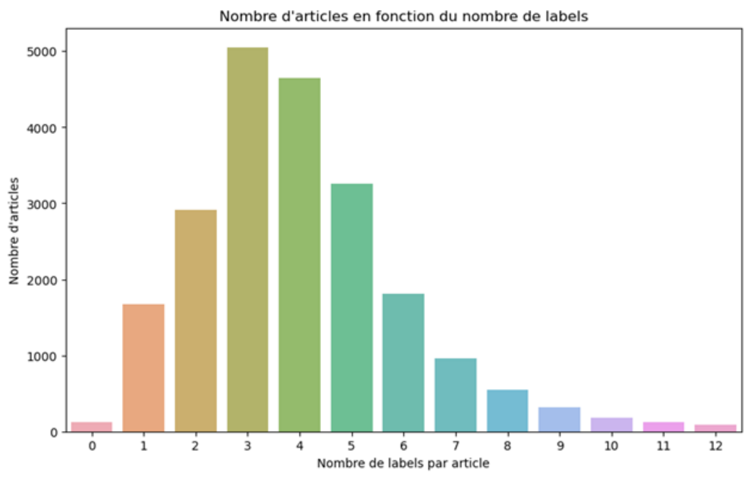
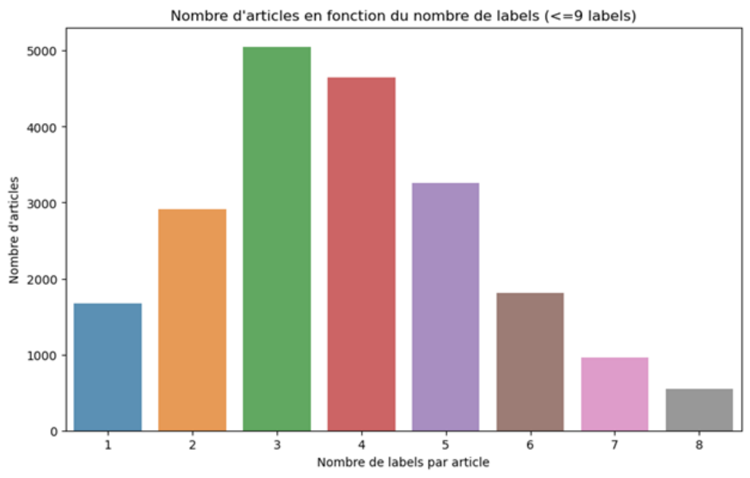


FIGURE 2 – Article distribution for labels

This filtering step reduced the total number of labels from 1,864 to 805, removing underrepresented concepts that the model would not have sufficient data to learn effectively.
Following this, we also removed documents that have no remaining labels after filtering or more than 9 labels (i.e., outliers in label count distribution).



(a) Label distribution before



(b) Label distribution after

This pruning step helps to ensure that the model is not overwhelmed by documents with excessive label density and prevents training on documents that are no longer usable after label filtering.

4 Text Cleaning and Normalization

The text data (title + abstract) was cleaned and normalized to prepare it for model ingestion. Preprocessing steps included :

— Lowercasing : All text was converted to lowercase to reduce vocabulary size and ensure uniformity.
— Non-alphabetical character removal : Using regular expressions, all characters that were not letters (e.g., punctuation, numbers, symbols) were removed to focus only on semantic content.
— Stopword removal : Common English stopwords (e.g., “the”, “and”, “is”) were removed using the NLTK stopword list to retain only meaningful words.
— Stemming : Words were reduced to their root forms using the Snowball stemmer. This reduces redundancy and helps the model generalize over different grammatical forms.

The following Python code from was used to perform these operations :

```
regex = re.compile(r'[^a-zA-Z]')
stop_words = set(stopwords.words('english'))

def remove_stopwords(text):
    no_stopword_text = [w for w in text.split() if not w in stop_words]
    return ' '.join(no_stopword_text)

def clean_text(text):
    text = text.lower()
    text = regex.sub(" ", text)
    text = ' '.join(text.split())
    return text

stemmer = SnowballStemmer("english")
def stemming(sentence):
    stemSentence = ""
    for word in sentence.split():
        stem = stemmer.stem(word)
        stemSentence += stem + " "
    return stemSentence.strip()

df_clean['text'] = df_clean['text'].apply(remove_stopwords)
df_clean['text'] = df_clean['text'].apply(clean_text)
df_clean['text'] = df_clean['text'].apply(stemming)

source : https://medium.com/analytics-vidhya/an-introduction-to-multi-label-text-classification-
b1bcb7c7364c
```

Text cleaning is essential in natural language processing tasks. It eliminates irrelevant characters, reduces vocabulary sparsity, and ensures the model focuses on meaningful semantic units. These steps contribute significantly to the effectiveness of downstream machine learning algorithms.

5 Preprocessing output summary

```
Number of articles before preprocessing: 21702
Number of labels before preprocessing: 1864

Number of articles after preprocessing: 20825
Number of retained labels: 805

Example of text before cleaning:
Dynamic Potential Sputtering of Lunar Analog Ma...
Generation of Low-inclination, Neptune-crossing...
Leveraging the Gravity Field Spectrum for Icy S...

Example of text after cleaning:
dynam potenti sputter lunar analog materi solar...
generat low inclin neptun cross tran neptunian ...
leverag graviti field spectrum ici satellit int...

Average text length (before cleaning): 224.18
Minimum length: 22
Maximum length: 483
```

```
Average text length (after cleaning): 154.88
Minimum length: 13
Maximum length: 361

Average number of labels per article (before cleaning): 3.99
Standard deviation: 2.01
Minimum number of labels: 0
Maximum number of labels: 12

Average number of labels per article (after cleaning): 3.81
Standard deviation: 1.68
Minimum number of labels: 1
Maximum number of labels: 8
```

II Model Selection and Experimental Settings

1 Overview

Given the complexity and scale of the multi-label classification task at hand with over 800 possible scientific keyword labels and the potential for each document to belong to multiple categories it is crucial to evaluate different modeling strategies. Our goal is to identify an architecture that not only performs well in terms of standard multi-label metrics, but also scales efficiently with the large number of classes and the size of the dataset. We implemented and compared six different modeling strategies : five traditional methods using scikit-learn and scikit-multilearn, and one transformer-based deep learning model using SciBERT. Our goal was to explore how both classical and modern NLP methods perform on a real-world, noisy scientific dataset.

The models evaluated are :

1. Binary Relevance (BR) with Logistic Regression
2. Classifier Chains (CC) with Logistic Regression
3. Label Powerset (LP) with Logistic Regression
4. Label Powerset (LP) with Random Forest
5. Label Powerset (LP) with MLP
6. SciBERT-based Neural Network for Multi-Label Classification

The first three model of this project (BR, CC and LP) follow the aproach of this article : `medium.com`

2 Binary Relevance (BR) with Logistic Regression

Binary Relevance is one of the most straightforward approaches to multi-label classification. The core idea is to decompose the multi-label problem into multiple independent binary classification tasks one for each label. Each classifier is trained separately to predict whether a specific label is present in a given document or not.

```python
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words=list(stop_words))),
    ('clf', BinaryRelevance(LogisticRegression(solver='sag')))
])
```

For our implementation, we used a `TfidfVectorizer` to convert the raw text into a sparse matrix representation which emphasizes the importance of terms that are more frequent in a specific document but less common across the corpus. A custom list of English stopwords was applied during vectorization to remove common, non-informative words. Each resulting TF-IDF vector served as input to a set of binary classifiers, one for each of the 805 labels. We used scikit-learn's LogisticRegression model with the 'sag' (Stochastic Average Gradient) solver, which is particularly suited for handling large-scale and sparse datasets efficiently. The output of this Binary Relevance approach is a vector of independent binary predictions, indicating the presence or absence of each label for a given instance.

Advantages

— Simplicity and Modularity : Each label is treated independently, making the model easy to implement, train, and debug.
— Scalability : Training can be parallelized across labels, which is beneficial when working with a large label space. (not the case for this implementation)

Drawbacks

— Ignores Label Correlation : This approach assumes label independence, which is rarely the case in scientific documents (e.g., "solar wind" is often associated with "magnetosphere").
— Limited Expressiveness : Lacks the ability to model complex interactions between labels, which may result in lower performance in domains where label dependencies are significant.

3 Classifier Chains (CC) with Logistic Regression

Classifier Chains (CC) is designed to overcome a key limitation of Binary Relevance : its inability to model label dependencies. In the CC approach, a sequence of binary classifiers is trained, where each classifier is responsible for predicting a single label. Each classifier in the chain receives not only the original input features, but also the predicted outputs of all previous classifiers in the sequence. This chaining mechanism enables the model to capture and exploit correlations between labels, which is particularly valuable in domains where labels frequently co-occur in structured or semantically meaningful ways.

To implement this method, we used a logistic regression classifier with the `sag` solver, chosen for its efficiency on large-scale problems. The text data was vectorized using TF-IDF representation, with a custom stopword list. The pipeline is shown below :

```
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words=stop_words)),
    ('clf', ClassifierChain(LogisticRegression(solver='sag')))
])
pipeline.fit(x_train, y_train)
predictions = pipeline.predict(x_test)
```

Despite its ability to model inter-label dependencies, Classifier Chains can be sensitive to the order in which labels are arranged. Errors made early in the sequence may propagate and negatively affect subsequent predictions. Furthermore, the sequential structure increases the computational cost of both training and inference compared to Binary Relevance, especially in large-scale multi-label tasks like ours. Nevertheless, Classifier Chains strike a promising balance between simplicity and dependency modeling.

Advantages

— Models inter-label dependencies, which can improve performance in cases where labels are not independent.
— Maintains the flexibility and simplicity of binary classifiers while enhancing them through sequential conditioning.

Drawbacks

— Sensitive to label order : suboptimal sequences may degrade performance.
— Computational cost increases with the number of labels, as each classifier depends on the outputs of the preceding ones.
— Error propagation may occur if early classifiers in the chain make poor predictions.

4 Label Powerset (LP) with Logistic Regression

Label Powerset (LP) is a problem transformation technique for multi-label classification that converts the task into a standard multi-class classification problem. Instead of training one classifier per label (as in Binary Relevance), LP treats each unique combination of labels in the training set as a single compound label. This transformation allows the model to inherently capture label dependencies, since it learns from co-occurring labels directly, rather than treating them independently.

However, this approach has significant limitations when applied to datasets with a large label space or high label cardinality, as is the case with the SciX corpus. In our dataset, with over 800 active labels and many possible combinations, the number of unique labelsets becomes extremely large. This results in a very sparse label distribution, where many label combinations occur only a handful of times, or even just once. Such sparsity makes it difficult for the classifier to generalize effectively, as it sees too few examples of most label combinations to learn robust decision boundaries. Furthermore, the computational cost increases with the number of classes, which negatively impacts both training efficiency and predictive performance.

```
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words=list(stop_words))),
    ('clf', LabelPowerset(LogisticRegression(max_iter=120)))
])
```

The training procedure for the Label Powerset (LP) model followed the same preprocessing pipeline as the Binary Relevance approach, relying on TF-IDF vectorization of the cleaned text data. As the LP strategy treats each unique combination of labels as a single new class, the logistic regression classifier was configured to allow up to 120 iterations to accommodate the increased complexity resulting from the expanded label space. While this approach can be effective for problems with a limited number of label combinations, it becomes problematic in settings like ours, where the number of unique label sets is high and the dataset is sparse. This label sparsity leads to many rare combinations that are hard for the model to generalize from, which may reduce performance despite the conceptual simplicity of the method.

Advantages

— Captures label dependencies implicitly. This is well suited for our case (labels can be linked to one another as explained before)

Drawbacks

— Scalability becomes an issue with increasing label diversity also many combinations may have low support or be unseen in the test set becoming non-relevent.

5 Label Powerset (LP) with Random Forest

Similar to the LP approach with logistic regression, this method transforms the multi-label problem into a multi-class task by treating each unique label combination as a single label. However, in this variant, a Random Forest classifier is used instead of logistic regression. Random Forest, being an ensemble of decision trees, can handle non-linear relationships and is generally more robust to noise and overfitting, especially on high-dimensional data.

Despite these strengths, the fundamental limitation of the LP method persists—labelset sparsity. In our case, the SciX corpus presents a very high number of unique label combinations, many of which occur infrequently. This poses a challenge for the Random Forest model as well, since it becomes difficult for trees to generalize from rare labelsets and the large number of classes increases model complexity and training time.

```
# Define a machine learning pipeline that includes TF-IDF vectorization followed by
# a multi-label classification using Label Powerset and a Random Forest classifier
pipeline = Pipeline([
    # Convert the input text into TF-IDF vectors, removing custom stopwords
    ('tfidf', TfidfVectorizer(stop_words=list(stop_words))),

    # Use the Label Powerset strategy with a Random Forest classifier
    ('clf', LabelPowerset(RandomForestClassifier(
        n_estimators=200,      # Use 200 decision trees in the forest
        max_depth=None,        # Allow trees to grow until all leaves are pure or contain <
    min_samples_split
        random_state=42,       # Fix the seed for reproducibility
        n_jobs=-1              # Use all available CPU cores for parallel processing
    )))
])
```

The Random Forest parameters were selected to balance depth and ensemble size, mitigating overfitting while allowing the model to capture complex patterns in the data. However, like with logistic regression, the model's performance was hindered by the extremely sparse distribution of compound labels.

Advantages

— Better captures non-linear patterns in feature space.
— More resilient to noise and overfitting than linear models.

Drawbacks

— Suffers from the same label sparsity issue as LP with logistic regression.
— High memory and computational cost due to large number of classes and deep trees.

6 Label Powerset (LP) with Multi-Layer Perceptron (MLP)

In this configuration, the Label Powerset method is combined with a Multi-Layer Perceptron (MLP), a class of feedforward artificial neural networks. MLPs are capable of modeling complex and non-linear interactions, which could potentially help in learning from the high-dimensional and sparse SciX dataset.

However, despite its expressive power, the MLP is still constrained by the underlying LP transformation. The vast number of unique label combinations in the dataset results in a very large output space, which increases the difficulty of optimization and requires a large amount of training data for each class something that is not available due to the sparsity of labelsets.

```
# Define a machine learning pipeline with TF-IDF vectorization followed by a Label Powerset
    classification model
pipeline = Pipeline([
    #Convert raw text into TF-IDF features, excluding custom stopwords
    ('tfidf', TfidfVectorizer(stop_words=list(stop_words))),

    #Use Label Powerset strategy with a Multi-Layer Perceptron classifier
    ('clf', LabelPowerset(MLPClassifier(
        hidden_layer_sizes=(256, 128, 64),  # Three hidden layers with 256, 128, and 64
    neurons respectively
        activation='relu',                 # ReLU activation function for non-linearity
        solver='adam',                     # Adam optimizer for weight updates
        max_iter=200,                      # Maximum number of training iterations
        random_state=42                    # Set random seed for reproducibility
    )))
])
```

The MLP was configured with three hidden layers and a moderately high iteration limit to accommodate the complexity of the task. Still, as with the other LP models, rare label combinations limit the model's ability to generalize effectively.

Advantages

- Can model highly non-linear and complex decision boundaries.
- Potential to learn hierarchical representations of input features.

Drawbacks

- Very sensitive to label imbalance and sparse labelsets.
- Requires extensive tuning and more computational resources to converge.

7 SciBERT for Multi-Label Classification

To leverage the contextual richness of scientific language, we also implement a transformer-based model : SciBERT. To capture richer semantics in scientific language, we developed a transformer-based model using `allenai/scibert_scivocab_uncased`, a domain-specific BERT model trained on 1.14M scientific papers. We fine-tune SciBERT on our dataset for a multi-label classification objective by adding a dense output layer with sigmoid activation on top of the [CLS] token representation. We modified the output layer for multi-label classification by applying a sigmoid function on the [CLS] token representation.

```python
class SciBERTMultiLabel(nn.Module):
    def __init__(self, n_labels):
        super().__init__()
        self.bert = AutoModel.from_pretrained("allenai/scibert_scivocab_uncased")
        self.dropout = nn.Dropout(0.2)
        self.classifier = nn.Linear(self.bert.config.hidden_size, n_labels)

    def forward(self, input_ids, attention_mask, labels=None):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        pooled_output = outputs.last_hidden_state[:, 0]
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        probs = torch.sigmoid(logits)
        ...
```

Training Pipeline :

- Tokenization : `AutoTokenizer` from HuggingFace Transformers with padding and truncation up to 256 tokens.
- Loss function : Binary Cross-Entropy (`BCELoss`) applied to sigmoid outputs.
- Optimizer : `AdamW` with a learning rate of 2×10^{-5}.
- Batch size : 8 for training, 16 for evaluation.
- Hardware : Trained on GPU if available, using PyTorch.

Training Loop : The model was trained for 3 epochs with weight updates on each batch. After each epoch, the model's state was saved. The training process leveraged a `DataLoader` to iterate over a custom `Dataset` that returned both tokenized input and multi-label targets.

Benefits :

- Strong performance on semantic understanding and contextual relationships.

— Excellent ability to generalize across noisy and high-dimensional labels.

Challenges :

— Training is resource-intensive and slow compared to traditional models.

— Model size and inference time can be a limitation in production settings.

III Evaluation Metrics

Evaluating multi-label classification models requires metrics that go beyond traditional accuracy used in single-label problems. In our study, we use three primary evaluation metrics : *Subset Accuracy*, *Hamming Loss*, and the *F1 score*. Each of these captures different aspects of the model's performance and helps us understand its strengths and limitations in the context of predicting multiple tags for scientific articles. Additionally, we briefly mention *Precision* and *Recall* as complementary metrics for deeper insight.

1 Subset Accuracy

Also known as exact match ratio, Subset Accuracy is the most stringent metric. It measures the proportion of samples for which the entire predicted label set exactly matches the true set of labels. It is defined as :

$$\text{Subset Accuracy} = \frac{1}{N}\sum_{i=1}^{N} \nVdash(Y_i = \hat{Y}_i)$$

where N is the total number of samples, Y_i is the true label set, $\hat{Y}_i$ is the predicted label set, and $\nVdash(\cdot)$ is the indicator function that returns 1 if its argument is true, and 0 otherwise.

Although this metric is very strict, it gives a clear indication of how often the model is perfectly correct. In our context, this is useful for evaluating if the model predicts all and only the relevant scientific tags for each article.

2 Hamming Loss

Hamming Loss calculates the fraction of labels that are incorrectly predicted, i.e., labels that are missed or wrongly assigned. It is defined as :

$$\text{Hamming Loss} = \frac{1}{N \cdot L}\sum_{i=1}^{N}\sum_{j=1}^{L} \nVdash(Y_{ij} \neq \hat{Y}_{ij})$$

where L is the number of labels, Y_{ij} is the true value (0 or 1) for label j of instance i, and $\hat{Y}_{ij}$ is the predicted value.

Hamming Loss is less strict than Subset Accuracy and gives a fine-grained view of the model's behavior across all labels. In our setting with over 800 labels, it is particularly useful because it captures partial correctness and penalizes both false positives and false negatives equally.

3 F1 Score (Micro-Averaged)

The F1 score balances Precision and Recall. In the micro-averaged version, the metric aggregates the contributions of all labels to compute the average F1. It is defined as :

$$\text{Precision}_{\text{micro}} = \frac{\sum_{i=1}^{L} TP_i}{\sum_{i=1}^{L} TP_i + FP_i} \qquad \text{Recall}_{\text{micro}} = \frac{\sum_{i=1}^{L} TP_i}{\sum_{i=1}^{L} TP_i + FN_i}$$

$$F1_{\text{micro}} = \frac{2 \cdot \text{Precision}_{\text{micro}} \cdot \text{Recall}_{\text{micro}}}{\text{Precision}_{\text{micro}} + \text{Recall}_{\text{micro}}}$$

The micro-averaged F1 score is suitable for imbalanced datasets, where some labels are much more frequent than others. It reflects overall performance in terms of both capturing relevant labels (recall) and avoiding irrelevant ones (precision).

4 Complementary Analysis

Beyond global evaluation scores, additional analyses can provide deeper insights into model performance and guide future improvements. One relevant perspective is to study how performance evolves with the number of labels : as the label space becomes more diverse, it becomes increasingly difficult for models to predict rare or highly specific label combinations. Analyzing subsets of labels can reveal how sensitive each model is to label sparsity. Similarly, examining the effect of training set size by training on 10%, 50%, or 100% of the data can highlight how datahungry each approach is and whether performance plateaus. Hyperparameter tuning, especially of the logistic regression's regularization strength (C), solver type, and maximum iterations, can also lead to significant improvements by avoiding underfitting or overfitting. Finally, substituting logistic regression with other classifiers, such as Support Vector Machines, Random Forests, or lightweight neural networks, may offer better performance on certain aspects like capturing non-linear relationships or reducing label noise sensitivity. These complementary analyses not only help optimize model performance but also increase interpretability and adaptability to the specificities of the dataset.

To sum up, the combination of Subset Accuracy, Hamming Loss, and micro-averaged F1 score gives a balanced view of model performance in a multi-label setting. Subset Accuracy emphasizes exact correctness, Hamming Loss quantifies general error rate, and F1 captures the trade-off between Precision and Recall. These metrics together ensure that both strict correctness and partial relevance are taken into account.

5 Models result on test dataset

For the results presented in this report, we limited our evaluation to the 50 most frequent labels in the dataset. This approach was adopted due to computational constraints (each algorithm computation time is approximately one hour), as training and evaluating models on the full label space which includes over 800 labels would require significantly more processing power and memory. While this simplification means that the reported results do not fully reflect the performance of the models in a complete real world scenario, it nonetheless provides a useful glimpse into how each model behaves on the most commonly occurring labels. It is important to keep in mind that these preliminary evaluations may not fully capture the robustness or limitations of the models when faced with rare or sparse label combinations. Additional experiments, using more computational resources and considering the entire label set, would be necessary to draw more definitive conclusions about the generalization capabilities of the different algorithms.

6 Model Performance Evaluation

In this section, we evaluate the performance of the six tested models using Subset Accuracy, Micro-averaged F1 Score, and Hamming Loss. The results are detailed below for each model, followed by a comparative analysis.

1 Binary Relevance Model

The Binary Relevance model treats each label as an independent binary classification problem. It achieved :

— **Accuracy** : 0.376
— **F1 Score** : 0.334
— **Hamming Loss** : 0.0195

This model performs well in terms of exact match and per-label prediction accuracy, likely due to its robustness and simplicity. However, it does not model label correlations, which limits its overall recall in highly interdependent label spaces.

2 Classifier Chain Model

The Classifier Chain model improves upon Binary Relevance by considering label dependencies through chaining binary classifiers. Its results are :

— **Accuracy** : 0.371
— **F1 Score** : 0.326
— **Hamming Loss** : 0.0197

While slightly behind Binary Relevance in overall performance, it retains a low Hamming Loss, indicating strong individual label predictions. The slight performance drop may come from error propagation along the chain.

3 Label Powerset Model (Logistic Regression)

This model transforms the multi-label problem into a multi-class task by encoding each unique label combination. It obtained :

— **Accuracy** : 0.368
— **F1 Score** : 0.280
— **Hamming Loss** : 0.0201

Although the accuracy is comparable to BR and CC, the F1 score is significantly lower. This model suffers from the combinatorial explosion of label sets, leading to sparse representation and overfitting on rare combinations.

4 Label Powerset with Random Forest

Using a Random Forest as the base classifier for Label Powerset yielded weaker performance :

— **Accuracy** : 0.327
— **F1 Score** : 0.063
— **Hamming Loss** : 0.0219

Despite the ensemble nature of the Random Forest, the model fails to generalize well. The low F1 score suggests that the model misses most relevant labels, possibly due to label imbalance and the sparsity of label combinations.

5 Label Powerset with Multi-Layer Perceptron (MLP)

This model leverages a neural network as a base classifier for Label Powerset :

— **Accuracy** : 0.294
— **F1 Score** : 0.359
— **Hamming Loss** : 0.0278

Despite a relatively low exact match accuracy, the MLP demonstrates the best F1 score among all models. This suggests that it captures partial label matches effectively, though it struggles to predict the exact full set.

6 SciBERT Model

SciBERT, a transformer-based deep learning model pretrained on scientific text, achieved :

— **Accuracy** : 0.025
— **F1 Score** : 0.246
— **Hamming Loss** : 0.0044

Although it has the lowest subset accuracy, SciBERT exhibits exceptional performance in minimizing per-label errors. The extremely low Hamming Loss implies it often predicts the right labels even if the complete set is incorrect expected behavior for a deep contextual model under high label cardinality.

7 Comparison and Discussion

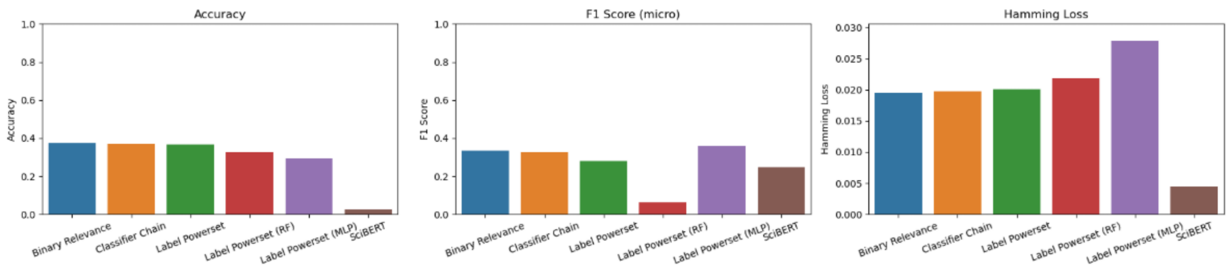


FIGURE 4 – Model comparison

When comparing all models, we observe several distinct performance patterns. Binary Relevance and Classifier Chain approaches provide solid results in terms of subset accuracy and overall robustness, indicating their reliability for predicting complete sets of labels in moderate complexity tasks. The MLP-based model, while less accurate in predicting exact label sets, excels in retrieving a larger number of relevant labels, which explains its superior F1 score. SciBERT demonstrates the best precision per individual label, achieving the lowest Hamming Loss among all models, though it tends to underperform in predicting full label sets, likely due to over-specialization. Finally, the Random Forest model underperforms in this multi-label context, suggesting its limitations in capturing the complex label dependencies and textual nuances present in the dataset.

IV Results Analysis

The evaluation of all implemented models reveals both their individual strengths and weaknesses when applied to multi-label classification on scientific text abstracts. This section interprets the performance metrics discussed earlier and identifies key patterns, challenges, and improvement opportunities.

General Observations

First, we observe that model performance tends to degrade significantly as the number of labels increases. This behavior is especially evident when the label space expands beyond 30 classes and becomes pronounced at 50 labels. Both traditional and deep learning models struggle to maintain precision and recall under such high-label cardinality. This suggests that the implemented models are not inherently designed for handling such a wide multi-label output space, and that additional architectural or training adaptations may be necessary.

Model Performance Summary

Among classical machine learning methods, the Binary Relevance (BR) and Classifier Chain (CC) models with logistic regression offered relatively balanced results, particularly in terms of subset accuracy and overall robustness. Their modular nature allows them to handle multi-label tasks reasonably well, although they lack an explicit mechanism to model inter-label dependencies. This limits their effectiveness in complex prediction scenarios, where label co-occurrence patterns are important.

The Label Powerset (LP) model with logistic regression achieved notably higher accuracy and F1 score (Accuracy = 0.738, F1 = 0.446), likely because it treats each unique label combination as a distinct class. However, this approach becomes computationally expensive and suffers from sparsity issues as the number of labels grows, making it unsuitable for larger label sets.

The use of Random Forest with the LP strategy yielded poorer results, highlighting that tree-based models may not be the best fit for multi-label classification in high-dimensional sparse input spaces like TF-IDF representations.

The Multi-Layer Perceptron (MLP) paired with LP performed well in terms of F1 score, indicating its ability to retrieve relevant labels across a variety of articles. However, its subset accuracy lagged behind, suggesting that while it captures parts of the true label set effectively, it struggles to predict the full correct combination consistently.

SciBERT Model Insights

The SciBERT-based model demonstrated the lowest Hamming Loss, meaning it was the most precise at predicting individual labels. However, its subset accuracy was relatively low, which implies that it often predicted only parts of the correct label set. This discrepancy could be attributed to limited training time the model was only fine-tuned for 3 epochs due to resource constraints. Additional training would likely improve its ability to generalize and capture more complete label sets.

Error Analysis and Difficult Labels

Error inspection revealed that specific labels were consistently underpredicted across models. These labels often corresponded to rare topics or domain-specific keywords that appeared in very few articles, making them hard to learn and generalize from. Conversely, frequent labels tended to be overpredicted, sometimes being assigned even when contextually inappropriate, indicating a bias toward high-frequency classes.

Recommendations for Improvement

To improve model performance on this task, several strategies could be considered :

— Label Embedding Techniques : Instead of treating labels independently, use models such as Joint Embedding Models or *Label Graph Neural Networks* to better capture relationships between labels.

— Transformer-based Models with More Epochs : Fully leveraging pretrained language models like SciBERT or Longformer with extended fine-tuning would likely enhance results, especially for capturing context-dependent label combinations.

— Hierarchical Classification : Introduce hierarchical label grouping or taxonomy-aware models to reduce the complexity of direct prediction over 50+ labels.

— Problem Transformation Approaches : Methods like *Classifier Chains with deep networks*, or more recent architectures like *Seq2Set Transformers* and *SetFit*, might improve generalization by accounting for label dependencies and generating sets rather than flat outputs.

— Data Augmentation and Balancing : Oversampling underrepresented labels or synthetically augmenting the dataset could help balance label frequency and reduce class imbalance issues.

In conclusion, while traditional models provide a strong baseline, tackling high-dimensional multi-label classification on scientific text effectively requires a combination of deep contextual understanding, inter-label dependency modeling, and sufficient training time. Future work should explore more specialized architectures and training schemes tailored for this demanding task.