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

-

Ma 412 - Mathematical Foundations for Statistical Learning

Ruth Aurora RUIZ CARRILLO
Aero 4 - SYS

27 December 2024

Abstract

Data classification is an integral part of everyday decision-making, enabling the optimization of tasks and processes. However, when addressing large-scale problems, more sophisticated tools and techniques are required to handle such classifications effectively. Multi-label classification, a prevalent challenge in machine learning, serves as the focus of this project, which aims to classify scientific literature from NASA's SciX corpus.

The project involves a comprehensive exploration of the dataset to understand its characteristics and complexities. Subsequently, various machine learning techniques suitable for multi-label classification will be analyzed. The most appropriate method will then be implemented, culminating in the development of a model capable of accurately classifying new data. This approach ensures a systematic and effective solution to the problem of multi-label classification within the context of scientific literature.

Introduction

In today's data-driven world, classification plays a crucial role in organizing and interpreting information to optimize workflows and decision-making processes. While humans intuitively classify data in everyday life, large-scale problems require sophisticated computational tools to handle the complexity and volume of information effectively. One such challenge in machine learning is multi-label classification, where each instance can be assigned multiple labels to capture its diverse attributes or themes. This approach is particularly relevant in the analysis of scientific literature, where documents often span multiple topics.

This project focuses on developing a system capable of predicting relevant keywords—such as "solar wind" and "lunar composition" from the titles and abstracts of scientific papers. Using multi-label text classification techniques, the goal is to train a machine learning model that can identify associations between document content and specific keywords. This enables the model to recognize and accurately label the diverse themes present in scientific texts, unlike single-label classification, which limits each document to a single category.

The dataset for this project is provided by the NASA ADS/SciX team and is hosted on the HuggingFace repository. It comprises the SciX corpus, a collection of titles and abstracts from published scientific papers, divided into training and test sets containing 18,677 and 3,025 documents, respectively. By leveraging this dataset, the project aims to explore and implement advanced machine learning techniques to tackle the complexities of multi-label classification.

Problem

The task is to design and implement a system that automates the classification of scientific literature by predicting relevant keywords based on the titles and abstracts of research papers. This is a multi-label text classification problem, meaning each document can be associated with multiple keywords to represent its diverse themes. The system must address the challenge of accurately recognizing and labeling these topics, which vary widely in complexity and frequency within the dataset.

Data Exploration and Preprocessing

The dataset is composed of five sections: bibcode, title, abstract, verified UAT IDs, and verified UAT labels. The training set contains 18,700 rows, while the validation set comprises 3,000 rows. Since we already possess a straightforward subset of data that can be readily classified (UAT labels and titles), this subset will be utilized during the preprocessing stage.

We begin by loading the adsabs/SciX_UAT_keywords dataset using the datasets library, providing access to a structured collection of scientific data. The dataset is printed to examine its structure, and the first training sample is displayed for a detailed view of its content, which includes fields like title and verified_uat_labels.

To analyze the class distribution of keywords in the dataset, the verified_uat_labels are extracted from all training samples and combined into a single list. The frequency of each keyword is calculated using the Counter function, and a bar chart is generated to visualize the keyword distribution, helping to identify potential class imbalances in the dataset.

Next, the code sets up preprocessing for text data by leveraging the nltk library. It downloads required resources, such as the punkt tokenizer and a list of English stop words. A custom function, preprocess_text, is defined to clean the text: it converts text to lowercase, tokenizes it into words, removes non-alphanumeric tokens and stop words, and recombines the processed words into a single string. This function is applied to the titles in the training set, producing a clean, tokenized dataset ready for vectorization.

To prepare the cleaned text for machine learning, the TfidfVectorizer is used to transform the processed titles into numerical features. This vectorizer captures the importance of each word in the text while limiting the feature space to a maximum of 5,000 terms. Simultaneously, the verified_uat_labels, which are multi-label annotations for each sample, are transformed into a binary matrix using the MultiLabelBinarizer. This step encodes the labels in a format suitable for multi-label classification tasks.

Finally, the processed dataset is split into training and testing subsets using train_test_split, allocating 80% of the data to training and 20% to testing. The split ensures that both features and labels are separated consistently, with a random state for reproducibility. To verify the preprocessing pipeline, the shapes of the feature and label matrices are printed, and the

first five processed titles are displayed, confirming that the text has been successfully cleaned and vectorized.

Possible Solutions

Six different models were considered for the possible architecture solution:

Support Vector Machines(SVM)

SVM is a supervised learning algorithm used primarily for binary classification tasks. It works by finding an optimal hyperplane that separates data points belonging to two different classes with the largest possible margin. This margin reduces misclassification during prediction. The kernel function is often used to transform the data into a higher-dimensional space, making it easier to separate classes that are not linearly separable in the original space. For multiclass problems, SVM requires the construction of multiple binary classifiers (e.g., one-vs-one or one-vs-all), which can increase computational cost.

This method was discarded due to the overhead of combining multiple classifiers for multiclass problems, which could be computationally expensive.

Nearest Neighbor(K-NN)

K-NN is a simple yet effective algorithm that classifies data points based on their proximity to other data points. It measures the distance (e.g. Euclidean, Manhattan, or Minkowski) between a given data point and its neighbors in the feature space. The classification of the point is determined by the majority class among its k nearest neighbors. The choice of k is critical; too small or too large a value can lead to underfitting or overfitting.

This method is particularly suitable for heterogeneous datasets but can become computationally expensive with large datasets or high-dimensional feature spaces.

Neural Networks(NN)

Neural networks consist of layers of interconnected nodes (neurons) that mimic the structure of the human brain. Data flows through an input layer, one or more hidden layers, and an output layer. Each neuron applies a weighted sum of its inputs followed by a non-linear activation function. Neural networks rely heavily on training data to learn patterns and relationships in the data. The learning process aims to minimize a cost function (e.g., mean squared error or cross-entropy loss), which quantifies the difference

between predicted and actual values.

While powerful and capable of capturing complex patterns, neural networks require significant computational resources, careful tuning, and sufficient data to avoid overfitting. Despite these challenges, they remain a strong contender due to their flexibility and predictive power.

Decision Tree

Decision trees classify data by recursively splitting the dataset based on feature values using decision rules. Each split reduces uncertainty (e.g., entropy or Gini impurity), and the process continues until all data points are classified or a stopping criterion (e.g., maximum depth or minimum number of samples per leaf) is met. Decision trees are easy to interpret and work well for small to medium-sized datasets. However, they are prone to overfitting, especially with large datasets, unless regularized (e.g., via pruning).

While simple and effective, decision trees may not scale well with complex datasets due to their tendency to grow exponentially with data size and complexity.

Logistic Regression

Logistic regression is a binary classification algorithm that models the probability of an outcome belonging to one of two classes. It uses the logistic (sigmoid) function to map predicted values to probabilities between 0 and 1. The model optimizes its parameters by maximizing the likelihood of the observed data, often using gradient descent. While simple and interpretable, logistic regression assumes a linear relationship between the input features and the log-odds of the outcome, which may not hold for complex datasets.

This method was deemed too computationally expensive for large datasets or datasets requiring non-linear decision boundaries.

Naïve Bayes

Naïve Bayes is a probabilistic classifier based on Bayes' theorem, with the assumption that features are conditionally independent given the class label. Despite this "naïve" assumption, the model performs surprisingly well for many applications, particularly text classification and spam detection. It calculates the posterior probability of each class for a given input and selects the class with the highest probability. Naïve Bayes is highly efficient, requires minimal computational resources, and scales well to large datasets.

This method appears to be the best option due to its speed, scalability, and suitability for massive datasets.

Model Selection

Three models were chosen and sketched to be implemented, to see which one came to a closer prediction. All three of the models use the preprocessed data code as base.

Neural Networks

The neural network is defined and constructed using TensorFlow's Keras API. It consists of an input layer with 512 nodes and ReLU activation, followed by a dropout layer to prevent overfitting. A second hidden layer with 256 nodes and another dropout layer is added, and the final output layer uses a sigmoid activation function to handle multi-label classification for the number of unique classes. The model is compiled with the Adam optimizer and binary cross-entropy loss, given the multi-label binary classification task. Training data is converted into dense arrays, and the model is trained for 10 epochs with a batch size of 32. Validation data is included to monitor performance during training.

Post-training, the model is evaluated on the test data, outputting the test loss and accuracy. Predictions are generated, and the inverse transformation of the binary predictions is used to map results back to label space. Performance metrics such as accuracy, precision, recall, and F1 score are calculated to assess the model's effectiveness. Finally, confusion matrices for each label are generated and visualized using Matplotlib.

Naïve Bayes

The MultiLabelBinarizer is updated to exclude the always-present labels.

A Naïve Bayes classification model is built using the MultinomialNB class wrapped with a OneVsRestClassifier for multi-label classification. The model is trained on the processed training data, and predictions are generated on the test data. Various evaluation metrics, including accuracy, precision, recall, and F1 score, are calculated to assess the model's performance.

Decision Tree

A Decision Tree-based multi-label classification model is created using MultiOutputClassifier, which wraps a DecisionTreeClassifier to handle multi-label outputs.

The model is trained on the processed training data and then used to predict the labels for the test data.

The model's performance is evaluated using standard metrics such as accuracy, precision, recall, and F1 score, calculated with the predictions and ground truth labels.

Evaluation and Metrics

An accuracy, precision, recall and F1 score where implemented on each of the models, obtaining the following results:

Neural Network

- Accuracy: 0.01 → Very poor; it means only 1% of labels were correctly predicted.
- Precision: 0.6 → Decent; this indicates that when the model predicts a label, it is correct 60% of the time.
- Recall: 0.04 → Very poor; it means the model is identifying only 4% of the actual labels.
- F1 Score: 0.09 → Low, but better than the other two models; the F1 score (harmonic mean of Precision and Recall) suggests that the model's performance is marginally better than the others.

Has the highest Precision but the lowest Recall. It's likely overemphasizing certain labels at the expense of missing most of the actual labels. It's not the best choice because of its poor Recall and F1 Score.

Naïve Bayes

- Accuracy: 0.001 → Terrible; only 0.1% of labels were correctly predicted.
- Precision: 0.8 → High; this means when the model predicts a label, it's correct 80% of the time. However, it might not be predicting much, as shown by the low recall.
- Recall: 0.001 → Extremely poor; it means only 0.1% of the actual labels were detected.
- F1 Score: 0.003 → The lowest of the three; this indicates a strong imbalance between Precision and Recall, rendering the model impractical.

High Precision but almost no Recall, leading to the lowest F1 Score. This suggests it is too conservative, only predicting when it's very confident, but failing to generalize across all classes.

Decision Tree

- Accuracy: 0.008 → Poor; only 0.8% of labels were correctly predicted.
- Precision: 0.2 → Very low; the model is correct only 20% of the time when predicting a label.
- Recall: 0.2 → Relatively higher; it captures 20% of the actual labels.
- F1 Score: 0.2 → The highest among the three models; this balance between Precision and Recall shows the Decision Tree is the most reliable of

the three models.

Though its Precision is lower than the other models, its Recall is much higher (and balanced with Precision), leading to the highest F1 Score. This makes it the best fit among the three for multi-label classification.

Results Analysis

The implementation of the Decision Tree model for multi-label classification on the adsabs/SciX_UAT_keywords dataset demonstrates a systematic approach to solving the problem, encompassing data preprocessing, feature extraction, and model evaluation. The pipeline effectively integrates the use of the DecisionTreeClassifier within a MultiOutputClassifier to handle the multi-label nature of the dataset. Despite the comprehensive approach, the evaluation metrics indicate significant room for improvement:

Accuracy:

The model's accuracy (0.008) is very low, suggesting that less than 1% of the labels were predicted correctly. This indicates that the model struggles to generalize across all labels.

Precision:

With a precision of 0.2, the model is only moderately reliable when it predicts a label, as 20% of its predictions are correct. This highlights the need for better feature-label association.

Recall:

The recall of 0.2, though still low, is relatively better than precision and accuracy, suggesting that the model captures some relevant labels, though many are missed.

F1 Score:

The F1 score of 0.2, which balances precision and recall, is the highest among the metrics but still insufficient for a robust classification system. It reflects an overall limited capability in handling the dataset's complexity.

Model Strengths:

The pipeline is well-structured, using TF-IDF vectorization to convert text data into numerical features and employing the MultiLabelBinarizer for encoding labels, which are critical steps in multi-label classification. The Decision Tree provides interpretable rules and may work well for smaller datasets or less complex problems.

Challenges:

Data Imbalance:

The label distribution visualization revealed imbalances in the dataset, which could hinder the model's ability to predict less frequent labels. Addressing this by oversampling minority classes, undersampling majority classes, or applying weighted loss functions could improve performance.

Model Complexity:

Decision Trees may struggle with high-dimensional feature spaces (as seen with 5,000 terms in the TF-IDF matrix). Using ensemble methods like Random Forest or Gradient Boosting can improve performance by reducing overfitting and capturing more nuanced patterns.

Evaluation Metrics:

The low scores across metrics suggest that further hyperparameter tuning (tree depth, split criteria, minimum samples per leaf...) and experimentation with feature engineering (limiting TF-IDF features further or applying dimensionality reduction techniques like PCA) are necessary.

In conclusion, the Decision Tree model, while structured and interpretable, currently underperforms for this task. To achieve better results, addressing label imbalances, refining feature selection, and experimenting with more advanced algorithms should be prioritized.

GitHub Repository

https://github.com/Auroooooooooora/Ma412_Project.git