Multi-label Classification of Pubmed Articles

Pavan Choudhari, Priya Garg | CS 6120

PubMed is a free search engine that comprises citations for biomedical literature from MEDLINE, life science journals, and online books. The search engine uses MeSH Headings to help researchers and experts find the resource papers for a target topic. However, these documents are manually annotated by Biomedical Experts. Considering the huge number of articles being added, manual annotation is difficult, time consuming and susceptible to human errors. In this project, we have used various Machine Learning and Deep Learning techniques to label the article texts and automate the process of manual annotation.

Data Extraction and Exploration:

The dataset has been pulled together from Huggingface but the root label categories are missing in the source. We use the descriptor 2022 file from PubMed official website to reverse map the root labels based on the descriptors within the raw data. Due to substantial numbers of labels present as a MeSH ID, this raises the issue of extremely large output space and severe label sparsity issues, therefore we decided to restrict our output space to the 15 root categories. An example of category D is shown in the below figures 1 and 2.

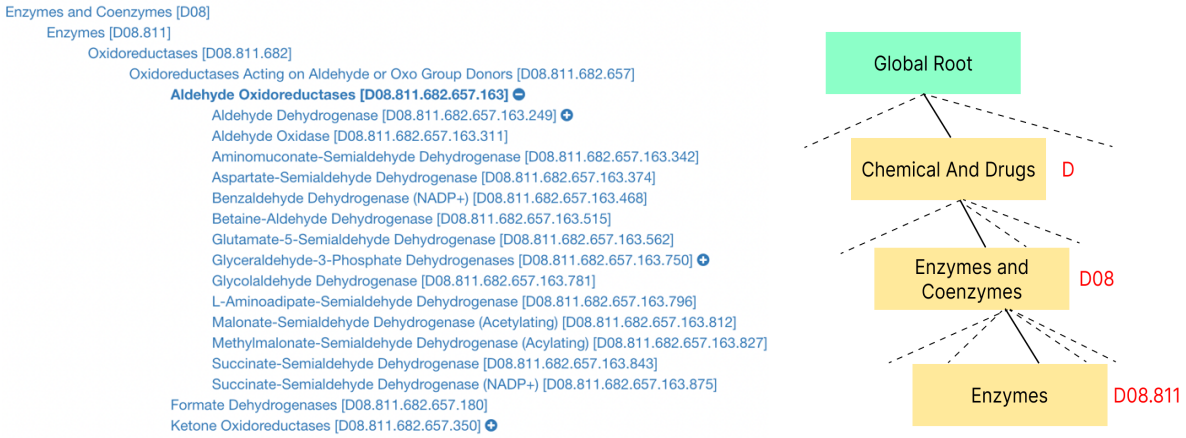

Fig 1. (left) If a research article is on aldehyde oxidoreductases, then figure 1, shows the tree structure for that topic, which can be traced back to its root domain, Chemicals and Drugs here as shown in figure 2 (right).

We have hosted the Dataset on Kaggle. There is a total of 190k records, out of which 70% records are used for training and 30% records are used for testing. With transfer learning, we also tested the model on an additional test dataset. Below is the label distribution of the training set.

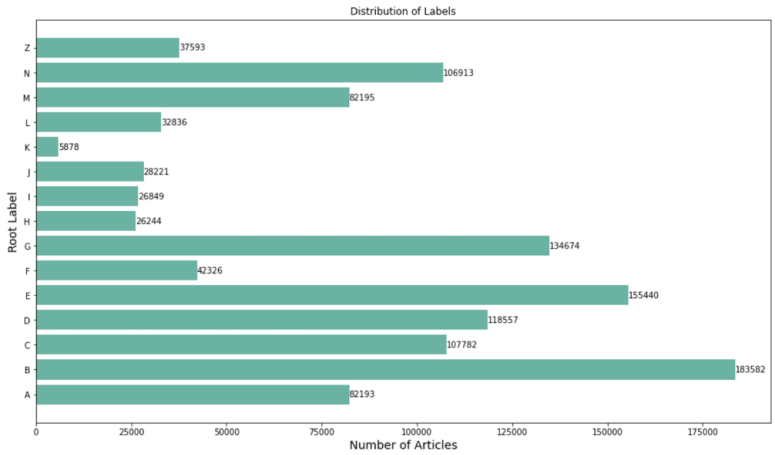

Fig 3. Distribution of labels in training set

Modeling:

The goal of this project was to experiment with different modeling techniques for multi-label classification. For multiclass classification, where you have a target that has mutually exclusive classes, it is straight forward to perform the classification using the sklearn package. But, for multi-label classification, we need a different approach. The experiments have been described below and code is available on GitHub.

Classification using Tf-Idf and Logistic Regression:

As the base model, we used Tf-Idf algorithm to convert the text abstracts in the training set to numbers. This method calculates the importance of a term by calculating the term frequency and inverse document frequency. The term frequency is the frequency of a particular word relative to the document. The inverse document frequency is the commonness of a word among all the documents in the corpus. In sklearn, we use smoothing, therefore idf is given as, idf(t) = log [(1 + n) / (1 + df(t))] + 1, where t is the term, n is the number of documents, and df(t) is the number of documents that contain the word t. Smoothing helps to avoid zero division problem, since not all documents contain the word t. Further, a few extra preprocessing steps, like lower casing, removing less frequent terms & stop words, and selecting only 20% of the most occurring words to form the vocabulary. We then build multi-label classifiers to classify our Tf-Idf features. The models have been discussed below:

a. OneVsRest Classification: This method assumes each label independent of each other, and builds a binary classifier for each category, where the interested label category is 'One' and all the remaining categories are considered as 'Rest'.
b. Classifier Chains: This method attempts to resolve the independence assumption of OnevsRest classifier by building one model for each class, but with a slight twist. The output predictions of each model act as input features for the next model in the chain, which helps the model consider the label correlations as shown below. We define a base logistic regression model that we will chain as shown in the figure above. In the end, we also build an ensemble model that is a combination of all the classifiers.

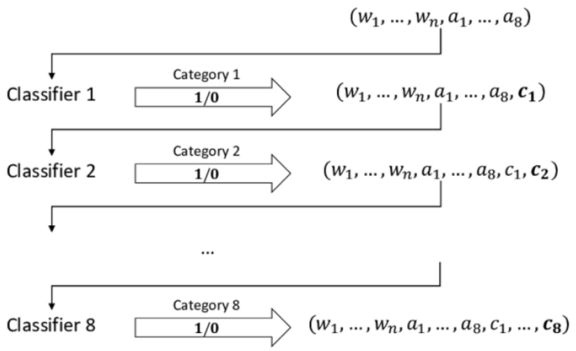

Fig 4. Chain Classifier

c. Label Powerset: This method builds a model for each combination of the label set. For example, if we have three classes {A,B,C} then we will have 8 models, i.e., 2^C, where C is number of classes, which in our case is three. So the 8 combinations are: {None,A,B,C,AB,AC,BC,ABC}. This method is not feasible for our case, because we have 15 classes, which is 2^15 = 32768 label classes, and not all combinations would have enough data to perform the classification.
d. Adaptive Algorithm: This method adapts the kNN classification algorithm to perform a multi-label classification. The idea is to find k nearest neighbors in the training set for an input test example using the kNN algorithm, then to use Bayesian inference to predict the output multi-label classification. This method is resource intensive, and for our problem yields poor results, but has shown excellent results when applied on smaller datasets.

Pretrained Word2Vec + Classifiers:

In this approach, instead of using a Tf-Idf, whose drawback is that it measures similarity in the word-count space and assumes independent evidence of similarity, we use a pretrained Word2Vec model called BioWordVec, which has been trained on 28,714,373 documents or about 4 billion tokens. We map each abstract text to a mean 200 feature vector and use these features to build multiple classifiers. We make sure to lowercase the sentence, remove stop words, and remove words that are not there in the word2vec model. We use Pycaret library to train classifiers for each root label category. The F1 score and Jaccard similarity score are used to compare the models. We will also save the models for future inference use.

As an example, let's start building the models for Mesh Category 'A'. Using Pycaret, we can build multiple models and choose the best one. From the following models for category 'A', we select the top 4 best models.

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lightgbm	Light Gradient Boosting Machine	0.8052	0.8869	0.7221	0.7874	0.7533	0.5928	0.5944	3.2490
lr	Logistic Regression	0.8023	0.8810	0.7071	0.7909	0.7466	0.5854	0.5878	1.6020
et	Extra Trees Classifier	0.8002	0.8802	0.6844	0.8015	0.7384	0.5784	0.5831	6.1200
rf	Random Forest Classifier	0.7991	0.8772	0.6918	0.7940	0.7394	0.5772	0.5808	23.8530

Fig 5. Top 4 model metrics

Transfer Learning:

The traditional machine learning models give a lot of pain when we do not have sufficient labeled data for the specific task or domain, we care about to train a reliable model. Transfer learning allows us to

deal with these scenarios by leveraging the already existing labeled data of some related task or domain. We try to store this knowledge gained in solving the source task in the source domain and apply it to our problem of interest. In this project, we have utilized transfer learning utilizing BIO-BERT model. We also applied Roberta for Sequence Classification.

Both the models have been implemented using PyTorch. The hyper-tuned Bert Sequence Classifier has 6 12 encoder stacks. Each encoder has an attention layer and 2 linear layers. The input Bert Embeddings are concatenated from word, position, and token type embeddings. Roberta builds on BERT and modifies key hyperparameters, removing the next-sentence pretraining objective and training with much larger mini-batches and learning rates.

The original training data is divided into 3 sets, train, validation, and test data. We hyper tune the model based on the evaluation of validation dataset. As expected, transfer learning gives us the best results on validation data. Figures 6 and 7 provide the comparison of Roberta and Bio-Bert Model.

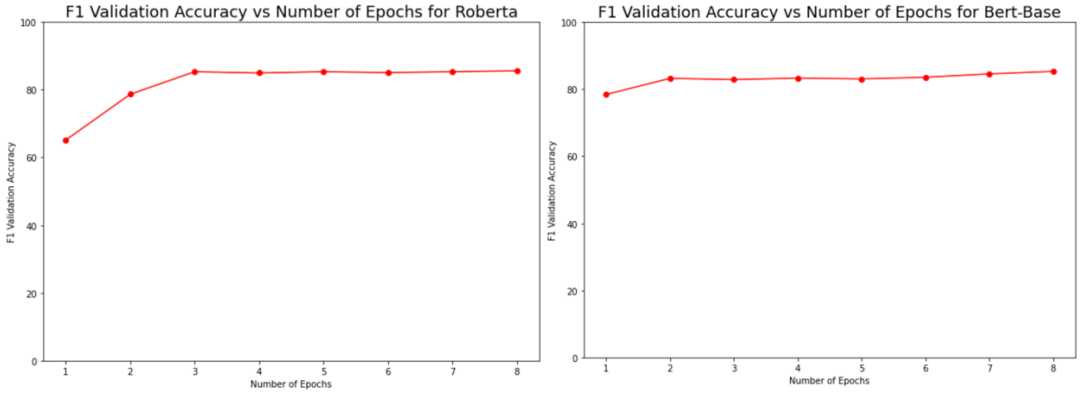

Fig 6 (left): Bio-Bert Validation. Fig 7 (right): Roberta Validation

Conclusions:

Compared with Binary Classification, the evaluation methods for multi-label classification are quite different. For the various techniques experimented above, the following metrics have been used to evaluate the performance in addition to accuracy score, micro and macro-f1.

Hamming loss: Average fraction of incorrect labels. (Perfect score is 0, Range is 0 to 1)
Jaccard similarity: Jaccard index, is the size of intersection of predicted labels and the true labels divided by the size of the union of the predicted & true labels. It ranges from 0 to 1, & 1 is the perfect score.
Levenshtein distance: Minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character. So here, Range = (0, 15) (#max=length of string)
Jaro Similarity: The value of Jaro distance ranges from 0 to 1, where 1 means the strings are equal and 0 means no similarity between the two strings. The main difference from Levenshtein distance is that Jaro-Winkler takes into account only matching characters and any required transpositions (swapping of characters). Also, it gives more priority to prefix similarity.

When using Tf-Idf to create numerical features for our base model, Logistic Regression, main 2 methods, OneVsRest & Classifier Chains, used for multi-label classification produce similar Jaccard index:

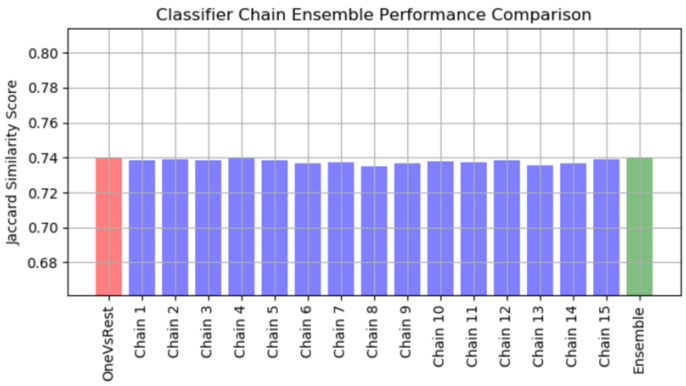

Fig 8: Comparison of OneVsRest and Classifier Chains

Below are the results from all experiments on 30% of original file (split into 70/30 for train and test). The low accuracy score is expected as the only the exact match of true and predicted label is counted as an accurate prediction. The macro-averaged F1 score is computed using the arithmetic mean of all the per-label F1 scores, is mainly used for multi-label classification. Unexpectedly, Tf-Idf (both with one vs rest and classifier chains) performed well looking at Jaccard Index. The Jaro mean is the average of Jaro scores of all records in test data. It is interesting to note the high Jaro Score, however since the Jaro score gives more weightage to prefix matching this pattern tells us that the former labels are being predicted well than the latter ones.

	Tf-Idf + OVR (LR)	Tf-Idf + Chain (LR)	w2v+LR	w2v+LGBM	w2v+RF	w2v+ET	w2v+Blend
Accuracy score	15.86%	16.33%	13.65%	13.99%	12.79%	12.65%	14.08%
F1 Macro	**71.38%**	**71.49%**	**68.87%**	**69.95%**	**65.15%**	**64.08%**	**68.02%**
F1 Micro	84.39%	84.39%	83.27%	83.42%	82.66%	82.48%	83.49%
Hamming_loss*	0.12%	0.12%	0.13%	0.13%	0.13%	0.13%	0.13%
Jaccard Index	**59.10%**	**59.20%**	**56.76%**	**57.59%**	**53.65%**	**52.81%**	**56.19%**
Jaro mean	92.51%	92.42%	92.18%	92.25%	92.01%	91.99%	92.27%
Lev mean*	1.76	1.98	1.88	1.87	1.94	1.95	1.85
Log loss*	46.16%	46.04%	48.38%	47.44%	49.80%	50.30%	47.76%

** Implies lower the better*

With transfer learning, we can test our models on a different test file. This test file contains 190k records. Below are the results for the same. As expected, Bio-Bert and Roberta perform better than the previous classifiers. F1 Scores and Jaccard Index have improved significantly. However, the results for both Roberta and Bio-Bert are similar.

	Accuracy Score	**F1 Macro**	F1 Micro	Hamming Loss*	**Jaccard Index**	Jaro Mean	Lev Mean*	Log Loss*
Roberta	16.91%	**75.32%**	85.29%	0.11%	**63.08%**	92.78%	1.69	42.21%
BERT	17.96%	**76.40%**	85.57%	0.11%	**64.17%**	92.03%	6.60	42.38%

** Implies lower the better*

Future Explorations:

Since we have extracted a large amount of data, more data can be used to train models on larger machines with more compute capacity. We could also explore using BioWordVec as a pretrained feature representation embedding for the transformer models (Bio-Bert and Roberta). This way we start with a good enough representation of the words.

Further, we could train classifiers for the sub-headings, and derive complex relationships between abstracts and the full MeSH tree structure. Each root category is further broken down into sub-categories, further subdivided into smaller ones. We could design the problem in such a way that we divide and conquer each category by building specialized classifiers that can take an abstract and perfectly generate the tree number. As a long shot extension to PubMed's exploration tool, we could also train a text generation model to help researchers author articles for categories. Finally, we could also perform Author Attribution task, where we dissect and analyze each Authors writing style, and add the functionality for researchers to explore each other's work.

Citations:

- RoBERTa: A Robustly Optimized BERT Pretraining Approach Paper (Link)
- Hugging Face: Roberta and Bert Overview paper (Link)
- Attention is all you need (Link)
- BioBERT: a pre-trained biomedical language representation model for biomedical text mining (Link)
- Multi-label Scientific Document Classification (Link)
- Efficient Estimation of Word Representations in Vector Space (Link)
- The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning) (Link)
- Semantic Sensitive TF-IDF to Determine Word Relevance in Documents (Link)
- Kevin Thomas, Rohan Paul, Mia Kanzawa PubMeSH: Extreme Multi-label Classification of Biomedical Research (Link)