DA5401: ML CONTEST

J Jayagowtham ME21B078,
Chaitanya V ED21B018

INITIAL THOUGHTS

After a first glance at the problem statement, it was discovered that the data is pretty huge for a single Machine Learning model to be trained - 2,00,000 embeddings of 1024 dimensions with 1400 labels each. We still were confident to crack the data problem with efficient data analysis. We went through a lot of research papers only to find out that deep learning models do pretty well on the ICD coding dataset. But since the course is focussed on ML models, we decided to dig deeper into the machine learning models that have proven to be working for the specific problem.

DATA INFERENCES

The data was thoroughly studied upon. The embeddings which were extracted from GatorTron were pretty much anonymous. But we decided to play around with the huge label space that we were presented with. We found that there were around 2500 embeddings which were duplicated and had contrasting labels. So, we dropped them from the dataset as we presumed that they are not significant to affect our model.The label space was huge and training a basic ML model on the dataset proved too much computationally and temporally intensive.So, we decided to do a stratified split on the dataset and do data inference on a smaller sample of (roughly around 40,000) data points whilst maintaining the same distribution of the labels. We used *IterativeStratification* from the *skmultilearn* library.

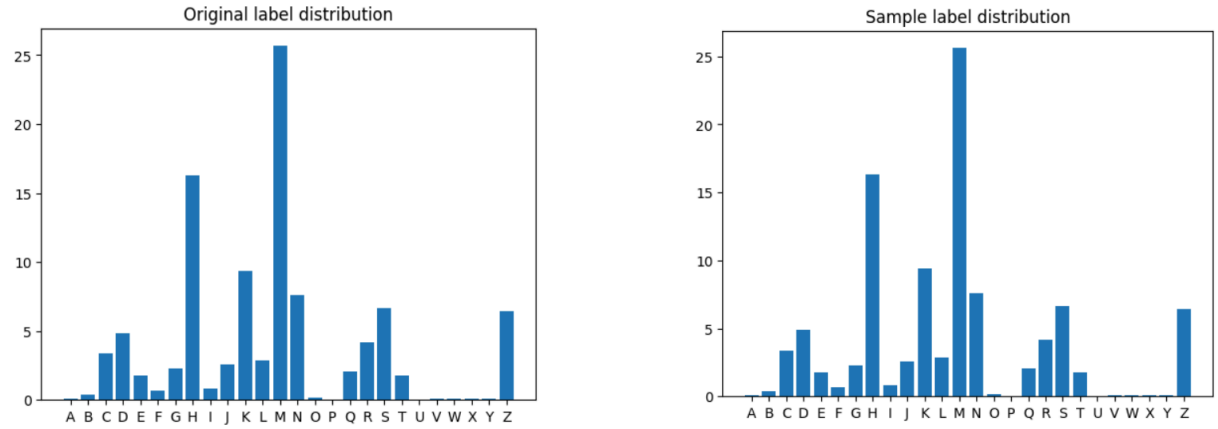

Fig 1: Label distribution of original data and sampled data with respect to letters (y-axis is the percentage of labels out of total)

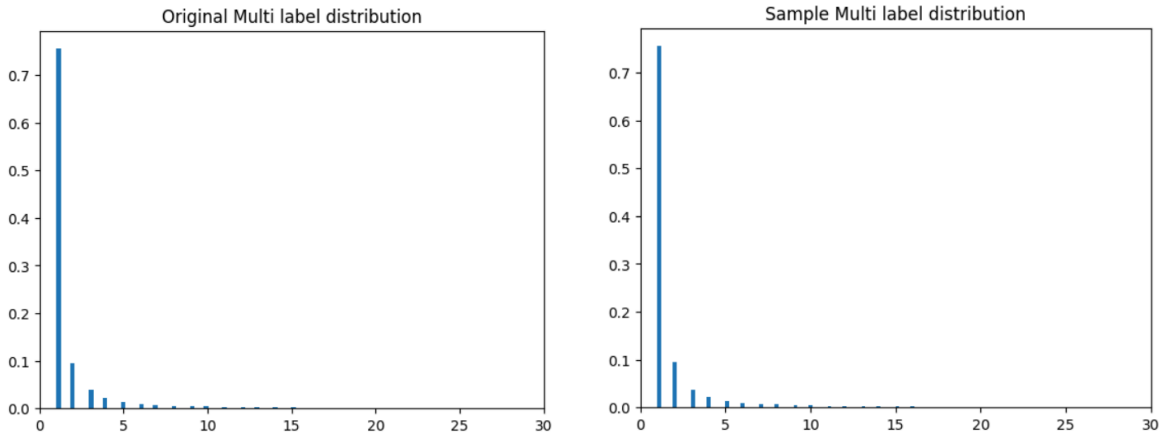

Fig 2: Multi Label distribution of original data and sampled data (x-axis is the number of data points that a multi label correspond to with the y-axis being the percentage of labels out of total)

From the above plots it is clear that there is heavy class imbalance among letters, but we presumed that since we are focused on optimizing the micro F2 score and not the macro version, even if we don't learn about the low frequency letters, we should still get a decent score. Also, around **70 percent** of multi labels only occur once. So, doing a label powerset method of training a classifier per multilabel even though computationally infeasible is also pointless since 70 percent of the time we will be training a model with only one datapoint.

We focussed more on the distribution of letters. A key observation is that the labels are hierarchical - they start with a letter and then have **5 to 6 numbers** with an extension being a letter or nothing. Upon eyeballing a sample and also getting inspired from a few papers, we decide that we will split the classification into two stages - predict the first letter and then correspondingly predict the multi label.

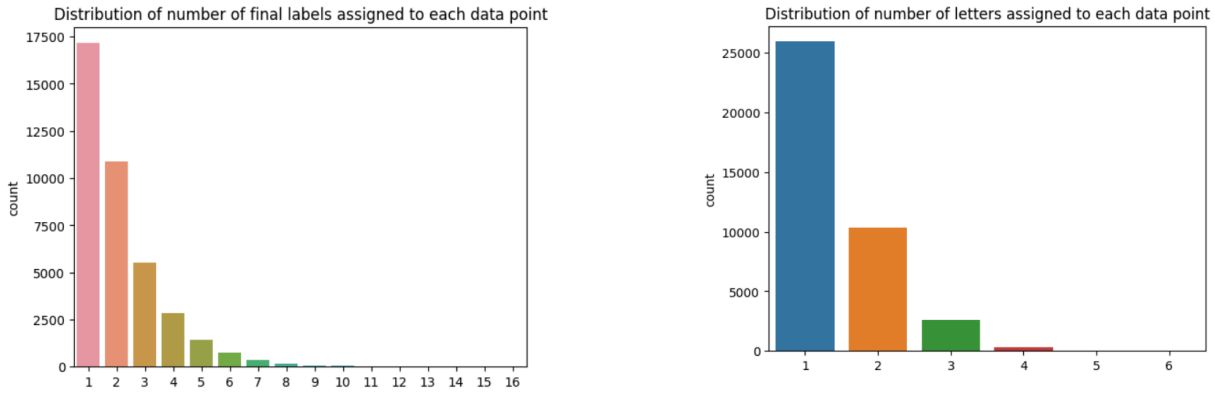

Fig 3: Number of labels(left) and letters(right) assigned to each data point

The key advantages of this idea is that one, it reduces the label space from **1400 to 24** ('P' and 'U' had no labels) in stage-1 and roughly around **300 multi labels** (228 for 'H') in stage-2 allowing our model to learn better by focusing on a sub-sample of labels. Second, after looking at the TSNE visualization aid from the professor, we hypothesized that we can model the second stage with binary relevance (training a binary classifier for each multi-label) as we will capture the label co-occurrence relationship in the first stage.

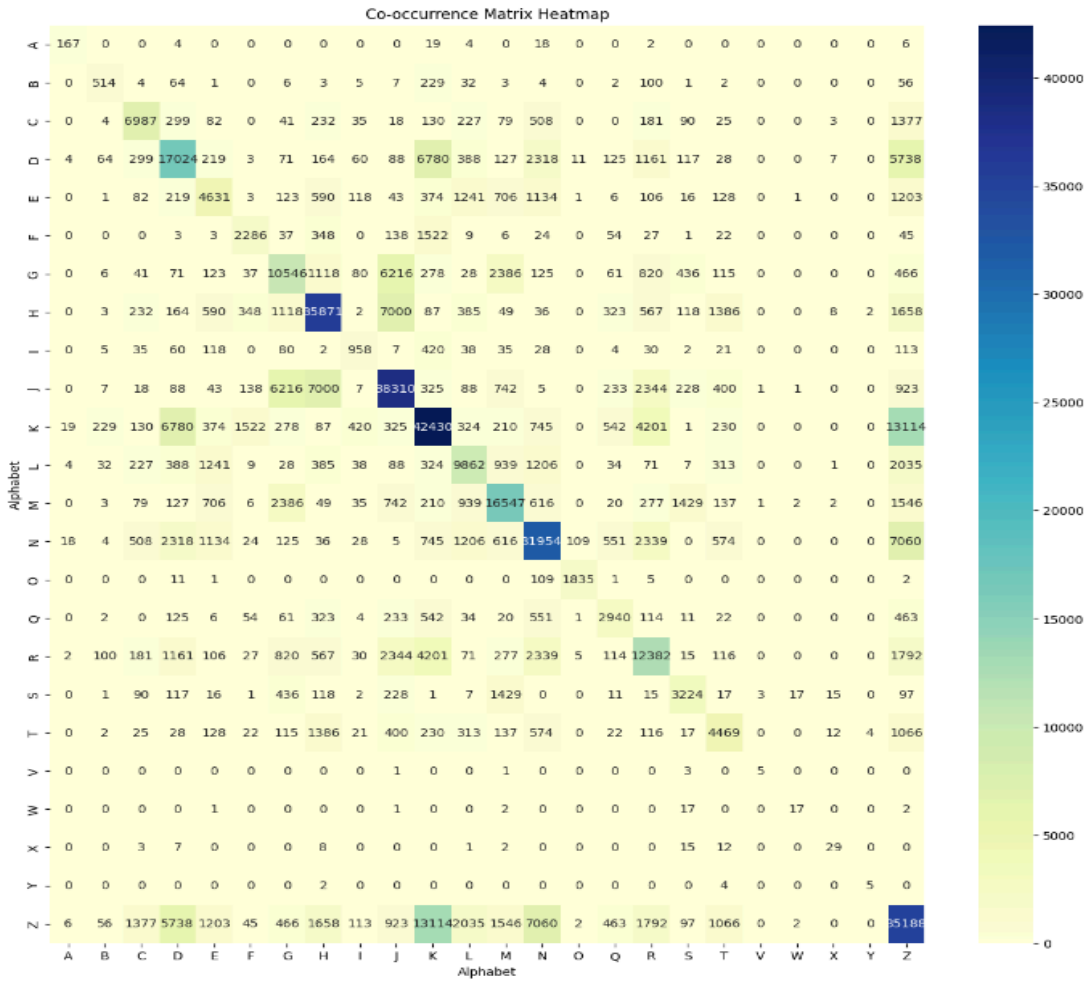

Fig 4: A very sparse co-occurence matrix indicating the possibility of capturing the label dependence in the first stage itself

We reduced the embedding space through PCA but since it was an embedding, it was already kind of a compressed representation in higher order, so it didn't help much. Even though our model was giving a good score on the validation set without tuning, it didn't perform as expected in the test set of the competition. We tried another approach where we will cluster the labels using **Kmeans** with each cluster representing a fixed number of representative letters and then for each cluster develop a model that can predict multi labels starting with the cluster representatives. From figure 3 (right) we can clearly see that with just **3 cluster representatives** we will be able to predominantly find all letters corresponding to a data point. Theoretically the number of clusters is $^{24}C_3$ but owing to the sparse occurrence of the majority of letters, we expected a good performance with as minimum as **20 clusters**.

In order to validate this method, we assumed that the second stage will retrieve all correct multi labels when we give it a cluster along with its representatives and we got the **micro f2 score** as **0.98**, **Hamming accuracy** as **0.99** and **Jaccard Accuracy** as **0.94**. To illustrate, if the data point's actual labels are A100, A200, B300, B400, C500, D600 and it is assigned to a cluster with representatives (A,B,D) then we assumed that the second stage can get us A100, A200,

B300, B400 and D600. We also trained a *RandomForestClassifier* (binary relevance) for the second stage with the same idea and received a **micro f2 score** as **0.89**, **Hamming accuracy** as **0.97** and **Jaccard Accuracy** as **0.64**. Even though we were missing out on less frequent letters, we were able to retain most of the micro f2 score as obtained theoretically thus prompting us to believe in this idea.

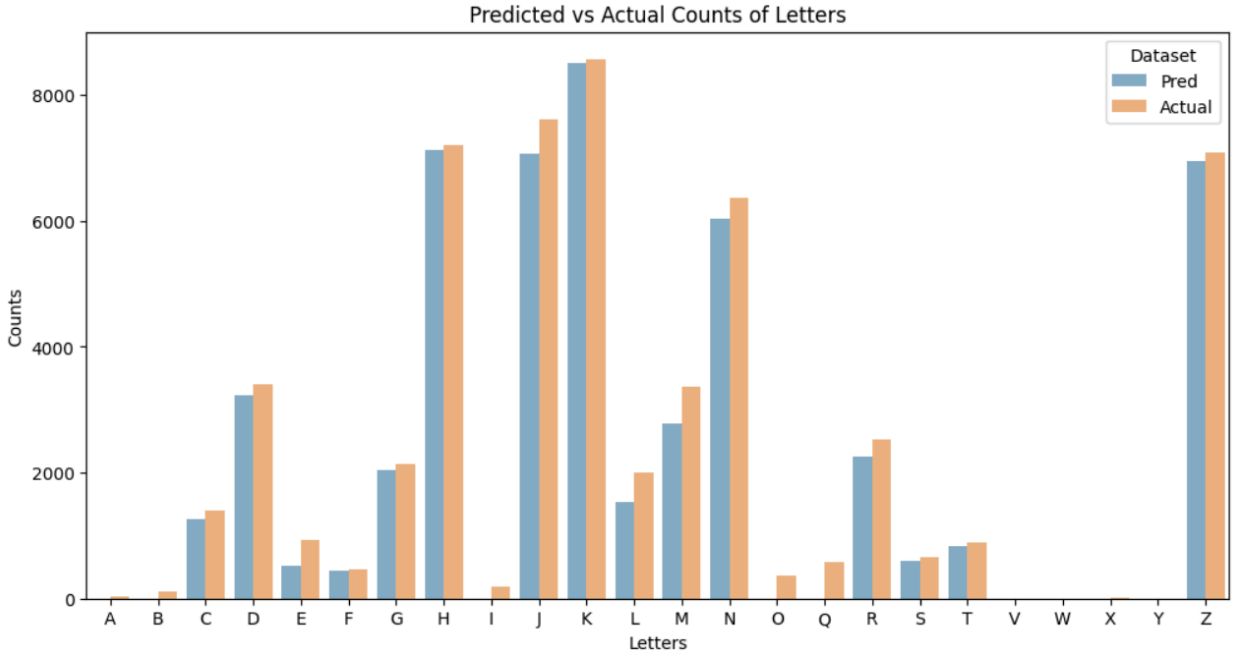

Fig 5: With the strong assumption of the 2nd stage, distribution of actual labels vs correctly predicted labels

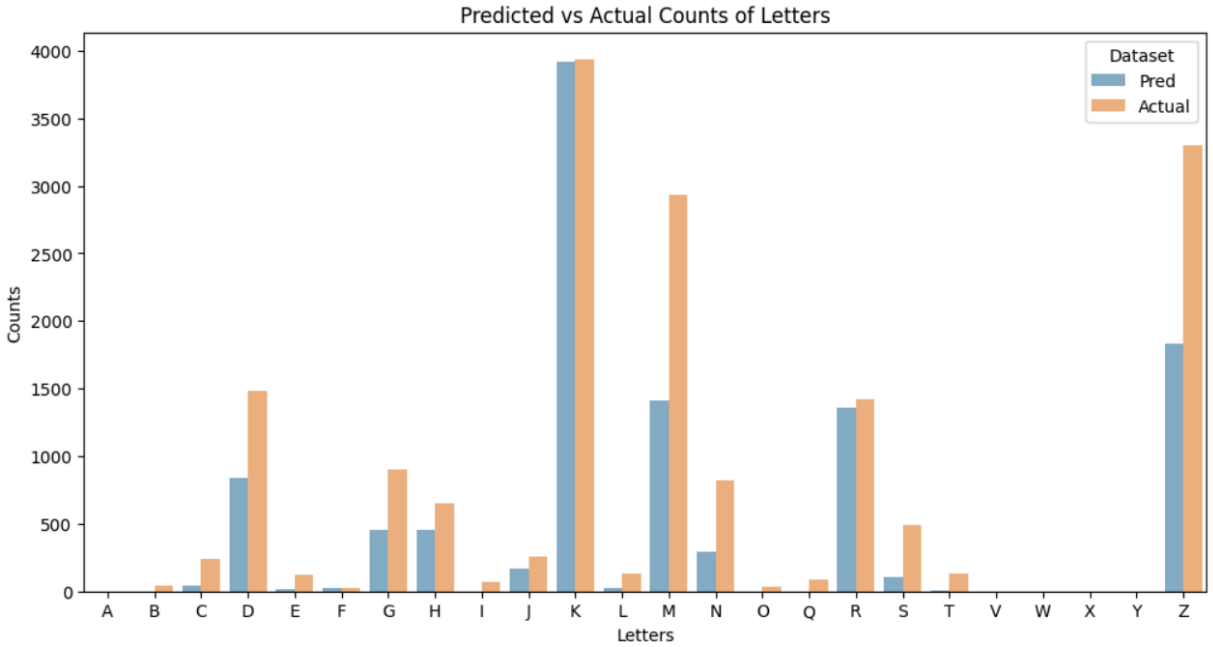

Fig 6: With a RandomForestClassifier for stage 2, distribution of actual labels vs correctly predicted labels

MODELS TRIED

1. **A Novel Two-Stage Approach for Complex Multilabel Classification: Random Forest for Letter Identification and XGBoost for Multilabel Prediction**
   To tackle the challenge of multilabel classification in this project, we were faced with an intriguing complexity: an overwhelming variety of label combinations, with frequent co-occurrences and several labels appearing together. Training a single model on such dense, intertwined data would likely lead to high computational demands, excessive training time, and potentially poor generalization. Our solution? A thoughtfully structured two-stage model that segments the problem into manageable layers while leveraging the strengths of distinct machine learning approaches at each stage.

   Stage 1: Letter Identification with Random Forest

   Upon analyzing the data, we observed a prominent pattern: certain labels (notably 'K') were more likely to appear together, with up to four labels frequently co-occurring. This insight led us to hypothesize that accurately predicting the primary or initial letter in a label sequence could help narrow the dataset to a smaller, focused subset for further analysis. This stratification would reduce the burden on the model, enabling it to learn finer details required for less common labels more effectively.

   For the initial letter identification, we selected **Random Forest** due to its inherent strengths like robustness to noise, feature importance insight and scalability.

   Stage 2: Multilabel Classification with XGBoost

   With the initial layer of complexity filtered out, we transitioned to the second stage — the multilabel classification of the remaining, potentially co-occurring letters. **XGBoost** was our model of choice here, chosen for its notable advantages in handling intricate, interdependent patterns.

   This two-stage, stacking-like model architecture provided an effective balance between **variance minimization** at the initial layer and **bias reduction** at the second. By focusing first on identifying the primary label and then specializing in multilabel classification, we crafted a pipeline capable of navigating the nuanced demands of this multilabel classification problem.

2. **Simple Neural Network (Both stages)**
   When we had to improve the train accuracy score more, we had to have a more complex model that could handle a vast dictionary of labels.

The architecture comprised 3 dense layers with 512, 256 and 128 neurons respectively, with ReLu activation, batch normalization, and a decreasing dropout from 0.4 to 0.3 at every layer to prevent overfit. Final output layer with sigmoid activation function, binary cross entropy loss and self defined f2_micro metric.

3. **BiGRU-NN model**

Going through research papers on the existing industrial solutions, an architecture from **Nature** journal caught our attention. Their classification model for ICD-10 coding leverages a four-layer neural network architecture that combines a recurrent neural network (RNN) with fully connected neural networks (FCNNs), designed specifically to handle the complexities of multilabel classification across extensive ICD-10 coding categories. The first layer of the model is a word embedding layer that transforms tokenized inputs into dense word vectors, capturing semantic relationships. This is followed by a **bidirectional GRU (BiGRU)** layer, which stands out as a powerful choice for this application, as explained below. Finally, two fully connected layers are employed, with the final layer's size tailored to the dimensionality of the classification task — 14,602 and 9,780 for CM and PCS labels, respectively, in the NTUH dataset.

Hyperparameters	Size
Bidirectional GRU[a] layer	256
Fully connected layer 1	700
Fully connected layer 2 CM/PCS[b]	14,602/9780
Dropout	0.2

Fig 7: Initial hyperparameters for the network we used

Why BiGRU Is an Optimal Choice for ICD-10 Classification

The **BiGRU layer** (bidirectional Gated Recurrent Unit) is an excellent fit for ICD-10 classification for several reasons. First, the **GRU's gating mechanisms** make it efficient in handling long sequences by selectively updating and forgetting information, which is essential given the complexity and context-heavy nature of medical text. GRUs require fewer parameters than LSTMs while still addressing the vanishing gradient problem, making them computationally efficient yet robust for tasks that need to remember contextual information over time.

The **bidirectional aspect** of BiGRU further enhances its suitability for ICD-10 classification. By processing information in both forward and backward directions, BiGRU captures contextual dependencies from both past and future tokens within the

sequence. This is crucial for medical text, where the context from surrounding words significantly impacts the meaning of individual terms and ultimately determines the correct ICD-10 code. For instance, "pain" may take on vastly different meanings based on preceding or following terms, such as "chronic," "acute," or "localized." A unidirectional RNN would struggle to capture this bidirectional dependency as effectively, making BiGRU a clear choice for maximizing the contextual understanding needed for ICD-10 classification.

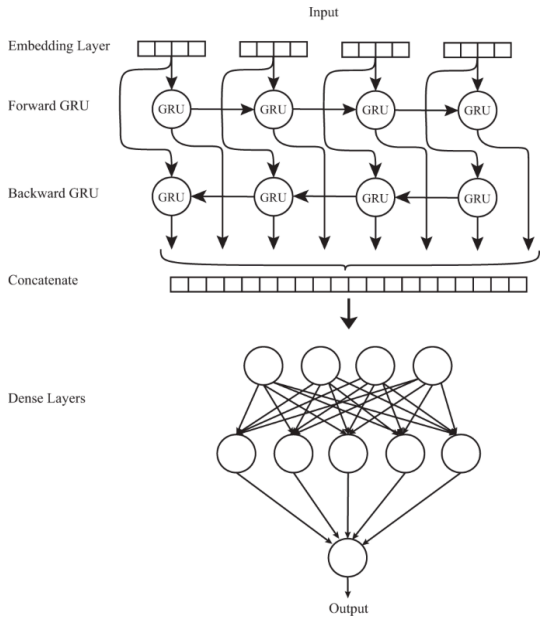

Fig 8: BiGru architecture overview

4. **Class-balanced dataset on BiGRU-NN model**
   Since some lowly represented letters had around 0.7 validation f2 score, we decided to improve the data balance using SMOTE, oversampling specific letters that we found underperforming like ‘A, B, I, V, W, X’

   Each of the letter representations was increased from about 20 to 500. But surprisingly, this did not increase the performance on the test set, which brought us to the conclusion that the test set representation doesn't have a major oversample of the least seen data, or the SMOTE method of synthetic data points did not add value.

5. **Attention mechanism on BiGRU-NN model**
   To further refine the model's capability, we incorporated a **Bahdanau attention mechanism**, which enables the model to dynamically focus on key segments of input text by assigning different weights to context pairs (ICD title–vector pairs). This approach enhances the model's performance by identifying and emphasizing keywords and phrases critical for accurate ICD-10 coding. By selectively attending to information that most

contributes to classification accuracy, the model better handles the vast label space and fine-grained distinctions required for ICD-10 classification. Adding a (64,64) context vector after a 256 BiGRU was expected to improve performance.

6. **Co-occurrence clustering**
   We later thought co-occurrence can be extended to more than just 'starting letter' but also with other labels that maybe co-occurring, that could reduce the number of models to train, and for better recall. Refer to Fig.4 for pictorial representation.
   Hence we clustered a few letters together, as some single letters had less than 10 corresponding data points. and lesser data isn't good for neural networks to learn. We clustered ['N','A'], ['K', 'B', 'I'] , ['S','V','W','X'] and ['T','Y'] with the first letter in each group being the representative in the final prediction.

FINAL PERFORMANCE

Our final model is a 2 stage BiGru-NN architecture with co-occurrence clustering in the second stage with the data free of repeated embeddings .

Stage	Info	Train Micro F2	Valid Micro F2
1	5 epochs, 128 batch size	0.951	0.925
2 20 epochs, 128 batch size	['C']	0.978	0.881
	['D']	0.985	0.834
	['E']	0.988	0.920
	['F']	0.906	0.831
	['G']	0.969	0.929
	['H']	0.903	0.750
	['J']	0.957	0.867
	['K', 'B', 'I']	0.979	0.865
	['L']	0.955	0.700
	['M']	0.973	0.847
	['N', 'A']	0.975	0.826
	['O']	0.992	0.977

	[‘Q’]	0.995	0.960
	[‘R’]	0.990	0.881
	[‘S’,’V’,’W’,’X’]	0.967	0.878
	[‘T’,’Y’]	0.972	0.783
	[‘Z’]	0.991	0.885
Final	Combined performance	0.955	0.838

REMARKS

We firmly believed in the idea of a machine learning model working wonders for this problem statement. But either due to our implementation not optimal or the problem isn’t intended to be solved using ML models, our performance wasn’t up to the mark. Since we were running out of time and ML models were consuming so much time on the huge dataset, we ultimately had to resort to neural networks which utilized Kaggle’s GPU enabling us to iterate faster. We also believe we could have approached this challenge better provided we knew about the working of neural networks which was not covered in this course. We are still curious to know whether any of our friends’ machine learning based approach worked and are eager to learn and identify our mistakes. We put in a lot of effort for this contest and it was both enjoyable and rewarding. We thank the DA5401 team for wonderfully facilitating the contest.

REFERENCES AND ACKNOWLEDGEMENTS

- https://xang1234.github.io/multi-label/ illustrating basic usage of skmultilearn
- https://machinelearningmastery.com/multi-label-classification-with-deep-learning/
- https://en.wikipedia.org/wiki/Multi-label_classification
- *Automatic multi label detection of ICD10 codes in Dutch cardiology discharge letters using neural networks - Arjan Sammani, Ayoub Bagheri, Peter G. M. van der Heijden, Anneline S. J. M. te Riele, Annette F. Baas, C. A. J. Oosters, Daniel Oberski & Folkert W. Asselbergs*
- *Boosting ICD multi-label classification of health records with contextual embeddings and label-granularity - Alberto Blanco, Olatz Perez-de-Viñaspre, Alicia Pérez, Arantza Casillas*
- *Automated ICD coding using extreme multi-label long text transformer-based models - Leibo Liu, Oscar Perez-Concha, Anthony Nguyen, Vicki Bennett, Louisa Jorm*
- *Enhanced ICD-10 code assignment of clinical texts: A summarization-based approach*
- *Yaoqian Sun, Lei Sang, Dan Wu, Shilin He, Yani Chen, Huilong Duan, Han Chen, Xudong Lu*
- Thanking Chat-GPT and Perplexity-AI for their search accumulation skills