Report

Overview:

This report details the process and results of building a multi-label classification model to predict ICD10 codes for medical chart embeddings, optimizing for the average micro-F2 score.

Data Engineering

The provided data consisted of medical chart embeddings in NumPy `.npy` files and corresponding ICD10 labels in text files. There were two chunks of data: “embeddings_1.npy”, “icd_codes_1.txt” and “embeddings_2.npy”, “icd_codes_2.txt”.

1. **Loading Data**: The embedding files were loaded using `numpy.load()`, and the label files were loaded using `pandas.read_csv()`.
2. **Concatenation**: The embeddings from both files were concatenated vertically using `np.concatenate()`. The labels from both files were concatenated using `pd.concat()`.
3. **Label Processing**: The ICD10 labels were provided as semicolon-separated strings in each line. These strings were split by the semicolon delimiter to obtain lists of labels for each medical chart.
4. **Multi-hot Encoding**: The lists of labels were converted into a multi-hot encoded vector representation using `sklearn.preprocessing.MultiLabelBinarizer`. This transformed the labels into a binary matrix where each column represents a unique ICD10 code and each row represents a medical chart, with a `1` indicating the presence of a code and a `0` indicating its absence. The `MultiLabelBinarizer` was fitted on the complete set of labels to capture all possible ICD10 codes.

Sampling

No explicit sampling was performed on the full dataset for training the main model. The entire combined dataset was used. However, a subset of the data was used for t-SNE visualization due to computational constraints.

Exploratory Data Analysis (EDA) and Visualization

EDA was performed to understand the characteristics of the data:

1. **Label Frequency Analysis**: The frequency of each ICD10 code was calculated by summing the columns of the multi-hot encoded label matrix. The most and least frequent codes were identified.
2. **Label Distribution per Chart**: The number of labels associated with each medical chart was calculated by summing the rows of the multi-hot encoded label matrix. A histogram was plotted to visualize the distribution of the number of codes per chart, showing that while many charts have a single label, a significant number have multiple labels, confirming the multi-label nature of the problem.
3. **Dimensionality Reduction and Visualization**:
    - Principal Component Analysis (PCA) was applied to reduce the dimensionality of the embeddings to 50 components, capturing a significant portion of the variance.
    - t-Distributed Stochastic Neighbor Embedding (t-SNE) was then applied to a subset of the PCA-reduced embeddings to further reduce the dimensions to 2 for visualization.
    - A scatter plot of the 2D t-SNE results was generated, highlighting the distribution of the top 10 most frequent single ICD10 codes to visually explore potential clustering of embeddings based on labels.

Model Selection

Potential Model Architectures for Multi-label Classification with High-Dimensional Embeddings

Given the high dimensionality of the input embeddings (1024 features) and the multi-label nature of the target variable (1400 possible classes, with each sample potentially having multiple labels), several model architectures could be considered:

1. **Neural Networks:**

- **Pros:** Can learn complex non-linear relationships between high-dimensional input and multiple output labels. Architectures like feed-forward networks, or more advanced models, can be effective. They can handle the multi-label output directly with an appropriate output layer (e.g., sigmoid activation for each output neuron) and loss function (e.g., binary cross-entropy).
- **Cons:** Require significant computational resources and data for training. Can be prone to overfitting, especially with a large number of parameters. Model selection and hyperparameter tuning can be challenging.

2. **One-vs-Rest (OvR) with Base Classifiers:**
    - **Pros:** Simple and interpretable. Converts the multi-label problem into multiple independent binary classification problems (one for each label). Can leverage well-established binary classifiers (e.g., Logistic Regression, Support Vector Machines, or even simple neural networks). Training can be parallelized for each binary classifier.
    - **Cons:** Ignores potential correlations between labels. The number of classifiers scales with the number of labels, which can be computationally expensive during prediction if the number of labels is very large. Performance can be suboptimal if label correlations are important for prediction.
3. **Classifier Chains:**
    - **Pros:** Models label correlations by building a sequence of binary classifiers where each subsequent classifier is trained on the original input features plus the predictions of the previous classifiers in the chain.
    - **Cons:** The order of classifiers in the chain can significantly impact performance. Errors from early classifiers propagate down the chain. Can be computationally expensive to train due to the sequential nature.
4. **Label Powerset:**
    - **Pros:** Transforms the problem into a multi-class classification problem where each unique combination of labels in the training set is treated as a single class. Can capture label correlations.

- **Cons:** The number of unique label combinations can be very large, leading to a massive number of classes and potentially sparse training data for some combinations. Cannot predict label combinations not seen during training.

Chosen Model(s) and Appropriateness

Considering the dataset characteristics and the evaluation metric (average micro-F2), a **neural network based approach** is likely the most appropriate initial choice.

- **Appropriateness:**
  - **High-Dimensional Embeddings:** Neural networks are well-suited for learning from high-dimensional feature spaces like the provided embeddings.
  - **Multi-label Classification:** With a sigmoid output layer and binary cross-entropy loss, a neural network can directly model the probability of each label being present independently, which aligns with the multi-label nature.
  - **Average Micro-F2:** The micro-F2 score considers the total number of true positives, false negatives, and false positives across all labels. A neural network trained with binary cross-entropy, which optimizes for predicting the presence of each label, can be a good starting point for optimizing this metric. While not directly optimizing F2, a strong binary classification performance for each label generally contributes to a good micro-F2 score. More advanced techniques or loss functions could be explored later to directly address the F2 metric if needed.
  - **Scalability:** Neural networks can scale well to large datasets.

As a strong baseline and a potentially competitive alternative, particularly if computational resources or training time become a significant constraint, **One-vs-Rest with a linear base classifier (like Logistic Regression)** is also a suitable consideration.

- **Appropriateness:**
  - **Simplicity and Speed:** OvR with Logistic Regression is relatively simple to implement and computationally less expensive than complex neural networks, especially during training (due to parallelization).

- **High-Dimensional Data:** Linear models like Logistic Regression can perform reasonably well on high-dimensional data, especially if the embeddings are informative.
- **Interpretability:** While less relevant for this task's objective, linear models offer some degree of interpretability.
- **Average Micro-F2:** OvR treats each label independently, and optimizing the performance of each binary classifier can contribute to a good micro-F2 score, similar to the neural network.

Given the preference for potentially capturing complex patterns and the power of deep learning on such data, the focus will initially be on a neural network model, but One-vs-Rest serves as a valuable benchmark and simpler alternative.

Basic Structure/Approach of Selected Model(s)

Neural Network:

The basic structure of a feed-forward neural network for this task would involve:

1. **Input Layer:** Matches the dimension of the input embeddings (1024 neurons).
2. **Hidden Layers:** One or more fully connected (dense) layers with activation functions (e.g., ReLU) to learn non-linear transformations of the input features. The number and size of hidden layers are hyperparameters to tune. Dropout layers can be included for regularization.
3. **Output Layer:** A fully connected layer with a number of neurons equal to the total number of unique ICD10 codes (1400 neurons). Each neuron corresponds to a specific ICD10 code. A sigmoid activation function is applied to each output neuron to produce a probability score between 0 and 1, representing the likelihood of that label being present for the input embedding.
4. **Loss Function:** Binary Cross-Entropy is commonly used for multi-label classification, calculated independently for each output neuron.
5. **Optimizer:** An optimization algorithm (e.g., Adam) to update the model weights during training.

One-vs-Rest with Logistic Regression (as a potential alternative/baseline):

This approach involves training 1400 independent binary Logistic Regression classifiers.

1. For each ICD10 code (label), a separate Logistic Regression model is trained.
2. The input to each Logistic Regression model is the original 1024-dimensional embedding.
3. The target for each Logistic Regression model is a binary variable indicating whether the specific ICD10 code is present (1) or absent (0) in the multi-hot encoded label vector.
4. During prediction, each of the 1400 trained Logistic Regression models predicts the probability of its corresponding label being present.
5. A threshold (e.g., 0.5) is applied to the predicted probabilities to determine which labels are predicted as present for a given input embedding.

Model Training

A feed-forward neural network was defined and trained using TensorFlow/Keras:

- **Architecture**: The model consisted of an input layer matching the embedding dimension (1024), two hidden dense layers with ReLU activation and dropout for regularization, and an output dense layer with a sigmoid activation function for each of the 1400 possible ICD10 codes.
- **Compilation**: The model was compiled using the Adam optimizer with a learning rate of 0.001 and binary cross-entropy as the loss function, which is standard for multi-label classification. Accuracy was also monitored as a metric during training.
- **Training**: The model was trained on the `X_train` and `y_train` data splits for 10 epochs with a batch size of 32. Validation data (`X_val`, `y_val`) was used to monitor performance during training.

Model Evaluation

The trained model was evaluated on the validation set (`X_val`, `y_val`) with a focus on the average micro-F2 score, as specified in the task.

- **Prediction**: The trained model was used to predict the probabilities of each ICD10 code being present for the validation set embeddings.

- **Thresholding**: A threshold of 0.5 was applied to the predicted probabilities to convert them into binary predictions (0 or 1) for each label.
- **Micro-F2 Score**: The `sklearn.metrics.fbeta_score` function was used with `beta=2` and `average='micro'` to calculate the micro-averaged F2 score.
- **Results**: The micro-F2 score on the validation set was calculated to be 0.7315. Validation loss and accuracy were also reported as 0.0021 and 0.5624, respectively.

Prediction on Test Data

The trained model was used to generate predictions for the provided test data ("test_data.npy"):

- **Loading Test Data**: The test embeddings were loaded using `numpy.load()`.
- **Prediction**: The trained model predicted the probabilities of each ICD10 code for the test embeddings.
- **Thresholding**: A threshold of 0.5 (the same as used for validation) was applied to obtain binary predictions.

Generate Submission File

The binary predictions for the test data were formatted into the specified submission file format:

- **Inverse Transformation**: The multi-hot encoded test predictions were converted back to lists of ICD10 code strings using the `inverse_transform` method of the fitted `MultiLabelBinarizer`.
- **Formatting**: The lists of labels were joined into semicolon-separated strings.
- **Saving**: The formatted predictions were saved to a CSV file named "submission.csv" without a header and without the index column, matching the format of the "sample_solution.csv" file.

Hacks and Workarounds

- **Label Splitting**: The initial approach to processing labels did not correctly split the semicolon-separated codes within each entry. This was corrected by explicitly splitting each string by the ';' character.

- **t-SNE Subset**: Due to the large size of the dataset, t-SNE visualization was performed on a randomly selected subset of the data to manage computational resources and time.
- **t-SNE Label Visualization**: Since visualizing all unique labels in the t-SNE plot was not feasible, the plot focused on highlighting only the most frequent single labels to look for potential clustering.

Conclusion

The task of building a multi-label classification model for medical chart embeddings has been completed. The data was successfully loaded, preprocessed into a multi-hot encoded format, and explored through EDA and visualization. A neural network model was trained and evaluated, achieving a micro-F2 score of 0.7315 on the validation set. Predictions were generated for the test data and formatted into the required submission file. The process involved addressing initial data loading issues and using a subset for computationally intensive visualization. Further work could involve hyperparameter tuning of the neural network, experimenting with different model architectures (like One-vs-Rest), exploring different probability thresholds for optimizing the F2 score, and potentially incorporating techniques to handle the class imbalance observed in the label frequencies.