**REPORT**

**Objective**

The objective is to present a modular implementation of a BERT-based machine learning model customized for clinical prediction tasks using the MIMIC III dataset, an electronic health record (EHR) dataset. The implementation includes comprehensive steps for data preprocessing, model construction, and training. Evaluation of the model's performance is conducted using relevant evaluation metrics to assess its effectiveness.

**Dataset Preparation and Preprocessing**

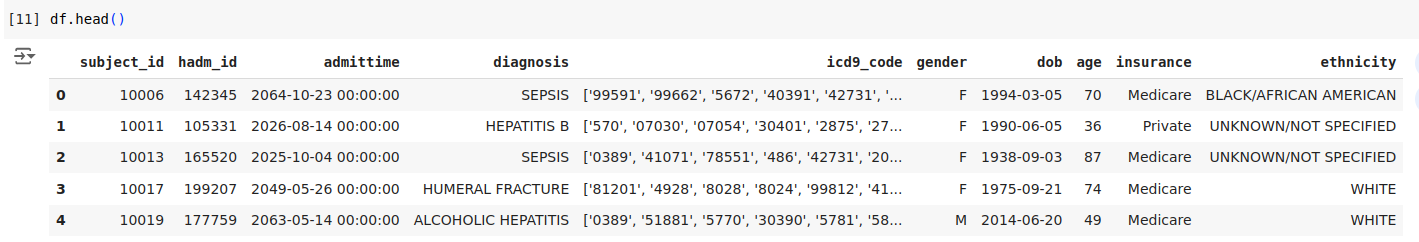
*Dataset used:* MIMIC III (<https://www.kaggle.com/datasets/asjad99/mimiciii>)

*Dataset Preparation:*

* Used the files: ‘ADMISSIONS.csv’, ‘DIAGNOSES\_ICD.csv’, and ‘PATIENTS.csv’
* Merged and grouped these CSV files based on the ‘subject\_id’, and the ‘hadm\_id’.
* Stored the icd9 codes for a specific visit in a list and then in the column ‘*icd9\_code’.*
* Calculated the age of each ‘*subject*’ by subtracting the year of birth (*dob*) from the year of admission (*admittime*).
* Columns included in the merged dataset: *'subject\_id'*, *'hadm\_id'*, *'admittime'*, *'diagnosis'*, *'icd9\_code'*, *'gender'*, *'dob'*, *'age'*, *'insurance'*, *'ethnicity'*.
* Dataset size increased for training and validation purposes. This is done considering the following:

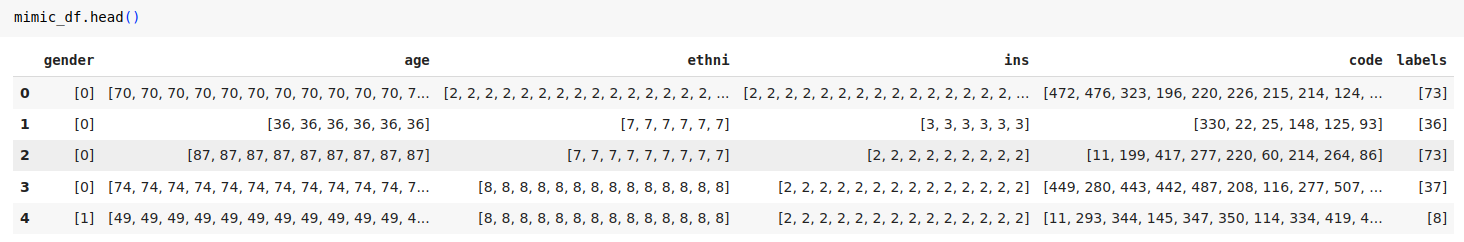
1. There are multiple visits of the same patient; they’ll have the same demographic details.
2. Patients can have different diagnoses during their next visit, but the sequence of icd9\_codes should be the combination of icd9 codes of the respective diagnosis from previous examples.
3. There can be new patients with random (from our dataset) demographics and diagnosis (in the same way as before).

*Dataset cleaning:* Removed patients which had age greater than 100.



*Dataset Preprocessing:*

* Label encoded the categorical columns (*'diagnosis'*, *'gender'*, *'insurance'*, and *'ethnicity'*) and converted them into numerical values.
* ICD9-Codes were processed to extract unique codes and label encoded accordingly.
* Filtering the dataset by selecting only the necessary columns.
* Columns were expanded to ensure that each categorical value was repeated appropriately, aligning with the length of the code list. This ensures consistency and prepares the data for model input.



Dataset splitted into 80-20 ratio for training and evaluation purposes.

**behrt\_model.py**

1. **BertEmbeddings Class:** This class creates representations (embeddings) for different pieces of patient information like words, age, gender, ethnicity, and insurance type. It combines these embeddings, normalizes them, and applies a technique to prevent overfitting, making the model robust to various inputs.
2. **BertModel Class:** This builds on the BertEmbeddings class by adding layers that process these embeddings to understand complex patterns in the data. It uses parts of the BERT model (a powerful text processing model) to do this, ensuring the model can learn from the EHR data effectively.
3. **BertForEHRPrediction Class:** This adapts the BertModel for specific medical prediction tasks. It adds a final layer to make predictions about patient outcomes.
4. **BertConfig and TrainConfig Classes:** These classes manage configuration settings. BertConfig stores settings related to the BERT model, like the size of different embeddings. TrainConfig stores settings for the training process, such as batch size (how many patient records to process at once), device settings (like using a GPU), and file paths for saving results.
5. **DataLoader Class:** This class handles loading and preparing the data for training. It reads patient information from a dataframe and processes it to be in the right format for the model. It also includes helper functions to pad sequences (make them the same length), and generate positional and segment indices needed for the model.

**Dataloader and Model Initialization**

* The BehrtDataLoader prepares the dataset such that it is ready to be fed into a BERT-based model for training or inference.
* It ensures that all sequences in the dataset have the same length (maximum sequence length) by adding zeros to shorter sequences.
* Masks are created to distinguish between actual data and padded data within sequences. These masks help the model focus only on real data elements during training and ignore padded values, improving efficiency and accuracy.
* Position (indicate the position of each token in the sequence) and segment (differentiate between different segments in the input data) indices are generated for each input data sequence. The sequence indices help the model to understand which parts of the input belong to which sequence or segment.
* The *BertEmbeddings* Class converts input data into dense vectors and generates embeddings for age, gender, ethnicity, insurance, and positional information using sinusoidal encoding. This class also applies layer normalization to standardize embeddings and dropout to prevent overfitting during training.
* The *BertModel* Class integrates embeddings from BertEmbeddings with BERT's transformer encoder to capture intricate patterns in the dataset.
* The model is initialized using the*BertForEHRPrediction* Class which extends the BertModel to include a dropout layer for regularization and a classification layer to produce logits for clinical prediction tasks.

**Model Training**

* The model is prepared for training by setting up the CrossEntropyLoss loss function and the Adamoptimizer.
* Within each epoch, a batch of data consisting of input features (IDs and embeddings for text and categorical data) and their corresponding labels are retrieved from the dataset.
* Before processing each batch, the optimizer is reset to clear out gradients from the previous batch.
* During the forward pass, the model computes predictions based on the input data, aiming to generate accurate outputs.
* After making predictions, the model checks how accurate its predictions are by comparing them to the actual results, calculating a measure of how well it's doing.
* Gradients of the loss function with respect to model parameters are calculated, facilitating adjustments to these parameters. This iterative process aims to optimize the model's performance by minimizing the loss.
* Throughout the epoch, the total loss incurred from all batches is summed up to give an overall measure of how well the model is performing.
* At the end of each epoch, the average loss per batch is computed, providing a standardized metric to assess the model's performance across different parts of the dataset.
* After the training, the final model is saved at a specified directory for further testing purposes.

**Model Testing**

* The trained model is loaded from the file located at the specified directory.
* A custom function is defined to calculate various evaluation metrics such as accuracy, precision, recall, F1-score, and confusion matrix based on model predictions and true labels.
* The model is set to evaluation mode to disable dropout and batch normalization layers which behave differently during training.
* For each batch of data in the test dataset, input tensors are moved to the appropriate computational device (e.g., GPU) and model predictions are generated (‘logits’) using the input data.
* Logits (raw model outputs) and corresponding labels from each batch are collected into lists.
* Collected logits and labels are concatenated into single tensors to facilitate comprehensive metric evaluation across the entire test dataset.
* The custom function calculates and returns key evaluation metrics such as accuracy, precision, recall, F1-score, and the confusion matrix based on the aggregated predictions and labels.

**Results**

For 10 epochs, Model’s Average training Loss:

Evaluation Metrics Values:

| Test Accuracy | 96.22% |
| --- | --- |
| Precision | 0.9685 |
| Recall | 0.9622 |
| F1 Score | 0.9590 |