Introduction

This project used a machine learning pipeline to perform classification tasks for the PubMed abstracts. When the text abstracts were passed to the model, it was expected to output the correct label of the abstract.

Method

1. Data exploration and preprocessing

Pandas were used to explore the data and extract the relevant columns (label and text) into a data frame. The shape of the data frame and the presence of null values were examined. There were a total of 27074 samples and 21 classes in the dataset. There was one null value in the text column, and this row was subsequently dropped.

The unique labels were extracted using the .unique() function and stored in a dictionary with indices (0~20) as keys and unique labels as values. The original labels were then replaced with the corresponding keys in the dictionary. Converting the labels makes it easier for the loss calculations in the later steps.

By plotting the frequencies of the classes using `.value_counts()` and `.plot()` functions of pandas, it was observed that the dataset is imbalanced. For example, there were approximately 4,000 (~15%) samples for class 17 ('Retinal Diseases'), but only less than 250 samples for class 15 ('Pupil Disorders') and class 4('Eye Hemorrhage') each.

The `parse_text` function was created to process the text data. This function performed the following steps:

- Converted the entire string to lowercase
- Used the NLTK regex tokenizer to tokenize the string and included only words
- Removed stop words by looking up tokens in the NLTK stop words collection
- Added the `<sos>` and `<eos>` tokens at the beginning and end of the sentence, respectively

The function was applied to the entire text column, and the cleaned tokens were stored in a new column.

The lengths of the tokens were analyzed and sorted in descending order. The statistical analysis of token lengths revealed that its distribution was right-skewed, with a mean of 128.04, a median of 116.00, and a standard deviation of 62.31. Rows with token lengths less than 32 or greater than 256 were dropped, as they represented small portions of the data, and removing them had minimal impact on the balance of labels, as shown by plotting the bar graphs. The trimmed dataset had 26102 samples in total. Finally, all the tokens were padded to 256 using back padding to enable the use of the `pack_padded_sequence()` function of PyTorch in model building.

2. Helper functions

Several functions are created to facilitate model training and evaluation.

K-fold was implemented by using the `k_fold_indices()` and `get_k_fold_data()`. The `k_fold_indices()` function splits the indices of cleaned data into n folds and returns n groups of indices for train, validation, and test datasets. The `get_k_fold_data()` function was used to retrieve the corresponding datasets based on the indices and convert them into NumPy arrays.

For handling tokens efficiently, a vocab class was created. This class could iterate through all tokens in the train dataset and extract and store the unique tokens. The minimum frequency of tokens could be specified to be considered a unique token and passed into the vocab class. The vocab class could also perform conversions between indices and tokens, as the inputs for the model must be a list of indices rather than words. When converting, if a token is not in the vocab class, it will be converted to `<unk>`.

A data iterator class was developed to feed the data to the model in specified batch sizes. This class took a data frame as input and outputted batched tokens (converted into indices by vocab class), labels, and token lengths as PyTorch tensors.

The `get_embed_matrix()` function was created to extract the vectors for unique words in the vocab class from pre-trained word embedding models and create a new embedding matrix to be loaded into the embedding layer of the model. Despite the higher cost in computing resources, the `word2vec google news 300` model was selected as the pre-trained embedding model for this pipeline due to its higher dimensionality (300) and slightly higher accuracy in model evaluation compared to the `fasttext en 100` and `glove 6B 50D models`

3. Model

The model was constructed with three layers: an embedding layer, a gated recurrent unit (GRU) layer, and a fully connected layer. The model took two inputs: a batch of lists of indices representing tokens and the lengths of the lists that exclude paddings. The embedding layer reads the model's input and looks up the corresponding vectors from the embedding matrix. After going through the embedding layer, the embedded input was passed to dropout to prevent overfitting issues while training the model. The `pack_padded_sequence()` function is then used to pack the embedded inputs according to the token length, which helps prevent the GRU's hidden state from being influenced by the paddings. The GRU processes the packed inputs, and the resulting hidden state is returned. Only the final hidden state is needed for classification, so it is passed to the fully connected layer. Finally, the fully connected layer output (logits) serves as the model's output.

GRU was selected for constructing the model due to its efficiency; it has two gates compared to the three gates in LSTM, resulting in faster processing times without sacrificing performance during experimentation with this pipeline. Compared to LSTM, GRU was found to have almost identical performance while requiring less computing time.

The optimization algorithm for training the model was `Adam`, and the loss function utilized was `CrossEntropyLoss`. This combination of logits and `CrossEntropyLoss` was equivalent to applying `log softmax` to the output of the fully connected layer and using negative log-likelihood as the criterion.

The hyperparameters for the model included: batch size, hidden size, learning rate, number of epochs, dropout probability, number of layers, and bidirectional for the GRU layer. Through experimentation, a batch size of 128 was found to be effective. The hidden size was set to 128, as a larger size may result in overfitting, and a smaller size may impair the model's capability of finding patterns in the data, which was necessary for classification. The learning rate was initially tuned on a logarithmic scale and then slightly adjusted. Additionally, an exponential learning rate scheduler was implemented with a decay rate of 0.9 to reduce the learning rate in later epochs for stability. A learning rate of 0.004 typically resulted in the highest accuracy or lowest loss within five epochs during training. However, the model typically began to overfit the training dataset after eight epochs, so the number of epochs was set to eight to end the training process earlier. In addition, the dropout probability was set to 0.4 to prevent overfitting while minimizing the loss of information. Finally, the number of layers and bidirectional in GRU were set to 1 and False, respectively, to avoid overcomplicating the model and resulting in overfitting.

4. Evaluation

The primary metric used in the evaluation was the cross-entropy loss, as the pipeline was designed for a classification task. The other metric used in the training process was accuracy, which was also used to monitor the model's performance. It was observed that the lowest loss did not always correspond to the highest accuracy.

K-fold validation was implemented to train the same model on different combinations of five folds of data and calculate the average loss and accuracy values to evaluate the model's overall performance.

Results

The average accuracy of the models with the best accuracy on the validation data was 79.294%. In comparison, the average accuracy of the models with the best loss on the validation data was 79.348%, which is only slightly different. Although the model was light-weighted (~15MB) but yet could achieve high accuracy. This pipeline outperformed other pipelines experimented with, for instance, the pipeline using LSTM.

Discussion

It is interesting to observe that a small model achieved good performance. However, there were a few limitations to this work. One limitation was the potential impact of imbalanced data on the model's accuracy. Increasing the number of samples for underrepresented classes (for instance, classes 4 and 15) may improve the model's overall performance. Another limitation was the significant variation in token lengths (with a standard deviation of 62.31). RNNs require fixed length input, which means that, in the case of this pipeline, some short sentences may be padded with more than 200 paddings, potentially leading to overfitting on padding tokens. While experimenting with this model, possible solutions found to this problem included (1) using front padding instead of back padding and using the final hidden state for prediction or (2) using `pack_padded_sequence()` to remove the padding and pack the sequences together. It may also be interesting to explore models such as seq2seq with attention or self-attention models, which do not have fixed sequence length limitations, on the classification task.