Student name: Biligsaikhan Khurtsbayar
Student Id: 2600220460-5
Course: Artificial Intelligence (G1)

Document classification with BERT model Natural Language Processing

I. Introduction

Natural language processing is, at its core, something humans do instinctively. When a child is born, they do not possess inherent abilities to read, listen, or converse. However, as they grow older, they naturally end up learning language. They acquire it slowly and logically, recognizing and memorizing patterns and hints throughout their lives. Similarly, when we take these innate abilities and attempt to apply them to a computer, the field of NLP is born. In recent years, since the introduction of Google's BERT model, this area of study has been expanding rapidly and trending to continue to do so.

II. Methodology

1. Classifications

To resolve the task of categorizing scientific abstracts into multi-classes, we first acquire datasets to train the BERT model. Out of the five scientific field options provided, I have strictly chosen to classify the abstracts from the study of physics, mathematics, and medicine. I have selected these classes with the deliberate intent of evaluating the model's performance given their apparent similarities and distinctness from each other in nature.

2. Data collection

For data collection, 1000 scholarly abstracts, consisting of 330 physics, 330 mathematics, and 340 medicinal, were all accomplished using the Arxiv repository's free-to-use API. As the API offered categorical and keyword query searching services, both query types were utilized as Arxiv's repository does not contain a medicine tag in their category. As the keywords for query search had to be distinct and representative of the medicinal field, it required a bit of research. To maintain a unique dataset from the other students, I have employed randomization tactics in the data collection.

I gathered an equivalent set of keywords and categories for each class. Further, I employ a random selection method to choose one category or keyword, resulting in 100 query results. Additionally, from this set of 100 query items, another random selection is applied to pick one abstract, which is only then stored in the class-specific '.csv' file. To further guarantee the uniqueness of each addition to each class dataset, the abstract URLs undergo a redundancy-checking process. Then, all the abstracts are combined into a single '.csv' file with their labels given depending on their previous '.csv' file name. Finally, redundancy-checking is applied once again across all class abstracts.

Student name: Biligsaikhan Khurtsbayar
Student Id: 2600220460-5
Course: Artificial Intelligence (G1)

3. Data Preprocessing

Before beginning the data preprocessing using the NLTK library methods, I took a preparatory step of extracting and storing LaTeX expressions detected within the abstracts. In this process, I replaced these expressions with placeholders, allowing for their seamless restoration to their original positions after the subsequent data preprocessing steps. This particular step is both specific and crucial, especially considering the abstracts from physics and mathematics papers had numerous mathematical formulas often presented in LaTeX format, typically enclosed within dollar signs ($). By isolation and temporary removal, these expressions are no longer at the risk of being deleted during the character and number removal process. This precaution ensures the preservation of mathematical integrity while enabling effective data cleanup.

As I have removed all the latex expressions, averaged_perceptron_tagger from the nltk library is used to tag words into their grammatical categories, such as nouns, verbs, and adjectives. This method is a prerequisite step for lemmatization. Lemmatization is a method in which words are normalized, in other words, stripping down to their root dictionary form based on the previously given tags. On the other hand, stemming does not require tagging, and it normalizes the text for languages that are less familiar or have limited resources. However, as it strips the word down by its suffixes, with no additional dictionary knowledge, the stemmer sometimes strips away too much, resulting in non-existent word forms. The goal of text normalization is to reduce the complexity of the data for the model to train by reducing every inflection of a word into a single root form.

After the word form reduction, the data is further processed by removing special characters and numbers, leaving only words to remain. Further, although not always necessary, every word is turned lowercase. Finally, to further reduce the complexity of the data, stop words are removed, or words that do not hold value for context have been removed.

Now that the abstracts are cleaned, normalized, and stripped down, we bring the latex expression, making it ready to be tokenized and used for NLP tasks. Now, to make its corresponding marker, we use one hot encoder for the labels.

4. Data Tokenization

As we have previously mentioned, in the introduction section, that in this project, we will be using BERT model. This tokenization process is one of the most important section, as BERT uses many very specific methods for its tokenization.

One of the two main mechanisms of BERT is its self attention mechanism. This mechanism enables all tokens in sequence to consider one another’s importance and assess each other’s importance and capturing their relationship. As BERT expects all input to have same shape, we use padding and truncation to generalize their shape. When truncating it takes note of sequence start [CLS] token and sequence separator [SEP] tokens and truncates the middle section of the sentence, as it deemed

to hold less important information than the end point of the sentences. As for padding, if we pad the sequences, until it reaches the common shape and don't specify anything, it will waste time paying attention to an empty token. So, to solve this issue come attention masks, as it makes it clear which tokens are padding and which ones are not.

In my BERT implementation, I utilized the tokenizer.encode_plus method, setting parameters such as truncation, padding, and a maximum length aligned with BERT's limit of 512 tokens. As I am using TensorFlow library instead of PyTorch, I set my return type as 'tf' tensors.

5. Cross-validation

Cross-validation, in short, is a statistical technique that uses machine learning to assess a given model's ability. To cross-validate, we are required to have multiple values that validate against each other. This is where k folds come into the picture. K folds evenly and divides the data into k equal parts. And each fold training or just running its own model and validating against each other. In Python straightforwardly dividing the data into splits can ruin your splits labeled data balance and meddle with the validation results. So, as a solution to this issue instead of k folds we use stratified k fold, which balances the data representations within the sets. Because the stratified option provides more evenly distributed splits, the model evolution usually tends to rise. By applying k-fold or better-stratified k-fold, we can ensure the robust and reliable nature of the metrics.

As for my implementation of the BERT model, I used stratified k-5 fold on my with 0.2 validation dataset. However, when using k-folds, there are a few considerations. Firstly, it takes considerable time to run and is computationally demanding. For students like us, who only own a MacBook Air, this was one of the significant limitations for exploring various paths.

6. Model fine-tuning

Transformer's BERT model, in total, has 12 layers and around 100 million pre-trained parameters. To transfer learning using this BERT model, we have to un-freeze some of the BERT model's layers parameters and let it learn from our dataset. When talking about fine-tuning or transfer learning BERT, usually the layers are coupled into 1-5 lower layers and 6-11 upper layers as they have similar purposes. However, this does not mean these layers do the same thing. Every 12 layers of the BERT model has its role for NLP.

During my data preprocessing, one of the preprocessing steps I took was removing special characters and numbers from the dataset. As the lower layer of BERT trains syntax and language structure, as my dataset itself, after preprocessing does not possess that aspect it is better to keep the lower layer frozen. So, by unfreezing the upper layers and freezing the lower layers of BERT, I make it take the word choice nuances without needing to learn about its syntaxes.

Student name: Biligsaikhan Khurtsbayar
Student Id: 2600220460-5
Course: Artificial Intelligence (G1)

7. BERT additional layers:

Adding layers to a pre-trained model, according to your necessities, is as much of a transfer learning approach as fine-tuning in the previous section. In my code, after preparing to fine-tune the BERT model's top layers with my dataset, I created some additional layers after it with the following purposes.

Dense layer:

Sequentially continuing from 12 layers of BERT, my program then connects it to a FeedForward Network Dense layer with 512 neurons, each activated by the Rectified Linear Unit function. A dense layer means, in this case, that each neuron in the layer is connected to every other neuron in the layer. Dense layers work to learn as much as complex mappings from the inputs to outputs. Moreover, using simple activation functions like relu, which turns every negative value into 0, solves one of the main obstacles called the vanishing gradient. In short, while enabling the dense layer to capture more intricate patterns by introducing non-linearity, relu also solves this vanishing gradient problem. As I have mentioned before, the gradient vanishing problem occurs in neural networks during backpropagation, as the gradient values go through each layer getting multiplied by the weights. In a case where the weights are less than 0, after some layers, the gradient turns negligibly small. Relu achieves to avoid this by saturating the negative values and deriving one from its positive inputs, effectively multiplying the gradient by one on each backpropagation.

Dropout layer:

Then continuing out from the dense layer, the sequence continues into dropout layer where we prevent our models to be overfitted by the datasets.l behind the dropout layer is that, during training, randomly chosen certain percentage of neuron outputs are set to 0. This means that the layer will make it doubtful of the model to memorize the data, since its neurons shutting down is determined by randomness. So, the model has to start learning the feature that makes its label.

Classification layer:

Then continuing out from the dropout layer, the sequence then continues into classification layer, where the model predicts the abstracts' labels. As we are doing a multi-classification document selection project, as the layer's activation function I chose softmax function, instead of sigmoid function. Sigmoid function is an activation function, which is used for binary classification as it returns only 0 or 1. As for softmax function, it takes all the sum of the output layer and normalizes it into probability distribution. And the classification label is given to the class that has the highest probability amongst the others

Student name: Biligsaikhan Khurtsbayar
Student Id: 2600220460-5
Course: Artificial Intelligence (G1)

III. Experiments

1.Data sampling

While starting to train the model, I have faced a great struggle of waiting for the model to finish training. As my hardware was unable to hastily test my code's operate-ability, I had to find a way to reduce my training time to at least test if my code is working. In my code, the labels and the abstracts were imported into the Jupyter Notebook using pandas. For testing my code, I used pandas to sample my data into a tenth of its size. As the pandas' sample takes random states as a variable, the created sub-sample of my original data was shuffled and reproducible. However, one more issue I faced while using this approach was that due to the minuscule size of the training data, the BERT model was easily overfitting for the task.

2. Overfitting Problem

Until I could successfully run the training model without encountering any issues, I was determined to continue using the sub-sampled data from pandas. However, in addressing the overfitting problem mentioned, I had to find a new approach to solve my problem. As stated in the Methodologies: BERT additional layers section, dropout layers are easily adjustable, a highly impactful approach to solving this issue. Initially, my Dropout rate was set to 0.2, but after finding the optimal Dropout rate for my subsampled data, I discovered that the Dropout rate, when optimized was specific to the dataset and its task, so I stopped pursuing this approach.

3. Max length solution

As mentioned in the other sections, the BERT model only takes tokens until the max length of 512. The more you set your training max-length closer to the 512, the more computationally taxing the task will be. So, to significantly reduce my training time issue, I had to reduce the max_length for training to 256 tokens, as recommended. It reduced the amount of training time by approximately four times. However, we still need to note that by cutting the max_length of the training tokens, we are removing a lot of context from the model training. In the end, it is always better to run with 512 tokens as max_length provided sufficient hardware.

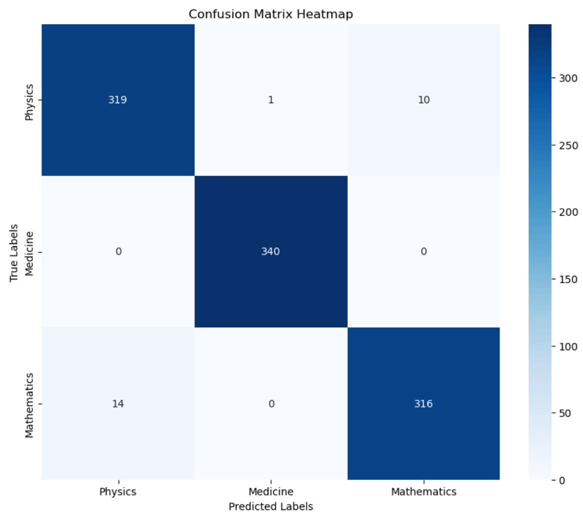
4. Learning rate scheduler

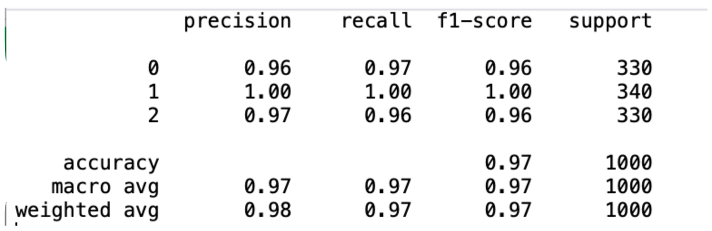
By both sub-sampling my data to the one-third of my original data size and setting the max_length of tokens to 256, I have achieved fast enough model training speed, and was finally able to run my code without any issue. As I was tackling the training speed issue for training, I was also trying to figure out how to achieve greater training and validation accuracy for each epoch in every folds. Initially, my accuracies were all stuck around 30 % in both training and validation. So, at that point, I figured that the learning rate was too big and the optimization was skipping over the minima to increase it’s accuracy. After reducing the learning rate through trial and error approach, although the accuracy improved decently, was not enough to take it as a properly trained model for document classification task.After, scourging through the internet for a bit , I discovered that we do not have to manually adjust the learning rates, and I discovered the existence of learning rate schedulers. Learning rate scheduler is a machine learning technique, which we dynamically, automatically reschedules the learning rate of the optimizer throughout training process.

5. Batch size issue

Although after the discovery of learning rate scheduler, everything is moving smoothly, it seems there is still another way to optimize training duration. I learned that batch size is most optimal when it is derived from the running computer’s memory. Through thorough experiments, starting from 8 batches scaling with 8 increments in batch size discovered that for my laptop for training model 16 batches was most optimal. However, this might not be completely true as I made judgement solely based on training time and memory usages.

IV. Results




```
              precision    recall  f1-score   support

           0       0.96      0.97      0.96       330
           1       1.00      1.00      1.00       340
           2       0.97      0.96      0.96       330

    accuracy                           0.97      1000
   macro avg       0.97      0.97      0.97      1000
weighted avg       0.98      0.97      0.97      1000
```

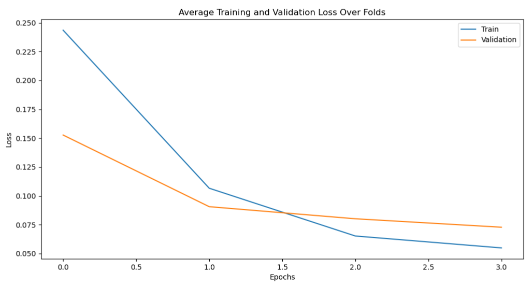
These are the results of my final run of training the BERT model on my dataset. For my final attempt, to get proper results, I stopped used dropout layers as I had set the max_length back to 512, training on my full dataset with 1000 abstracts, and utilizing cross-validation, there was minimal chance that the model would overfit.

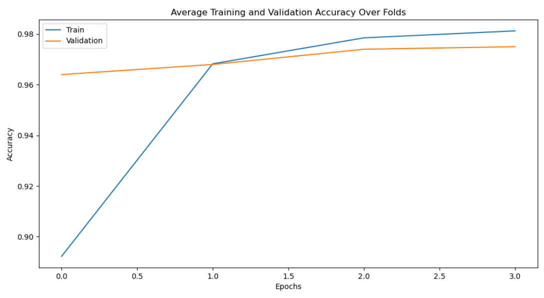
With the learning rate scheduler ready to properly optimize my accuracy, with no worries of overfitting the dataset to the model, and a patience ready to wait the full 4 epochs 5 folds training duration I started my final training.

After thoroughly setting up to have an overall acceptable results, for my final full length run, I have successfully achieved my goal. As we can see, from this heat map, either due to not removing the latex expressions in data cleaning or just for the fact that physics and mathematics having considerable similarities, the model have mismatched the labels of these classes quite a bit.

Student name: Biligsaikhan Khurtsbayar
Student Id: 2600220460-5
Course: Artificial Intelligence (G1)

Moreover, as expected, the medicine class almost did not have any wrong labels in its predictions. This might have been due to the data collection method, we used specifically for medicine class, as arxiv's category did not included medicine, it only by itself had to find the abstracts through word search query in Arxiv.





Now that we established, through the confusion matrix and classification report, that the model performed well with a 97% accuracy across all classes, the next step is to evaluate the model's fitness by analyzing its loss over folds.

Here we can see that my validation loss decrease, and validation accuracy increased over the folds. This is a sign of the model fitting positively to the task. Within the to diagrams we can also notice that the training and validation averages over the folds have intersected with each other. This means around the 2 fold that the model has reached to a stable solution for the task.

V. Discussion

By tinkering to decrease the model's training time and reduce its calculation complexities, I have discovered many previously unknown purposes for the BERT setup. After learning from my experiments, I can now know specifics on how to avoid making the model and the dataset to be unfit for each other. I have successfully learned to calibrate BERT model to its training dataset sizes and needs. As for the trained model, even with overwhelming accuracy of 97%, as hypothesized at the beginning of the report, mathematics and physics have been wrongfully categorized into each other considerable amount of times. If I have to do such tasks to increase classification accuracy between mathematics and physics papers, I will look for a way to give less weight or importance to its Latex expressions. My initial guess is that the tokenization for the formulas were dealt as symbols not as expressions. So, an excessive amount of tokens might have been generated through the tokenization process giving those specific sign over importance.

VI Conclusion

Through trying to utilize BERT for transfer learning document classification task, I have gained vast knowledge in both the field of neural networks and machine learning. Although transfer learning might look as nothing much needs to be done, I now understand that it requires significant amount of domain knowledge and coding expertise to account for many possible logical and practical errors.