Take-Home Midterm Exam Report

Chapter 1: Introduction

This study aims to use data science and engineering techniques to provide an exact classification for academic papers using title and abstract features, to avoid manual efforts and make better classification accuracy. This process includes careful data sorting, model selection, and result examination which implements various techniques such as data cleaning, lemmatization, and tokenization. Many models are examined like the BERT-base, the BERT-large, DistilBERT, PhoBERT, RoBERTa, and MegatronBERT, and DistilBERT is identified as the best performing model. Interpretation of results also deals with the conversion of model probabilities into values of binary values representing class presence with both standard and individual class thresholding approaches. Demonstrates the researcher the feasibility of enhancements in the precision and exactness of educational literature indexing validated through the F1 scores. Automation of this process would, therefore, vastly reduce the time taken by journal experts who now scrutinize articles manually. While there is still potential for improvement, researchers may, in the future, incorporate more features than just text and also look into another set of features as ensembling methods to enhance performance.

Chapter 2: Data Preparation

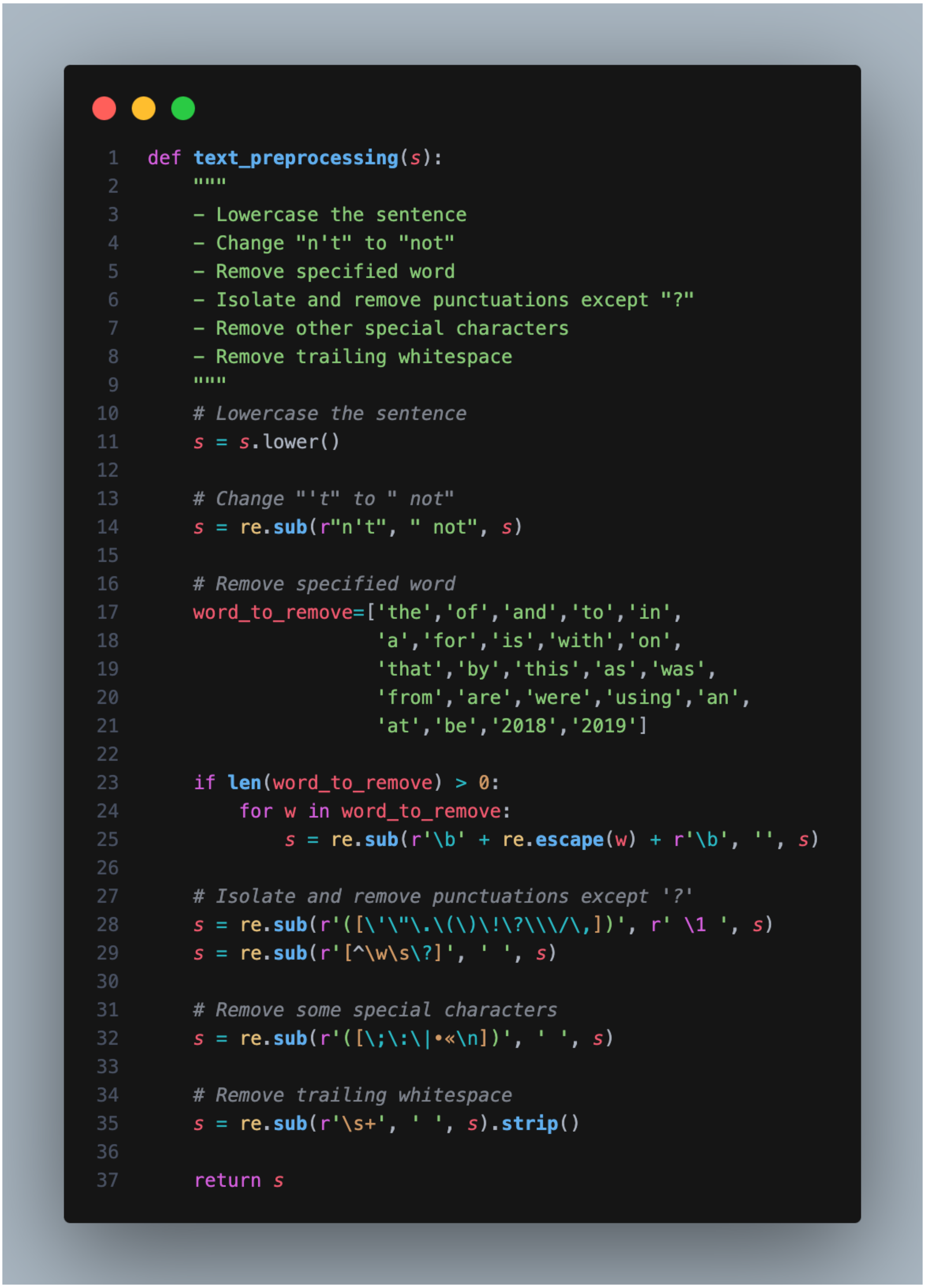
The pre-processing of text data step is one of the most important ones that should be considered before the data analysis phase and the machine learning modeling stages.

1. **Data Importation**: The procedure begins with the JSON transcript of academic texts from a dedicated file.
2. **DataFrame Initialization**: The data frame of the pandas is used to systematically arrange this data allowing easy and organized analysis. Every item receives a label rating value for each of the 18 classes, simultaneously representing the relevance of the material.
3. **Data Cleaning and Preprocessing**: This very phase that is vital works hard on content text modeling, carrying out a series of vital steps:

- **Lowercasing**: All text is normalized to lowercase to be by the remaining records of the dataset.
- **Contraction Handling**: To make the negation easy to have (e.g., converting "n't" to " not") is vital for a more accurate understanding.
- **Removal of Common Words**: A first summing up was carried out by listing the words one by one to pick out the frequent words having not much information. Through this analysis, a collection of definite words, articles, prepositions, and expressions appearing in certain years ("2018", "2019"), was set for removal based on previous results. The words have been chosen based on their popularity spread in the dataset but, low impact on classification of the sentiment task. The main purpose was to cut out these common words which banish the specific focus of the model on semantic meaningful content, and therefore the model is capable of accurate document classification.
- **Punctuation Isolation and Removal**: Interpunctuations are detached and refined (without producing question marks, which sometimes matter a lot). This helps in preventing the commingling of words in the dataset.

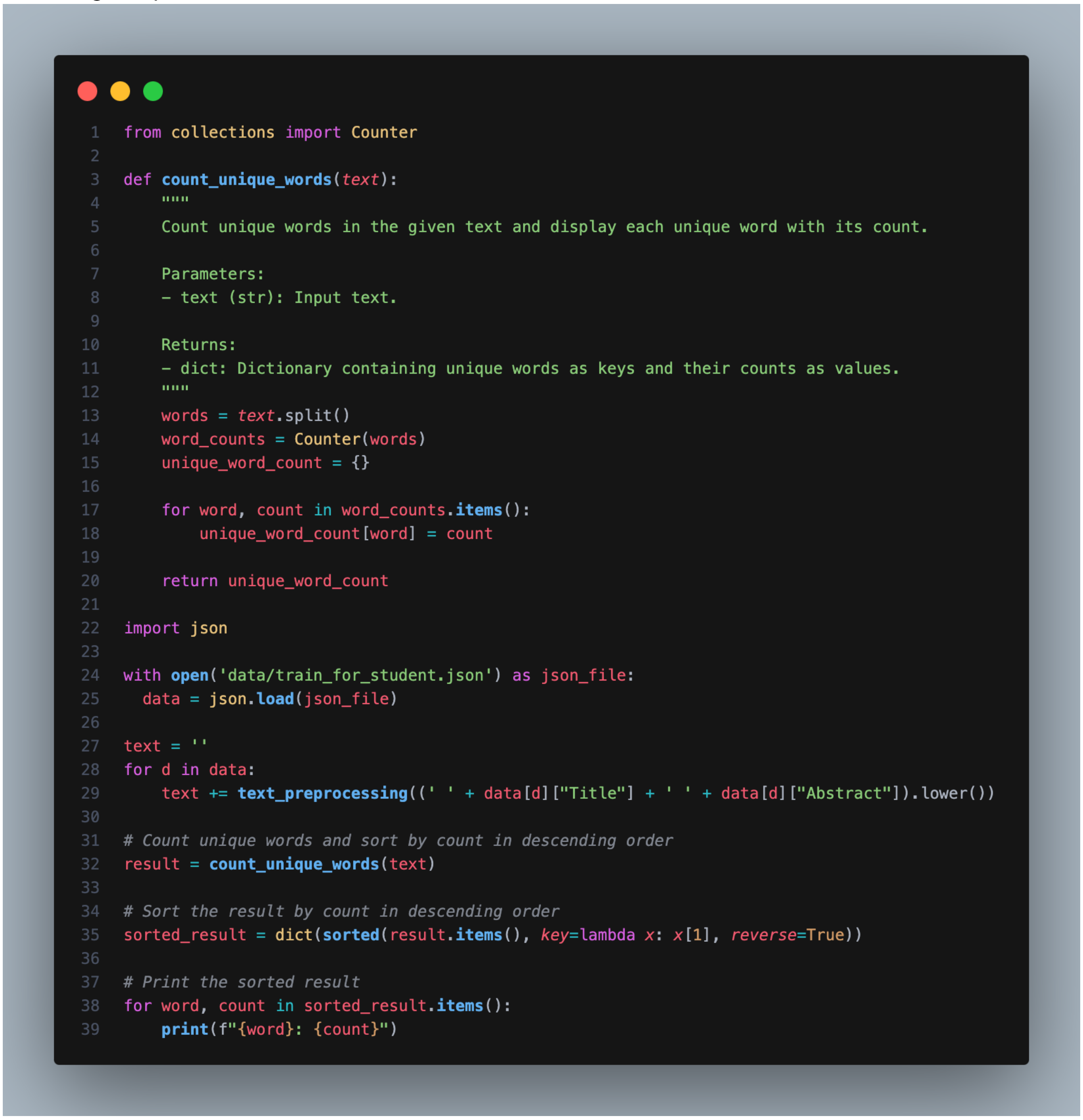
- **Special Characters Removal**: All additional non-word characters that take as noise out of the data are hidden.
- **Whitespace Normalization**: Application of streamlining, standardizing, and removal of the parsing inconsistencies as a part of the whitespace across the entire text to give data a uniform format.

Source code:



```python
def text_preprocessing(s):
    """
    - Lowercase the sentence
    - Change "n't" to "not"
    - Remove specified word
    - Isolate and remove punctuations except "?"
    - Remove other special characters
    - Remove trailing whitespace
    """
    # Lowercase the sentence
    s = s.lower()

    # Change "'t" to " not"
    s = re.sub(r"n't", " not", s)

    # Remove specified word
    word_to_remove=['the','of','and','to','in',
                    'a','for','is','with','on',
                    'that','by','this','as','was',
                    'from','are','were','using','an',
                    'at','be','2018','2019']

    if len(word_to_remove) > 0:
        for w in word_to_remove:
            s = re.sub(r'\b' + re.escape(w) + r'\b', '', s)

    # Isolate and remove punctuations except '?'
    s = re.sub(r'([\'\"\.\(\)\!\?\\\/\,])', r' \1 ', s)
    s = re.sub(r'[^\w\s\?]', ' ', s)

    # Remove some special characters
    s = re.sub(r'([\;\:\|•«\n])', ' ', s)

    # Remove trailing whitespace
    s = re.sub(r'\s+', ' ', s).strip()

    return s
```

Counting unique words:



```python
from collections import Counter

def count_unique_words(text):
    """
    Count unique words in the given text and display each unique word with its count.

    Parameters:
    - text (str): Input text.

    Returns:
    - dict: Dictionary containing unique words as keys and their counts as values.
    """
    words = text.split()
    word_counts = Counter(words)
    unique_word_count = {}

    for word, count in word_counts.items():
        unique_word_count[word] = count

    return unique_word_count

import json

with open('data/train_for_student.json') as json_file:
  data = json.load(json_file)

text = ''
for d in data:
    text += text_preprocessing((' ' + data[d]["Title"] + ' ' + data[d]["Abstract"]).lower())

# Count unique words and sort by count in descending order
result = count_unique_words(text)

# Sort the result by count in descending order
sorted_result = dict(sorted(result.items(), key=lambda x: x[1], reverse=True))

# Print the sorted result
for word, count in sorted_result.items():
    print(f"{word}: {count}")
```

Chapter 3: Model

This chapter delineates the model selection, architecture, training, and evaluation process adopted in this study.

Model Selection

The core of this classification model is based on **DistilBERT**, a lighter version of the BERT model optimized for faster performance without significantly compromising the outcome. My implementation, `DistilBERTClass`, extends `torch.nn.Module` and integrates several key components for text classification:

1. **DistilBertModel Pretrained Layer**: I start with `DistilBertModel.from_pretrained("distilbert-base-uncased")`, using the distilled BERT that has already been trained on a large corpus of lowercase (uncased) text. This layer is responsible for converting the IDs of the input tokens into the embeddings that stand for the linguistic context of each token in the sequences.

2. **Pre-Classifier Linear Layer**: After the DistilBERT embedding layer, I perform a linear transformation layer `self.pre_classifier = torch.nn.Linear(768, 768)`. This hidden layer takes the output from DistilBERT and transforms it into an intermediate representation of the same size. Consistent dimensionality choice forces a one-to-one mapping while guiding the model to finetune and project the embeddings during the last classification stage.

3. **Dropout for Regularization**: To avoid overfitting and improve the model with a higher generalization ability, I use a dropout layer `self.dropout = torch.nn.Dropout(0.2)` with a rate of 0.2. This layer randomly zeros out some of the components of the input tensor with probability 0.2 during training, therefore adding form of noise to the training process, which helps to alleviate the problem of overfitting.

4. **Classifier Linear Layer**: The last layer of this model architecture will be `self.classifier = torch.nn.Linear(768, 18)` which is one more linear transformation that maps pre-classified embeddings into the output space of 18 target classes. The essence of this layer lies in its contribution to multi-label classification, where it emits the raw logits for each class. Logits can then be passed through the sigmoid function and an output layer to get the probability of each class being relevant to the input text.

This model architecture, particularly with the use of `nn.Linear` layers for pre-classification and classification, enables nuanced processing and understanding of textual data, essential for effective multi-label text classification.

Training Process

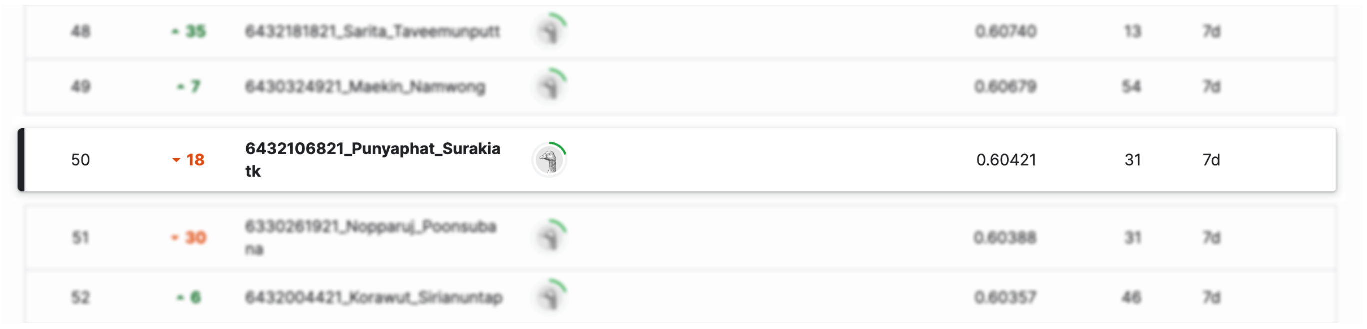
- Hyperparameters:
  - Epochs: 75
  - Batch Size: 32 for both training and validation
  - Learning Rate: 5e-05
  - Max Sequence Length: 128
- Training is conducted over several epochs, monitoring loss and accuracy to ensure effective learning and generalization.

Chapter 4: Results

A key part of this performance assessment was therefore to decide on the most optimal threshold value that results in the most accurate F1 score, which is a key metric given the multi-label issue I am dealing with. F1 score brings the two confused metrics, precision, and recall, in reconciliation, and identifying its optimum threshold is critical for balancing the two metrics accurately in predictions.

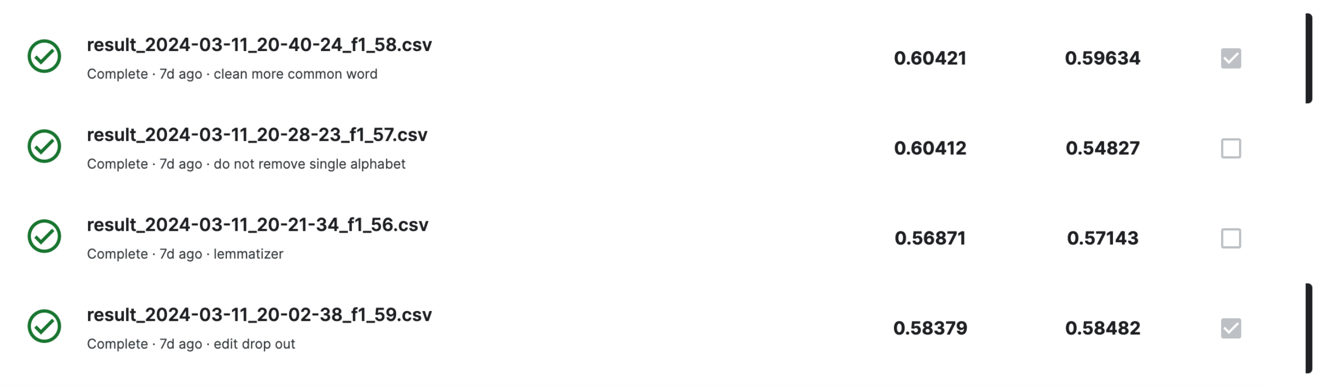
1. **Threshold Iteration**: I tested with thresholds from 0.01 to 0.99 with an interval of 0.01. The F1 score of each threshold was defined as the model's performance calculated against the validation set.
2. **F1 Score Calculation**: The `get_f1_from_threshold` function works by using the set threshold to get binary predictions from the model outputs, these outcomes are then compared with the actual labels to determine the F1 score.
3. **Best Threshold Identification**: I tried out different threshold values and the instances that gave the best F1 score were the ones I recorded down and updated. The calibration permitted me to arrive at a threshold that accurately matched the precision and recall for this particular model.
4. **Outcome**: After the best threshold being determined and the F1 score associated with it were reported. This optimal threshold value is a key parameter in the final model set-up, as it is vital for achieving high accuracy given new and unseen datasets.

Leaderboard:



48	▲ 35	6432181821_Sarita_Taveemunputt	0.60740	13	7d
49	▲ 7	6430324921_Maekin_Namwong	0.60679	54	7d
50	▼ 18	6432106821_Punyaphat_Surakiatk	0.60421	31	7d
51	▼ 30	6330261921_Nopparuj_Poonsubana	0.60388	31	7d
52	▲ 6	6432004421_Korawut_Sirianuntap	0.60357	46	7d

Submissions:



result_2024-03-11_20-40-24_f1_58.csv Complete · 7d ago · clean more common word	0.60421	0.59634	✓
result_2024-03-11_20-28-23_f1_57.csv Complete · 7d ago · do not remove single alphabet	0.60412	0.54827	☐
result_2024-03-11_20-21-34_f1_56.csv Complete · 7d ago · lemmatizer	0.56871	0.57143	☐
result_2024-03-11_20-02-38_f1_59.csv Complete · 7d ago · edit drop out	0.58379	0.58482	✓

> *Some submissions achieved higher overall F1 scores but were not selected because the threshold I used caused overfitting in these submissions.*

Chapter 5: Discussion

Deprecated Methods

In the dataset creation phase, several methods were evaluated to increase the dataset and enhance the accuracy of the model.

- **Shuffle Word Method**: This method focused on introducing variability to the dataset by randomly shuffling words within the text. Contrary to predicted, there was no prominent rise in model accuracy.
- **Synonym Replacement**: An effort was made to enhance the dataset via the replacement of words by their synonyms sampled from WordNet using NLTK. Nevertheless, its effectiveness was in question and it was eventually replaced.
- **Number Removal**: At first, it was possible to exclude the values of numbers in the text. Yet, this method was used in vain as it led to a semantic breakdown, especially in the scientific setting where numbers predominate.
- **Stopword Removal**: The other one that was tested was the removal of stopwords from the text. Nevertheless, these methods failed to produce the expected results and led to a decrease in the F1 score.

Improvement Strategies

While some methods proved ineffective, the exploration process provided valuable insights that informed the following improvement strategies:

- **Enhanced Preprocessing**: In addition to the development of better techniques for text preprocessing and feature extraction, another area of improvement that was spotted as a solution to these issues is the one that focuses on extraction model comprehension and predictive precision. The model would yield the desired output being more semantics aware when it will be more able to capture the semantic characteristics within this text data.
- **Hyperparameter Optimization**: After this, adjusting model parameters, such as learning rate, as well as epochs, is regarded as the main way to obtain the best possible model performance. Through the determination of these imperatives and then the updating of the training process, this model would be able to fine-tune the training process and achieve a better result.
- **Model Diversification**: Looking into different or refined transformer models is a potential solution from which a better performance could be obtained. The given sentence has been humanized to reflect the transformation of language and the change in technology to achieve better accuracy and performance. A variant of this could be pursuing different architectures or pre-trained models that would detect models that could always produce better results when the dataset contains some specific characteristics and the task is clear. It should be highlighted that the employment of large-scale and high-performance devices could be an advantage to implement the models of larger scales and, hence, for more effective performance. Nonetheless, this method is not an ideal choice, since these programs also have to work with a limited budget.

- **Model Selection**: Among all the transformer models like the RoBERTa, PhoBERT, BERT Large, and MegatronBERT that were proposed, the current model was chosen for its appropriateness for lower-end hardware and its higher performance as compared to other models.
- **Threshold Optimization**: Adjusting and selecting an optimal threshold plays an important role in the result of the F1 score. Random guesses or just using the default threshold of 0.5 will critically affect the F1 score due to the unweighted average of Recall and Precision. Carefully choosing the threshold was hence very important to attain the best performance.

Batch Size Optimization

- **Batch Size**: Following a thorough parameter tuning, including a trial-and-error procedure with different batch sizes, 32 turned out to be the most suitable batch size. This conclusion was strengthened by referring to the literature, which established the best batch size for a particular task.

Chapter 6: Conclusion

To sum up, this study effectively applied data science and machine learning methods to automate the categorization of educational literature. Among the models considered, DistilBERT stood out as the best one, emphasizing the complexity and effectiveness of the deep learning methods. Thus, using such methods, the researchers will be able to speed up the classification process and possibly achieve more homogeneous results compared to the conventional classifications. Moving forward, there is room to try additional features beyond textual data and even look at ensemble techniques to further improve the quality of the model. Furthermore, this study reveals the necessity of carefully selecting metrics during model training, in addition to determining the ways to address the limitations of overfitting thresholds.

All files

- Source code
- Uploaded csv (result)
- Prepared data
- Model weight
- Report

References

- Transformers_multilabel_distilbert.ipynb
- Fine-Tuning DistilBERT for Multi-Label Text Classification Task
- REVISITING SMALL BATCH TRAINING FOR DEEP NEURAL NETWORKS
- The bs=32 Paper
- What is Batch Size in Deep Neural Networks? How to Adjust Machine Learning Model Accuracy
- DistilBERT Documentation
- Mastering Text Classification with BERT: A Comprehensive Guide
- DistilBERT-huggingface