Task-1: Research Paper Publishability Assessment

1. Introduction

In this project, we address the problem of classifying documents based on their publishability. Given a dataset consisting of documents from two categories—"Publishable" and "Non-Publishable"—we aim to build a machine learning model that can predict whether a new document belongs to the "Publishable" or "Non-Publishable" class. The problem is approached using natural language processing (NLP) and deep learning, leveraging pre-trained transformer models like BERT to perform text classification.

The main goal of this project is to create a robust and efficient model for classifying document publishability, which could be valuable in various real-world applications, such as automated content moderation or sorting academic papers for publication.

2. Data Preparation

The dataset consists of PDF documents categorized into two subfolders: "Publishable" and "Non-Publishable." The first step involves extracting the text from these PDF files and cleaning it for use in training a machine learning model.

Data Extraction Process:

We utilized the `pdfplumber` library to extract text from PDF files. The `read_pdfs_from_folder` function recursively searches through the folder structure to extract the contents of the PDFs. Text extraction for each PDF is performed on all pages of the document, with each document being processed as a whole.

Text Cleaning:

Once the text is extracted, it undergoes cleaning using regular expressions. The following cleaning steps are applied:

- Multiple spaces or newlines are replaced with a single space.
- Leading and trailing spaces are removed.
- "Page X" patterns (where X is a number) are removed.
- Any lines consisting only of author names, document titles, or similar irrelevant text are filtered out.

After cleaning, the data is split into two lists—"Publishable" and "Non-Publishable"—containing the cleaned texts for each category.

3. Dataset Splitting and Preprocessing

To prepare the dataset for training, the text data is split into training and validation sets. The `train_test_split` function from scikit-learn is used, with 80% of the data allocated to the training set and 20% to the validation set. This ensures that the model is evaluated on unseen data during training.

Additionally, text data is tokenized using the BERT tokenizer. The BERT model requires text to be tokenized into a specific format, where each word or subword is mapped to an integer ID from a pre-trained vocabulary. The text is padded or truncated to a fixed length (512 tokens) to ensure uniform input size for the model.

4. Model Architecture

For the text classification task, we use the BERT model, specifically the pre-trained `bert-base-uncased` version. BERT (Bidirectional Encoder Representations from Transformers) is a transformer-based model that has achieved state-of-the-art results in many NLP tasks, including text classification.

Model Configuration:

- The BERT model is fine-tuned for the classification task. We add a classification head on top of the pre-trained BERT model, which consists of a linear layer that maps the output of the transformer model to the target classes (Publishable or Non-Publishable).
- The model outputs a probability distribution for each class, and the class with the highest probability is selected as the predicted label.

Optimizer:

The optimizer used for training is `AdamW`, a variant of the Adam optimizer, which works well for transformer-based models. A learning rate of 1e-5 is used, which is typical for fine-tuning BERT models.

Training Process:

The training loop involves the following steps:

- For each batch of data, the model performs a forward pass, computing the loss and predictions.
- The loss is backpropagated to update the model weights using the AdamW optimizer.
- After each epoch, the model's performance is evaluated on the validation set to track overfitting and ensure generalization.

The model is trained for 20 epochs, with training and validation loss as well as accuracy metrics being tracked throughout the process.

5. Results and Evaluation

After training, the model is evaluated on the validation set, and the results are summarized in terms of accuracy, loss, classification report, and confusion matrix.

Training and Validation Accuracy:

The training and validation accuracies indicate the performance of the model in correctly classifying documents into their respective categories. A high training accuracy suggests that the model has learned the task effectively, while a high validation accuracy indicates that the model generalizes well to unseen data.

Classification Report:

The classification report provides detailed metrics such as precision, recall, F1-score, and support for each class. These metrics give us an understanding of the model's ability to correctly classify each class and its overall performance.

Confusion Matrix:

The confusion matrix shows the number of true positives, true negatives, false positives, and false negatives. This matrix is useful for assessing the types of errors the model is making and can guide future improvements to the model.

6. Predictions

Once the model is trained, it is used to make predictions on a test set of PDF documents, which were not seen during training. The `predict_publishability` function processes each document in the test set by tokenizing the text and passing it through the trained model. The model outputs a prediction (0 for Non-Publishable and 1 for Publishable), and these predictions are stored along with the document names.

The predictions are saved to a CSV file, which allows easy export and further analysis of the results.

7. Challenges and Limitations

- **Data Quality:** The quality of the extracted text depends heavily on the quality of the PDFs. If the PDFs contain complex layouts or scanned images instead of text, the extraction process may not work as expected, leading to incomplete or noisy data.
- **Class Imbalance:** If the dataset contains a disproportionate number of Publishable or Non-Publishable documents, the model may be biased toward predicting the majority class. This issue can be mitigated by using techniques like oversampling, undersampling, or class weights.
- **Generalization:** While the model performs well on the validation set, its ability to generalize to unseen, real-world documents may vary. Further testing on diverse datasets is necessary to evaluate its robustness.

8. Future Work

To improve the model's performance, several strategies could be considered:

- **Data Augmentation:** More labeled data, especially for the Non-Publishable class, could help balance the dataset and improve the model's accuracy.
- **Model Fine-Tuning:** Experimenting with different BERT variants (e.g., RoBERTa, DistilBERT) or other transformer models could yield better results.
- **Error Analysis:** Conducting a detailed error analysis would help identify common mistakes made by the model, which could guide further improvements to the training process.

9. Conclusion

In conclusion, this project successfully demonstrates the application of deep learning and NLP for document classification. By using a pre-trained BERT model and fine-tuning it on a specific task, we achieved a system capable of classifying documents as Publishable or Non-Publishable. The results show promising accuracy and can be further improved with more data and fine-tuning. This approach can be extended to various document classification tasks in academic publishing and content management.

Appendix: Code and CSV Output

The predictions made by the model have been saved in a CSV file. You can access the file for further analysis and review. The CSV contains two columns: the document name and its predicted class.

Task-2: Conference Paper Classification

1. Objective:

The aim of this project is to recommend suitable conferences for research papers based on their content using natural language processing techniques. We utilize feature extraction techniques like `CountVectorizer` (and later explored BERT for potential improvements) to classify papers and suggest conferences based on text similarity.

2. Data:

1. **Papers Dataset**: The dataset includes research papers from multiple subfolders such as `CVPR`, `EMNLP`, `KDD`, `NeurIPS`, and `TMLR`. These are considered the reference papers that help in classifying the test papers.
2. **Test Papers**: These papers are classified using the trained models and methods based on text similarity.

3. Methodology:

1. **Text Extraction**:
    - We used `pdfplumber` to extract text from PDF files within each subfolder.
    - Texts are cleaned using a custom function to remove unnecessary spaces, page numbers, and author names.
2. **Feature Extraction**:
    - For feature extraction, `CountVectorizer` is used to convert the raw paper and conference content into a vector of word counts. This enables us to compare papers with conference profiles by analyzing common keywords.
3. **Cosine Similarity**:
    - The similarity between a paper and a conference is calculated using cosine similarity, a metric that measures the cosine of the angle between two vectors (representing the paper and conference).
4. **Conference Recommendation**:
    - Based on the similarity score, the most relevant conference is selected.
    - The top conferences are ranked, and justifications are generated based on the similarities between the paper and conference profiles.
5. **Justification**:

- For each recommended conference, a justification is generated explaining why the paper aligns well with the conference based on keyword overlap.

4. Key Steps in the Process:

1. **Text Extraction and Preprocessing**:
    - PDFs were read, and the extracted text was cleaned by removing unnecessary characters, page numbers, and extra spaces.
2. **Using CountVectorizer**:
    - `CountVectorizer` is applied to extract keywords from both the conference profiles and the test papers. The extracted terms are used to form vectors for each paper and conference profile.
3. **Similarity Calculation**:
    - The cosine similarity is used to compute how similar a given paper is to a particular conference profile based on shared keywords.
4. **Conference Recommendation**:
    - The paper is then compared to conference profiles from datasets like `CVPR`, `NeurIPS`, `KDD`, `EMNLP`, and `TMLR`.
    - Based on the highest similarity score, the top recommended conferences are returned.
5. **Final Classification**:
    - For each paper, the top recommended conference(s) are returned along with a rationale for why each recommendation is made.

5. Code Breakdown:

- The code is structured around the extraction of text from PDFs, followed by the cleaning process. The cleaned text is processed using `CountVectorizer`, which is then used to compute similarity scores between paper and conference content.
- Conferences are selected based on the highest similarity score, and justifications for the selection are provided based on keyword overlap between the paper and the conference.

6. Example Workflow:

1. **Extract Text**: Extract paper content from test papers using `pdfplumber`.
2. **Process Text**: Clean the text to remove any unwanted parts.
3. **Feature Extraction**: Convert the cleaned text to vectors using `CountVectorizer`.
4. **Calculate Similarity**: Measure the similarity between a paper and each conference.

5. **Recommend Conferences**: Select the conference with the highest similarity score and generate a justification.

7. Output:

1. **CSV File**: After classification, the results are stored in a dictionary that is then converted into a DataFrame and saved as a CSV file for further analysis. The columns in the CSV file include:
    - `Id`: Paper identifier
    - `Conference`: The recommended conference
    - `Rationale`: Justification for the recommendation
2. **Downloadable CSV**: The CSV file is made available for download in the notebook.

8. Other Approaches:

1. **Use of BERT/Transformer Models**: While the `CountVectorizer` approach works, it may be beneficial to explore more sophisticated models like BERT or RoBERTa, which provide contextual embeddings and can capture deeper semantic relationships between papers and conference profiles.
2. **Extended Feature Extraction**: Exploring other NLP techniques such as TF-IDF or Word2Vec could improve the vector representation, especially for longer and more complex texts.
3. **Model Optimization**: Fine-tuning the model using additional conference-specific data could improve the accuracy of recommendations.

9. Conclusion:

The current system uses a robust method of recommending conferences for research papers based on keyword similarity. The model leverages text vectorization with `CountVectorizer` and computes similarity scores to make recommendations. The use of justifications for recommendations ensures transparency in the process, which can be valuable for researchers seeking to find the most suitable conference for their work.

The current approach can be enhanced by incorporating more sophisticated models like BERT or RoBERTa to capture more nuanced relationships between paper topics and conference themes, improving overall recommendation accuracy.

Combining Outputs for Final Table

After processing and classifying the papers, two separate CSV files were generated:

1. **First CSV** (`dict1`): Contains the following columns:
    - `Id`: Paper identifier
    - `Conference`: The recommended conference for the paper
    - `Rationale`: Justification for the conference recommendation
2. **Second CSV** (`dict2`): Also contains similar columns but with additional classifications for the 'Publishable' category (i.e., whether the paper is deemed publishable or not).

These two output CSV files were **combined into one final table**, which includes the following columns:

- **Paper ID**: Unique identifier for the paper
- **Publishable**: The classification of whether the paper is publishable or not
- **Conference**: The recommended conference for the paper based on similarity analysis
- **Rationale**: Justification explaining why the paper is recommended for the chosen conference

This final table consolidates all the information, providing a comprehensive view of the paper's classification and recommendation along with the rationale behind each decision.