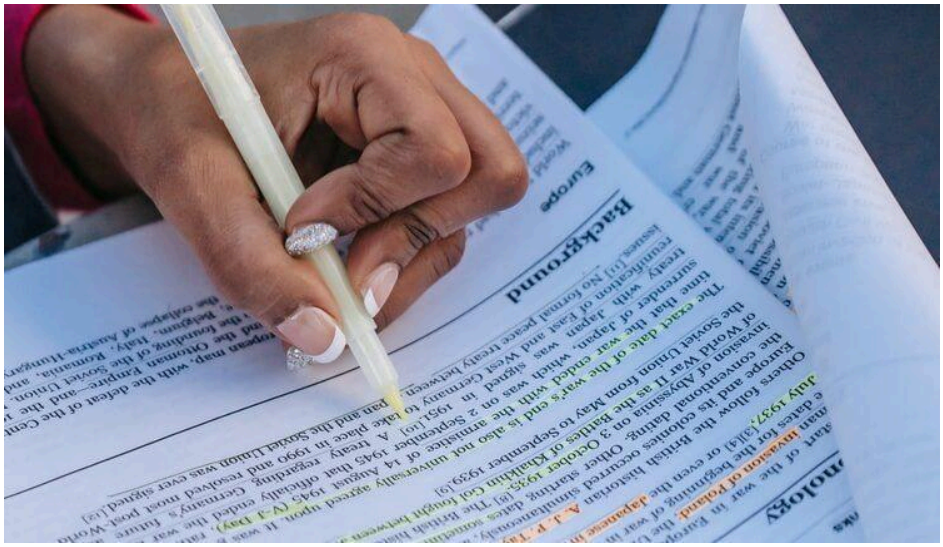
Transformer-Based Classification for Academic Paper Abstracts

Problem Statement:

An academic journal receives a large volume of paper submissions from researchers across various disciplines and topics. The editorial team must manually classify each submission based on its abstract to route it to the appropriate reviewers and organize accepted papers into themed issues. This manual process is time-consuming and prone to human error, potentially delaying the peer review process and leading to inconsistencies in categorization. The journal requires a transformer model-based solution to automatically classify academic papers into predefined categories using their abstracts, streamlining the review process and ensuring accurate categorization.



Pre requisites

For this project, we will be using Jupyter Notebook and or Google Colab.
To install the Anaconda navigator and to know how to use Jupyter Notebook watch the video.
Link: Click here to watch the video: Link
Install the necessary libraries numpy, pandas and NLTK, by using the following command in your command prompt or anaconda prompt.
'pip install numpy'
'pip install numpy'
'pip install nltk'
'pip install transformers'

Objective

The objective of this project is to develop a transformer-based classification model that automatically categorizes academic paper abstracts into predefined fields of study. This solution streamlines the peer review process by routing submissions to the appropriate experts and organizing accepted papers into themed issues.

Tasks:

- Install the necessary libraries
- Load pre-trained model and tokenizer
- Classify a Single Academic Abstract

Task 1 :Install & Import the necessary libraries

Install the required libraries using the following command:

```
[13] !pip install transformers

Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.38
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from tra
Requirement already satisfied: huggingface-hub<1.0,>=0.19.3 in /usr/local/lib/python3.10/dis
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from tra
Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/python3.10/dist-pack
```

In the given code snippet, install the "transformers" library via pip for access to transformer-based models.

Task 2: Load pre-trained model and tokenizer

Load a pre-trained transformer model and tokenizer suitable for text classification tasks. You can choose a model such as DistilBERT fine-tuned for classification tasks.

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification, pipeline

# Define the model name (e.g., DistilBERT fine-tuned for classification)
model_name = "distilbert-base-uncased-finetuned-sst-2-english"

# Load the tokenizer and model
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)

# Create a text classification pipeline
classifier = pipeline("text-classification", model=model, tokenizer=tokenizer)
```

This code snippet sets up a text classification pipeline using the Hugging Face Transformers library. It imports `AutoTokenizer`,

‘AutoModelForSequenceClassification`, and `pipeline` from the library. The model name is set to `distilbert-base-uncased-finetuned-sst-2-english`, a fine-tuned DistilBERT model for sentiment analysis. The tokenizer and model are loaded using `from_pretrained` to load the pre-trained model and tokenizer. Finally, a text classification pipeline is created using the specified model and tokenizer for easy classification of text data.

Task 3:Classify a Single Academic Abstract

Define a function to classify a single academic abstract using the fine-tuned transformer model. You can test the model's performance on individual abstracts.

```python
# Define a function to classify a single academic abstract
def classify_abstract(abstract):
    # Use the classifier pipeline to classify the abstract
    result = classifier(abstract)
    # Extract the label (predicted category) from the result
    label = result[0]['label']
    return label
```

The function `classify_abstract` takes a single academic abstract as input and uses a pre-trained transformer model to classify it. It passes the abstract to the `classifier` pipeline, which returns the classification results as a list. The function extracts the predicted label (category) from the first result in the list and returns it. This allows for easy classification of individual academic abstracts using the transformer model.

```python
# Test the function with a specific abstract
input_abstract = "A study on the impact of renewable energy sources on modern power grids."
predicted_category = classify_abstract(input_abstract)

# Print the predicted category
print(f"The abstract is classified as: {predicted_category}")
```

```
The abstract is classified as: POSITIVE
```

The above code snippet demonstrates how to classify a specific academic abstract and print its predicted category using a pre-trained transformer model. The code begins by defining an `input_abstract` about renewable energy sources and power grids. The abstract is passed to the `classify_abstract` function, which uses the model to classify the abstract and returns the predicted category. The final line of the code prints the predicted category in a readable format, allowing you to see how the model classifies the input abstract.

Conclusion:

The project successfully implemented a transformer-based classification model for academic paper abstracts. By automatically categorizing abstracts into predefined fields of study, the model streamlines the peer review process and aids in organizing papers into themed issues. This approach enhances efficiency and ensures a more structured review and publication process, benefiting researchers, reviewers, and publishers alike.