National College of Ireland

Configuration Manual

MSc Research Project
MSc Data Analytics

Kiruthika Suresh
Student ID: x23189916

School of Computing
National College of Ireland

Supervisor: Musfira Jilani

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Kiruthika Suresh		
Student ID:	x23189916		
Programme:	MSc Data Analytics	**Year:**	2024
Module:	MSc Research Project		
Supervisor:	Musfira Jilani		
Submission Due Date:	12/12/2024		
Project Title:	Enhancing Personalized News Recommendations with a Hybrid Model: Integrating BERT, Neural Collaborative Filtering and Attention Mechanisms		
Word Count:	**852**		
Page Count:	6		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Kiruthika Suresh
Date:	12th December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	☑
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☑
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☑

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Kiruthika Suresh
Student ID: x23189916

1 Introduction

This Configuration manual provides a step-by-step guide for setting up necessary environments and tools to execute the research code for a hybrid news recommendation system on Google Colab Pro. The project involves combining BERT-based content embeddings, Neural Collaborative Filtering (NCF), and an attention mechanism to enchance personalized news recommendations. Google Colab Pro offers a cloud-based environment with GPU/TPU support, making it ideal for running machine learning models efficiently. This guide will walk through accessing the required MIND dataset, installing necessary libraries and executing the code in Google Colab for training and evaluating the recommendation model.

2 Environmental Setup

Google Colab Pro is an ideal platform for running machine learning experiments as it provides access to GPUs and TPUs which are crucial for training deep learning models like BERT and Neural Collaborative filtering (NCF). Below are the steps to set up the environment on Colab.

2.1 Hardware Requirements:

Since Google Colab Pro runs on cloud-based infrastructure, no specific hardware setup is required. However, for optimal performance, ensuring the following:

- Google Account: A google account is required to access Colab Pro.
- Internet Connection: A stable and fast internet connection to handle large datasets such as the MIND dataset and compute intensive operations like training deep learning models (BERT, NCF).
- Google Colab Pro Subscription: Access to Colab Pro is recommendation for higher memory and longer runtimes, which are essential for large scale model training and experimentation.

2.2 Software Requirements:

To execute your hybrid recommendation system code efficiently in Google Colab Pro, the following software dependencies and libraries need to be installed:

- Python: 3.7 or later (typically pre-installed in Colab)

- Libraries
    - TensorFlow or PyTorch: Depending on preference for BERT and NCF model implementation. TensorFlow is commonly used for BERT, but PyTorch is also a good option.
    - Transformers: A library from hugging face for loading pre-trained BERT models and fine-tuning them.
    - Keras: For building and training neural network models
    - Scikit-learn: For data processing, model evaluation, and utilities like tarin test splitting.
    - Numpy: For numerical operations and handling arrays.
    - Pandas: For data manipulation
    - Matplotlib/ Seaborn: For visualization results, metrics and training curves.
    - SciPy: for advanced scientific computing and statistical analytics.

- Google Drive: Since we are using Google Colab, we need to mount Google drive to access datasets like MIND dataset and save model outputs.

3 Tools Required and Setup

3.1 Accessing Google Colab Pro:

Sign in to Google account. Go to Google Colab, we have Google Colab Pro subscription, we can access the enhanced GPU/TPU features. Start a new Colab notebook by clicking on "New Notebook" from the Colab dashboard.

3.2 Accessing Datasets from Google Drive:

To run the code with our datasets in Google Colab, we need to access the datasets stored on Google Drive. We can mount Google Drive in Colab as follows:

```
from google.colab import drive
drive.mount('/content/drive')
```

This command will prompt to authenticate your Google account. Once authenticated, you will be able to access the datasets stored in your Google Drive.

Behaviour Dataset:	https://drive.google.com/file/d/1LAnJOVW5o2wAXoLck0RzV2rtrhK4fsM/view?usp=sharing
News Dataset:	https://drive.google.com/file/d/192W13zzkM_JDoJQQ1nszUca2hIgd2S4C/view?usp=sharing
News_with_embedding:	https://drive.google.com/file/d/1BBwpTz3nQPjI5IkX0Npf8WAkxQZUwNG_/view?usp=sharing

3.3 Running the Code:

Once the datasets and libraries are set up, we can execute code in Colab. Ensure Code follow the structure required for your research:

1. Import necessary libraries.
2. Load datasets into pandas dataframes.
3. Preprocess the data.
4. Train models and evaluted results.

4 GPU Setup Verification:

To ensure that TensorFlow and Pytorch are utilizing the GPU for computation which is efficient for deep learning model training w ecan verify the GPU availability with the following code:

4.1 TensorFow and PyTorch GPU Verification:

```
import tensorflow as tf
print("TensorFlow GPU available: ", tf.config.list_physical_devices('GPU'))

import torch
print("PyTorch GPU available: ", torch.cuda.is_available())
if torch.cuda.is_available():
    print("PyTorch is using GPU: ", torch.cuda.get_device_name(0))
```

Expected Output: if TensorFlow and PyTorch detects the GPU, the output will be similar to:

```
TensorFlow GPU available:  [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
PyTorch GPU available:  True
PyTorch is using GPU:  Tesla T4
```

If the GPU is available, we will see a message indicating the GPU model, such as Tesla T4 which is available in Google Colab Pro.

5 Configuration and Setup for Model Training:

5.1 Mounting Google Drive and Loading Data:

```python
from google.colab import drive
import pandas as pd

# Mount Google Drive
drive.mount('/content/drive')
behaviors_column_names = ['impression_id', 'user_id', 'time', 'history', 'impressions']
behaviors_df = pd.read_csv('/content/drive/MyDrive/behaviors.tsv', sep='\t', header=None, names=behaviors_column_names)

news_column_names = ['news_id', 'category', 'subcategory', 'title', 'abstract', 'url', 'entity_title', 'entity_abstract']
news_df = pd.read_csv('/content/drive/MyDrive/news.tsv', sep='\t', header=None, names=news_column_names)

# Printing the first few rows to verify
print("Behaviors Data Sample:")
print(behaviors_df.head())

print("\nNews Data Sample:")
print(news_df.head())
```

This code will load the datasets from Google Drive and print the first few rows to verify the data.

5.2 BERT Embeddings for News Data:

Load the pre-trained BERT tokenizer model and apply them to extract embeddings for news articles titles and abstracts:

```python
import torch
from transformers import BertTokenizer, BertModel

# Step 1: Check for GPU availability and set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print("Using device:", device)

# Step 2: Load the pre-trained BERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = BertModel.from_pretrained('bert-base-uncased').to(device)  # Move the model to GPU

# Step 3: Define the function to get BERT embeddings for a given text on GPU
def get_bert_embedding(text):
    # Check if text is valid and not None
    if isinstance(text, str) and text:
        # Tokenize and create input tensors on GPU
        inputs = tokenizer(text, return_tensors='pt', truncation=True, padding=True, max_length=128).to(device)

        # Get the hidden states from the BERT model
        with torch.no_grad():
            outputs = bert_model(**inputs)
        embeddings = outputs.last_hidden_state.mean(dim=1).squeeze()
        return embeddings.cpu().numpy()
    else:
        return [0] * 768  # Return a zero vector for invalid text

sample_text = "This is a sample text for BERT embeddings."
embedding = get_bert_embedding(sample_text)
print("BERT embedding shape:", embedding.shape)
print("BERT embedding:", embedding)
```

This code will processes the news titles and abstracts to generate embeddings using the BERT model.

5.3 Saving Embeddings to CSV:

After generating embeddings for news articles, save them back to dataset and store them in CSV file:

```
news_df['title_embedding'] = news_df['title_embedding'].apply(lambda x: x.tolist() if isinstance(x, np.ndarray) else x)
news_df['abstract_embedding'] = news_df['abstract_embedding'].apply(lambda x: x.tolist() if isinstance(x, np.ndarray) else x)
news_df['combined_embedding'] = news_df['combined_embedding'].apply(lambda x: x.tolist() if isinstance(x, np.ndarray) else x)

# Save the DataFrame to a CSV file
news_df.to_csv('/content/drive/MyDrive/news_with_embeddings.csv', index=False)

# Verify the save and show a sample of the combined embedding
print("Sample of the combined embedding:")
print(news_df[['title', 'combined_embedding']].head())
```

5.4 Processing History and Impressions Columns:

Clean and process the behaviour data, including history and impressions and create dictionary for mapping news_id to embeddings.

```
# Step 1: Process the 'history' Column

# Replace NaN values with empty strings and ensure all entries are lists of clean 'news_id's
behaviors_df['history'] = behaviors_df['history'].fillna('').apply(
    lambda x: [news_id.strip(",[]") for news_id in x.split()] if isinstance(x, str) else []
)

# Verify that all entries in 'history' are now lists
print("After Processing 'history' Column:")
print(behaviors_df[['impression_id', 'user_id', 'history']].head())
print("\nData types in 'history' column:")
print(behaviors_df['history'].apply(type).value_counts())
print("\n----------------------------------------\n")
```

```
# Step 2: Process the 'impressions' Column

# Function to split impressions into (news_id, click_label) tuples
def process_impressions(impression_str):
    if isinstance(impression_str, str):
        impressions = impression_str.split()
        return [(imp.split('-')[0], int(imp.split('-')[1])) for imp in impressions]
    else:
        # If impressions are not a string, return an empty list
        return []

# Apply the function to the 'impressions' column
behaviors_df['impressions_processed'] = behaviors_df['impressions'].apply(process_impressions)

# Show a sample to verify the processing
print("After Processing 'impressions' Column:")
print(behaviors_df[['impressions', 'impressions_processed']].head())
print("\n----------------------------------------\n")
```

This Processes the history and impressions column to create structured data for model training.

5.5 Mapping Embeddings to history and impressions:

Create mappings for history and impressions to their respective embeddings using the previously saved news embeddings.

```
# Step 4: Define Mapping Functions with Safeguards

# Function to map 'history' news_ids to embeddings
def map_history_to_embeddings(history):
    if isinstance(history, list):
        return [news_embeddings_dict.get(news_id, [0]*768) for news_id in history]
    else:
        # In case 'history' is not a list, return an empty list
        return []

# Function to map impressions news_ids to embeddings with click labels
def map_impressions_to_embeddings(impressions):
    if isinstance(impressions, list):
        return [(news_embeddings_dict.get(news_id, [0]*768), click) for news_id, click in impressions if news_id in news_embeddings_dict]
    else:
        # If 'impressions' is not a list, return an empty list
        return []
```

5.6 Applying Mapping Functions:

Apply mapping functions to add embeddings columns to the behaviour data:

```
# Step 5: Apply Mapping Functions to Generate Embeddings


# Apply the mapping functions to create 'history_embeddings' and 'impressions_embeddings'
behaviors_df['history_embeddings'] = behaviors_df['history'].apply(map_history_to_embeddings)
behaviors_df['impressions_embeddings'] = behaviors_df['impressions_processed'].apply(map_impressions_to_embeddings)

# Display a sample to verify the embeddings mapping
print("After Mapping Embeddings:")
print(behaviors_df[['history_embeddings', 'impressions_embeddings']].head())
print("\nData types in 'history_embeddings' column:")
print(behaviors_df['history_embeddings'].apply(type).value_counts())
print("\nData types in 'impressions_embeddings' column:")
print(behaviors_df['impressions_embeddings'].apply(type).value_counts())
print("\n-----------------------------------------\n")
```

6 Conclusion:

This Configuration manual helps in settings up environment, verifying GPU availability, loading and processing datasets, extracting BERT embeddings for news articles, and mapping embeddings to behaviour data. This setup is crucial for building the recommendation model that leverages both content-based and collaborative filtering approaches.