National College of Ireland

Configuration Manual

MSc Research Project
MSc. in Artificial Intelligence

Visakh Vijayakumar Nair
Student ID: 23198273

School of Computing
National College of Ireland

Supervisor: Rejwanul Haque

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Visakh Vijayakumar Nair……
Student ID:	23198273……..……
Programme:	MSc in Artificial Intelligence **Year:** 2024……..
Module:	Practicum ……..……
Lecturer:	Dec 12, 2024 ……..……
Submission Due Date:	……..……
Project Title:	Beyond Accuracy: A Comparative Analysis of Recommendation Models Incorporating Quantitative and Qualitative Evaluation
Word Count:	1,312…… **Page Count:** 10 ……..……..……

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: ……

Date: 12 Dec, 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

Visakh Vijayakumar Nair
Student ID: 23198273

1 Introduction

This configuration manual provides detailed instructions for setting up and running the research project titled "**Beyond Accuracy: A Comparative Analysis of Recommendation Models Incorporating Quantitative and Qualitative Evaluation Metrics**". The project aims to develop two different recommendation models and compare both their predictive accuracy and their ability to deliver a qualitative user experience.

2 Section 2

2.1 Hardware Configuration

In this research study a local machine (Apple MacBook Pro) is used as Infrastructure as a Service (IaaS). The configuration of the local machine used is given below in the Table 1.

Hardware Configuration	
Operating System	MacOS 14.3.1
RAM	32 GB
Virtual CPUs	10
Hard Disk Storage	480

2.2 Software Configuration

To complete this study, some external software is used. The list of all the software and their versions are given below.

Software Configuration	
Conda	23.7.4
Jupyter Notebook	7.2.2
Python	3.9.19
MongoDb	7.0.8

Conda:
Conda is a versatile package and environment management tool that is used to create independent and recreatable environments for this project. With the support of Conda, dependencies for different models and tools are managed efficiently, ensured compatibility, and preventing conflicts between libraries.

Jupyter Notebook:
Jupyter Notebook is used as a development interface for this project, which provides an interactive environment to write, test, and document code. Its ability to combine executable code, visualizations, and markdown notes was invaluable for experimentation and detailed analysis.

MongoDB:
To store and manage the project dataset MongoDB is used, which can handle unstructured data. Its flexible schema and ability to handle large-scale, unstructured data made it a suitable choice for this research. MongoDB's querying capabilities are practically useful for efficiently retrieving and analysing the data required for model training and evaluation.

Python:
Python is used as the core programming language used for the development and analysis of the recommendation models. Its extensive libraries, such as TensorFlow, PyTorch, and Scikit-learn, facilitated the implementation and evaluation of both recommendation models.

Here are the python libraries that we used in this project.

Python Libraries Used	
scikit-learn	1.5.2
tensorflow	2.17.0
torch	2.0.1
seaborn	0.13.2
pymongo	4.10.1
numpy	1.23.5
keras	3.5.0
pandas	1.5.3

3 Project Development

3.1 Data Preparation

The data is stored in a MongoDB database, so to access the data and pre-process it we use Python and PyMongo. At first, we need to import required libraries, then connect with MongoDB server. After that, access the *findups_daily_next_world* database.

```
from pymongo import MongoClient
```

```
from bson import ObjectId
```

```
client = MongoClient("mongodb://localhost:27017/")
```

Figure 1: Importing Libraries for accessing MongoDb Database

3.1.1 Creating *read_logs* collection

In our database we have three different collections: *users, news, news_action_logs*. To train a model, we need all the required information in one single dataset. For that, we build a new MongoDB Collection named *read_logs*.

For that, we write a set of programmes which will traverse through each entry of *news_action_logs*, then access the *news_id*, *user_id*, *time_stamp* and weather the user interacted with the news or not (*is_read*). After that, by matching up the *news_id* with the '*_id*', the *news_content* will be added to the dataset. After completing all these steps, the data will be added to the MongoDB database as a new collection and named it as *read_logs*.

```
count = 0
read_log = []
for news in news_logs:
    read_entry = {}
    read_entry['news_id'] = news['newsId']
    read_entry['user_id'] = news['user']
    read_entry['time_stamp'] = news['updatedOn']

    if news['isBookmarked'] or news['isLiked'] or news['isShared'] or news['isRead']:
        read_entry['is_read'] = True
    else:
        read_entry['is_read'] = False

    news_id = ObjectId(read_entry['news_id'])
    news = list( db.extracted_news.find({'_id': news_id}) )

    if len(news) < 1:
        continue

    read_entry['news_content'] = news[0]['news_content']

    read_log.append(read_entry)

db.read_logs.insert_many(read_log)
```

Figure 2: The set of code for generating read_logs dataset.

3.1.2 Embedding Textual Data

As we can see, the most important data, *news_content*, is a textual data. This textual data set cannot be use as an input for the machine learning or neural network model training. So we need to convert the textual form to embeddings.

To embed the text content, here we use BERT. For that, the BERT modules, *BertTokenizer* and *BertModel*, are imported from the *transformers* library. Then we load pre-trained BERT tokenizer and model, '*bert-base-uncased*'.

A separate function will be written with a sole responsibility to provide embedding values of given text content. The function, named *get_news_embedding,* which have one single parameter '*text_content*', will convert the textual value to tokens. After that the tokens will be pass through the *BertModel* and convert it into embedding, which have 768 dimensions.

The embeddings of each *news_content* will be identified using this function with the support of Python's Lambda feature.

```
from transformers import BertTokenizer, BertModel

# Load pre-trained BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')

def get_news_embedding(text_content):
    inputs = tokenizer(text_content, return_tensors='pt', truncation=True, padding=True)
    outputs = model(**inputs)
    # Use the CLS token representation as the embedding
    return outputs.last_hidden_state[:, 0, :].detach().numpy()

# Assuming you have a dataframe `news_df` with `news_id` and `news_content`
news_df['embedding'] = news_df['news_content'].apply(get_news_embedding)
```

Figure 3: Generating embeddings for each *news_content*.

3.2 Transformation

In this project, we build two different recommendation models to compare its performance. For the second model **Behavioural Pattern Learning Model** requires sequential data, which provides *n* news that user already interested in. Following a target news that user to predict whether the user read or ignore. And label column which shows, weather the user read the news or not.

We need to check this functionality on various sequence, by adjusting the value of *n* (the number of news that the user read before. A function has been written to generate such sequential dataset, by which can adjust the number of news that user read.

```python
def get_sequense_data(min_checks):
    user_news = {}
    data = []

    # min_checks = 7

    for index, news in news_df.iterrows():
        row = []

        news_array = user_news.get(news['user_id'], [])
        if len(news_array) < min_checks:
            news_array.append(news['embedding'])
        else:
            news_array.pop(0)
            news_array.append(news['embedding'])

        user_news[news['user_id']] = news_array
        # print(news_array)
        if len(news_array) == min_checks:
            row = [news['user_id']]

            for i in range(min_checks):
                row = row + user_news[news['user_id']][i][0].tolist()

            # row = row.tolist()
            row.append(news['is_read'])


            data.append(row)
    return pd.DataFrame(data)
```

Figure 4: Function to generate sequence dataset.

Here is a sample dataset that the function generated with 4 interacted news, 1 target news and label column.

```python
pd_news = get_sequense_data(5)
```

```python
pd_news.drop(0, axis=1, inplace=True)
```

```python
pd_news.head()
```

3832	3833	3834	3835	3836	3837	3838	3839	3840	3841
396573	0.085705	-0.307321	-0.282962	0.178083	-0.290556	-0.588054	0.578497	0.403499	False
226925	0.279387	-0.371205	0.068680	0.189348	-0.081013	-0.402511	0.288362	0.795604	False
.122274	0.629863	-0.042809	0.335417	-0.405624	-0.351386	-0.007708	0.752452	-0.157104	False
303019	0.386305	-0.491156	0.551000	-0.415115	-0.429968	-0.068254	0.772537	-0.080966	False
088700	0.292305	-0.205667	0.032945	-0.071510	-0.172701	-0.572488	0.201037	0.201912	False

Figure 5: Sequential Dataset

4 Model Development

Here build two recommendation model to compare its performance in quantitative and qualitative basis.

4.1 Embedding-Based Ranking Model

The working of Embedding-Based Ranking Model is that it creates a user profile by accessing all the news that read by the user. First, the sum of all the embeddings of those news will be taken, and then the mean value of it will be calculated. That value will be considered as *user_profile*.

```
import numpy as np

def get_user_profile(user_id, read_log_df, news_df):
    # Get all news read by the user
    read_news_ids = read_log_df[(read_log_df['user_id'] == user_id) & (read_log_df['is_read'] == True)]['news_id']
    read_news_embeddings = news_df[news_df['news_id'].isin(read_news_ids)]['embedding'].values
    # Average embeddings to create user profile
    user_profile = np.mean(np.vstack(read_news_embeddings), axis=0)
    return user_profile
```

Figure 6: function for creating user profile.

Here, the model will recommend *top_n* number of news for the users which have the embedding values closer to the user_profile. For that, the user_profile will be calculated. Then all the news that user didn't interact will be identified. After that the distance between the user_profile and the embedding of each news will be calculated using *cosine_similarity* function from *sklearn*.

```
from sklearn.metrics.pairwise import cosine_similarity

def recommend_news(user_id, read_log_df, news_df, top_n=5):
    # Get user profile
    user_profile = get_user_profile(user_id, read_log_df, news_df)
    # Get embeddings of all news not read by the user
    unread_news_df = news_df[~news_df['news_id'].isin(read_log_df[read_log_df['user_id'] == user_id]['news_id'])]
    unread_news_embeddings = np.vstack(unread_news_df['embedding'].values)
    # Compute cosine similarity
    similarities = cosine_similarity([user_profile], unread_news_embeddings)[0]
    # Get top N recommendations
    top_n_idx = similarities.argsort()[-top_n:][::-1]
    recommended_news = unread_news_df.iloc[top_n_idx]
    return recommended_news
```

Figure 7: *recommend_news* function.

By adjusting the top_n parameter value, the function can predict as many news as we required. We also need to pass *user_id*, list of news that the user already read (*read_news*) and all the news list (*news_df*).

4.2 Behavioural Pattern Learning Model

The working of this model is that it creates a sequence dataset by combining the embeddings *n* previously read news by the user, and a target news. Followed that, a label column which shows weather user interact with the target news or not. To create such database, we already created a function *get_sequence_data(),* (check Figure 5).

After some trial and error, it is identified that the combination of 4 previously interacted news and 1 target news sequence dataset can train a better performing model.

4.2.1 Splitting Data into Train and Test

The data is separated like all the embedding sequences are moved to **X**. Then the labels are assigned as **y**.

```
X = pd_news.iloc[:, :-1].values
y = pd_news.iloc[:, -1].values
```

Figure 8: X and y Data separation.

Then the data is converted into training and testing datasets by 80:20 ratio. That means 80% of total data is taken for training and remaining 20% is assigned for testing.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 9: Training - Testing data splitting.

4.2.2 Creating Weighted Class

Since the dataset is unbiased, we need to build a weighted class to avoid bias issues in model.

```
class_weights = class_weight.compute_class_weight('balanced', classes=np.unique(y_train), y=y_train)
class_weights_dict = dict(enumerate(class_weights))
```

Figure 10: Generating weighted class for avoiding bias.

4.2.3 Design Neural Network

To design and implement neural network, here we use TensorFlow library. It processes structured input data to predict a binary outcome, such as user engagement with a specific item.

Input Layer:

The input layer accepts a feature vector with a size equal to the number of features in the training data (X_train.shape[1]). This ensures compatibility with the dataset's dimensionality.

Hidden Layers:

The first hidden layer contains **768 neurons** with a ReLU (Rectified Linear Unit) activation function. This layer learns to capture high-level patterns from the input data.

```
model = tf.keras.models.Sequential([
    tf.keras.layers.InputLayer(shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(768, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')  # Binary classification
])
```

Figure 11: Neural Network Design

The second hidden layer, with **128 neurons**, refines these patterns further, extracting intermediate-level features.

The third hidden layer, with **64 neurons**, extracts the information into meaningful, lower-dimensional representations to improve the model's ability to generalize.

Output Layer:
The final layer consists of a single neuron with a **sigmoid activation function**, producing a probability value between 0 and 1. This represents the likelihood of a positive classification, such as whether the user will engage with the target article or item.

This architecture is well-suited for capturing complex relationships in high-dimensional data while maintaining a focus on binary outcomes. It balances feature extraction and decision-making through its layered design.

4.2.4 Model Compile
The model is compiled with configurations tailored for binary classification tasks:

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy', 'recall'])
```

Figure 12: Model compile by adding optimizer, loss function and metrics.

Optimizer:

The Adam optimizer is selected for its efficiency in handling large datasets and its ability to adapt learning rates during training dynamically. This helps the model converge faster and more effectively compared to traditional optimization techniques.

Loss Function:

The binary cross-entropy loss function is used, which is ideal for binary classification problems. It quantifies the difference between predicted probabilities and actual binary labels, guiding the model to minimize errors during training.

Evaluation Metrics:

Accuracy is included to measure the overall correctness of the model's predictions.

Recall is added to emphasize the model's ability to correctly identify positive instances (true positives), which is particularly important in scenarios where missing a relevant recommendation (false negatives) can have significant implications.

This compilation setup ensures the model is optimized for both training efficiency and performance evaluation, with a focus on critical aspects of binary classification.

4.2.5 Model Training

Then the model trained on 20 epochs with 24 batch size, and 20% data is taken for validation purpose. The training will also be influenced by the weighted class, so it can avoid bias due to the imbalance of dataset.

```
model.fit(X_train, y_train, epochs=20, batch_size=24, validation_split=0.20, class_weight=class_weights_dict)
```

Figure 13: Model training with weighted class.