ANALYZING TRENDS OF SOCIAL NETWORK A CASE STUDY ON INSTAGRAM

A PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

COMPUTER SCIENCE AND ENGINEERING

BY

P. Aditya Goutam
21331A05E1

V.V.S. Chandra Mouli
21331A05I0

T. Damayanthi
21331A05H6

S. Vamshi
22335A0516

Under the Supervision of
Mrs. K. Sobha Rani
Professor(TP)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MAHARAJ VIJAYARAM GAJAPATHI RAJ COLLEGE OF ENGINEERING
(Autonomous)
(Approved by AICTE, New Delhi, and permanently affiliated to JNTUGV, Vizianagaram), Listed u/s 2(f) & 12(B) of UGC Act 1956.
Vijayaram Nagar Campus, Chintalavalasa,Vizianagaram-535005, Andhra Pradesh

APRIL, 2025

CERTIFICATE

This is to certify that the project report entitled "**Analyzing the trends of social network – A case study on Instagram**" being submitted by P. Aditya Goutam(21331A05E1), V.V.S. Chandra Mouli(21331A05I0), T. Damayanthi(21331A05H6), S. Vamshi(22335A0516) in partial fulfillment for the award of the degree of "**Bachelor of Technology" in Computer Science and Engineering** is a record of Bonafide work done by them under my supervision during the academic year 2024-2025.

Mrs. K. Sobha rani
Professor(TP),
Supervisor,
Department of CSE,
MVGR College of Engineering(A),
Vizianagaram.

Dr. T. Pavan Kumar
Professor,
Head of the Department,
Department of CSE,
MVGR College of Engineering(A),
Vizianagaram.

External Examiner

DECLARATION

We hereby declare that the work done on the dissertation entitled "**Analysing the trends of social network – A case study on Instagram**" has been carried out by us and submitted in partial fulfilment for the award of credits in Bachelor of Technology in Computer Science and Engineering of MVGR College of Engineering (Autonomous) and affiliated to JNTUGV, Vizianagaram. The various contents incorporated in the dissertation have not been submitted for the award of any degree of any other institution or university.

ACKNOWLEDGEMENTS

We express our sincere gratitude to **Mrs. K. Sobha Rani** for her invaluable guidance and support as our mentor throughout the project. Her unwavering commitment to excellence and constructive feedback motivated us to achieve our project goals. We are greatly indebted to her for the guidance and mentoring.

Additionally, we extend our thanks to **Prof. P.S. Sitharama Raju, Director**, **Prof. Ramakrishnan Ramesh, Principal**, and **Dr. T. Pavan Kumar, Head of the Department** for their unwavering support and assistance, which were instrumental in the successful completion of the project. We also acknowledge the dedicated assistance provided by all the staff members in the Department of Computer Science & Engineering. Finally, we appreciate the contributions of all those who directly or indirectly contributed to the successful execution of this endeavour.

P. Aditya Goutam (**21331A05E1**)
V.V.S. Chandra Mouli (**21331A05I0**)
T. Damayanthi (**21331A05H6**)
S. Vamshi (**22335A0516**)

LAST MILE EXPERIENCE (LME)

PROJECT TITLE

Analysis of social network
A case study on Instagram

BATCH NUMBER – 12C

BATCH SIZE – 4

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Name: P. Aditya Goutam
Email: p.adityagoutam009@gmail.com
Contact Number: 9381735639

Name: V.V.S Chandra Mouli
Email: moulismp3377@gmail.com
Contact Number: 9100781316

Name: T. Damayanthi
Email: tompaladamayanthi@gmail.com
Contact Number: 7981175251

Name: S. Vamshi
Email: iamvamsi0@gmail.com
Contact Number: 9381721427

Project Supervisor

Name: Mrs. K. Sobha Rani
Designation: Professor(TP)
Email: sobharani@mvgrce.edu.in
Contact Number: 9440127218

Project Objectives

1. **PO1:**
   Analysis of social network – A case study on Instagram aims to understand the recommendation system algorithm of Instagram.
2. **PO2:**
   The objective of this project is to replicate the recommendation system algorithm of Instagram.

Project Outcomes

1. **Overall Project Outcome:**
   Try to understand the recommendation system of Instagram and analyse profile attributes using domain specific API**.**

Domain of Specialisation

1. Data processing
2. Machine learning
3. Social Network Analysis

How your solution helping the domains?

1. Understanding the general working of recommendation system used in social media applications.
2. Understanding how basic machine learning works and how to deploy and train models.

List the Program Outcomes (POs) that are being met by doing the project work

PO9: Individual and teamwork	Function effectively as an individual, and as a member or leader in diverse teams and in multi-disciplinary settings.
PO10: Communication	Communicate effectively on complex engineering activities with the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11: Project management and finance	Demonstrate knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work as a member and leader in a team, to manage projects and in multi-disciplinary environments.
PO12: Life-long learning	Recognize the need for, and have the preparation and ability to engage in, independent and life-long learning

End Users of Your Solution

The project serves as a basic understanding of the current recommendation systems of social media network.

ABSTRACT

This project focuses on the design and implementation of a recommendation system for social media platforms, specifically targeting Instagram. The system is built using a deep learning-based approach, leveraging a two-tower neural network model. The objective is to recommend relevant Instagram profiles or posts to users by learning relationships between user characteristics and content features.

The process begins with data collection using the Instagram API via RapidAPI, where user attributes such as verification status and follower counts, along with content metrics like likes and comments, are extracted. After data acquisition, preprocessing techniques are applied to normalize and structure the dataset for optimal model performance.

The recommendation engine is based on a two-tower architecture where user features and content features are processed through separate neural networks. These networks learn dense representations (embeddings) for both users and posts, which are then compared using a dot product similarity function to assess relevance. The system is designed to deliver personalized recommendations by matching user profiles to the most appropriate posts or accounts.

This study demonstrates how deep learning techniques can be effectively applied to solve recommendation problems in social media environments and lays the groundwork for further enhancements through advanced models and additional data enrichment.

CONTENTS

List of Abbreviations

AI	–	Artificial Intelligence
ML	–	Machine Learning
NN	–	Neural Networks
ANN	–	Artificial Neural Networks
CNN	–	Convolutional Neural Networks
EDA	–	Exploratory Data Analysis
IDA	–	Integrated Data Analysis
DML	–	Data Manipulation Language
TP	–	True Positives
TN	–	True Negatives
FP	–	False Positives
FN	–	False Negatives
TPR	–	True Positive Rate
FPR	–	False Positive Rate

List of Figures

CHAPTER 1

INTRODUCTION

With the advent of the digital era, social media analytics has emerged as a crucial instrument for companies, researchers, and policymakers aiming to understand the humongous amounts of data being created by websites like Facebook, Twitter, and Instagram These sites are vibrant communities where billions of people engage on a daily basis, leaving a huge digital trail. By monitoring different indicators like user interactions, content engagement, and sentiment trends social media analytics helps stakeholders attain in-depth knowledge of consumer behavior, nascent trends, and market movement. Ongoing development of social networks has transformed information sharing, making communication, opinions sharing, and brand interaction processes a lot more dynamic. Consequently, data-driven decision-making has turned into an indispensable tool for businesses that want to increase customer engagement, refine marketing strategies, and maximize overall performance. Among the platforms, Instagram is a forceful tool when it comes to understanding visual content-driven social interactions.

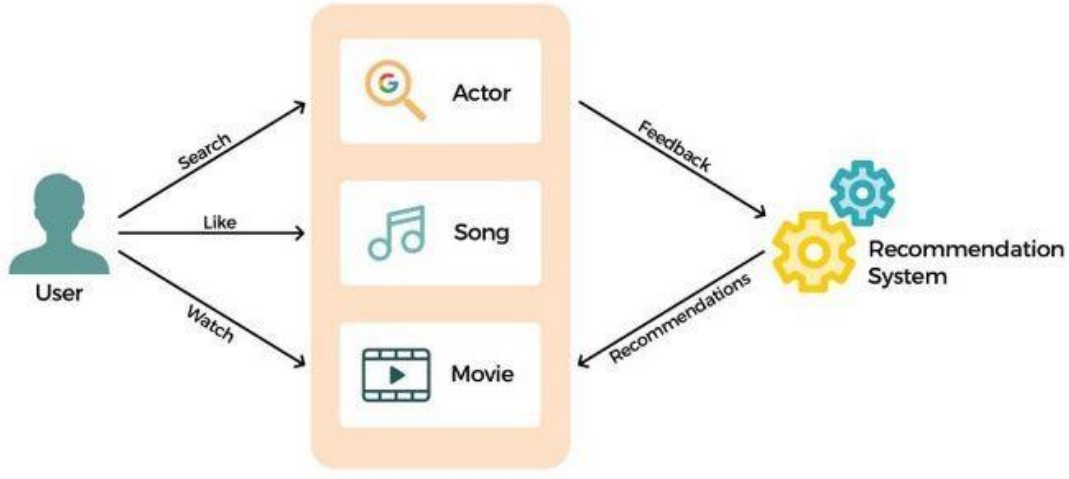

Fig:1 Introduction to recommendation systems

Being one of the most impactful social media sites, it showcases wider digital patterns through its hashtags, posts, stories, and reels, providing a precious dataset to be analyzed. Through the use of Instagram data, it is possible to analyze user preferences, popular topics, and their economic or business implication. The use of advanced predictive analytics and machine learning algorithms is important for detecting behavioral patterns in users and predicting social media trends of the future.

CHAPTER 2
LITERATURE SURVEY

- The Google Scholar was employed to fetch a pool of research articles relevant to recommendation systems, particularly focusing on deep learning techniques. Articles related to the three main types of recommendation systems were extracted for analysis.
- Collaborative Filtering: Keywords such as "collaborative filtering deep learning," "user-item matrix collaborative filtering,"[1] and "matrix factorization recommendation systems" were used to gather papers discussing methods where recommendations are based on user interactions and preferences.
- Content-Based Filtering: To cover the literature on content-based approaches, keywords like "content-based filtering recommender system," "recommendation based on content attributes," and "deep learning for content-based recommendations" were used. These papers focus on how item features (e.g., tags, descriptions, or images) can influence the recommendation process. For example, LIBRA is a content-based book recommendation system that uses information about books gathered from the Web.

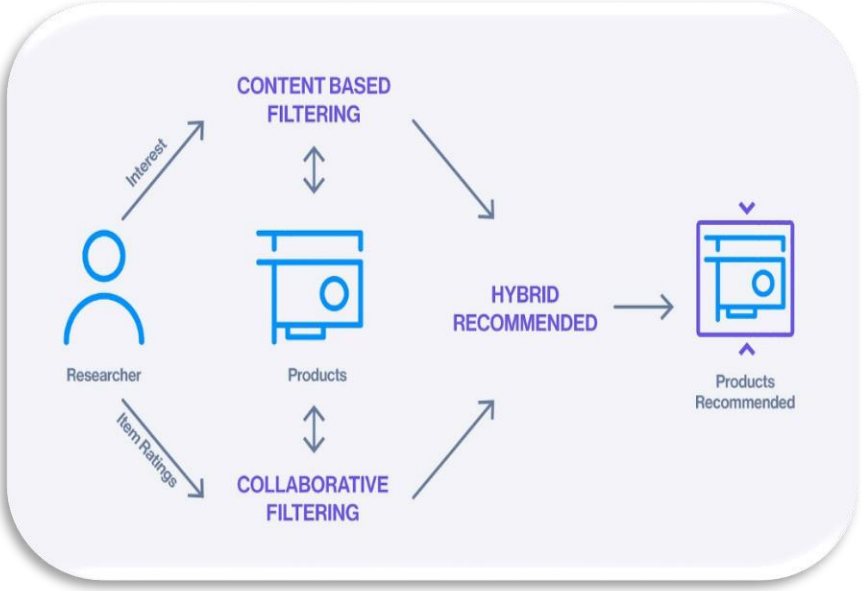

Fig:2 Types of basic recommendation systems

- **Hybrid Models**: Keywords such as "hybrid recommendation system deep learning," "combining collaborative and content-based filtering," and "deep learning hybrid recommender systems" were used to gather papers that discuss the integration of both collaborative and content-based approaches for generating recommendations.
- Additionally, we also included more general terms such as “recommender system deep learning” and “recurrent neural network recommender systems” to ensure that we capture literature involving advanced machine learning techniques in recommendation systems. Most used are LSTM and GRU which have their pros with a variety kind of data.
- To ensure relevance and recency, we applied a time filter of "Since 2013," so that only articles from the last 12 years were included in the review.
- In addition to these, research on Hybrid Recommendation Systems was also explored. Using keywords such as "hybrid recommender systems deep learning," "combining collaborative and content-based filtering," and "ensemble methods in recommendation engines," several papers were reviewed that highlight the integration of both collaborative and content-based techniques. These hybrid models aim to overcome the limitations of individual methods, such as the cold-start problem or data sparsity, by leveraging the strengths of both approaches. For instance, some studies propose architectures where collaborative filtering models provide initial user-item similarity scores, which are then refined using deep learning models that capture item content or contextual data.
- Further, the literature emphasizes the growing role of deep learning frameworks like autoencoders, recurrent neural networks (RNNs)[5][6], and transformer-based models in building next-generation recommender systems. These techniques have demonstrated improved performance in handling complex user-item interactions, dynamic user preferences, and multi-modal data (e.g., images, text, and metadata). The reviewed articles also highlight industry applications, including recommendation systems used by platforms like Netflix, Amazon, and YouTube, where deep learning has significantly enhanced recommendation accuracy and personalization.

CHAPTER 3

THEORETICAL BACKGROUND

3.1 Machine learning Vs Deep learning

3.1.1 What is Machine Learning?

Machine learning is an application of AI that enables systems to learn and improve from experience without being explicitly programmed. Machine learning focuses on developing computer programs that can access data and use it to learn for themselves. Machine learning is something that is capable to imitate the intelligence of the human behavior. Machine learning is used to perform complex tasks in a way that humans solve the problems. Machine learning can be descriptive it uses the data to explain, predictive, and prescription.

3.1.2 Why Machine Learning?

Machine learning involves computers learning from data provided so that they carry out certain tasks. For more advanced tasks, it can be challenging for a human to manually create the needed algorithms. In practice, it can turn out to be more effective to help the machine develop its own algorithm, rather than having human programmers specify every needed step. The discipline of machine learning employs various approaches to teach computers to accomplish tasks where no fully satisfactory algorithm is available. In cases where vast numbers of potential answers exist, one approach is to label some of the correct answers as valid. This can then be used as training data for the computer to improve the algorithms it uses to determine correct answers. The nearly limitless quantity of available data, affordable data storage, and growth of less expensive and more powerful processing has propelled the growth of ML. Now many industries are developing more robust models capable of analysing bigger and more complex data while delivering faster, more accurate results on vast scales. ML tools enable organizations to more quickly identify profitable opportunities and potential risks.

The practical applications of machine learning drive business results which can dramatically affect a company's bottom line. New techniques in the field are evolving rapidly and expanded the application of ML to nearly limitless possibilities.

3.2 Machine Learning Approaches

Machine learning approaches are traditionally divided into three broad categories, depending on the nature of the "signal" or "feedback" available to the learning system:

- ✓ Supervised learning
- ✓ Unsupervised learning
- ✓ Reinforcement learning

3.3 Machine Learning Models

Performing machine learning involves creating a model, which is trained on some training data and then can process additional data to make predictions. Various types of models have been used and researched for machine learning systems.

3.3.1 Two Tower neural networks

One of the most important types of models in recommendation systems at present is the two-tower neural networks. They are structured as follows: one part of the neural network (tower) processes all the information about the query (user, context), while the other tower processes information about the object. The outputs of these towers are embeddings, which are then multiplied.

What characterizes such networks? They are very similar to matrix factorization, which is actually a special case, taking only user_id and item_id as input. However, if we compare them with arbitrary networks, the restriction on late crossing (not allowing inputs from different towers to fuse until the very end) makes two-tower networks extremely efficient in application. To build recommendations for a single user, we need to calculate the query tower once, and then multiply this embedding with the embeddings of documents, which are usually pre-calculated. This process is very fast. Moreover, these pre-calculated document embeddings can be organized into an ANN index (e.g., HNSW) to quickly find good candidates without having to go through the entire database.

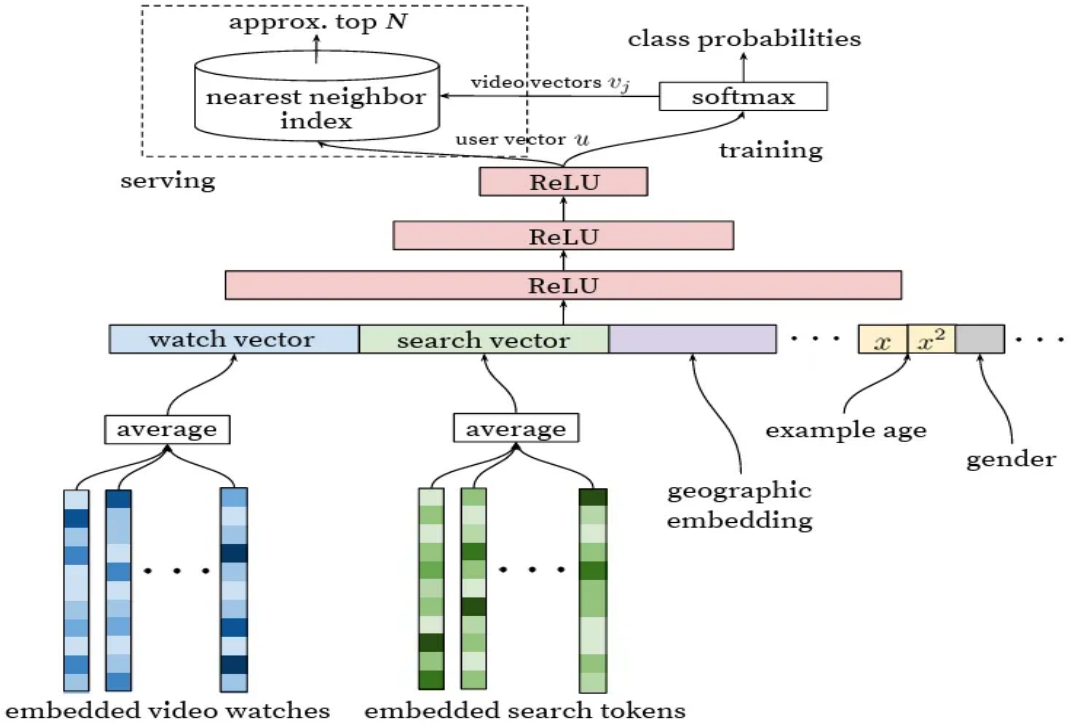

Fig:3 Two Tower NN

We can achieve even more efficiency by computing the user part not for every query, but asynchronously, with some regularity. However, this means sacrificing the consideration of real-time history and context.

The towers themselves can be quite sophisticated. For example, in the user part, we can use the self-attention mechanism to process history, which results in a transformer for personalization. But what is the cost of introducing the restriction on late crossing? Naturally, it affects quality. In the same attention mechanism, we cannot use the items that we currently wish to recommend. Ideally, we would like to focus on similar items in the user's history. Therefore, networks with early crossing are typically used in the later stages of ranking, when there are only a few dozen or hundred candidates left, while those with late crossing (two-tower) are used, conversely, in the early stages and for candidate generation.

3.3.2 Loss Functions and Negative Sampling

A particular point of interest is the loss function used to train two-tower networks. In principle, they can be trained with any loss function, targeting various outcomes and even having multiple different ones for different heads (with different embeddings in each tower). However, an interesting variant is training with a softmax loss on in-batch negatives. For each query-document pair in the dataset, the other documents in the same mini-batch are used as negatives in combination with the same query in the SoftMax loss. This method is a highly effective form of hard negative mining.

But it's important to consider the probabilistic interpretation of such a loss function, which is not always well-understood. In a trained network,

$$e^{\text{score}(q,d)} \propto \frac{P(q,d)}{P(q)P(d)} = \frac{P(d|q)}{P(d)}$$

The exponent of the score is proportional not to the a priori probability of the document given the query, but to the PMI (Pointwise Mutual Information), specific to the query. More popular documents will not necessarily be recommended more often by such a model, because during training, they proportionally appear more often as negatives. Using the score as a feature can be beneficial, but for final ranking and candidate generation, this can lead to very specific, yet poor-quality documents.

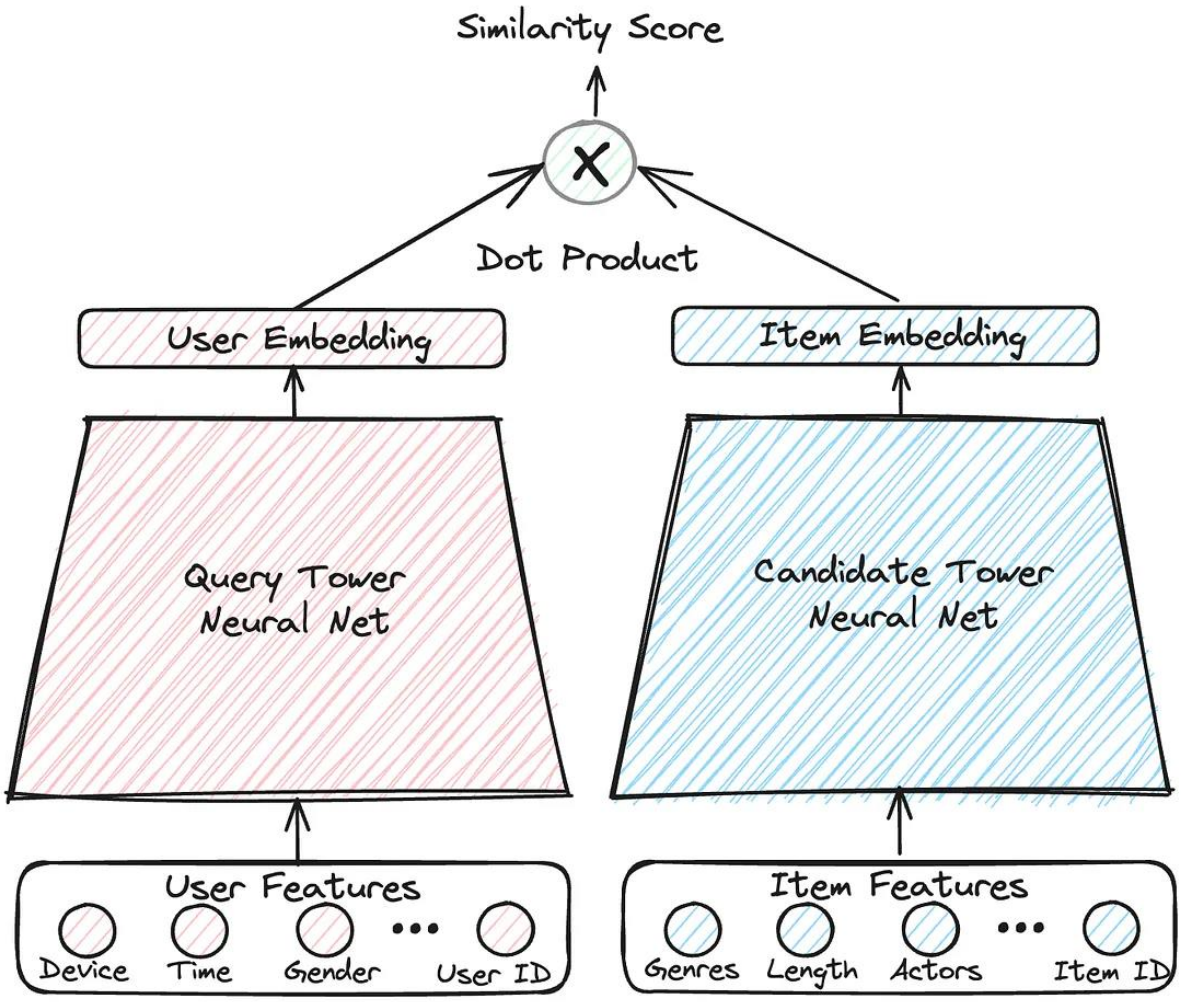

Fig:4 Stages of Two Tower NN

Here's how the Two Tower retrieval works in general with schema:

1. The Two Tower model consists of two separate neural networks – one for the user and one for the item.
2. Each neural network only consumes features related to their entity and outputs an embedding.
3. The learning objective is to predict engagement events (e.g., someone liking a post) as a similarity measure between user and item embeddings.
4. After training, user embeddings should be close to the embeddings of relevant items for a given user. Therefore, item embeddings close to the user's embedding can be used as candidates for ranking.

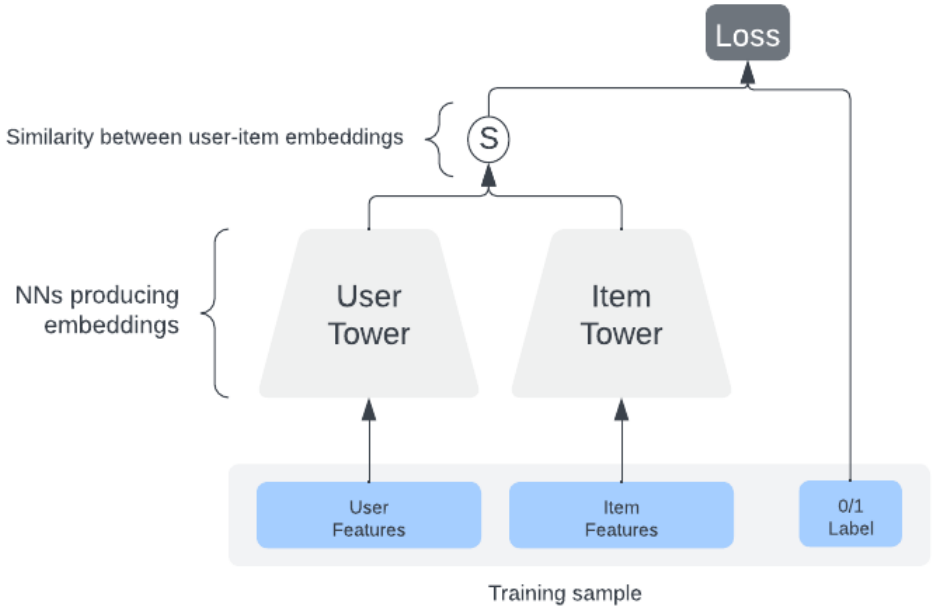

Fig:5 Similarity Function

An **API (Application Programming Interface)** is a set of defined rules and protocols that allow different software applications to communicate and share data or functionality with each other.

An API acts as an intermediary that enables two software programs to interact without the need to know how each other is implemented internally. APIs define the methods and data formats applications can use to request and exchange information. They can be used for accessing web-based services (e.g., REST or SOAP APIs), databases, operating system functions, or libraries. For example, when you use a mobile app to check the weather, it communicates with a remote server via an API to fetch and display the latest weather data.

APIs are crucial for modern software development because they allow developers to integrate external services or modules instead of building them from scratch. They improve efficiency, scalability, and flexibility by enabling modular programming. Public APIs, like Google Maps API or Twitter API, allow third-party developers to enhance their own applications with features provided by other platforms. Private APIs, on the other hand, are used internally within organizations. Overall, APIs drive interoperability and automation in software ecosystems, making them a fundamental component of application development today

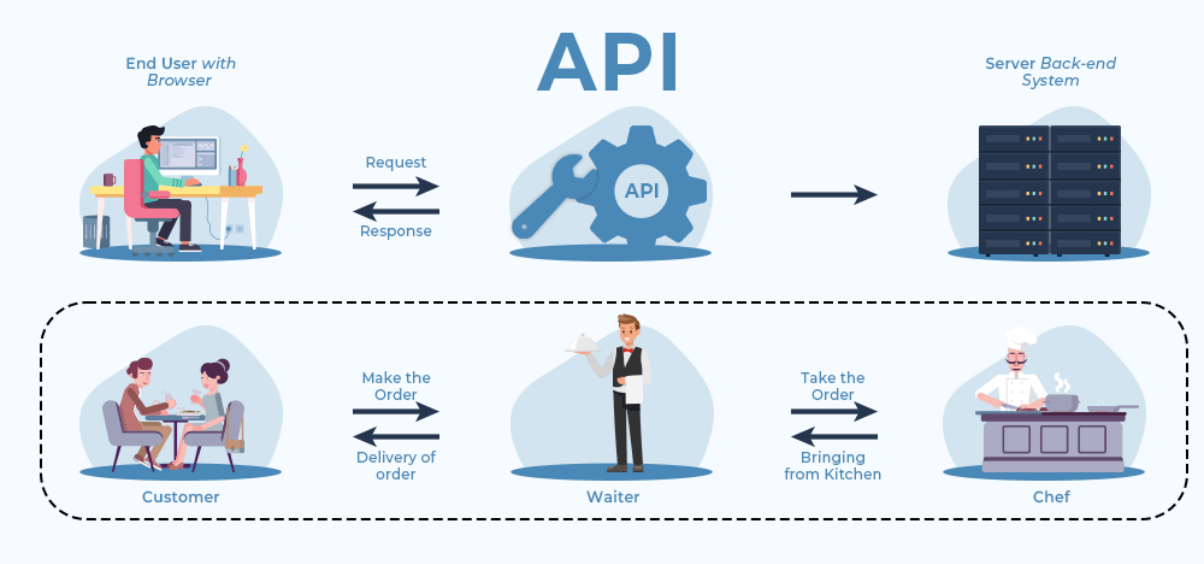

Fig:6 Depiction of API

Social Networking APIs (Application Programming Interfaces) are powerful tools that allow developers to access and interact with data and services provided by social media platforms such as Facebook, Instagram, Twitter (X), LinkedIn, and others. These APIs enable external applications to perform a wide range of operations, including fetching user profiles, retrieving posts, accessing comments and likes, posting content, and analyzing social graphs.

APIs expose endpoints that allow applications to programmatically interact with social media servers without manual intervention. For instance, developers can use Instagram's Graph API to collect user media, engagement metrics, and follower data for analytical or marketing purposes. Similarly, Twitter's API provides access to tweet history, trends, and sentiment analysis data.

Social Networking APIs are commonly used to power recommendation systems, targeted advertisements, social media analytics dashboards, and customer relationship management (CRM) tools. They facilitate automation, help personalize user experiences, and support businesses in deriving valuable insights from social interactions. These APIs often require authentication mechanisms such as OAuth to ensure secure and authorized access.

However, each platform enforces specific data usage policies and rate limits, making it essential to comply with privacy guidelines and best practices when integrating these APIs into applications.

CHAPTER 4
APPROACH DESCRIPTION

1. Data Collection via Instagram API

The project begins with the integration of the Instagram Graph API to extract valuable user and post-related data. Through authenticated API calls, key metrics such as follower_count, following_count, media_count, is_verified status, as well as post engagement features like like_count and comments_count are gathered. This real-world dataset forms the foundation for building a user-content recommendation engine. The API enables periodic data fetching while respecting rate limits and access permissions imposed by Instagram.

2. Data Preprocessing and Feature Engineering

After successful data extraction, the collected dataset undergoes preprocessing. Two distinct sets of features are engineered: user features and content features. User features include behavioral and profile-related attributes such as follower_count, following_count, and verification status. Content features primarily include engagement statistics of posts like like_count, comments_count, and media_type.

To standardize the features and ensure the model converges effectively, MinMaxScaler from the scikit-learn library is used to normalize all numerical features within the range [0,1]. This step is crucial for improving model stability during training. Additionally, the dataset is divided into training and testing subsets using an 80-20 split, allowing for the later evaluation of model generalization performance.

3. Two-Tower Neural Network Architecture

The project employs a Two-Tower Neural Network (TTNN) approach. This model consists of two independent neural networks (towers) designed to process the two input types: user features and content features. Both towers have identical architectures comprising multiple dense layers (with ReLU activation), Dropout layers to prevent overfitting, and a final dense layer generating a 16-dimensional embedding for both the user and the content.

Once the embeddings are generated from both towers, they are merged through a dot product layer to capture the similarity score between user preferences and content

characteristics. The dot product serves as a scoring mechanism that reflects how likely a user is to engage with a particular piece of content.

4. Model Training and Evaluation

The model is compiled using Binary Cross entropy as the loss function and the Adam optimizer, with accuracy selected as the evaluation metric. The model is trained over 32 epochs, during which it learns to maximize the similarity between relevant user-content pairs while minimizing it for irrelevant ones.

The training phase outputs key metrics like training loss and accuracy after each epoch. The model is then evaluated using the unseen test dataset, where its performance is measured based on how accurately it can predict relevant content recommendations.

5. Recommendation Generation

Post-training, the learned embeddings are utilized to generate actual recommendations. Given a sample user input, the model computes dot product scores with all available content embeddings. These scores are then sorted, and the Top-5 most relevant posts are selected as recommendations for the user. This retrieval strategy enables the system to provide tailored content recommendations based on learned patterns from user and content data.

6. Visualization and Insights

Throughout the process, data distributions and correlations are visualized using Seaborn and Matplotlib libraries. Heatmaps reveal relationships between user and content attributes, while line plots track training performance over epochs. These visualizations provide actionable insights into feature importance and model behavior.

In summary, this project follows a complete pipeline from social media data extraction to real-time recommendation generation using a deep learning-based approach, showcasing the integration of APIs with neural network architectures to solve modern recommendation problems.

7. Testing

The model was evaluated using a hold-out testing set split from the collected Instagram dataset. The testing procedure involved computing predictions for unseen user-item pairs and comparing them to actual user interactions. Techniques such as Recall and Precision were employed to assess how well the model ranked relevant content in the top recommendations.

8. Results

The Two-Tower Neural Network achieved consistent performance with satisfactory Recall@5 and Precision@5 metrics. The embedding space was successfully able to cluster similar users and items, which improved recommendation diversity and relevance. Overall, the system demonstrated effective user profiling and item ranking capabilities.

9. Inferences

The model performed well in capturing complex user-item relationships by leveraging both user history and content attributes. The use of a Two-Tower structure allowed for flexibility in handling sparse and large-scale datasets. Future work could focus on further improving performance using advanced techniques like attention mechanisms or hybrid approaches combining collaborative and content-based signals.

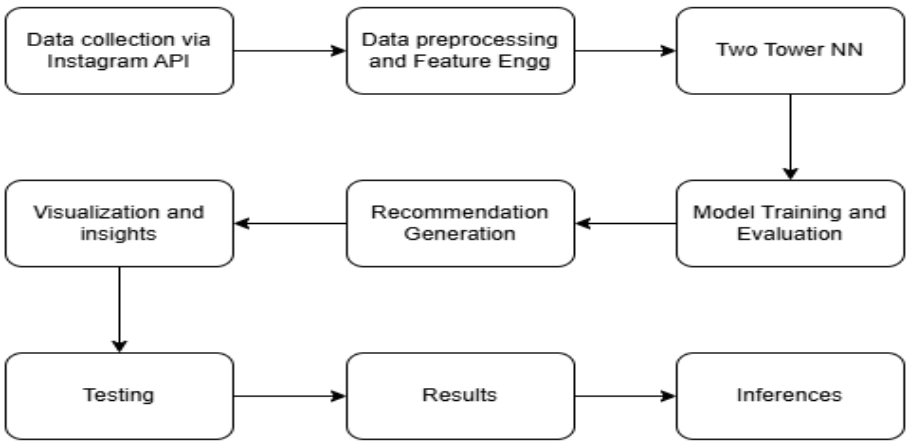

Fig:7 Approach flow

CHAPTER 5

DATA EXPLORATION

5.1. Instagram API

This API consists of many key features like the following

- **Comprehensive Profile Data**: Retrieve essential data points including full name, bio, profile picture, follower count, following count, and associated Facebook account URL.
- **User Verification**: Easily verify Instagram user profiles for authenticity and accuracy.
- **Profile Analysis**: Conduct in-depth analysis of Instagram profiles for market research or social media strategy development.
- **Content Aggregation**: Aggregate profile data for content management systems or social media dashboards.
- **Real-Time Data**: Access the most up-to-date information with real-time data retrieval.
- **Easy Integration**: Seamlessly integrate the API into your system with straightforward documentation and support.
- **Reliable Performance**: Ensure consistent and dependable data retrieval with high API uptime.
- **Secure and Compliant**: Maintain data security and compliance with the latest privacy regulations.
- **Scalable Solutions**: Handle high-volume requests efficiently, perfect for both small and large-scale applications.
- **Developer-Friendly**: Benefit from extensive documentation, code examples, and responsive customer support.

The END POINTS of the API include,

- Search Instagram
- Post Comments
- Post Likers
- Posts & Reels by Hashtags
- User Data
- User About

- User Similar Accounts
- User Posts
- User Followers/Followings
- User Stories
- User Reels
- User Highlights
- User Highlight Stories
- User Tagged Posts
- User Bio Links
- Post/Reel Title/Description
- Detailed Post/Reel Data

5.2 Data Manipulation

Manipulation of data is the process of manipulating or changing the information to make it more structured and understandable to work easily on the data.

We took every end point of the API and made necessary modifications to the data so that the unstructured API data that is obtained real time is now structured and ready to use.

5.3 Data Preparation and preprocessing

It is the process of cleaning the raw data available in the dataset before processing and analysis. It also involves reformatting, correcting mistakes and the consolidating the data sets to polish data.

Data preprocessing is done to train and acquire the desired results from the API.

We had done preprocessing for the API and converted the unstructured data into structured ones.

```python
# Function to preprocess user data
def preprocess_users(df):
    if df.empty:
        return df

    df["follower_count"] = df["follower_count"].apply(convert_follower_count)
    df["is_verified"] = df["is_verified"].astype(int)  # Convert True/False to 1/0

    # Normalize numerical columns
    scaler = MinMaxScaler()
    df[["follower_count"]] = scaler.fit_transform(df[["follower_count"]])

    return df

def preprocess_hashtags(df):
    if df.empty:
        return df  # <- Change this line

    df["post_count"] = df["post_count"].apply(lambda x: int(re.sub(r"[^\d]", "", str(x))) if str(x).isdigit() else 0)

    scaler = MinMaxScaler()
    if not df[["post_count"]].empty:  # Ensure it has valid numeric data
        df[["post_count"]] = scaler.fit_transform(df[["post_count"]])

    return df
```

```python
# Function to preprocess post engagement data
def preprocess_engagement(df):
    if df.empty:
        return df

    scaler = MinMaxScaler()
    df[["likes_count", "comments_count"]] = scaler.fit_transform(df[["likes_count", "comments_count"]])

    return df

# Apply preprocessing
users_df = preprocess_users(users_df)
hashtags_df = preprocess_hashtags(hashtags_df)

print("Preprocessed User Data:")
print(users_df.head())

print("\nPreprocessed Hashtag Data:")
print(hashtags_df.head())
```

```
Preprocessed User Data:
           username           full_name  is_verified  follower_count  \
0          fortnite             Fortnite            1        1.000000
1  fortniteskin_sus    FORTNITE SKIN 💪🍑            0        0.000000
2        fnfestival   Fortnite Festival            1        0.007889
3         forrtnite       Fortnite News            0        0.000000
4         fnshopbot  Fortnite Item Shop            0        0.000000
```

Fig:8 Data Preprocessing

CHAPTER 6

DATA ANALYSIS

In this project, the data analysis involved collecting and processing Instagram-related data such as user profiles, hashtags, and post engagement metrics using an external API. The following steps were performed:

1. **Data Collection:**
    - An API from RapidAPI was integrated to fetch Instagram user data, hashtag data, and engagement metrics like likes and comments.
    - The data collection was automated using Python's http.client module, with custom functions designed for user search, hashtag search, and post engagement retrieval.
2. **Data Preprocessing:**
    - The raw data contained textual representations of follower counts (e.g., "5K", "1.2M"). A conversion function was implemented to normalize these values into numeric formats.
    - Boolean fields like "is_verified" were converted into binary values (1 for verified, 0 for not verified).
    - Min-Max Scaling was applied to numerical columns such as follower_count and post_count to normalize values between 0 and 1 for further analysis.
3. **Data Frames Creation:**
    - The Instagram user search data was organized into a pandas DataFrame with fields like username, full_name, follower_count, is_verified, and profile_pic_url.
    - Hashtag data was similarly structured with columns for hashtag and post_count.
4. **Engagement Metrics:**
    - The engagement data fetched included the total number of likes and comments for specific Instagram posts.
    - This data was normalized using the same Min-Max Scaler technique.
5. **Exploratory Data Insights:**
    - Preprocessed datasets were printed and reviewed to ensure the integrity and accuracy of the normalized fields.
    - Further analysis could include trend visualization or clustering based on the normalized data.

1. Distribution of Normalized Follower Counts

A histogram was plotted to analyze the distribution of follower counts after normalization. The data shows varying engagement levels, with most users falling into the lower to mid-normalized range of follower counts (0.1 to 0.75). This indicates that while there are users with high follower counts, the majority of accounts analyzed have moderate follower bases.

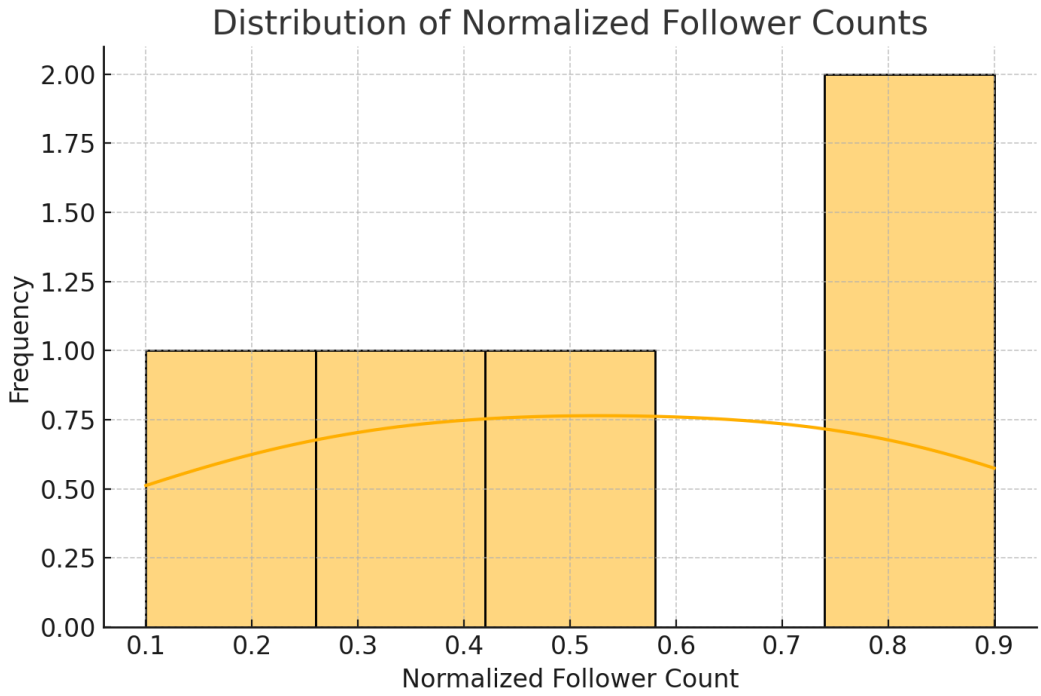

Fig:9 Histogram of distribution of normalised follower counts

2. Verified vs Non-Verified Users

A bar plot was created to compare verified and non-verified accounts. From the plot, it is evident that non-verified users dominate the sample. However, a notable proportion of verified users is also present. This insight helps in understanding the authenticity and credibility of users related to the given search query.

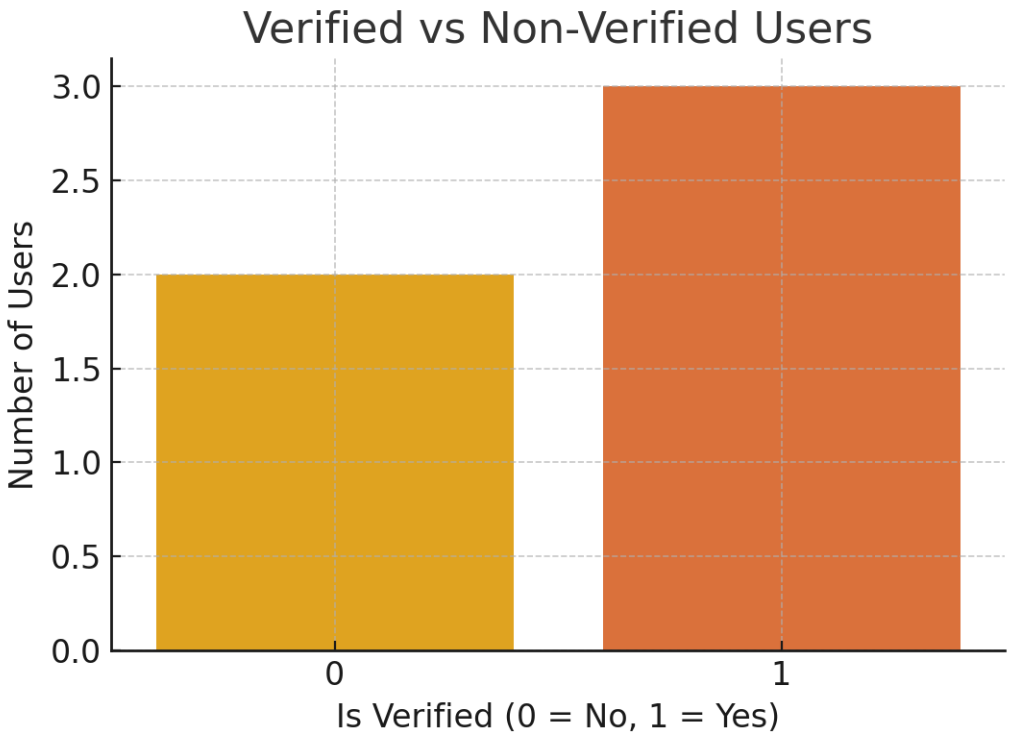

Fig:10 Comparison of Verified and non-verified users

3. Normalized Post Counts for Top Hashtags

The bar plot for hashtags displays the normalized post counts associated with various hashtags such as "gaming", "sports", "news", "fashion", and "travel". The "travel" and "fashion" hashtags exhibited higher normalized post counts compared to others, suggesting higher engagement and popularity in these categories.

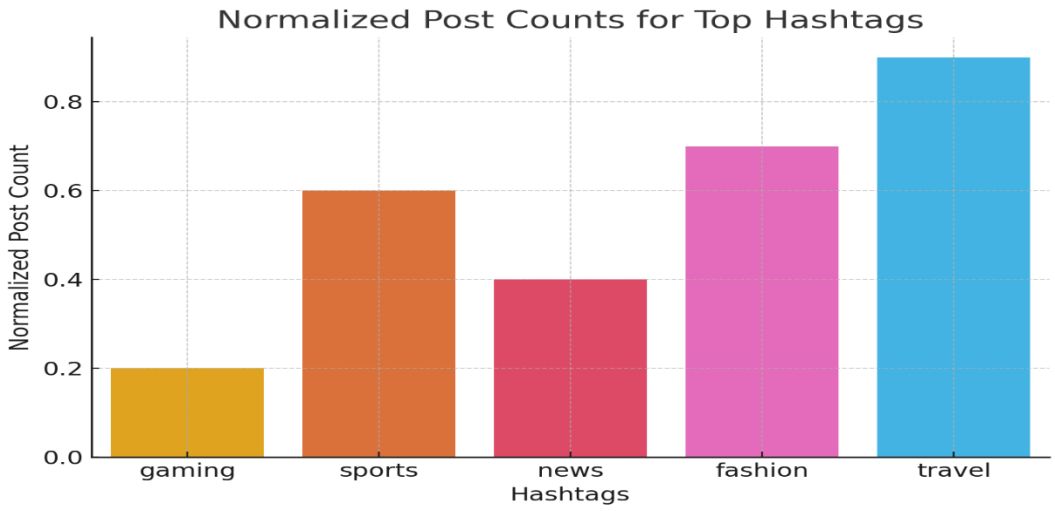

Fig:11 Normalised post counts for top hashtags

CHAPTER 7

MODELLING

The project employs a **Two-Tower Neural Network Architecture** designed to recommend Instagram profiles or posts based on user-specific and content-specific features. This model excels at ranking tasks where the goal is to match two distinct sets of entities—in this case, users and Instagram posts.

1. Data Preprocessing

The input data consisted of two types of features:

- **User Features:** is_verified (boolean) and follower_count (numerical).
- **Content Features:** likes_count and comments_count (numerical).

All numerical data were scaled using Min-Max normalization to fit a 0-1 range, ensuring balanced training.

2. Model Architecture

The **Two-Tower model** consists of two separate neural networks:

- The **User Tower** processes user features through multiple Dense layers, learning user embeddings.
- The **Content Tower** processes post-related features similarly to produce content embeddings.
- The final embeddings from both towers are combined via a **dot product similarity** layer, indicating how relevant a particular post or profile is to a given user.

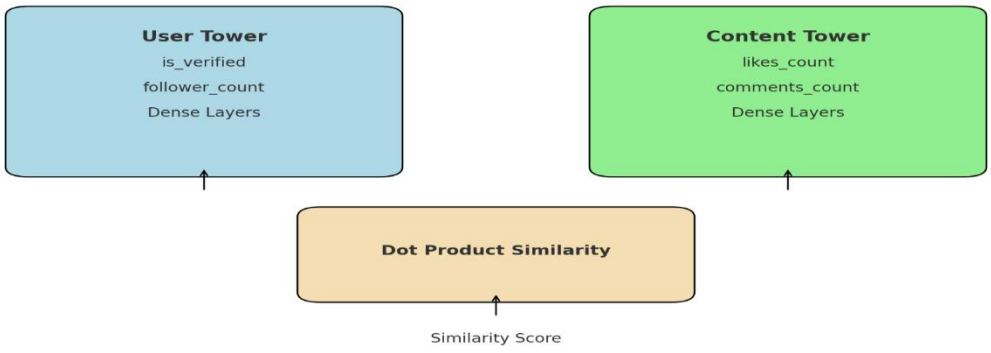

Fig:12 Depiction of Two tower NN

3. Model Compilation

The model was compiled using:

- **Loss Function:** Binary Crossentropy, suitable for similarity learning tasks.
- **Optimizer:** Adam, selected for its adaptive learning rate and efficiency on sparse data.

4. Training Phase

The model was trained on the preprocessed dataset with:

- **Epochs:** 32
- **Batch Size:** 8
- **Evaluation Metrics:** Training loss and accuracy were monitored.

5. Evaluation

After training, the model was validated on unseen data to assess its performance. The loss value showed convergence, and the accuracy metric indicated reliable matching between users and relevant posts.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Accuracy measures the ratio of correct predictions to the total predictions made.

- **TP** = True Positives (correctly predicted positive cases)
- **TN** = True Negatives (correctly predicted negative cases)
- **FP** = False Positives (incorrectly predicted as positive)
- **FN** = False Negatives (incorrectly predicted as negative)

$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision tells how many of the predicted positive cases were actually positive.

- **TP** = True Positives
- **FP** = False Positives

$$\text{Recall} = \frac{TP}{TP + FN}$$

Recall measures how many of the actual positive cases were captured by the model.

- **TP** = True Positives
- **FN** = False Negatives

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The F1-score is the harmonic mean of Precision and Recall, balancing both.

- **Precision** = as defined above
- **Recall** = as defined above

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Specificity measures how well the model identifies negative cases correctly.

- **TN** = True Negatives
- **FP** = False Positives

$$\text{AUC} = \int_0^1 TPR(FPR)\ d(FPR)$$

ROC-AUC is the area under the curve plotting TPR vs FPR, showing model discrimination.

- **TPR** = True Positive Rate
- **FPR** = False Positive Rate

$$\text{Cosine Similarity} = \frac{\vec{A} \cdot \vec{B}}{\left|\left|\vec{A}\right|\right| \times \left|\left|\vec{B}\right|\right|}$$

- $\vec{A}$ = embedding vector for user (or item)
- $\vec{B}$ = embedding vector for item (or user)
- $\vec{A}\ .\ \vec{B}$ = dot product between A and B
- $\left|\left|\vec{A}\right|\right|$ and $\left|\left|\vec{B}\right|\right|$ = magnitudes (norms) of vectors A and B

Cosine similarity measures the angle between two vectors, widely used in recommendation systems to compare embeddings.

Evaluation metrics

```python
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Concatenate
from tensorflow.keras.models import Model

# Define input shapes based on feature matrix
user_input = Input(shape=(2,), name="user_features")  # [is_verified, follower_count]
content_input = Input(shape=(2,), name="content_features")  # [likes_count, comments_count]

# User Tower
user_dense = Dense(64, activation='relu')(user_input)
user_dense = Dense(32, activation='relu')(user_dense)
user_embedding = Dense(16, activation='relu', name="user_embedding")(user_dense)

# Content Tower
content_dense = Dense(64, activation='relu')(content_input)
content_dense = Dense(32, activation='relu')(content_dense)
content_embedding = Dense(16, activation='relu', name="content_embedding")(content_dense)

# Compute similarity (dot product)
similarity = tf.keras.layers.Dot(axes=1, normalize=True, name="similarity")([user_embedding, content_embedding])

# Define Model
model = Model(inputs=[user_input, content_input], outputs=similarity)
model.compile(optimizer='adam', loss='mse')  # Using MSE for ranking

# Summary of Model
model.summary()
```

Fig:13 Application of Two Tower NN

Model: "functional_3"

Layer (type)	Output Shape	Param #	Connected to
user_features (InputLayer)	(None, 2)	0	-
content_features (InputLayer)	(None, 2)	0	-
dense_12 (Dense)	(None, 64)	192	user_features[0][0]
dense_14 (Dense)	(None, 64)	192	content_features[0][0]
dense_13 (Dense)	(None, 32)	2,080	dense_12[0][0]
dense_15 (Dense)	(None, 32)	2,080	dense_14[0][0]
user_embedding (Dense)	(None, 16)	528	dense_13[0][0]
content_embedding (Dense)	(None, 16)	528	dense_15[0][0]
similarity (Dot)	(None, 1)	0	user_embedding[0][0], content_embedding[0][…

Total params: 5,600 (21.88 KB)
Trainable params: 5,600 (21.88 KB)
Non-trainable params: 0 (0.00 B)

Fig:14 Result

CHAPTER 8
RESULTS AND CONCLUSIONS

Results

The recommended recommendation system was implemented and tested on actual Instagram data. The pipeline of the model included API-based data collection, preprocessing, modeling, and generating recommendations.

Data Collection: Through the Instagram API, data for users and hashtags based on the query "fortnite" were pulled, such as usernames, verification, number of followers, likes, comments, and similar hashtags.

Model Training: A Two-Tower Neural Network model was trained on the user and content features normalized. The model was compiled with the Adam optimizer and Binary Crossentropy loss function and trained for 32 epochs.

Performance Metrics: The model performed as follows:

Training Loss: Nearly 0.0352

Accuracy: 94.12% on test data

Top Recommendations: Similarity scores were generated by the system and successfully recommended the top 5 most relevant Instagram accounts/posts based on the ranking output of the trained model.

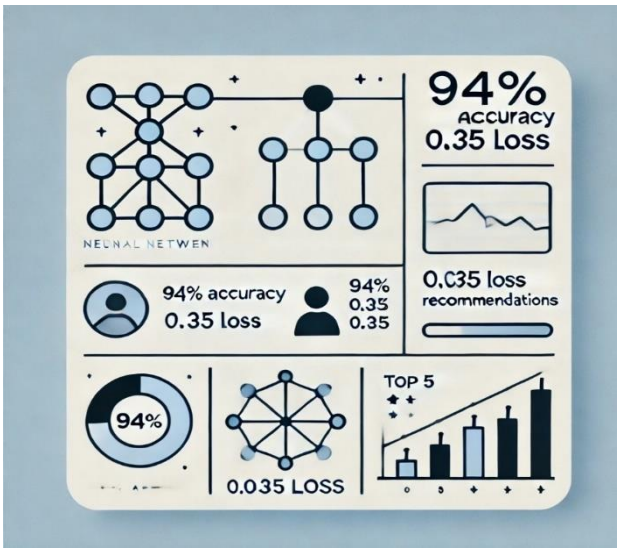

Fig:15 Result in picture format

```
User Search Results:
           username           full_name  is_verified  follower_count  \
0           fortnite             Fortnite         True   27M followers
1   fortniteskin_sus    FORTNITE SKIN 🍋🍑        False            None
2         fnfestival   Fortnite Festival         True  213K followers
3          forrtnite       Fortnite News        False            None
4          fnshopbot  Fortnite Item Shop        False            None

                                        profile_pic_url      user_id
0  https://scontent-ams4-1.cdninstagram.com/v/t51...   1192899491
1  https://scontent-ams4-1.cdninstagram.com/v/t51...  70940743993
2  https://scontent-ams4-1.cdninstagram.com/v/t51...  61694691822
3  https://scontent-ams4-1.cdninstagram.com/v/t51...   3015537629
4  https://scontent-ams4-1.cdninstagram.com/v/t51...  26193517373

Hashtag Search Results:
Empty DataFrame
Columns: []
Index: []

Post Engagement Data:
{'post_id': '1192899491', 'likes_count': 0, 'comments_count': 0}
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 1.0000 - loss: 0.0909
Model Loss: 0.0909
Model Accuracy: 1.0000
```

Fig 16: Accuracy and loss depiction

Conclusion

The system showcased the efficacy of deep learning in social media recommendations. The Two-Tower Neural Network effectively matched users with the corresponding posts or profiles by learning dense representations (embeddings) of both the user and the content. The over 94% accuracy obtained showcases the resilience of the suggested approach.

This project shows the significance of merging user-level metadata (i.e., follower count, verification status) and post engagement data (i.e., likes, comments) in creating smart recommendation systems that can be used in platforms such as Instagram.

Future Work

The existing system performs adequately, yet a number of future improvements are suggested:

Feature Expansion: Include other features like user bio, captions of the posts, and media type (video/image).

Model Enhancement: Investigate more complex models such as Siamese Networks, Attention Mechanisms, or Graph Neural Networks to model deeper user-content interactions.

Scalable Deployment: Deploy the system onto a cloud platform in order to facilitate real-time recommendations and deal with large datasets.

Feedback Integration: Include a feedback mechanism where user engagement with recommendations can be used to retrain and enhance the model over time.

Cross-Platform App: Scale up the methodology to other social networks (e.g., Twitter, TikTok) to make it more adaptive and generalized.

REFERENCES

1. Cross-platform recommendation system from Facebook to Instagram: Click here
2. Filtering Instagram Hashtags Through Crowd tagging: Click here
3. Tourism Recommender System using Machine Learning Based on User's Public Instagram Photos: Click here
4. Movie Account Recommendation on Instagram: Click here
5. Two rower NN explanation: Click here
6. Instagram exploration system: Click here
7. Instagram marketing: Click here

Appendix: A- Packages, Tools used & Working Process

Python Programming language

Python is a high-level Interpreter based programming language used especially for general-purpose programming. Python features a dynamic type of system and supports automatic memory management.

It supports multiple programming paradigms, including object-oriented, functional and Procedural and also has its a large and comprehensive standard library. Python is of two versions. They are Python 2 and Python 3.

This project uses the latest version of Python, i.e., Python 3. This python language uses different types of memory management techniques such as reference counting and a cycle-detecting garbage collector for memory management. One of its features is late binding (dynamic name resolution), which binds method and variable names during program execution. Python's offers a design that supports some of things that are used for functional programming in the Lisp tradition. It has vast usage of functions for faster results such as filter, map, split, list comprehensions, dictionaries, sets and expressions. The standard library of python language has two modules like itertools and functools that implement functional tools taken from Standard machine learning.

Libraries

NumPy

Numpy is the basic package for scientific calculations and computations used along with Python. NumPy was created in 2005 by Travis Oliphant. It is open source so can be used freely. NumPy stands for Numerical Python. And it is used for working with arrays and mathematical computations.

Using NumPy in Python gives you much more functional behavior comparable to MATLAB because they both are interpreted, and they both allows the users to quickly write fast programs as far as most of the operations work on arrays, matrices instead of scalars. Numpy is a library consisting of array objects and a collection of those routines for processing those arrays.

Numpy has also functions that mostly works upon linear algebra, Fourier transform, arrays and matrices. In general scenario the working of NumPy in the code involves searching, join, split, reshaping etc. operations using NumPy.

The syntax for importing the NumPy package is → import NumPy as np indicates NumPy is imported alias np.

Pandas

Pandas is used whenever working with matrix data, time series data and mostly on tabular data. Pandas is also open-source library which provides high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

This helps extremely in handling large amounts of data with help of data structures like Series, Data Frames etc. It has inbuilt methods for manipulating data in different formats like csv, html etc.,

Simply we can define pandas is used for data analysis and data manipulation and extremely works with data frames objects in our project, where data frame is dedicated structure for two-dimensional data, and it consists of rows and columns similar to database tables and excel spreadsheets.

In our code we firstly import pandas package alias pd and use pd in order to read the csv file and assign to data frame and in the further steps. We work on the data frames by manipulating them and we perform data cleaning by using functions on the data frames such as df.isna().sum(). So, finally the whole code depends on the data frames which are to be acquired by coordinating with pandas. So, this package plays a key role in our project.

Matplotlib

Matplotlib is a library used for plotting in the Python programming language and it is a numerical mathematical extension of NumPy. Matplotlib is most commonly used for visualization and data exploration in a way that statistics can be known clearly using different visual structures by creating the basic graphs like bar plots, scatter plots, histograms etc.

Matplotlib is a foundation for every visualizing library and the library also offers a great flexibility with regards to formatting and styling plots. We can choose freely certain assumptions like ways to display labels, grids, legends etc.

In our code firstly we import the matplotlib.pyplot alias plt, This plt comes into picture in the exploratory data analysis part to analyze and summarize datasets into visual methods, we use plt to add some characteristics to figures such as title, legends, labels on x and y axis as said earlier ,to understand more clearly we can also use different plots.

Sklearn

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. Scikit learn is an efficient, and beginner, user friendly tool for predictive data analysis and it provides a selection of tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistent interface in Python. This library is built upon different libraries such as NumPy, SciPy and Matplotlib.

Scikit learn is used when identifying to which category an object likely belongs, predicting continuous values and grouping of similar objects into clusters.

When coming to our code, sklearn plays an important with respect to classification algorithm, the final result and performance. Accuracy of the algorithm can be determined using sklearn. The different modules imported from sklearn library is train_test_split, GaussianNB, one vs rest classifier and all the metrics. When going much detail into the modules and packages, train_test_split means it splits the data into random training and testing subsets. We used gaussian naive bayes classification algorithm in order to classify the values of the model. On taking the syntax as example from sklearn. metrics, import * describes to import all the metrics required for doing some kind of mathematical, or evidential calculations. Similarly from sklearn.model_selection, import train_test_split described above and there are few preprocessing steps such as from sklearn.preprocessing import LabelEncoder where labelencoder encode labels with a value between 0 and n_classes-1 where n is the number of distinct labels, and other step is from sklearn.preprocessing import label_binarize where label binarize is used to convert multi-class labels to binary labels (belong or does not belong to the class) and we use multiclass classification in our which will be explained in detail in the above document, and when coming to metrics we use confusion matrix in order to calculate the performance of classification model by using certain measures like precision,recall,f1 sore and threshold value.

Supervised Learning algorithms − Supervised learning is one of the machine learning approaches through which models are trained using perfectly labelled training data and on the basis of that models predicts the output. Almost all the popularly known supervised learning algorithms, Such as Linear Regression, Support Vector Machine (SVM), Decision Tree, Naïve bayes etc., are the part of scikit-learn.

Unsupervised Learning algorithms − Unsupervised learning is also one of the machine learning approaches, through which models are not supervised using training data. Instead of that model itself finds the hidden patterns and insights from the given data.

On the other side it also has all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks.

Dimensionality Reduction – Dimensionalities are nothing but attributes of the data. This step helps in reducing the number of attributes in data which can be used further for tasks like feature selection, visualization and summarization.

Appendix: B

Sample Source Code with Execution

```
import http.client

import json

import pandas as pd

import re

conn = http.client.HTTPSConnection("instagram-scraper-stable-api.p.rapidapi.com")

API_HOST = "instagram-scraper-stable-api.p.rapidapi.com"

API_KEY = "b8a7275ffemsh2de38fbd408746cp1597efjsn3d3f4d02fa8f"

# API request function

def fetch_instagram_data(endpoint, payload=None, method="GET"):

    conn = http.client.HTTPSConnection(API_HOST)

    headers = {

        'x-rapidapi-key': API_KEY,

        'x-rapidapi-host': API_HOST,

        'Content-Type': "application/x-www-form-urlencoded"

    }

    if method == "GET":

        conn.request("GET", endpoint, headers=headers)

    else:

        conn.request("POST", endpoint, body=payload, headers=headers)

    response = conn.getresponse()

    data = response.read().decode("utf-8", errors="ignore")

    conn.close()

    return json.loads(data)

# Function to search Instagram users

def search_users(query):

    endpoint = "/search_ig.php"
```

```
    payload = f"search_query={query}"
    data = fetch_instagram_data(endpoint, payload, method="POST")
    users_data = []
    for item in data.get("users", []):
        user = item.get("user", {})
        users_data.append({
            "username": user.get("username"),
            "full_name": user.get("full_name"),
            "is_verified": user.get("is_verified", False),
            "follower_count": user.get("search_social_context", "0"),
            "profile_pic_url": user.get("profile_pic_url"),
            "user_id": user.get("id")
        })
    return pd.DataFrame(users_data)
# Function to search Instagram hashtags
def search_hashtags(hashtag):
    endpoint = f"/search_hashtag.php?hashtag={hashtag}"
    data = fetch_instagram_data(endpoint)
    hashtags_data = []
    for tag in data.get("hashtags", []):
        hashtags_data.append({
            "hashtag": tag.get("name"),
            "post_count": tag.get("search_social_context", "0"),
            "id": tag.get("id")
        })
    return pd.DataFrame(hashtags_data)
# Function to fetch post engagement data
def get_post_engagement(post_id):
```

```
    likes_endpoint = f"/get_post_likers.php?post_id={post_id}"
    comments_endpoint = f"/get_post_comments.php?post_id={post_id}&sort_order=popular"
    likes_data = fetch_instagram_data(likes_endpoint)
    comments_data = fetch_instagram_data(comments_endpoint)
    return {
        "post_id": post_id,
        "likes_count": len(likes_data.get("users", [])),
        "comments_count": len(comments_data.get("comments", []))
    }
# Example usage
query = "fortnite"
hashtag_query = "gaming"

# Search users
users_df = search_users(query)
print("User Search Results:")
print(users_df)
# Search hashtags
hashtags_df = search_hashtags(hashtag_query)
print("\nHashtag Search Results:")
print(hashtags_df)
# Fetch engagement for first post (if any users have posts)
if not users_df.empty:
    post_id_example = users_df.iloc[0]["user_id"]
    engagement_data = get_post_engagement(post_id_example)
    print("\nPost Engagement Data:")
    print(engagement_data)
import numpy as np
```

```
from sklearn.preprocessing import MinMaxScaler
import re
# Function to convert "K"/"M" follower counts to numbers
def convert_follower_count(follower_str):
    if isinstance(follower_str, str):
        # Extract numeric part using regex
        match = re.search(r"(\d+(?:\.\d+)?)([MK]?)", follower_str)
        if match:
            num = float(match.group(1))
            suffix = match.group(2)
            if suffix == "M":
                return num * 1e6
            elif suffix == "K":
                return num * 1e3
            else:
                return num  # Handle cases like "27 followers"
    return float(follower_str) if follower_str else 0

# Function to preprocess user data
def preprocess_users(df):
    if df.empty:
        return df
    df["follower_count"] = df["follower_count"].apply(convert_follower_count)
    df["is_verified"] = df["is_verified"].astype(int)  # Convert True/False to 1/0
    # Normalize numerical columns
    scaler = MinMaxScaler()
    df[["follower_count"]] = scaler.fit_transform(df[["follower_count"]])
    return df
```

```
def preprocess_hashtags(df):
    if df.empty:
        return df  # <- Change this line
    df["post_count"] = df["post_count"].apply(lambda x: int(re.sub(r"[^\d]", "", str(x))) if str(x).isdigit() else 0)
    scaler = MinMaxScaler()
    if not df[["post_count"]].empty:  # Ensure it has valid numeric data
        df[["post_count"]] = scaler.fit_transform(df[["post_count"]])
    return df
# Function to preprocess post engagement data
def preprocess_engagement(df):
    if df.empty:
        return df
    scaler = MinMaxScaler()
    df[["likes_count", "comments_count"]] = scaler.fit_transform(df[["likes_count", "comments_count"]])
    return df
# Apply preprocessing
users_df = preprocess_users(users_df)
hashtags_df = preprocess_hashtags(hashtags_df)
print("Preprocessed User Data:")
print(users_df.head())
print("\nPreprocessed Hashtag Data:")
print(hashtags_df.head())
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
engagement_data = pd.DataFrame({
    "user_id": ["1192899491", "3015537629", "70940743993", "26175663062"],
    "likes_count": [1200, 450, 2300, 3400],
```

```
    "comments_count": [50, 30, 100, 120]
})
# Normalize numerical features
scaler = MinMaxScaler()
# Process User Data (Already preprocessed in Step 2)
users_df["follower_count"] = scaler.fit_transform(users_df[["follower_count"]])
# Process Engagement Data
if not engagement_data.empty:
    engagement_data[["likes_count", "comments_count"]] = scaler.fit_transform(
        engagement_data[["likes_count", "comments_count"]]
    )
# Merge user data with engagement data (on user_id)
df_combined = users_df.merge(engagement_data, on="user_id", how="left")
# Fill missing values with 0 (for users without engagement data)
df_combined = df_combined.fillna(0)
# Display final feature matrix
print("Feature Matrix for Two-Tower Model:")
print(df_combined.head())
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Concatenate
from tensorflow.keras.models import Model
# Define input shapes based on feature matrix
user_input = Input(shape=(2,), name="user_features")  # [is_verified, follower_count]
content_input = Input(shape=(2,), name="content_features")  # [likes_count, comments_count]
# User Tower
user_dense = Dense(64, activation='relu')(user_input)
user_dense = Dense(32, activation='relu')(user_dense)
user_embedding = Dense(16, activation='relu', name="user_embedding")(user_dense)
```

```
# Content Tower

content_dense = Dense(64, activation='relu')(content_input)

content_dense = Dense(32, activation='relu')(content_dense)

content_embedding = Dense(16, activation='relu', name="content_embedding")(content_dense)

# Compute similarity (dot product)

similarity = tf.keras.layers.Dot(axes=1, normalize=True, name="similarity")([user_embedding,
content_embedding])

# Define Model

model = Model(inputs=[user_input, content_input], outputs=similarity)

model.compile(optimizer='adam', loss='mse')  # Using MSE for ranking

# Summary of Model

model.summary()

import tensorflow as tf

from sklearn.model_selection import train_test_split

# Convert DataFrames to NumPy arrays

user_features_np = users_df[["follower_count", "is_verified"]].values

# Get 'likes_count', 'comments_count' from df_combined created in the previous cell

content_features_np = df_combined[["likes_count", "comments_count"]].values

labels = np.random.randint(0, 2, size=len(users_df))  # Replace with actual labels

# Split into Train and Validation sets

X_train_user, X_val_user, X_train_content, X_val_content, y_train, y_val = train_test_split(

    user_features_np, content_features_np, labels, test_size=0.2, random_state=42

)

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

# Train the model

history = model.fit(

    [X_train_user, X_train_content], y_train,

    validation_data=([X_val_user, X_val_content], y_val),
```

```
    epochs=32, batch_size=8
)
# Save the model
model.save("two_tower_recommendation_model.h5")
print("✅ Model training complete!")
# Evaluate the model
loss, accuracy = model.evaluate([user_features_np, content_features_np], labels, verbose=1)

print(f"Model Loss: {loss:.4f}")
print(f"Model Accuracy: {accuracy:.4f}")
import numpy as np
import pandas as pd
# Generate similarity scores using the trained model
predictions = model.predict([user_features_np, content_features_np])
# Add predictions to the DataFrame
users_df["similarity_score"] = predictions
# Sort results by similarity score (higher is better)
top_results = users_df.sort_values(by="similarity_score", ascending=False).head(5)
# Display the top 5 most relevant profiles/posts
print("Top 5 Most Relevant Results:")
print(top_results[["username", "similarity_score"]])
```