HARPY AEROSPACE INTERNSHIP:

AIOT PROJECT

3 RECOMMENDATION SYSTEMS

OUTPUT AND CODE SCREENSHOTS

SUBMITTED BY

NETHRA V S

REG NO. 2022506115

B Tech Information Technology

Madras Institute of Technology

GNN RECOMMENDATION MODEL

CODE:

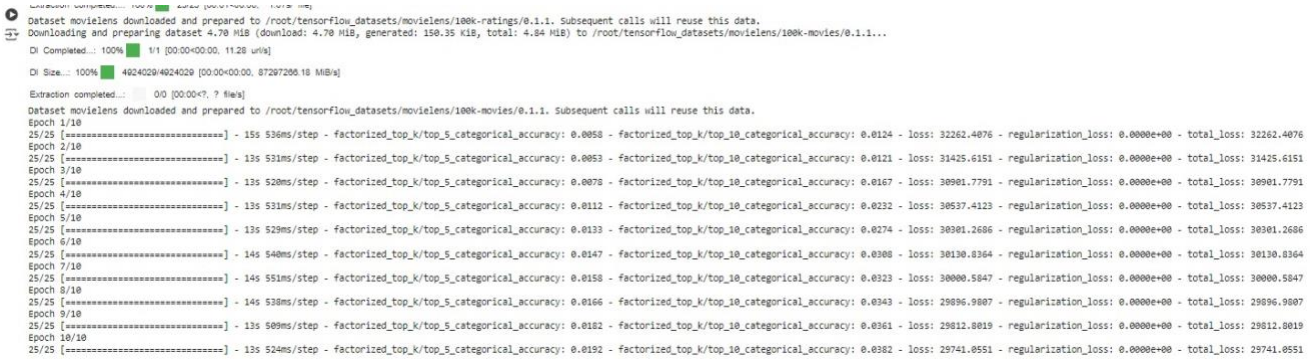
```
# Install the necessary libraries
!pip install -q tensorflow-recommenders matplotlib

import tensorflow as tf
import tensorflow_datasets as tfds
import tensorflow_recommenders as tfrs
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import layers

# Load the MovieLens dataset
ratings = tfds.load("movielens/100k-ratings", split="train")
movies = tfds.load("movielens/100k-movies", split="train")

# Prepare the data
ratings = ratings.map(lambda x: {
    "movie_title": x["movie_title"],
    "user_id": x["user_id"],
    "timestamp": x["timestamp"]
})

movies = movies.map(lambda x: x["movie_title"])

# Define the user and movie model with additional features.
user_ids_vocabulary = tf.keras.layers.StringLookup()
movie_titles_vocabulary = tf.keras.layers.StringLookup()

user_ids_vocabulary.adapt(ratings.map(lambda x: x["user_id"]))
movie_titles_vocabulary.adapt(movies)

# Convert the movie titles to a TensorFlow Dataset
movies = tf.data.Dataset.from_tensor_slices(list(movies))

# Define the GNN layer
class GNNLayer(layers.Layer):
    def __init__(self, units):
        super(GNNLayer, self).__init__()
        self.units = units
        self.dense = layers.Dense(units)
```

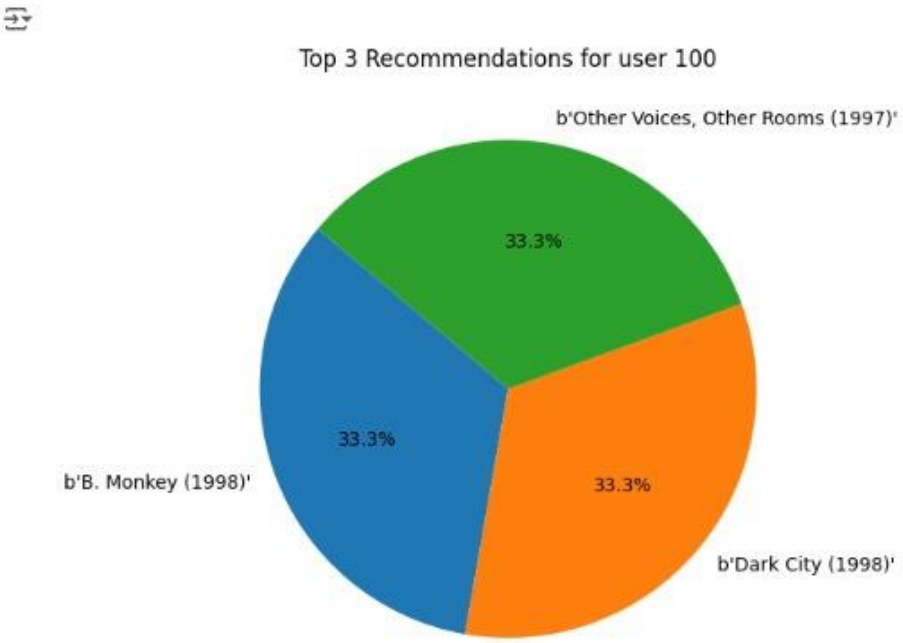
```
    def call(self, inputs, edge_index):
        x = inputs
        row, col = edge_index[:, 0], edge_index[:, 1]
        out = tf.math.unsorted_segment_sum(x[col], row,
num_segments=tf.shape(x)[0])
        return self.dense(out)

class GNNModel(tfrs.Model):
    def __init__(self, user_model, movie_model, task):
        super().__init__()
        self.user_model = user_model
        self.movie_model = movie_model
        self.task = task

    def call(self, features):
        user_embeddings = self.user_model(features["user_id"])
        movie_embeddings = self.movie_model(features["movie_title"])
        edge_index = tf.stack([features["user_id"],
features["movie_title"]], axis=1)
        gnn_layer = GNNLayer(64)
        user_embeddings = gnn_layer(user_embeddings, edge_index)
        movie_embeddings = gnn_layer(movie_embeddings, edge_index)
        return self.task(user_embeddings, movie_embeddings)

    def compute_loss(self, features, training=False):
        user_embeddings = self.user_model(features["user_id"])
        movie_embeddings = self.movie_model(features["movie_title"])
        return self.task(user_embeddings, movie_embeddings)

# Define user and movie models
user_model = tf.keras.Sequential([
    user_ids_vocabulary,
    tf.keras.layers.Embedding(user_ids_vocabulary.vocabulary_size(), 64),
    tf.keras.layers.Dense(32, activation="relu")
])

movie_model = tf.keras.Sequential([
    movie_titles_vocabulary,
    tf.keras.layers.Embedding(movie_titles_vocabulary.vocabulary_size(),
64),
    tf.keras.layers.Dense(32, activation="relu")
])

# Define the task
```

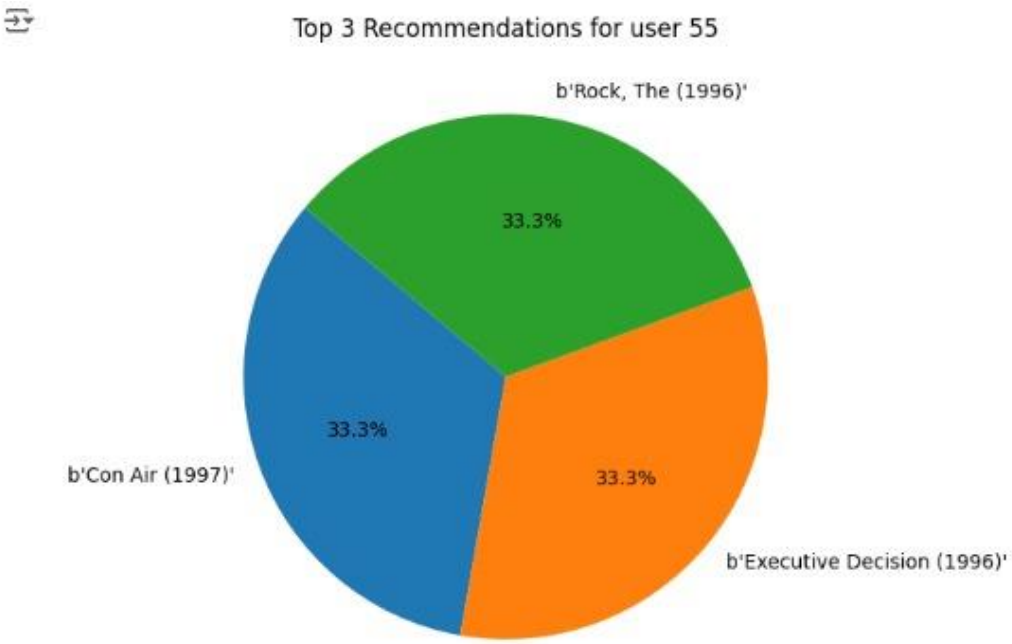
```
task = tfrs.tasks.Retrieval(metrics=tfrs.metrics.FactorizedTopK(
    candidates=movies.batch(128).map(movie_model),
    ks=[5, 10]
))

# Create and compile the model
model = GNNModel(user_model, movie_model, task)
model.compile(optimizer=tf.keras.optimizers.Adam(0.01))

# Train the model and capture the training history
history = model.fit(ratings.batch(4096), epochs=10, verbose=1)

# Set up brute-force search for retrieval
index = tfrs.layers.factorized_top_k.BruteForce(model.user_model)
index.index_from_dataset(
    movies.batch(100).map(lambda title: (title, model.movie_model(title)))
)

# Function to get recommendations for a specific user and plot a pie chart
def plot_recommendation_pie_chart(user_id):
    _, titles = index(np.array([user_id]))
    top_3_titles = titles[0, :3].numpy()
    unique_titles, counts = np.unique(top_3_titles, return_counts=True)

    plt.figure(figsize=(6, 6))
    plt.pie(counts, labels=unique_titles, autopct='%1.1f%%',
startangle=140)
    plt.title(f'Top 3 Recommendations for user {user_id}')
    plt.show()

# Get recommendations for a specific user and plot the pie chart
plot_recommendation_pie_chart("55")

# Get recommendations for another user and plot the pie chart
plot_recommendation_pie_chart("100")

# Plot the training loss and top-k accuracy
plt.figure(figsize=(12, 6))

# Plot training loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Loss')
plt.title('Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

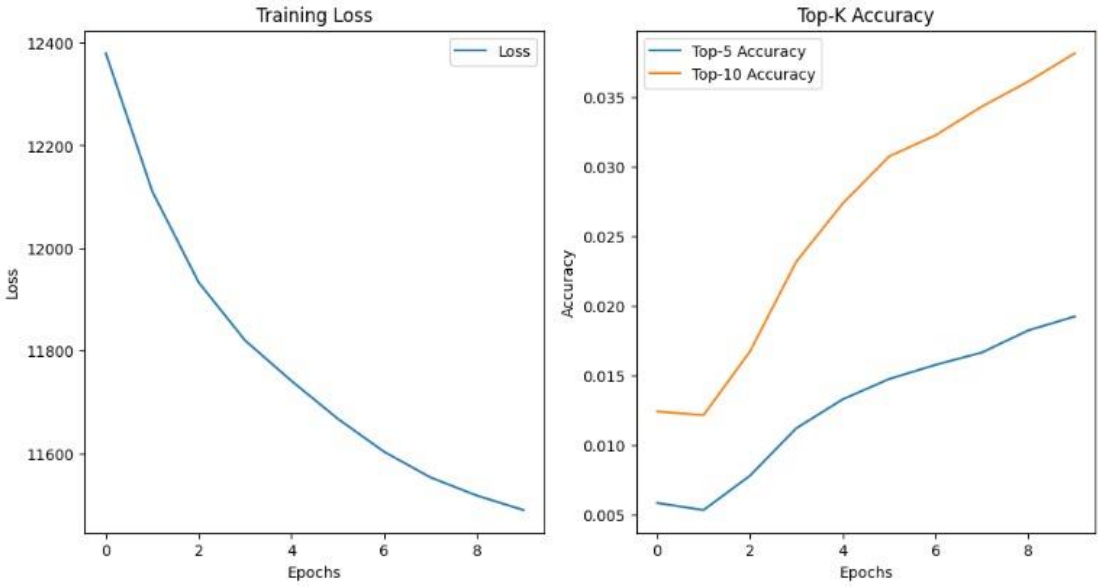
```
plt.legend()

# Plot top-5 and top-10 accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['factorized_top_k/top_5_categorical_accuracy'],
label='Top-5 Accuracy')
plt.plot(history.history['factorized_top_k/top_10_categorical_accuracy'],
label='Top-10 Accuracy')
plt.title('Top-K Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
```

OUTPUT:




Top 3 Recommendations for user 100
b'Other Voices, Other Rooms (1997)'
33.3%
33.3%
b'B. Monkey (1998)'
33.3%
b'Dark City (1998)'


Top 3 Recommendations for user 55
b'Rock, The (1996)'
33.3%
33.3%
b'Con Air (1997)'
33.3%
b'Executive Decision (1996)'


Training Loss
Loss
12400
12200
12000
11800
11600
Loss
0
2
4
6
8
Epochs
Top-K Accuracy
Top-5 Accuracy
Top-10 Accuracy
0.035
0.030
0.025
0.020
0.015
0.010
0.005
Accuracy
0
2
4
6
8
Epochs

2. KNN model (K-Neighbourhood model)

CODE:

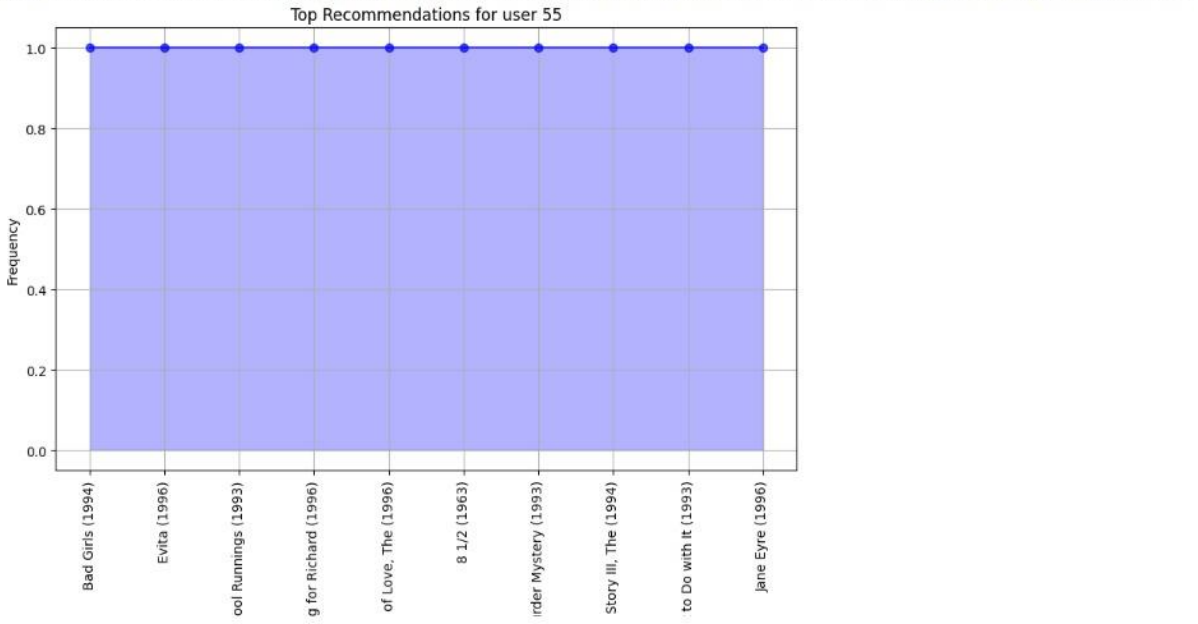
```
# Install necessary libraries
!pip install -q tensorflow-recommenders matplotlib

import tensorflow as tf
import tensorflow_datasets as tfds
import tensorflow_recommenders as tfrs
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import layers

# Load the MovieLens dataset
ratings = tfds.load("movielens/100k-ratings", split="train")
movies = tfds.load("movielens/100k-movies", split="train")

# Prepare the data
ratings = ratings.map(lambda x: {
    "movie_title": x["movie_title"],
    "user_id": x["user_id"],
    "timestamp": x["timestamp"]
})

movies = movies.map(lambda x: x["movie_title"])

# Define the KNN layer
class KNNLayer(layers.Layer):
    def __init__(self, k):
        super(KNNLayer, self).__init__()
        self.k = k

    def call(self, inputs, queries):
        queries = tf.expand_dims(queries, axis=1)
        distances = tf.reduce_sum(tf.square(inputs - queries), axis=-1)
        _, indices = tf.math.top_k(-distances, k=self.k)
        return tf.reduce_mean(tf.gather(inputs, indices), axis=1)

# Define the KNN model for movie recommendations
class KNNModel(tfrs.Model):
    def __init__(self, user_model, movie_model, task, k):
        super().__init__()
```

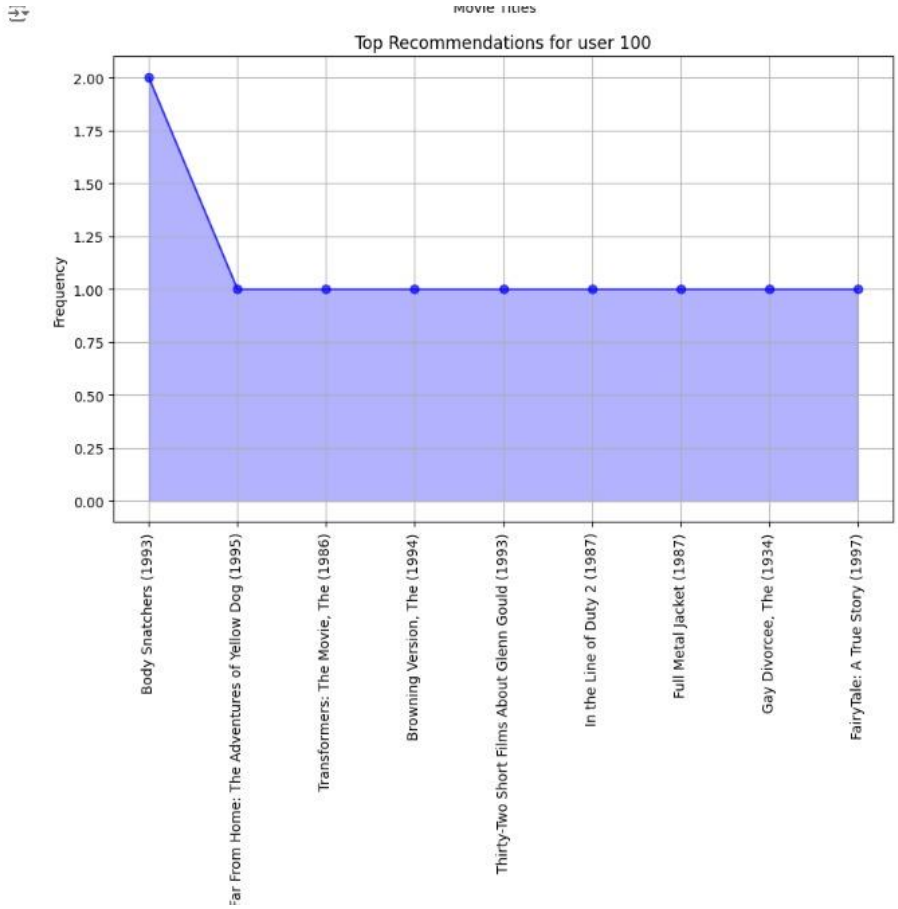
```
        self.user_model = user_model
        self.movie_model = movie_model
        self.task = task
        self.k = k
        self.knn_layer = KNNLayer(k)

    def call(self, features):
        user_embeddings = self.user_model(features["user_id"])
        movie_embeddings = self.movie_model(features["movie_title"])
        return self.task(user_embeddings, self.knn_layer(movie_embeddings,
user_embeddings))

    def compute_loss(self, features, training=False):
        user_embeddings = self.user_model(features["user_id"])
        movie_embeddings = self.movie_model(features["movie_title"])
        return self.task(user_embeddings, self.knn_layer(movie_embeddings,
user_embeddings))

# Define user and movie models with embeddings
user_ids_vocabulary = tf.keras.layers.StringLookup()
movie_titles_vocabulary = tf.keras.layers.StringLookup()

user_ids_vocabulary.adapt(ratings.map(lambda x: x["user_id"]))
movie_titles_vocabulary.adapt(movies)

user_model = tf.keras.Sequential([
    user_ids_vocabulary,
    tf.keras.layers.Embedding(user_ids_vocabulary.vocabulary_size(), 64),
    tf.keras.layers.Dense(32, activation="relu")
])

movie_model = tf.keras.Sequential([
    movie_titles_vocabulary,
    tf.keras.layers.Embedding(movie_titles_vocabulary.vocabulary_size(),
64),
    tf.keras.layers.Dense(32, activation="relu")
])

# Define the retrieval task
task = tfrs.tasks.Retrieval(
    metrics=tfrs.metrics.FactorizedTopK(
        candidates=movies.batch(128).map(movie_model),
        ks=[5, 10]
    )
)
```

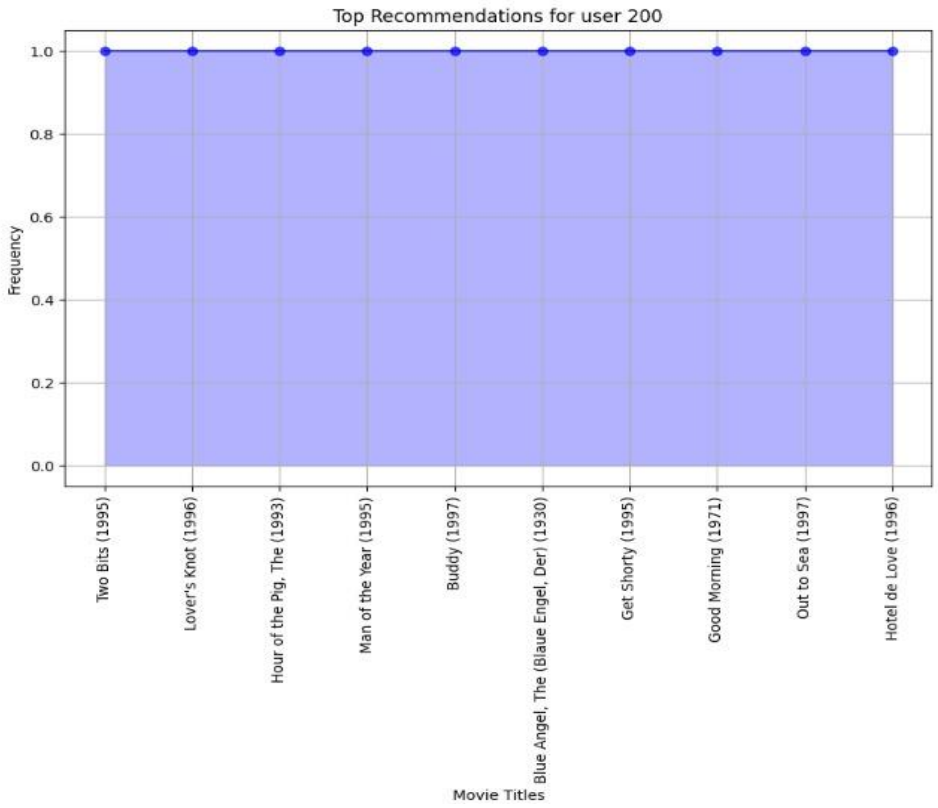
```
# Create and compile the KNN model
k = 10  # Number of nearest neighbors
knn_model = KNNModel(user_model, movie_model, task, k)
knn_model.compile(optimizer=tf.keras.optimizers.Adam(0.01))

# Train the KNN model and capture the training history
history = knn_model.fit(ratings.batch(4096), epochs=10, verbose=1)

# Set up brute-force search for retrieval
index = tfrs.layers.factorized_top_k.BruteForce(knn_model.user_model)
index.index_from_dataset(
    movies.batch(100).map(lambda title: (title,
knn_model.movie_model(title)))
)

# Function to get recommendations for a specific user and plot a frequency
polygon
def plot_recommendation_frequency_polygon(user_id):
    _, titles = index(np.array([user_id]))
    top_titles = titles[0].numpy()

    title_counts = {}
    for title in top_titles:
        title_str = title.decode('utf-8')  # Convert bytes to string
        if title_str in title_counts:
            title_counts[title_str] += 1
        else:
            title_counts[title_str] = 1

    sorted_titles = sorted(title_counts.items(), key=lambda x: x[1],
reverse=True)
    top_titles, counts = zip(*sorted_titles)

    plt.figure(figsize=(10, 6))
    plt.plot(top_titles, counts, marker='o', linestyle='-', color='b',
alpha=0.7)
    plt.fill_between(top_titles, counts, color='b', alpha=0.3)
    plt.title(f'Top Recommendations for user {user_id}')
    plt.xlabel('Movie Titles')
    plt.ylabel('Frequency')
    plt.xticks(rotation=90)
    plt.grid(True)
    plt.show()
```

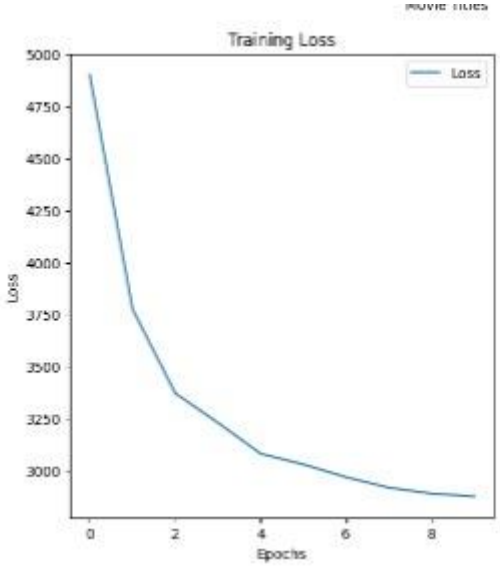
```
# Get recommendations for specific users and plot the frequency polygon
users_to_plot = ["55", "100", "200"]  # Example list of user IDs
for user_id in users_to_plot:
    plot_recommendation_frequency_polygon(user_id)

# Plot the training loss and top-k accuracy
plt.figure(figsize=(12, 6))

# Plot training loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Loss')
plt.title('Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# No need to plot top-k accuracy for KNN model

plt.show()
```

OUTPUT:


Top Recommendations for user 55
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
Bad Girls (1994)
Evita (1996)
ool Runnings (1993)
g for Richard (1996)
of Love, The (1996)
8 1/2 (1963)
rder Mystery (1993)
Story III, The (1994)
to Do with It (1993)
Jane Eyre (1996)


Movie Titles
Top Recommendations for user 100
Frequency
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Body Snatchers (1993)
ar From Home: The Adventures of Yellow Dog (1995)
Transformers: The Movie, The (1986)
Browning Version, The (1994)
Thirty-Two Short Films About Glenn Gould (1993)
In the Line of Duty 2 (1987)
Full Metal Jacket (1987)
Gay Divorcee, The (1934)
FairyTale: A True Story (1997)


Top Recommendations for user 200
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
Two Bits (1995)
Lover's Knot (1996)
Hour of the Pig, The (1993)
Man of the Year (1995)
Buddy (1997)
Blue Angel, The (Blaue Engel, Der) (1930)
Get Shorty (1995)
Good Morning (1971)
Out to Sea (1997)
Hotel de Love (1996)
Movie Titles


Training Loss
Loss
5000
4750
4500
4250
4000
3750
3500
3250
3000
0
2
4
6
8
Epochs

3. Session Based Recomendation Model

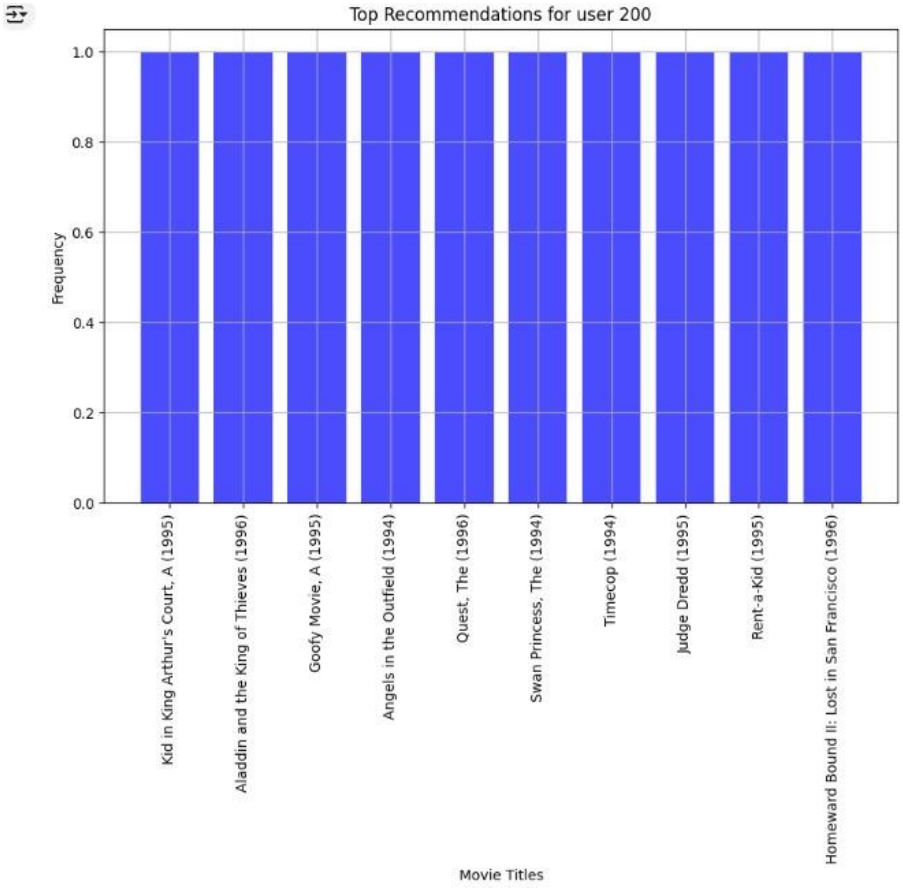
CODE:

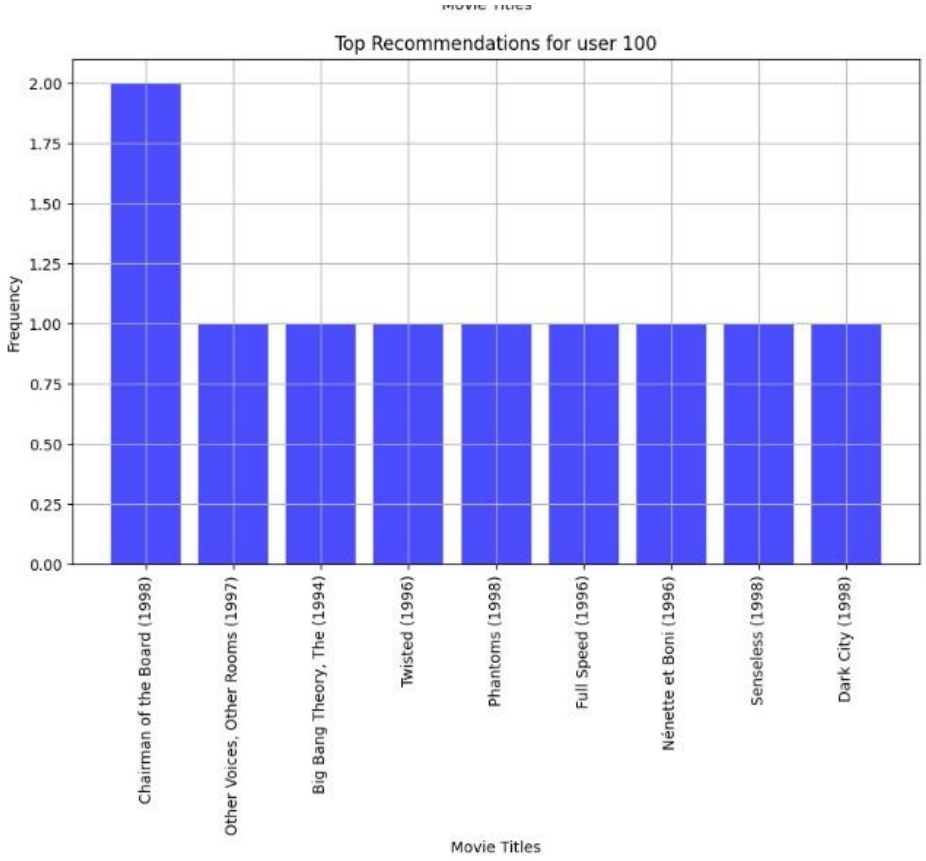
```
# Install necessary libraries
!pip install -q tensorflow-recommenders matplotlib

import tensorflow as tf
import tensorflow_datasets as tfds
import tensorflow_recommenders as tfrs
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import layers

# Load the MovieLens dataset
ratings = tfds.load("movielens/100k-ratings", split="train")
movies = tfds.load("movielens/100k-movies", split="train")

# Prepare the data
ratings = ratings.map(lambda x: {
    "movie_title": x["movie_title"],
    "user_id": x["user_id"],
    "timestamp": x["timestamp"]
})

movies = movies.map(lambda x: x["movie_title"])

# Define user and movie models with embeddings
user_ids_vocabulary = tf.keras.layers.StringLookup()
movie_titles_vocabulary = tf.keras.layers.StringLookup()

user_ids_vocabulary.adapt(ratings.map(lambda x: x["user_id"]))
movie_titles_vocabulary.adapt(movies)

user_model = tf.keras.Sequential([
    user_ids_vocabulary,
    tf.keras.layers.Embedding(user_ids_vocabulary.vocabulary_size(), 64),
    tf.keras.layers.Dense(32, activation="relu")
])

movie_model = tf.keras.Sequential([
    movie_titles_vocabulary,
    tf.keras.layers.Embedding(movie_titles_vocabulary.vocabulary_size(),
64),
```

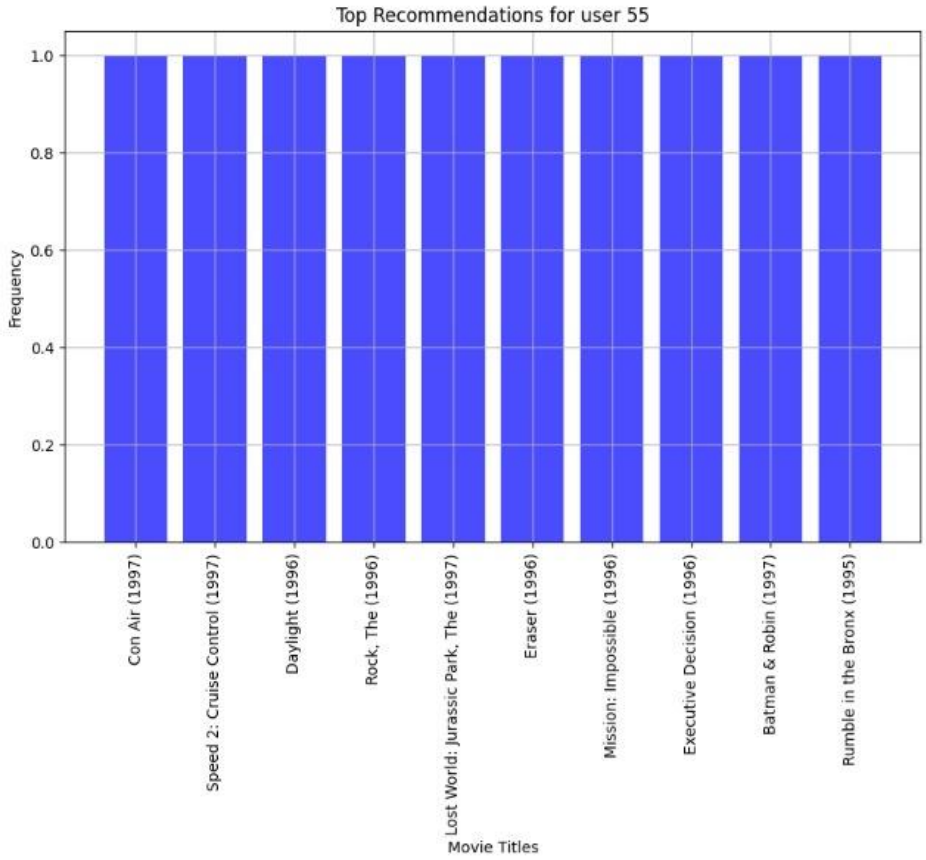
```
    tf.keras.layers.Dense(32, activation="relu")
])

# Define the retrieval task
task = tfrs.tasks.Retrieval(
    metrics=tfrs.metrics.FactorizedTopK(
        candidates=movies.batch(128).map(movie_model),
        ks=[5, 10]
    )
)

# Create and compile the session-based recommendation model
class SessionBasedModel(tfrs.Model):
    def __init__(self, user_model, movie_model, task):
        super().__init__()
        self.user_model = user_model
        self.movie_model = movie_model
        self.task = task

    def compute_loss(self, features, training=False):
        user_embeddings = self.user_model(features["user_id"])
        movie_embeddings = self.movie_model(features["movie_title"])
        return self.task(user_embeddings, movie_embeddings)

session_model = SessionBasedModel(user_model, movie_model, task)
session_model.compile(optimizer=tf.keras.optimizers.Adam(0.01))

# Train the session-based recommendation model and capture the training
history
history = session_model.fit(ratings.batch(4096), epochs=10, verbose=1)

# Set up brute-force search for retrieval
index = tfrs.layers.factorized_top_k.BruteForce(session_model.user_model)
index.index_from_dataset(
    movies.batch(100).map(lambda title: (title,
session_model.movie_model(title)))
)

# Function to get recommendations for a specific user and plot a column
graph
def plot_recommendation_column_graph(user_id):
    _, titles = index(np.array([user_id]))
    top_titles = titles[0].numpy()

    title_counts = {}
```

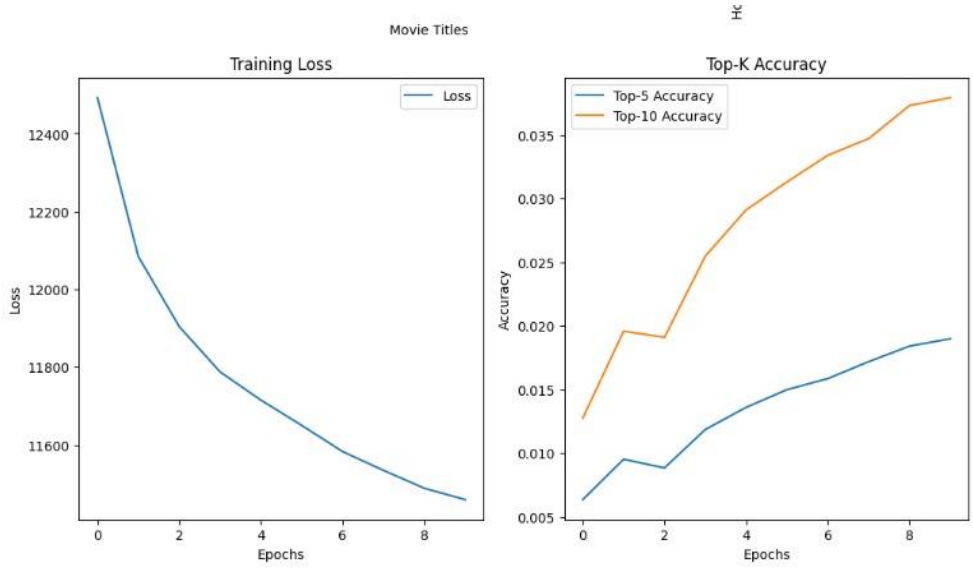
```
    for title in top_titles:
        title_str = title.decode('utf-8')  # Convert bytes to string
        if title_str in title_counts:
            title_counts[title_str] += 1
        else:
            title_counts[title_str] = 1

    sorted_titles = sorted(title_counts.items(), key=lambda x: x[1],
reverse=True)
    top_titles, counts = zip(*sorted_titles)

    plt.figure(figsize=(10, 6))
    plt.bar(top_titles, counts, color='blue', alpha=0.7)
    plt.title(f'Top Recommendations for user {user_id}')
    plt.xlabel('Movie Titles')
    plt.ylabel('Frequency')
    plt.xticks(rotation=90)
    plt.grid(True)
    plt.show()
# Get recommendations for specific users and plot the column graph
users_to_plot = ["55", "100", "200"]  # Example list of user IDs
for user_id in users_to_plot:
    plot_recommendation_column_graph(user_id)

# Plot the training loss and top-k accuracy
plt.figure(figsize=(12, 6))

# Plot training loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Loss')
plt.title('Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Plot top-5 and top-10 accuracy
plt.subplot(1, 2, 2)
plt.plot(history.history['factorized_top_k/top_5_categorical_accuracy'],
label='Top-5 Accuracy')
plt.plot(history.history['factorized_top_k/top_10_categorical_accuracy'],
label='Top-10 Accuracy')
plt.title('Top-K Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend() plt.show()
```

OUTPUT


Top Recommendations for user 200
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
Kid in King Arthur's Court, A (1995)
Aladdin and the King of Thieves (1996)
Goofy Movie, A (1995)
Angels in the Outfield (1994)
Quest, The (1996)
Swan Princess, The (1994)
Timecop (1994)
Judge Dredd (1995)
Rent-a-Kid (1995)
Homeward Bound II: Lost in San Francisco (1996)
Movie Titles


Top Recommendations for user 100
Frequency
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Chairman of the Board (1998)
Other Voices, Other Rooms (1997)
Big Bang Theory, The (1994)
Twisted (1996)
Phantoms (1998)
Full Speed (1996)
Nénette et Boni (1996)
Senseless (1998)
Dark City (1998)
Movie Titles


Top Recommendations for user 55
Frequency
1.0
0.8
0.6
0.4
0.2
0.0
Con Air (1997)
Speed 2: Cruise Control (1997)
Daylight (1996)
Rock, The (1996)
Lost World: Jurassic Park, The (1997)
Eraser (1996)
Mission: Impossible (1996)
Executive Decision (1996)
Batman & Robin (1997)
Rumble in the Bronx (1995)
Movie Titles


Movie Titles
Training Loss
Loss
12400
12200
12000
11800
11600
Epochs
Top-K Accuracy
Top-5 Accuracy
Top-10 Accuracy
Accuracy
0.035
0.030
0.025
0.020
0.015
0.010
0.005
Epochs