HARPY AEROSPACE SUMMER INTERSHIP PROGRAMME

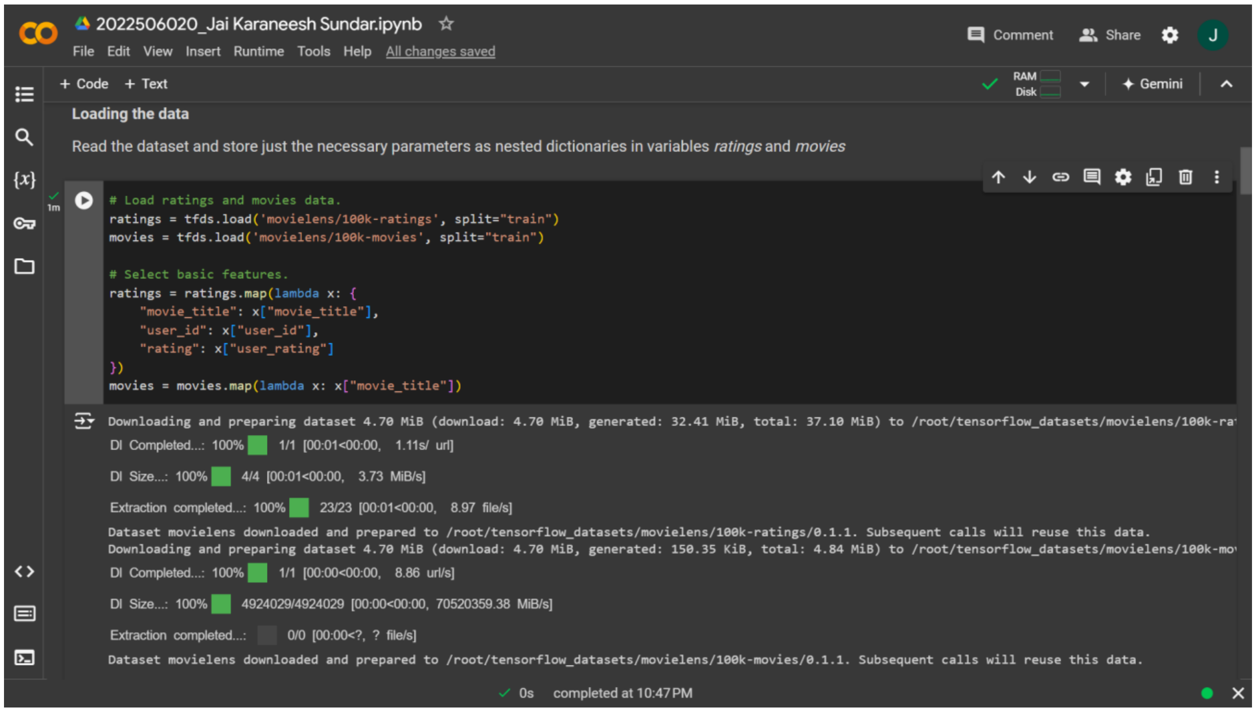
# RECOMMENDATION SYSTEM

AIOT PROJECT

**JAI KARANEESH SUNDAR**
**2022506020**
**B.TECH - INFORMATION TECHNOLOGY**
**MADRAS INSTITUTE OF TECHNOLOGY**
**CHENNAI**

To compare the output recommendations between different ML models working and training on the same dataset.

## LOADING THE DATASET


2022506020_Jai Karaneesh Sundar.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk
Gemini

**Loading the data**

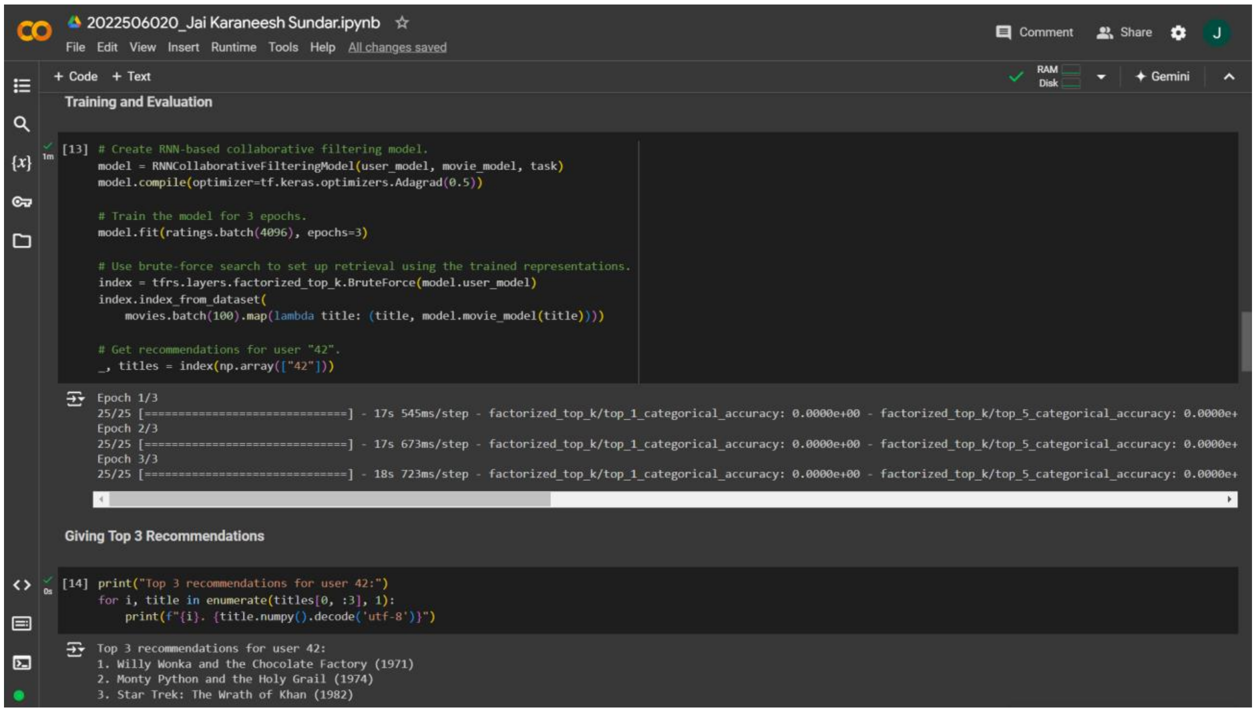
Read the dataset and store just the necessary parameters as nested dictionaries in variables *ratings* and *movies*

```
# Load ratings and movies data.
ratings = tfds.load('movielens/100k-ratings', split="train")
movies = tfds.load('movielens/100k-movies', split="train")

# Select basic features.
ratings = ratings.map(lambda x: {
    "movie_title": x["movie_title"],
    "user_id": x["user_id"],
    "rating": x["user_rating"]
})
movies = movies.map(lambda x: x["movie_title"])
```

```
Downloading and preparing dataset 4.70 MiB (download: 4.70 MiB, generated: 32.41 MiB, total: 37.10 MiB) to /root/tensorflow_datasets/movielens/100k-ra
Dl Completed...: 100% 🟩 1/1 [00:01<00:00,  1.11s/ url]
Dl Size...: 100% 🟩 4/4 [00:01<00:00,  3.73 MiB/s]
Extraction completed...: 100% 🟩 23/23 [00:01<00:00,  8.97 file/s]
Dataset movielens downloaded and prepared to /root/tensorflow_datasets/movielens/100k-ratings/0.1.1. Subsequent calls will reuse this data.
Downloading and preparing dataset 4.70 MiB (download: 4.70 MiB, generated: 150.35 KiB, total: 4.84 MiB) to /root/tensorflow_datasets/movielens/100k-mo
Dl Completed...: 100% 🟩 1/1 [00:00<00:00,  8.86 url/s]
Dl Size...: 100% 🟩 4924029/4924029 [00:00<00:00, 70520359.38 MiB/s]
Extraction completed...:  0/0 [00:00<?, ? file/s]
Dataset movielens downloaded and prepared to /root/tensorflow_datasets/movielens/100k-movies/0.1.1. Subsequent calls will reuse this data.
```

✓ 0s completed at 10:47 PM

## RECOMMENDATION SYSTEM 1: RECURRENT NEURAL NETWORK (RNN) MODEL


2022506020_Jai Karaneesh Sundar.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk
Gemini

**Training and Evaluation**

```
[13] # Create RNN-based collaborative filtering model.
     model = RNNCollaborativeFilteringModel(user_model, movie_model, task)
     model.compile(optimizer=tf.keras.optimizers.Adagrad(0.5))

     # Train the model for 3 epochs.
     model.fit(ratings.batch(4096), epochs=3)

     # Use brute-force search to set up retrieval using the trained representations.
     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))))

     # Get recommendations for user "42".
     _, titles = index(np.array(["42"]))
```

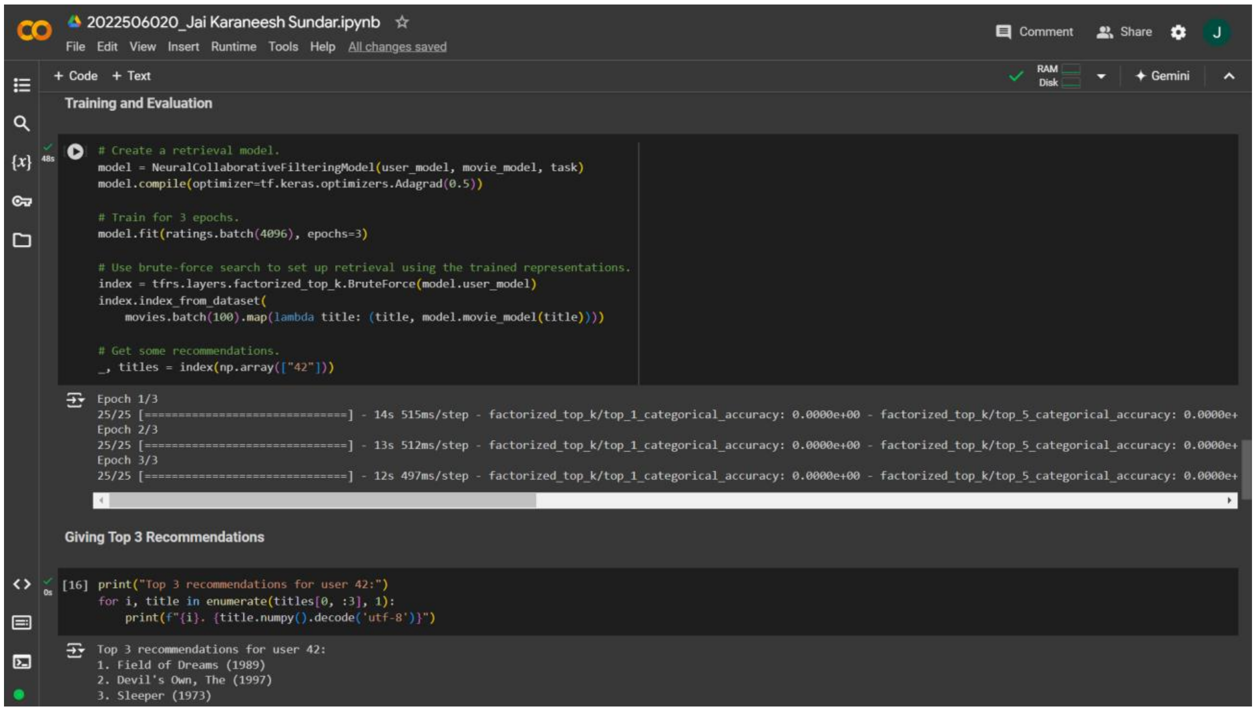
```
Epoch 1/3
25/25 [==============================] - 17s 545ms/step - factorized_top_k/top_1_categorical_accuracy: 0.0000e+00 - factorized_top_k/top_5_categorical_accuracy: 0.0000e+
Epoch 2/3
25/25 [==============================] - 17s 673ms/step - factorized_top_k/top_1_categorical_accuracy: 0.0000e+00 - factorized_top_k/top_5_categorical_accuracy: 0.0000e+
Epoch 3/3
25/25 [==============================] - 18s 723ms/step - factorized_top_k/top_1_categorical_accuracy: 0.0000e+00 - factorized_top_k/top_5_categorical_accuracy: 0.0000e+
```

**Giving Top 3 Recommendations**

```
[14] print("Top 3 recommendations for user 42:")
     for i, title in enumerate(titles[0, :3], 1):
         print(f"{i}. {title.numpy().decode('utf-8')}")
```

```
Top 3 recommendations for user 42:
1. Willy Wonka and the Chocolate Factory (1971)
2. Monty Python and the Holy Grail (1974)
3. Star Trek: The Wrath of Khan (1982)
```

## RECOMMENDATION SYSTEM 2: NEURAL COLLABORATIVE FILTERING (NCF) MODEL


2022506020_Jai Karaneesh Sundar.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk
Gemini

Training and Evaluation

```
# Create a retrieval model.
model = NeuralCollaborativeFilteringModel(user_model, movie_model, task)
model.compile(optimizer=tf.keras.optimizers.Adagrad(0.5))

# Train for 3 epochs.
model.fit(ratings.batch(4096), epochs=3)

# Use brute-force search to set up retrieval using the trained representations.
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))))

# Get some recommendations.
_, titles = index(np.array(["42"]))
```

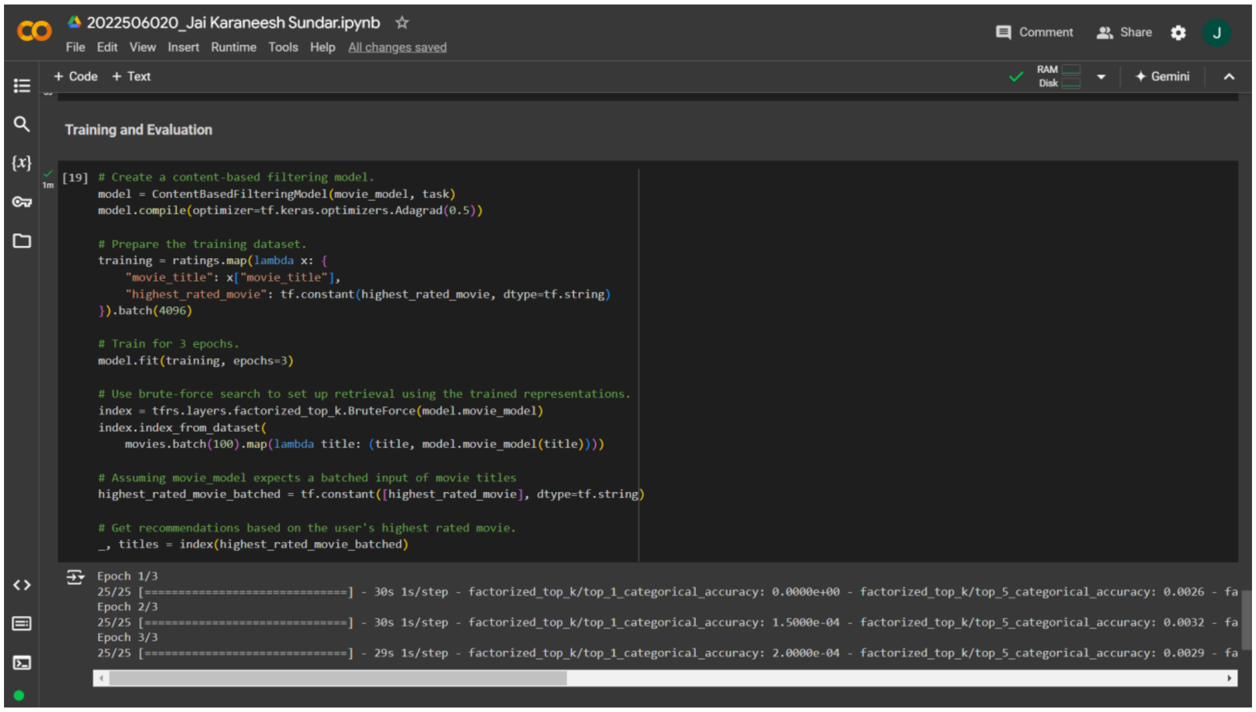
```
Epoch 1/3
25/25 [==============================] - 14s 515ms/step - factorized_top_k/top_1_categorical_accuracy: 0.0000e+00 - factorized_top_k/top_5_categorical_accuracy: 0.0000e+
Epoch 2/3
25/25 [==============================] - 13s 512ms/step - factorized_top_k/top_1_categorical_accuracy: 0.0000e+00 - factorized_top_k/top_5_categorical_accuracy: 0.0000e+
Epoch 3/3
25/25 [==============================] - 12s 497ms/step - factorized_top_k/top_1_categorical_accuracy: 0.0000e+00 - factorized_top_k/top_5_categorical_accuracy: 0.0000e+
```

Giving Top 3 Recommendations

```
[16] print("Top 3 recommendations for user 42:")
     for i, title in enumerate(titles[0, :3], 1):
         print(f"{i}. {title.numpy().decode('utf-8')}")
```

```
Top 3 recommendations for user 42:
1. Field of Dreams (1989)
2. Devil's Own, The (1997)
3. Sleeper (1973)
```

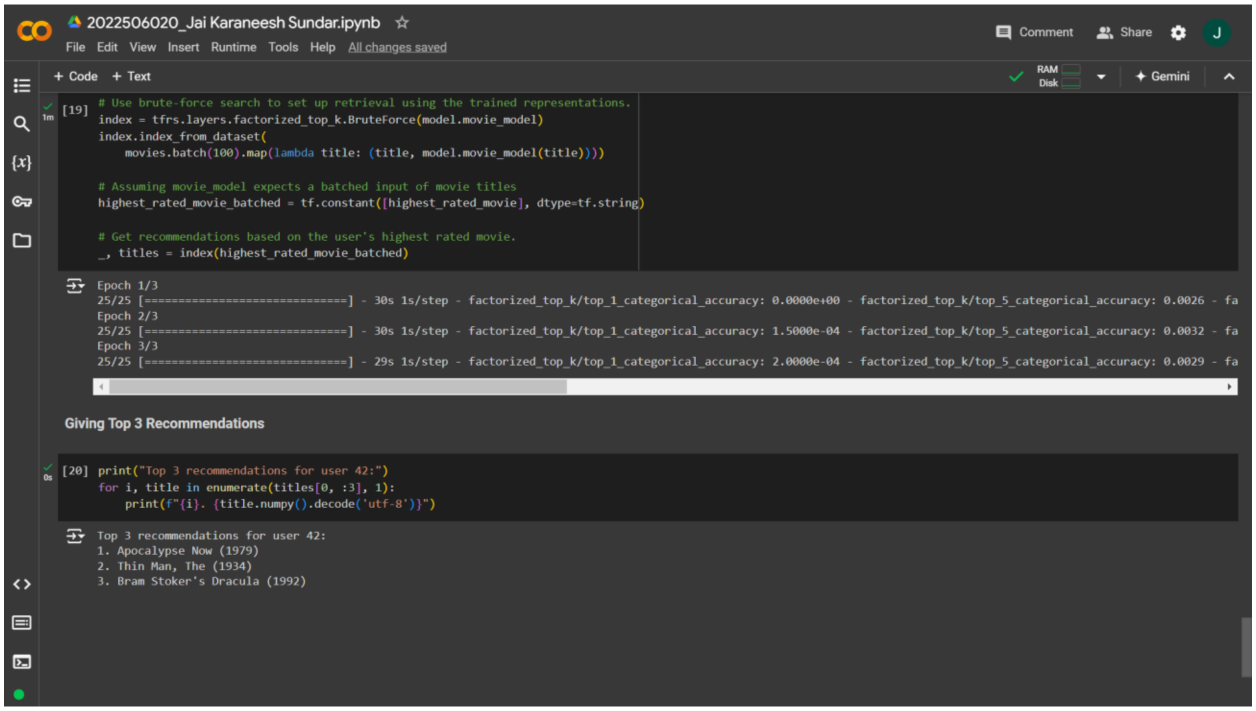
## RECOMMENDATION SYSTEM 3: CONTENT BASED FILTERING MODEL


2022506020_Jai Karaneesh Sundar.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk
Gemini

Training and Evaluation

```
[19] # Create a content-based filtering model.
     model = ContentBasedFilteringModel(movie_model, task)
     model.compile(optimizer=tf.keras.optimizers.Adagrad(0.5))

     # Prepare the training dataset.
     training = ratings.map(lambda x: {
         "movie_title": x["movie_title"],
         "highest_rated_movie": tf.constant(highest_rated_movie, dtype=tf.string)
     }).batch(4096)

     # Train for 3 epochs.
     model.fit(training, epochs=3)

     # Use brute-force search to set up retrieval using the trained representations.
     index = tfrs.layers.factorized_top_k.BruteForce(model.movie_model)
     index.index_from_dataset(
         movies.batch(100).map(lambda title: (title, model.movie_model(title))))

     # Assuming movie_model expects a batched input of movie titles
     highest_rated_movie_batched = tf.constant([highest_rated_movie], dtype=tf.string)

     # Get recommendations based on the user's highest rated movie.
     _, titles = index(highest_rated_movie_batched)
```

```
Epoch 1/3
25/25 [==============================] - 30s 1s/step - factorized_top_k/top_1_categorical_accuracy: 0.0000e+00 - factorized_top_k/top_5_categorical_accuracy: 0.0026 - fa
Epoch 2/3
25/25 [==============================] - 30s 1s/step - factorized_top_k/top_1_categorical_accuracy: 1.5000e-04 - factorized_top_k/top_5_categorical_accuracy: 0.0032 - fa
Epoch 3/3
25/25 [==============================] - 29s 1s/step - factorized_top_k/top_1_categorical_accuracy: 2.0000e-04 - factorized_top_k/top_5_categorical_accuracy: 0.0029 - fa
```


2022506020_Jai Karaneesh Sundar.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk
Gemini

```
[19] # Use brute-force search to set up retrieval using the trained representations.
     index = tfrs.layers.factorized_top_k.BruteForce(model.movie_model)
     index.index_from_dataset(
         movies.batch(100).map(lambda title: (title, model.movie_model(title))))

     # Assuming movie_model expects a batched input of movie titles
     highest_rated_movie_batched = tf.constant([highest_rated_movie], dtype=tf.string)

     # Get recommendations based on the user's highest rated movie.
     _, titles = index(highest_rated_movie_batched)
```

```
Epoch 1/3
25/25 [==============================] - 30s 1s/step - factorized_top_k/top_1_categorical_accuracy: 0.0000e+00 - factorized_top_k/top_5_categorical_accuracy: 0.0026 - fa
Epoch 2/3
25/25 [==============================] - 30s 1s/step - factorized_top_k/top_1_categorical_accuracy: 1.5000e-04 - factorized_top_k/top_5_categorical_accuracy: 0.0032 - fa
Epoch 3/3
25/25 [==============================] - 29s 1s/step - factorized_top_k/top_1_categorical_accuracy: 2.0000e-04 - factorized_top_k/top_5_categorical_accuracy: 0.0029 - fa
```

**Giving Top 3 Recommendations**

```
[20] print("Top 3 recommendations for user 42:")
     for i, title in enumerate(titles[0, :3], 1):
         print(f"{i}. {title.numpy().decode('utf-8')}")
```

```
Top 3 recommendations for user 42:
1. Apocalypse Now (1979)
2. Thin Man, The (1934)
3. Bram Stoker's Dracula (1992)
```