HARPY AEROSPACE SUMMER INTERNSHIP

AIOT ZONE PROJECT

# MOVIE RECOMMENDATION SYSTEM

NIVETHA T

2022510007

BTech Artificial Intelligence & Data Science

MADRAS INSTITUTE OF TECHNOLOGY

MODEL-1

```
[17]
def build_user_model(user_ids_vocabulary):
    return tf.keras.Sequential([
        user_ids_vocabulary,
        tf.keras.layers.Embedding(user_ids_vocabulary.vocabulary_size(), 64),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(64, activation='relu'),
    ])

def build_movie_model(movie_titles_vocabulary):
    return tf.keras.Sequential([
        movie_titles_vocabulary,
        tf.keras.layers.Embedding(movie_titles_vocabulary.vocabulary_size(), 64),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(64, activation='relu'),
    ])
```

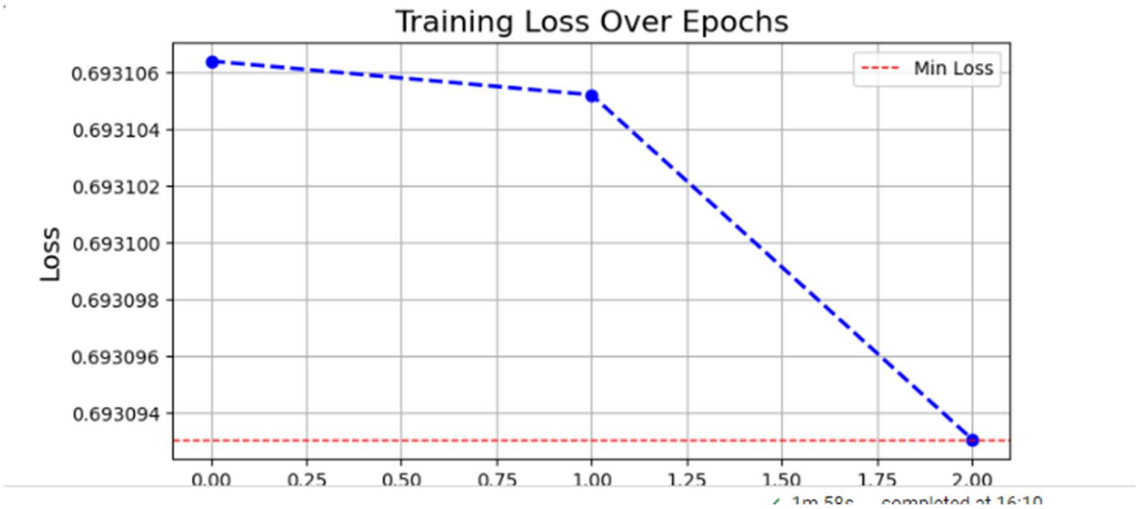
```
index = tfrs.layers.factorized_top_k.BruteForce(model.user_model)
index.index_from_dataset(
    movies.batch(100).map(lambda title: (title, model.movie_model(title)))
)
```

```
Epoch 1/3
25/25 [==============================] - 38s 2s/step - loss: 0.6931 - factorized_top_k/top_1
Epoch 2/3
25/25 [==============================] - 37s 1s/step - loss: 0.6931 - factorized_top_k/top_1
Epoch 3/3
25/25 [==============================] - 38s 2s/step - loss: 0.6931 - factorized_top_k/top_1
<tensorflow_recommenders.layers.factorized_top_k.BruteForce at 0x7f4b67c7f160>
```

```
[20] # Train the model
model.fit(ratings.batch(4096), epochs=3, callbacks=[history])
```

```
Epoch 1/3
25/25 [==============================] - 38s 2s/step - loss: 0.6931 - factorized_top_k/top_1
Epoch 2/3
25/25 [==============================] - 38s 2s/step - loss: 0.6931 - factorized_top_k/top_1
Epoch 3/3
25/25 [==============================] - 47s 2s/step - loss: 0.6931 - factorized_top_k/top_1
<keras.src.callbacks.History at 0x7f4b65ce5300>
```

```
# Plot the training loss with colors
plt.figure(figsize=(8, 4))
plt.plot(history.losses, color='blue', marker='o', linestyle='dashed', linewidth=2, markersize=6)
plt.title('Training Loss Over Epochs', fontsize=16)
plt.xlabel('Epoch', fontsize=14)
plt.ylabel('Loss', fontsize=14)
plt.grid(True)
plt.axhline(y=min(history.losses), color='red', linestyle='--', linewidth=1, label='Min Loss')
plt.legend()
plt.show()
```




✓ 1m 58s completed at 16:10

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

# Print recommendations as bulleted points
print("Top 10 recommendations for user 42:")
for i, title in enumerate(titles[0, :10].numpy(), start=1):
    print(f"{i}. {title.decode('utf-8')}")
```

```
Top 10 recommendations for user 42:
1. For Richer or Poorer (1997)
2. G.I. Jane (1997)
3. Some Folks Call It a Sling Blade (1993)
4. Old Man and the Sea, The (1958)
5. Target (1995)
6. Trees Lounge (1996)
7. Diva (1981)
8. Mother Night (1996)
9. Fall (1997)
10. Deconstructing Harry (1997)
```

MODEL-2

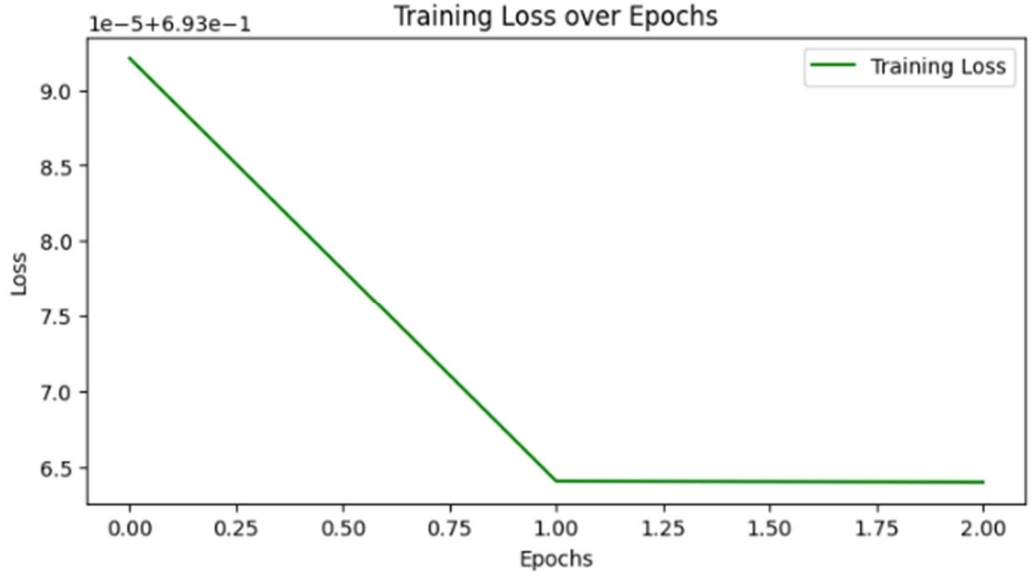
## Define the model

```
[25] class MovieLensModel(tfrs.Model):
         def __init__(self, user_model, movie_model):
             super().__init__()
             self.user_model = user_model
             self.movie_model = movie_model
             self.dropout = tf.keras.layers.Dropout(0.2)
             self.task = tfrs.tasks.Retrieval()

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

             user_embeddings = self.dropout(user_embeddings, training=training)
             movie_embeddings = self.dropout(movie_embeddings, training=training)

             return self.task(user_embeddings, movie_embeddings)
```

```
[28] # Train the model
     model.fit(ratings.batch(4096), epochs=3)
```

```
Epoch 1/3
25/25 [==============================] - 38s 1s/step - loss: 0.6931 - factorized_
Epoch 2/3
25/25 [==============================] - 38s 2s/step - loss: 0.6931 - factorized_
Epoch 3/3
25/25 [==============================] - 37s 1s/step - loss: 0.6931 - factorized_
<keras.src.callbacks.History at 0x7f4b65ccc0a0>
```

```
[29] plt.figure(figsize=(8, 4))
     plt.plot(history['loss'], label='Training Loss', color='green')  # Change color here
     plt.xlabel('Epochs')
     plt.ylabel('Loss')
     plt.legend()
     plt.title('Training Loss over Epochs')
     plt.show()
```




```
# Use 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))
)

# Get recommendations for a specific user (user id = "42")
_, titles = index(np.array(["42"]))
print("Top 10 recommendations for user 42:")
for i, title in enumerate(titles[0, :10], start=1):
    print(f"{i}. {title.numpy().decode('utf-8')}")
```

```
Top 10 recommendations for user 42:
1. For Richer or Poorer (1997)
2. G.I. Jane (1997)
3. Some Folks Call It a Sling Blade (1993)
4. Old Man and the Sea, The (1958)
5. Target (1995)
6. Trees Lounge (1996)
7. Diva (1981)
8. Mother Night (1996)
9. Fall (1997)
10. Deconstructing Harry (1997)
```

MODEL-3

```python
# Part 4: Define the BPR Model
class BPRModel(tf.keras.Model):
    def __init__(self, user_model: tf.keras.Model, movie_model: tf.keras.Model, task: tfrs.tasks.Retrieval):
        super().__init__()
        self.user_model = user_model
        self.movie_model = movie_model
        self.task = task

    def call(self, features: Dict[str, tf.Tensor], training=False) -> tf.Tensor:
        user_embeddings = self.user_model(features["user_id"])
        positive_movie_embeddings = self.movie_model(features["movie_title"])

        # Generate negative samples
        negative_movie_embeddings = self.movie_model(tf.random.shuffle(features["movie_title"]))

        # Calculate the BPR loss
        positive_scores = tf.reduce_sum(user_embeddings * positive_movie_embeddings, axis=1)
        negative_scores = tf.reduce_sum(user_embeddings * negative_movie_embeddings, axis=1)
        bpr_loss = -tf.reduce_mean(tf.math.log(tf.nn.sigmoid(positive_scores - negative_scores)))

        self.add_loss(bpr_loss)  # Add BPR loss to model's losses

        return self.task(user_embeddings, positive_movie_embeddings)
```

```
Epoch 1/3
25/25 [==============================] - 42s 2s/step - loss: 0.6931 - factorized_top_k/top_1_
Epoch 2/3
25/25 [==============================] - 38s 1s/step - loss: 0.6931 - factorized_top_k/top_1_
Epoch 3/3
25/25 [==============================] - 39s 2s/step - loss: 0.6931 - factorized_top_k/top_1_
```

```
matplotlib.legend.Legend at 0x7f4b746ced70>
```

```python
# Set up retrieval using brute-force search with 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 a specific user
_, titles = index(np.array(["42"]))
print("Top 10 recommendations for user 42:")
for i, title in enumerate(titles[0, :10], 1):
    print(f"{i}. {title.numpy().decode('utf-8')}")
```

```
Top 10 recommendations for user 42:
1. Rough Magic (1995)
2. Touch (1997)
3. Misérables, Les (1995)
4. Race the Sun (1996)
5. Sex, Lies, and Videotape (1989)
6. Niagara, Niagara (1997)
7. Desperate Measures (1998)
8. Desperate Measures (1998)
9. Chasing Amy (1997)
10. Chasing Amy (1997)
```