Maaz Saad

Professor Miguel V. Martin

MBAI 5310G

3 April 2023

Movie Recommendation System in Python

There was a time when people had to rely on video store clerks for film recommendations, hoping for the best (Kniazieva). Although not that long ago, it feels like centuries have passed since then. The change in the way people consume media has been revolutionary in recent years, thanks to the rise of streaming platforms. Nowadays, all the films in the world have been digitized and are easily accessible through streaming platforms such as Netflix, Crave, and Disney Plus (Kniazieva). These platforms present an opportunity to keep viewers engaged for longer periods of time. Data scientists have utilized machine learning tools to recommend films to people based on their past viewing history and other metrics (Kniazieva). This is an impressive achievement, and companies like Netflix have developed accurate algorithms that are well-received by the public.

For my final project, I decided to build a movie recommendation system because I have been passionate about films and cinema since childhood. Although there may have been other projects that I could have worked on and possibly done a better job on, this opportunity was too good to pass up. I wanted to work on a project that I was truly passionate about, and films have been my constant companion through thick and thin. Before delving deeper into my own project, I would like to provide some technical background on the concept and ideology behind a machine learning-powered movie recommendation system.

There are several strategies and sets of algorithms that can be used to filter through thousands of films and provide recommendations to users (Kniazieva). The two most common approaches are **content-based filtering** and **collaborative filtering** (Kniazieva).

Content-based filtering involves reviewing a user's past film selections to determine what film they are likely to watch next (Kniazieva). For example, the model can analyze a user's viewing history and determine that they only watch films by Alfred Hitchcock, so the algorithm can recommend more films by the same director. Similarly, the algorithm can determine that a user only watches action flicks and provide recommendations accordingly. It's important to note that this type of filtering only uses data collected directly from the user for analysis (Kniazieva).

Collaborative filtering is a strategy that analyzes patterns across a set of users (Kniazieva). This approach considers the interactions of all users in the system, and the final prediction is a function of the entire data set rather than a single person (Kniazieva). For instance, if two users are deemed similar by the algorithm and user one has watched a film that user two hasn't, the algorithm will recommend that film to user two (Kniazieva).

I utilized the MovieLens 1M Dataset, which can be accessed through this link. The dataset includes 1 million ratings for 4,000 movies, which were provided by 6,000 users (Grouplens). Ideally, I would have preferred to work with the 25M dataset, which includes 25 million ratings. However, splitting that larger dataset would have taken a considerable amount of time, whereas working with the 1M dataset only took approximately 8 minutes.

In this section, I will review the code that I wrote for this project, explaining the purpose of each block of code and how it fits into the bigger picture. Additionally, I will assess my code in the context of the sources that I used for guidance throughout the project. Finally, I will examine some ethical considerations related to my project and suggest ways to address them.

I first used a tutorial on TensorFlow's website that breaks down the process of recommending movies. In addition, I consulted two similar guides that incorporate the TensorFlow tutorial in their own way. These three sources are cited in the references section of this report and also in the accompanying .ipynb file.

```
# Load the data and split it into training and test set

# Ratings data
ratings = tfds.load("movielens/1m-ratings", split="train")

# Movies data
movies = tfds.load("movielens/1m-movies", split="train")

# We only use the ratings data & within that, we use the movie title & user id

ratings = ratings.map(lambda x: { "movie_title": x["movie_title"],
    "user_id": x["user_id"],})
movies = movies.map(lambda x: x["movie_title"])

# We split the data in 80% training and 20% testing sets

tf.random.set_seed(42)
shuffled = ratings.shuffle(100_000, seed=42, reshuffle_each_iteration=False)

train = shuffled.take(80_000)
test = shuffled.skip(80_000).take(20_000)

# Extract unique sets of movie titles and user IDs

movie_titles = movies.batch(1_000)
user_ids = ratings.batch(1_000_000).map(lambda x: x["user_id"])

unique_movie_titles = np.unique(np.concatenate(list(movie_titles)))
unique_user_ids = np.unique(np.concatenate(list(user_ids)))
```

In this first block of code, we load and preprocess the MovieLens 1M dataset for our recommendation system. Firstly, we load the ratings and movies data and store them in their respective variables. Then, we extract only the movie title and user ID from the ratings data using a lambda function. Similarly, we use a lambda function to extract only the movie title from the movies data. Next, we split the data into training and testing sets. The ratings data is randomly shuffled and divided into an 80% training set and a 20% test set. After that, we batch the movie titles into sets of 1,000 and the ratings into sets of 1,000,000. We then map the ratings data to extract only the user ID. Finally, we combine the batches into a single array and extract unique movie titles and user IDs. This preprocessing ensures that our recommendation system has access to relevant and accurate data, which will improve its performance.

```
# Build the query tower (Set up the retrieval model)

embedding_dimension = 32

user_model = tf.keras.Sequential([
    tf.keras.layers.StringLookup(
      vocabulary=unique_user_ids, mask_token=None),
  tf.keras.layers.Embedding(len(unique_user_ids) + 1, embedding_dimension)])

# Build the movie candidate tower

movie_model = tf.keras.Sequential([
  tf.keras.layers.StringLookup(
      vocabulary=unique_movie_titles, mask_token=None),
  tf.keras.layers.Embedding(len(unique_movie_titles) + 1,
                            embedding_dimension)])

#Evaluate the performance of the model

metrics = tfrs.metrics.FactorizedTopK(candidates=movies.batch(128).
                                      map(movie_model))

# Evaluate the loss used to train the model

task = tfrs.tasks.Retrieval(metrics=metrics)
```

In the next block of code, we will build the retrieval model. Firstly, we set the embedding dimension to specify the size of the embedding vectors for both users and movies. Next, we use a sequential model to convert user IDs to integers. An embedding layer is then employed to map the integer indices to embedding vectors. We do the same for movie titles by converting them to integers. We then evaluate the performance of the recommendation model using the top-k metric. In this case, we used unique movie titles to assess the performance. Finally, we evaluate the loss used to train the model on the metrics parameter, which is essentially the performance metric defined in the previous step. We will use the computed loss to implement the model's training loop. This retrieval model is a crucial component of our recommendation system. It enables us to generate relevant recommendations for users by

finding the most similar movies to the ones they have already watched. The evaluation metrics provide us with valuable insights into the model's performance, which we can use to fine-tune and optimize the system.

```
# Put it together into a model

class MovielensModel(tfrs.Model):

  def __init__(self, user_model, movie_model):
    super().__init__()
    self.movie_model: tf.keras.Model = movie_model
    self.user_model: tf.keras.Model = user_model
    self.task: tf.keras.layers.Layer = task

  def compute_loss(self, features: Dict[Text, tf.Tensor],
                   training=False) -> tf.Tensor:
# We pick out the user features and pass them into the user model
    user_embeddings = self.user_model(features["user_id"])
# And pick out the movie features & pass them into the movie model,
# getting embeddings back
    positive_movie_embeddings = self.movie_model(features["movie_title"])

# The task computes the loss and the metrics
    return self.task(user_embeddings, positive_movie_embeddings)
```

In the next block of code, we will combine all the previous components into a complete model. The MovielensModel will incorporate the user model and movie model as input, which are the two towers in the retrieval model. In the "init" method, we initialize the three attributes. The compute_loss method will return the computed loss, which we will use to optimize the model during training. Within the compute_loss function, we obtain the user embeddings by passing the user IDs through the user model. Similarly, we get the movie embeddings by passing the movie titles through the movie model. Finally, the task computes the loss and metrics based on the user embeddings and positive movie embeddings.

```
# Fit and evaluate the model

model = MovielensModel(user_model, movie_model)
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.1))
cached_train = train.shuffle(100_000).batch(8192).cache()
cached_test = test.batch(4096).cache()
model.fit(cached_train, epochs=10)
```

```
l_accuracy: 0.0924 - factorized_top_k/top_10_categorical_accuracy: 0.1549

l_accuracy: 0.1512 - factorized_top_k/top_10_categorical_accuracy: 0.2437

l_accuracy: 0.1600 - factorized_top_k/top_10_categorical_accuracy: 0.2604

l_accuracy: 0.1596 - factorized_top_k/top_10_categorical_accuracy: 0.2679

l_accuracy: 0.1706 - factorized_top_k/top_10_categorical_accuracy: 0.2846

l_accuracy: 0.1703 - factorized_top_k/top_10_categorical_accuracy: 0.2889

l_accuracy: 0.1708 - factorized_top_k/top_10_categorical_accuracy: 0.2913

l_accuracy: 0.1711 - factorized_top_k/top_10_categorical_accuracy: 0.2944

l_accuracy: 0.1715 - factorized_top_k/top_10_categorical_accuracy: 0.2974

l_accuracy: 0.1722 - factorized_top_k/top_10_categorical_accuracy: 0.2985
```

Once the model is defined, we can use Keras to fit and evaluate it. Firstly, we create an instance of the model using the user model and movie model we created earlier. Next, we compile the model using the Adam optimizer with a learning rate of 0.1. To speed up the training process, we create cached versions of the training and test sets. The training set is shuffled, batched into groups of 8,192, and cached, while the test set is batched into groups of 4,096 and cached. Finally, we fit the model to the training set and run it for 10 epochs. This trains the model on the training data, updating the model parameters to decrease the computed loss. The model will be evaluated automatically after each epoch using the cached test data. We can see the loss decreasing and the top-k retrieval metrics being updated as the model trains. For example, a top-10 categorical accuracy metric of 0.3 would indicate that, on average, the true positive is in the top 10 retrieved items 30% of the time. We can observe the accuracy metric improving across the epochs.

```
# Evaluate the model on the test set

model.evaluate(cached_test, return_dict=True)
```

```
5/5 [==============================] - 21s 2s/step - factorized_top_k/top_
{'factorized_top_k/top_1_categorical_accuracy': 4.999999873689376e-05,
 'factorized_top_k/top_5_categorical_accuracy': 0.0006500000017695129,
 'factorized_top_k/top_10_categorical_accuracy': 0.0022499999031424522,
 'factorized_top_k/top_50_categorical_accuracy': 0.02800000086426735,
 'factorized_top_k/top_100_categorical_accuracy': 0.06129999831318855,
 'loss': 44066.7890625,
 'regularization_loss': 0,
 'total_loss': 44066.7890625}
```

In this step, we evaluate the trained model on the test set. We pass the cached test data as input to the model, which uses the trained model to make predictions on the test set. The output is the loss and accuracy metrics of the predictions. However, we observe that the accuracy metrics for the test set are somewhat lower than those for the training set. I will discuss this further in the conclusion section of the report.

```
# We can make predictions now

index = tfrs.layers.factorized_top_k.BruteForce(model.user_model)
index.index_from_dataset(tf.data.Dataset.zip((movies.batch(100),
                                              movies.batch(100).map(model.movie_model))))

# Get recommendations

_, titles = index(tf.constant(["20"]))
print(f"Recommendations for user 20: {titles[0, :3]}")

_, titles = index(tf.constant(["68"]))
print(f"Recommendations for user 68: {titles[0, :3]}")

_, titles = index(tf.constant(["99"]))
print(f"Recommendations for user 99: {titles[0, :3]}")

_, titles = index(tf.constant(["129"]))
print(f"Recommendations for user 129: {titles[0, :3]}")

_, titles = index(tf.constant(["212"]))
print(f"Recommendations for user 212: {titles[0, :3]}")
```

```
Recommendations for user 20: [b'Saving Private Ryan (1998)' b'Matrix, The (1999)' b'Sommersby (1993)']
Recommendations for user 68: [b'Persuasion (1995)' b'Chasing Amy (1997)' b'Scary Movie (2000)']
Recommendations for user 99: [b'Next Best Thing, The (2000)' b'X-Men (2000)' b'Twelve Monkeys (1995)']
Recommendations for user 129: [b'True Crime (1999)' b'Fantastic Voyage (1966)' b'Donnie Brasco (1997)']
Recommendations for user 212: [b'Eraser (1996)' b'Kissing a Fool (1998)' b'Jerry Maguire (1996)']
```

In the final step, we are ready to use our model to make movie recommendations to the users in the dataset. First, we set up a brute-force retrieval index and pass in the user model to this class. Next, we create an index from the dataset by zipping the movie titles and the movie embeddings generated by the model's movie tower. After setting up the index, we can obtain recommendations for specific users by passing their user ID to the index.

It is essential to remember that building machine learning models is a privilege, and not a right. Therefore, we must build models that are free from discrimination and bias. We must strive to incorporate large and diverse datasets into our models and ensure that they do not disadvantage specific groups of people based on their gender, religion, or race. For my project, I chose a dataset with one million entries, which is a decent-sized dataset. I will now analyze my model and its implications in light of Google's AI principles.

Keeping the principles of responsible AI practices in mind, I believe that the algorithm I developed is socially beneficial. If implemented on a website after undergoing some optimization, it can be a source of comfort in people's lives. Fortunately, my data did not include any sensitive or confidential variables that could harm the privacy of an individual. If necessary, I would be willing to share my code with users who wish to see how the recommendation process works, essentially rendering my model not a black box.

The purpose of my algorithm is to recommend movies to people who love films. The intention of my machine learning model is not to cause harm or gather information about my

users. Furthermore, in future iterations of this project, I can incorporate more data points and conduct more testing on the data to improve its results. The goal is to provide recommendations that make users happy and appreciate the tool that I have built.

Although I believe the nature of my model spares it from being misused, we cannot predict how the world works. Hackers may gain access to a person's viewing history and use it against them. Therefore, it is essential to use data that has been stripped of all personal variables, which I did in this project.

In conclusion, I have developed and deployed a movie recommendation system to the best of my abilities, utilizing the knowledge acquired during this course. Although I had considered other project ideas, such as building a chatbot using NLTK and Keras and using image recognition for medical diagnosis, I ultimately settled on the recommendation algorithm as it was something I believed I would enjoy, and I did. To guide me through this project, I utilized TensorFlow's tutorial and other sources to understand the code's purpose and its application to my situation.

I acknowledge that there is still a lot of room for improvement in this model, and it can be further optimized with more training and testing data that incorporates more variables about movies, such as genre, directors, and performers, as well as more variables about the users, such as their age, employment status, and income. Introducing these metrics may improve the accuracy of the recommendation system, but it may also increase the model's susceptibility to cyber-attacks and misuse of data. Therefore, proper planning and execution will be necessary in future iterations of this *AI Programming* project.

This side project is not directly related to this project, but it is somewhat within the same realm. I have an immense passion for films, thanks to my father, and it surpasses all other interests. As luck would have it, I produced a short film in my dorm room during the past winter break when I had no other commitments. I was able to showcase the film to my father right at the very end.

If you are interested in watching the film, please follow the link below. Although it is less than three minutes long, every second was meticulously planned, and it enabled me to gain a deeper appreciation for the art of filmmaking.

https://github.com/maazsaad1440/Filmmaking-Project

Works Cited

Kniazieva, Y. (2022, April 14). Guide to movie recommendation systems using machine learning. https://labelyourdata.com/articles/movie-recommendation-with-machine-learning

Movielens. (2013, September 6). GroupLens. https://grouplens.org/datasets/movielens/

Our principles. (n.d.). Google AI. Retrieved April 3, 2023, from https://ai.google/principles/

Responsible AI practices. (n.d.). Google AI. Retrieved April 3, 2023, from https://ai.google/responsibilities/responsible-ai-practices/

Recommending movies: Retrieval | TensorFlow Recommenders. (n.d.). TensorFlow. Retrieved April 3, 2023, from https://www.tensorflow.org/recommenders/examples/basic_retrieval

Movies recommendation systems with tensorflow. (2022, October 12). Paperspace Blog. https://blog.paperspace.com/movie-recommender-tensorflow/

Bilalsedef. (2022, February 13). Creating movie recommender system with tensorflow recommenders(Tfrs). Medium. https://medium.com/@bilalsedef/creating-movie-recommender-system-with-tensorflow-recommenders-tfrs-acc19fd9f7a0

General format—Purdue owl®—Purdue university. (n.d.). Retrieved April 3, 2023, from https://owl.purdue.edu/owl/research_and_citation/mla_style/mla_formatting_and_style_guide/mla_general_format.html