<Exploring the Dynamics of Movie Recommendation Systems: A Research Report>

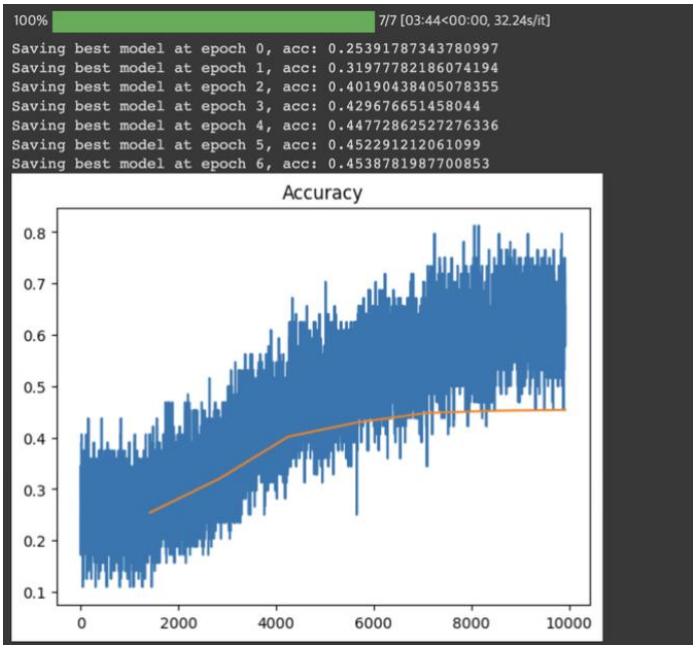
Taeyoon-Lee(Art & Technology, 20231099)

Entering the modern era, the film industry has experienced rapid growth, and with the advancement of digital technology, we now have easy access to a diverse array of movies anytime, anywhere. However, this diversity has brought about challenges for users in navigating through the multitude of options, _ the emergence of movie recommendation systems. These systems aim to address the difficulty users face in choosing movies by understanding their individual preferences and providing personalized recommendations. As a result, movie recommendation systems have gained significant attention in recent times.

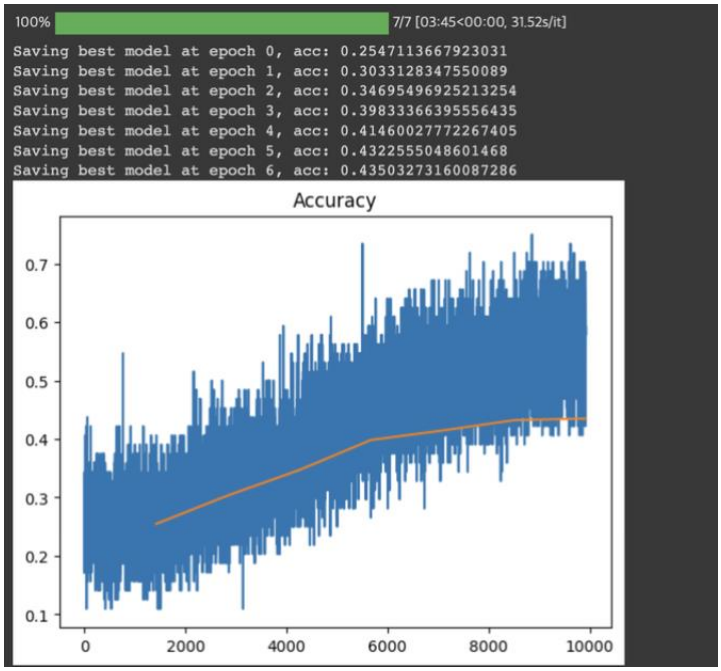
In this study, our goal is to conduct a thorough analysis of the performance and operational mechanisms of movie recommendation systems. Additionally, we aim to enhance our understanding of the algorithms and technologies utilized in these systems by tackling coding problems related to the technologies employed in the respective recommendation system. Through this research, we seek to elevate our comprehension of how recommendation systems leverage algorithms and technologies to understand user preferences.

Before analyzing the recommendation system, I took steps to sufficiently enhance its performance by setting values for embedding vectors and training epochs. Initially, I configured the embedding vector values, followed by establishing the training epoch values. Concerning the embedding vector values, if set too small, it could lead to underfitting issues, causing the model to exhibit low performance across training, validation, and test data. On the other hand, if set too large, overfitting problems might arise, showcasing excellent performance only on the training data but poor results on validation and test data. Therefore, determining an appropriate value involved finding a balance, avoiding extremes of both too large and too small values.

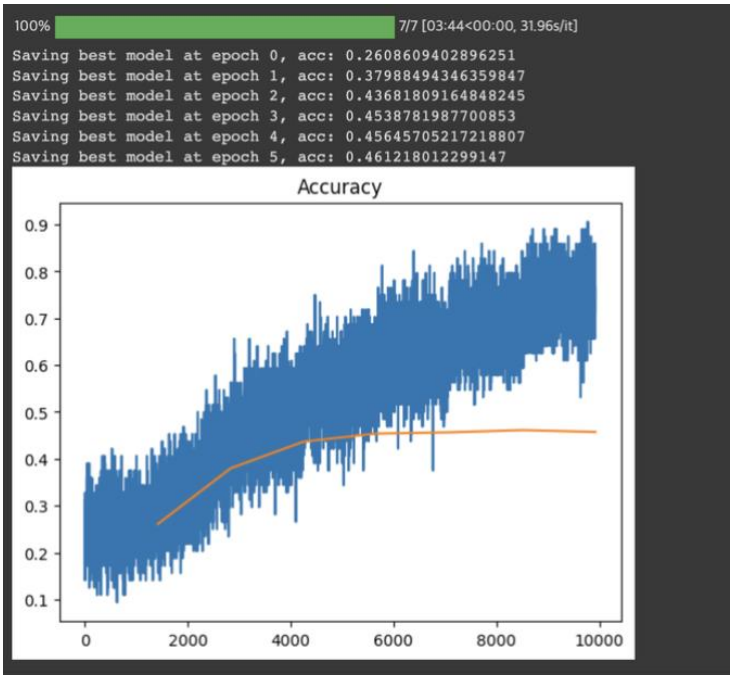
Initially, I conducted training with the default value of 50 for the embedding vectors. As a result, the maximum accuracy on the validation data reached 45.3%.



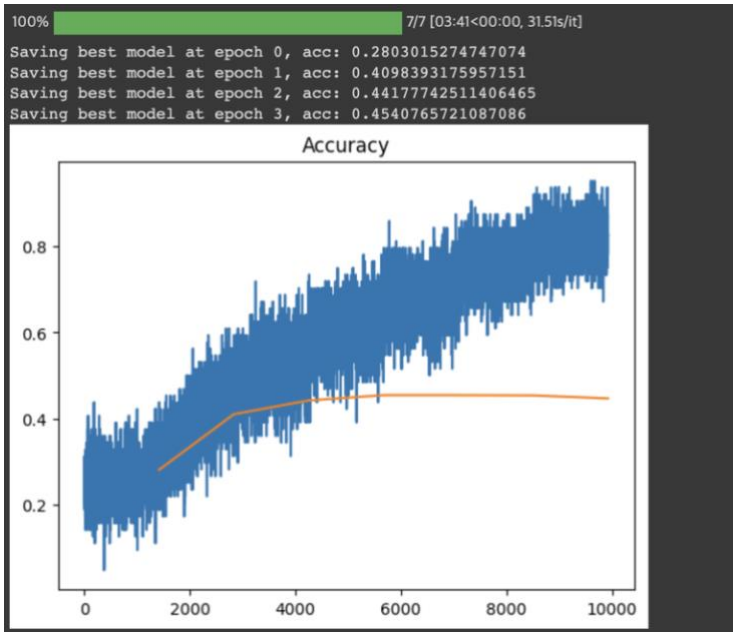
Subsequently, I adjusted the value to 25, and the maximum accuracy decreased further to 43.5%, indicating a reduction compared to the setting with 50. Observing this outcome, I concluded that this scenario aligns with the situation described earlier, where the embedding vector value is too small.



Therefore, I increased the value beyond 50 to set it at 100. The result showed a maximum accuracy of 46.1%. Since this accuracy was notably higher than when it was set to 50, I decided to proceed with a higher value, aiming for further improvement.



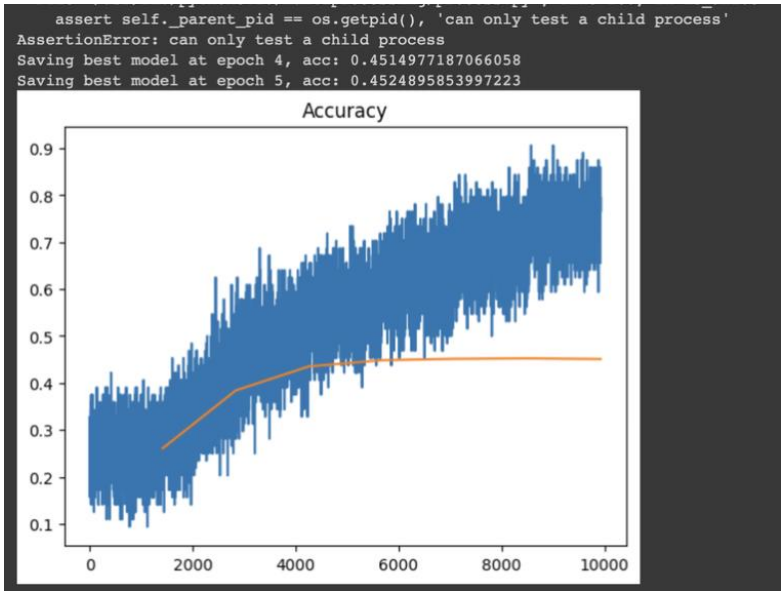
Consequently, I set the value to 150 and attempted the training again, resulting in a maximum accuracy of 45.4%. Since the maximum accuracy was lower than when it was set to 100, I concluded that this scenario corresponds to the situation described earlier, where the embedding vector value is too large.



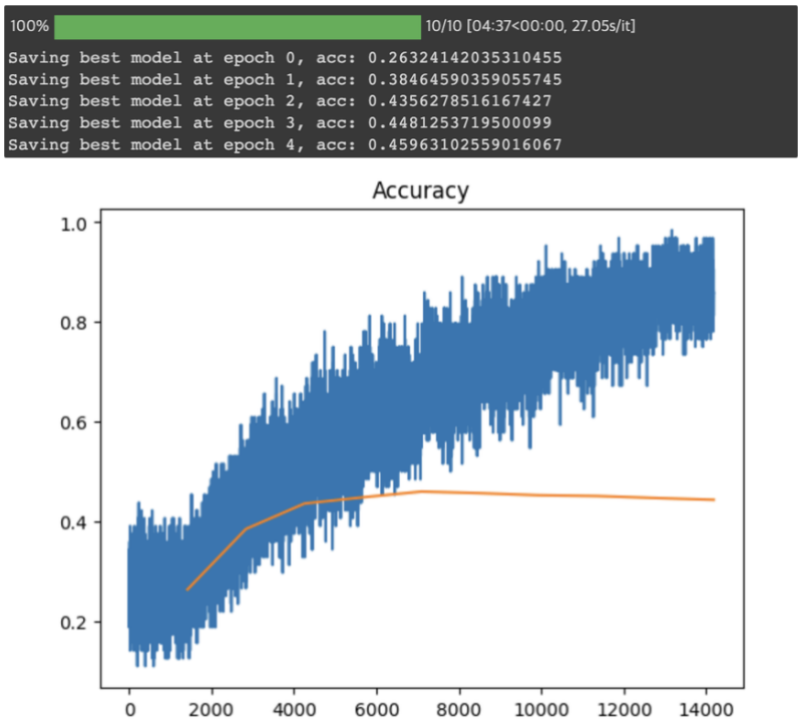
Therefore, after conducting several additional experiments with various values, I determined that maintaining the epoch value and setting the embedding vector value to 100 significantly improves the model's accuracy. Consequently, I ultimately decided to set it to 100.

After configuring the embedding vector values, I proceeded to set the training epoch values. Just

like the embedding vector values, if the epoch value is set excessively high, it may lead to overfitting, where the model performs well only on the training data but poorly on validation and test data. Conversely, if set too low, underfitting may occur, resulting in low performance across all data. Therefore, it was crucial to choose a value that was neither too large nor too small. Thus, initially keeping the existing epoch value, I trained and evaluated the model.

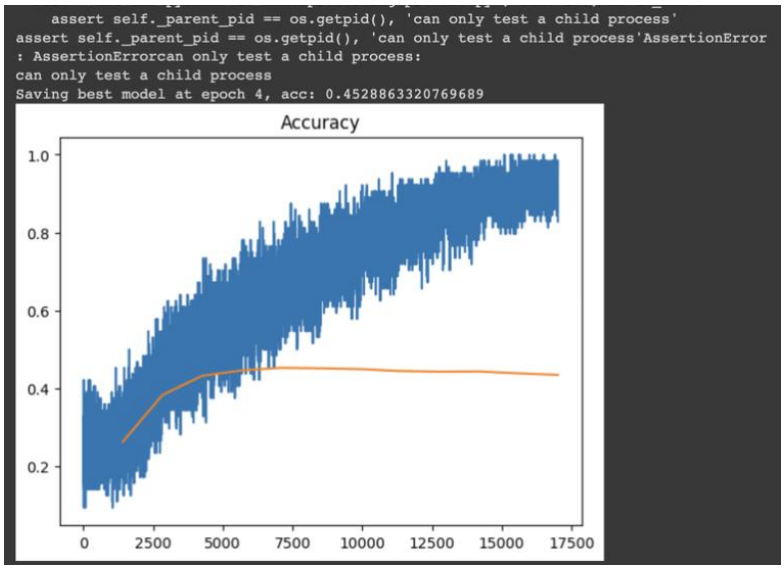


Having conducted training and evaluation with the original epoch value of 7, the maximum accuracy on the validation data turned out to be 45.2%. Consequently, I increased the epoch value to 10 and proceeded with training and evaluation of the model once again.

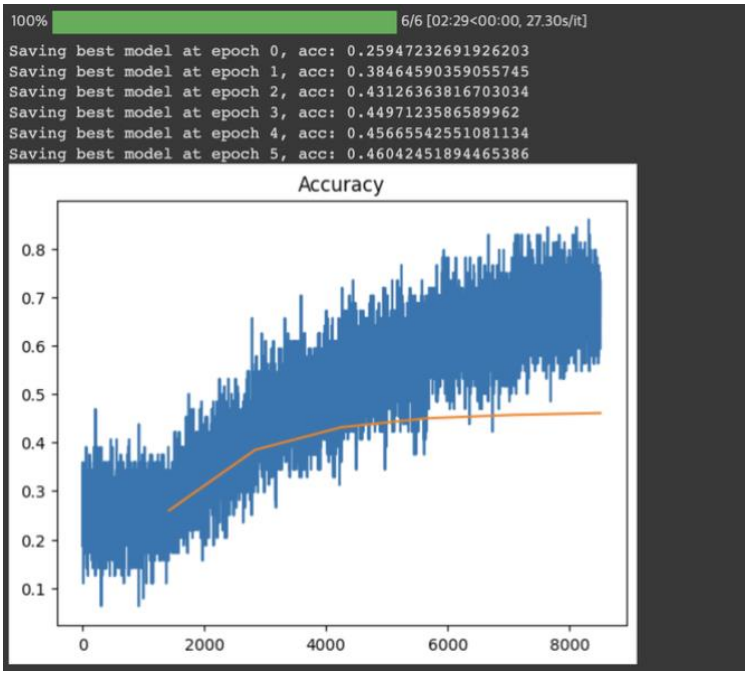


As a result, the maximum accuracy reached 45.9%. Subsequently, I further increased the epoch value

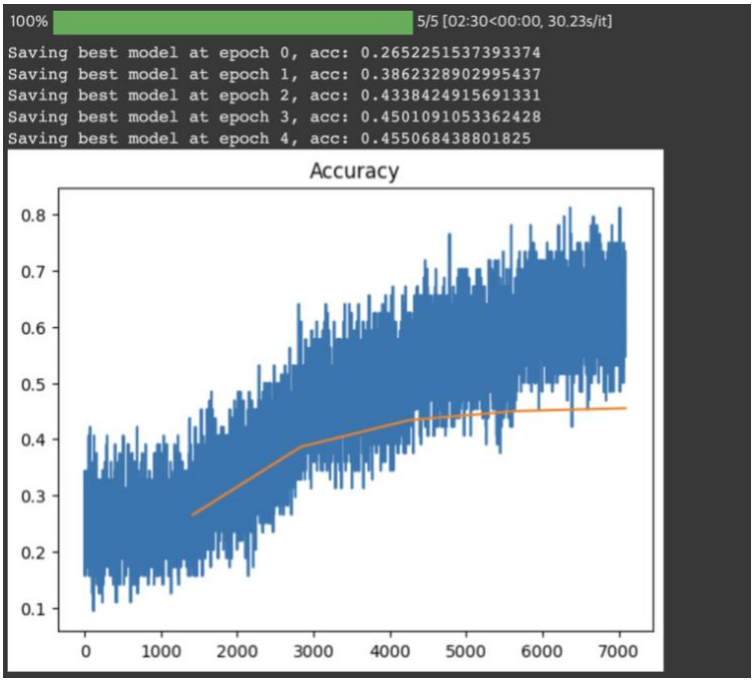
to 12 and conducted another round of model training and evaluation.



With the epoch value set to 12, the model's maximum accuracy was 45.2%. This performance is lower than when the epoch value was set to 10 and, notably, like the accuracy observed when using the default value of 7. Observing this, I considered the possibility of overfitting, as mentioned earlier. Consequently, I conducted an experiment by setting the epoch value even lower, at 6.



As a result, the maximum accuracy reached 46%, marking the highest accuracy among the values tested so far. Encouraged by this, I proceeded with another round of training and evaluation, this time setting the epoch value to an even smaller value of 5.



The result showed a maximum accuracy of 45.5%, leading me to consider the possibility of underfitting, as mentioned earlier. Therefore, I concluded that the epoch value should be between 5 and 12. After experimenting with several values within this range, I found that setting the epoch value to 6 yielded the highest maximum accuracy. Consequently, I decided to finalize the epoch value as 6.

After configuring the embedding vector values and the epoch value as described above, I executed code to calculate bias values for each movie. This allowed me to observe bias values for the top 10 movies with high bias values and the bottom 10 movies with low bias values. However, it's essential to note that I could observe bias values only for these selected movies, not for all movies.

	title	bias
0	Shawshank Redemption, The (1994)	1.035303
1	Forrest Gump (1994)	0.967811
2	Star Wars: Episode IV - A New Hope (1977)	0.808996
3	Raiders of the Lost Ark (Indiana Jones and the...	0.781739
4	Schindler's List (1993)	0.762642
5	Dark Knight, The (2008)	0.761209
6	Fugitive, The (1993)	0.750874
7	Pulp Fiction (1994)	0.727368
8	Usual Suspects, The (1995)	0.705632
9	Goodfellas (1990)	0.690136
...	...	...
9714	Catwoman (2004)	-0.329555
9715	Problem Child (1990)	-0.330743
9716	Problem Child 2 (1991)	-0.335127
9717	Spice World (1997)	-0.339245
9718	Speed 2: Cruise Control (1997)	-0.356902
9719	Superman IV: The Quest for Peace (1987)	-0.359788
9720	Karate Kid, Part III, The (1989)	-0.361408
9721	I Know What You Did Last Summer (1997)	-0.363016
9722	Flintstones in Viva Rock Vegas, The (2000)	-0.374432
9723	Godzilla (1998)	-0.389445

Upon examining the top 10 movies with high bias values, it became evident that these movies are still widely recognized as masterpieces by the public and continue to be popular works. Some of these movies even included those that I personally still enjoy on occasion. In contrast, when reviewing the bottom 10 movies with low bias values, it was apparent that these movies, unlike the top 10 mentioned earlier, are generally less well-known to the public or are infamous for being poorly received.

Analyzing movie-specific bias values as described above revealed that the use of bias values in a movie recommendation system serves the purpose of effectively considering the characteristics of each movie and the individual preferences of users. In other words, by incorporating bias values, the recommendation system can consider the relative value of each movie, providing users with more diverse and personalized recommendations. This approach allows the system to induce high

user satisfaction by considering various user preferences, ultimately enhancing the quality of movie recommendations.

After analyzing the bias values for each movie, I executed code to select a specific movie and output a list of movies similar to the chosen one. This list included the titles, genres, and similarity scores of movies that were deemed similar to the selected movie. The results of the code execution were as follows:

Titles of selected movie_id is: Old Boy (2003)

	title	genre	similarity
0	Hustler, The (1961)	Drama	0.754303
1	Apocalypse Now (1979)	Action|Drama|War	0.743772
2	12 Angry Men (1957)	Drama	0.730833
3	Logan (2017)	Action|Sci-Fi	0.716872
4	Dr. Strangelove or: How I Learned to Stop Worr...	Comedy|War	0.706620
5	Fog of War: Eleven Lessons from the Life of Ro...	Documentary|War	0.705123
6	Godfather, The (1972)	Crime|Drama	0.697009
7	Fight Club (1999)	Action|Crime|Drama|Thriller	0.696775
8	Exit Through the Gift Shop (2010)	Comedy|Documentary	0.696562
9	Grand Budapest Hotel, The (2014)	Comedy|Drama	0.695734
...	...	...	...
9714	Problem Child 2 (1991)	Comedy	-0.568063
9715	Amityville II: The Possession (1982)	Horror	-0.568319
9716	Undercover Blues (1993)	Comedy|Crime	-0.570858
9717	Catwoman (2004)	Action|Crime|Fantasy	-0.580928
9718	Spice World (1997)	Comedy	-0.582956
9719	Wicker Man, The (2006)	Horror|Mystery|Thriller	-0.584779
9720	Ernest Goes to Camp (1987)	Comedy	-0.606591
9721	Jason X (2002)	Horror|Sci-Fi|Thriller	-0.607453
9722	Disaster Movie (2008)	Comedy	-0.612330

Firstly, I analyzed the output results. The movie "Oldboy," in a broad sense, depicts the male protagonist being attacked by an antagonist and seeking revenge. The film captures the relationship between the male and female protagonists, an overall dark atmosphere, and numerous action scenes. Another notable feature is the significant plot twist involving the male protagonist in the latter part of the movie, and it holds the characteristic of being rated as not suitable for youth.

Examining the top 10 movies of similar genres recommended by the system (including rank 0), the movie "Logan," ranked 4th, narrates the story of the mutant male protagonist, Logan, and the female protagonist, Laura, who possesses similar abilities. Together, they engage in battles against mercenaries hired by a scientist conducting mutant experiments, aiming to free Laura and other

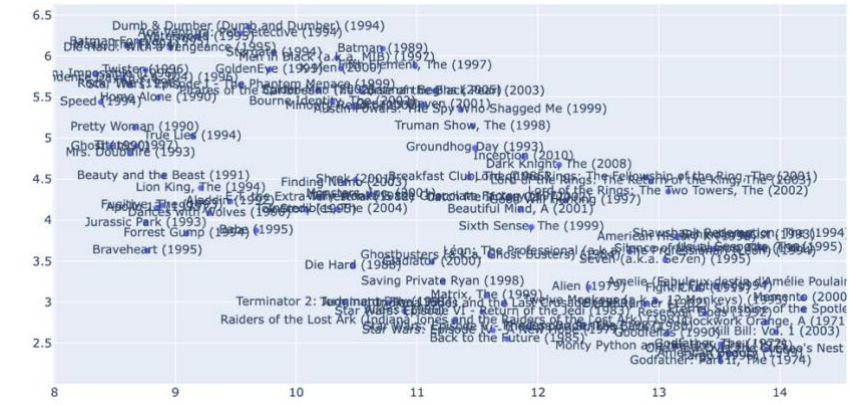
mutant children from the scientist's control. Just like "Oldboy," "Logan" also carries a dark atmosphere and portrays the strong bond between the male protagonist, Logan, and the female protagonist, Laura. Additionally, the movie provides interest to the audience through various action scenes. Like "Oldboy," "Logan" is also rated as not suitable for youth.

Additionally, I explored the movie "Fight Club," which ranked 8th. The film portrays the protagonist, Edward Norton, who leads a mundane life as an insurance company employee. During a business trip, he encounters Tyler Durden, and they inexplicably find a sense of liberation through one-on-one physical combat. This leads to the formation of the "Fight Club," where individuals use fistfights to release societal anxiety and pressure. The movie delves into the story that unfolds after the establishment of the "Fight Club." Like "Oldboy," "Fight Club" explains the relationship between the male protagonist, Edward Norton, and the female protagonist, Marla Singer. It also features numerous action scenes for the audience. Although not rated as not suitable for youth, "Fight Club" shares the characteristic of presenting a significant plot twist involving the male protagonist, like "Oldboy," in the latter part of the movie.

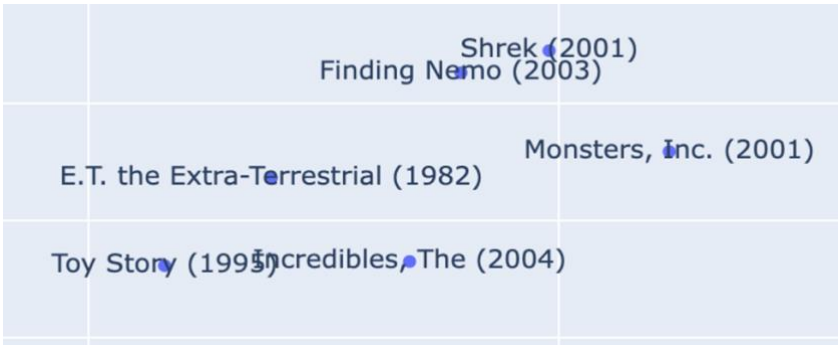
This time, I examined the bottom 10 movies that the system deemed not like the movie "Oldboy." One of the movies in the bottom 10, "Spice World," is a comedy film that portrays the success story and everyday life of the real-life girl group 'Spice Girls.' It is evident that in terms of genre, atmosphere, and plot, "Spice World" does not align with "Oldboy."

As seen above, movies like "Logan" and "Fight Club," which exhibit similarity in overall aspects such as atmosphere and components to the movie "Oldboy," are ranked in the top 10. On the other hand, movies like "Spice World," which do not share such similarities, are ranked in the bottom 10. Based on this, I conclude that the recommendation system effectively presented a list of movies like "Oldboy."

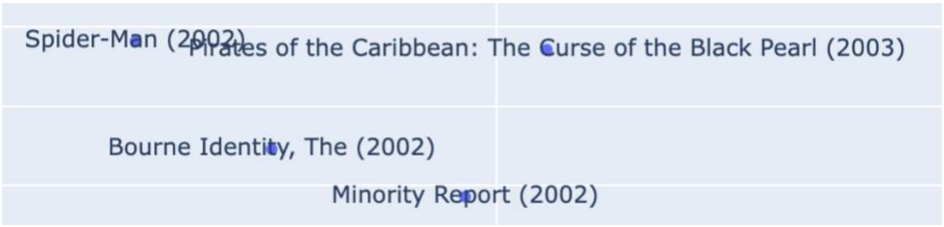
After executing the code to output a list of movies similar to a selected movie and then running the code to visually represent the similarity between movies in the entire movie dataset, the results showed multiple clusters of similar movies. I analyzed several of these clusters, and the output looked like the following:



Firstly, the cluster in the above image caught my attention because all four movies in that cluster were released in 1996. Among these four movies, "Mission: Impossible" and "The Rock" share similarities in having antagonists and showcasing various action scenes. On the other hand, "Independence Day" and "Twister" exhibit a common theme of confrontation, one between humans and extraterrestrial life and the other between humans and the forces of a tornado. This highlights a common thread of conflict between humans and non-humans. Additionally, "Independence Day," which involves action scenes with extraterrestrial beings, is grouped with "Mission: Impossible" and "The Rock," leading to the inclusion of "Twister" due to its perceived similarity to "Independence Day." Consequently, while these four movies may not be entirely similar, they share some commonalities, and the system appears to have effectively grouped them based on these shared characteristics.



Next, I examined the cluster above. Upon looking at the titles, I immediately thought these six movies were very similar. "Shrek," "Monsters, Inc.," "Finding Nemo," "E.T.," "Toy Story," and "The Incredibles" are all animated or fantasy genre films, targeted at families and children. Additionally, they share the common feature of having simple yet emotionally resonant stories that have been loved by almost everyone, regardless of age or gender, both in the past and, arguably, still today.



Finally, I examined the cluster above. The four movies, "Spider-Man," "Pirates of the Caribbean: The Curse of the Black Pearl," "The Bourne Identity," and "Minority Report," primarily revolve around heroism, adventure, espionage, and science fiction, featuring various action scenes. Additionally, they share the common elements of having antagonists and depicting relationships between male and female protagonists. Thus, it is evident that these four movies are similar. Therefore, I believe the model has effectively grouped similar movies. In conclusion, after examining these three clusters, it is evident that the recommendation system model has appropriately grouped similar movies together.

After executing the code to visually represent the similarity between movies in the entire movie dataset and observing the results, it became apparent that the recommendation system model, while not perfect, generally grouped similar movies effectively. An intriguing aspect is that the model discerned the similarity between movies solely based on user rating data. To understand how the model could successfully determine the similarity between movies using only user rating data, I examined the code of the model, conducted research on relevant materials on the internet, and thus, resolved my questions regarding the functioning of the model.

Firstly, the movie recommendation system model operating in this code is fundamentally based on matrix factorization. Matrix factorization decomposes the interaction between users and movies into latent factors, where these latent factors represent the characteristics of users and movies. In this process, the model strives to decompose both the overall structure and local similarities between users and movies. This means that matrix factorization attempts to learn not only overall similarities but also local patterns and similarities, preserving both global structure and local similarities. In other words, it simultaneously learns latent features while maintaining both global structure and local clustering among similar movies. The learned latent features are then utilized to measure similarity, enabling the identification of similar user and movie data. The model learns tendencies where users with similar rating patterns prefer similar movies.

After running and observing the code that visualizes the similarity between various movies, you proceeded to analyze different dimensions within the embedded movie data. By examining the top 10 movies with large values and the bottom 10 movies with small values for each dimension, you aimed to understand what characteristics or features each dimension represents in the movies. Let's delve into the discussion of the 100th dimension as an example.

Three Colors: White (Trzy kolory: Bialy) (1994)	Comedy|Drama	0.600256
Red Dawn (1984)	Action|Drama|War	0.554982
Last Life in the Universe (Ruang rak noi nid m...	Drama|Romance	0.552685
Rabbit-Proof Fence (2002)	Adventure|Drama	0.541035
Disaster Movie (2008)	Comedy	0.533501
Leaving Las Vegas (1995)	Drama|Romance	0.514955
South Park: Bigger, Longer and Uncut (1999)	Animation|Comedy|Musical	0.510430
Little Man (2006)	Comedy	0.504410
Big Kahuna, The (2000)	Comedy|Drama	0.487904
Seven Years in Tibet (1997)	Adventure|Drama|War	0.479757

The above list comprised the top 10 movies with large values in the 100th dimension. Overall, these movies tended to focus on human emotions, ethical dilemmas, or the meaning of life.

My Cousin Vinny (1992)	Comedy	-0.631397
Happy Gilmore (1996)	Comedy	-0.649803
Island, The (2005)	Action|Sci-Fi|Thriller	-0.654523
3:10 to Yuma (2007)	Action|Crime|Drama|Western	-0.671369
X2: X-Men United (2003)	Action|Adventure|Sci-Fi|Thriller	-0.672488
Princess Diaries, The (2001)	Children|Comedy|Romance	-0.674961
Police Academy 4: Citizens on Patrol (1987)	Comedy|Crime	-0.683248
Clueless (1995)	Comedy|Romance	-0.747795
21 Jump Street (2012)	Action|Comedy|Crime	-0.762631
Catch Me If You Can (2002)	Crime|Drama	-0.785232

On the other hand, the bottom 10 movies in the list generally emphasized comedy and entertainment elements, providing humor or tension in their own unique ways across various genres. Therefore, I concluded that the 100th dimension represents a characteristic focusing on human emotions, ethical dilemmas, and the meaning of life.

I encountered a coding problem where I had to complete a function that calculates accuracy after analyzing a movie recommendation system model based on matrix factorization from various aspects. The function to be completed takes two tensors as input: one containing a list of predicted values and the other containing a list of actual target values corresponding to those predictions. The function checks each predicted value, rounds it, and treats it as a correct prediction if the rounded value matches the corresponding target value. Additionally, the function calculates accuracy by dividing the number of correct predictions by the total number of predictions. The code for the given problem was as follows:

```python
def get_accuracy(pred,target):
  '''
  pred (torch.Tensor): Estimated ratings of data samples. Shape of [n,1], where n is number of data samples
  target (torch.Tensor): Ground-truth ratings of data samples. Shaoe of [n,1], where n is number of data samples

  The order of data sample for pred and target is same.

  output (float): Number of correct estimations divided by number of data samples

  Hint: You can get the round value for each element of Tensor by torch.round(atensor)
  '''
  # TODO: Complete this function

  return

'''
You don't have to change the code below
'''

dummy_estimation = torch.Tensor([[2.532], [1.672], [3.741], [4.512], [2.701] ])
dummy_target = torch.Tensor([[2], [1], [4], [5], [2] ])
accuracy = get_accuracy(dummy_estimation, dummy_target)

print(f"The accuracy is {accuracy}")
```

Regarding this, I completed the "get_accuracy" function as follows.

```python
def get_accuracy(pred,target):
  '''
  pred (torch.Tensor): Estimated ratings of data samples. Shape of [n,1], where n is number of data samples
  target (torch.Tensor): Ground-truth ratings of data samples. Shaoe of [n,1], where n is number of data samples

  The order of data sample for pred and target is same.

  output (float): Number of correct estimations divided by number of data samples

  Hint: You can get the round value for each element of Tensor by torch.round(atensor)
  '''
  # TODO: Complete this function

  return

'''
You don't have to change the code below
'''

dummy_estimation = torch.Tensor([[2.532], [1.672], [3.741], [4.512], [2.701] ])
dummy_target = torch.Tensor([[2], [1], [4], [5], [2] ])
accuracy = get_accuracy(dummy_estimation, dummy_target)

print(f"The accuracy is {accuracy}")
```

Firstly, the total number of predictions is, in fact, the same as the total number of predicted values. Since the "pred" tensor, which contains the predicted values, has a row length equal to the total number of predicted values and a column length of 1, I stored the row length of "pred" represented by "pred.shape[0]" in the variable "all_datas" to save the total number of predictions. Next, I extracted the rounded predicted values one by one from the "pred" tensor and stored them in the variable "k". Additionally, I declared a variable "count" to store the number of correct predictions, initializing it to 0 as no correct predictions have been counted yet. Then, through a loop, I compared each rounded predicted value (k) with the corresponding target value extracted from "target". If the two values were equal, I incremented the "count" variable by 1, aiming to increase the count of correct predictions by 1. After evaluating all predictions, I then printed the final accuracy by dividing the fully stored count variable, representing the total number of correct predictions, by the variable "all_datas", which holds the total number of predictions. Consequently, it verified that the function operates correctly.

```
The accuracy is 0.4
```

After solving the accuracy calculation problem, I encountered the second coding problem, which required completing a function that takes embedded user data, movie data, user biases, and movie biases as input. The task is to use this information to predict and output a user's rating for a movie. The provided code for the problem was as follows:

```
def estimates_rating(user_embedding, movie_embedding, user_bias, movie_bias):
  '''
  user_embedding (torch.Tensor): Trained embeddings for a user, in shape of [n_factors]
  user_embedding (torch.Tensor): Trained embeddings for a movie, in shape of [n_factors]
  user_bias (torch.Tensor): Trained bias for a user, in shape of [1]
  movie_bias (torch.Tensor): Trained bias for a user, in shape of [1]

  output (torch.Tensor): Estimated score of the user for the movie
  '''
  # TODO: Complete this function
  # You don't have to consider sigmoid_range for this problem
  # The scaling function will be applied after this function

  return

selected_user_index = 0
selected_movie_index = 0


'''
You don't have to change the code below
'''

selected_user_emb = model.user_embedding.weight[selected_user_index]
selected_movie_emb = model.movie_embedding.weight[selected_movie_index]
selected_user_bias = model.user_bias.weight[selected_user_index]
selected_movie_bias = model.movie_bias.weight[selected_movie_index]

estimated_rating = estimates_rating(selected_user_emb, selected_movie_emb, selected_user_bias, selected_movie_bias)
estimated_rating = model.scaled_sigmoid(estimated_rating).item()

print(f"Estimated rating of user {selected_user_index} for movie {movies.iloc[selected_movie_index]['title']} is {estimated_rating}")
```

Regarding this, I have completed the "estimates_rating" function as follows:

```
def estimates_rating(user_embedding, movie_embedding, user_bias, movie_bias):
  '''
  user_embedding (torch.Tensor): Trained embeddings for a user, in shape of [n_factors]
  user_embedding (torch.Tensor): Trained embeddings for a movie, in shape of [n_factors]
  user_bias (torch.Tensor): Trained bias for a user, in shape of [1]
  movie_bias (torch.Tensor): Trained bias for a user, in shape of [1]

  output (torch.Tensor): Estimated score of the user for the movie
  '''
  # TODO: Complete this function
  # You don't have to consider sigmoid_range for this problem
  # The scaling function will be applied after this function

  dotted_result = torch.dot(user_embedding, movie_embedding)

  dotted_result = dotted_result.reshape([1])

  dotted_result += user_bias + movie_bias # d

  final_result = dotted_result

  return final_result
```

First, I stored the result of the dot product of the embedded user data and movie data in "dotted_result". To enable adding user bias and movie bias to "dotted_result", I reshaped "dotted_result" to have a shape of torch.Size([0]), After that, I added the user bias "user_bias" and movie bias movie bais and stored the result in "dotted_result". Finally, I added this value to "final_result". By outputting "final_result", I completed the "estimates_rating" function, which calculates the user's rating for a movie based on the embedded user data, embedded movie data, user bias, and movie bias. It was confirmed to work correctly.

```
Estimated rating of user 0 for movie Toy Story (1995) is 4.700512409210205
```

Based on the insights gained from solving coding problems related to the analysis of the movie recommendation system, I was able to deepen my understanding of how recommendation systems work and their performance. This experience was particularly valuable in acquiring more detailed knowledge about matrix factorization, a crucial component in recommendation systems. Solving coding problems provided insights into various technical aspects to improve the accuracy of the recommendation system model. Specifically, activities such as adjusting embedding vector values and epochs enhanced my understanding of practical methodologies for improving the performance of deep learning models.

These experiences will serve as important guidelines when implementing and optimizing future recommendation systems. Customized strategies for performance improvement and appropriate hyperparameter tuning can be effectively applied in future projects or practical applications. Therefore, based on the knowledge gained from this project, I anticipate focusing more on the implementation and performance enhancement of recommendation systems in the future, contributing to improving user experiences.