SUMMER TRAINING PROJECT REPORT

(Term June-July 2025)

Hybrid Movie Recommendation System Using Deep Learning with Content-Based and Collaborative Filtering

Submitted by

R. Jagan Mohan Reddy
Registration Number :12404777

Course Code: PETV79

Under the Guidance of

Mahipal Singh Papola Assistant Professor

School of Computer Science and Engineering

CERTIFICATE

This is to certify that R Jagan Mohan Reddy, Registration Number 12404777, has successfully completed the summer course "Machine Learning Made Easy: From Basics to AI Applications" during June-July 2025 at Lovely Professional University. The student developed a project titled " Hybrid Movie Recommendation System Using Deep Learning with Content-Based and Collaborative Filtering" under my supervision.

SIGNATURE OF FACULTY

SIGNATURE OF STUDENT

R. JAGAN MOHAN REDDY

SIGNATURE OF HEAD OF THE DEPARTMENT

ACKNOWLEDGEMENT

I want to give a big thank you to my course instructor, Mr. Mahipal Singh Papola, for helping me through the summer course at Lovely Professional University. His teaching made machine learning easy to understand, and his guidance helped me build my movie recommendation project. I'm super grateful to the School of Computer Science and Engineering for giving me this chance to learn something I love, like movies and anime. My friends and family were always there, cheering me on, which meant a lot. This project was a great experience, and I learned so much about machine learning and how to make something useful.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Course Profile

1.2 Overview of Training Domain

1.3 Objective of the Project

CHAPTER 2: TRAINING OVERVIEW

2.1 Tools and Technologies Used

2.2 Areas Covered During Training

2.3 Weekly Work Summary

CHAPTER 3: PROJECT DETAILS

3.1 Title of the Project

3.2 Problem Definition

3.3 Scope and Objectives

3.4 System Requirements

3.5 Architecture Diagram

3.6 Data Flow Diagram

CHAPTER 4: IMPLEMENTATION

4.1 Tools Used

4.2 Methodology

4.3 Modules

4.4 Screenshots

4.5 Code Snippets

CHAPTER 5: RESULTS AND DISCUSSION

5.1 Output / Report

5.2 Challenges Faced

5.3 Learnings

CHAPTER 6: CONCLUSION

6.1 Summary

CHAPTER 1: INTRODUCTION

1.1 Course Profile

The summer course "Machine Learning Made Easy: From Basics to AI Applications" was offered by Lovely Professional University's School of Computer Science and Engineering. It lasted five weeks, from June to July 2025, and was taught by Mr. Mahipal Singh Papola, an Assistant Professor. The course was all about teaching students like me how to use machine learning, starting from the basics and moving to real projects. I learned how to work with data, build models, and use them to solve problems. Since I love watching movies and anime shows, I chose to build a movie recommendation system for my project. It was a fun way to use what I learned.

The course had lectures, hands-on assignments, and a final project. It was perfect for someone like me who was new to machine learning. We went over things like how to clean data, train models, and even put them online for others to use. By the end, I felt confident using tools like Python and PyTorch to create something useful. The project let me combine my love for movies with new skills, which made the whole experience exciting. I chose this project because I often find it hard to pick new movies or anime to watch, and I wanted a system that could suggest ones I'd like based on my tastes.

1.2 Overview of Training Domain

Machine learning is about teaching computers to learn from data and make smart decisions or predictions. In this course, we focused on using machine learning to solve real problems, like recommending movies or products. My project is a recommendation system, which suggests movies based on what users like or what's similar to other movies. There are two main ways to do this: collaborative filtering, which looks at user ratings, and content-based filtering, which looks at movie details like genres or descriptions. My project combines both to make suggestions that are more accurate and personal.

I got excited about this topic because I watch a lot of movies and anime, and finding new ones can be tough. A good recommendation system makes it easier by suggesting movies that match my tastes or are similar to ones I've enjoyed, like action-packed anime or sci-fi films. The course taught me how to use data and tools like Python and PyTorch to build this kind of system. It was really cool to learn how to make a computer understand what I might like to watch and suggest those things to me.

1.3 Objective of the Project

The main goal of my project was to create a movie recommendation system that does two things:

- Suggest the top N movies for a user based on their past ratings, so I can get personalized suggestions.

- Find the top N movies similar to a movie I like, based on its content, like genres or themes.

Since I enjoy movies and anime, I wanted a system that feels personal and helps me discover new shows or films. The system should be easy to use, with a web page where users can enter their ID or a movie title and get recommendations. It uses machine learning to make smart suggestions, combining what users like with movie details to give the best results possible.

CHAPTER 2: TRAINING OVERVIEW

2.1 Tools and Technologies Used

To build my project, I used several tools and technologies that we learned in the course:

- Python: The main programming language I used to write all the code.

- PyTorch: A tool for building neural networks, which I used for my recommendation model.

- Pandas and NumPy: For working with data, like organizing movie ratings and details.

- Scikit-learn: For splitting data, scaling numbers, and checking how well the model works.

- Flask: To create a web page where users can get movie recommendations.

- Matplotlib: For making graphs to show the model's results, like how close predictions are to actual ratings.

These tools were new to me before the course, but they made it possible to build a complete system from start to finish. Learning them was a big part of the course, and they helped me make my project work.

2.2 Areas Covered During Training

The course taught me a lot of important things that I used in my project:

- Data Preprocessing: Cleaning and preparing data so the computer can use it properly. This was key for handling movie data.

- Supervised Learning: Teaching the computer to predict things, like movie ratings, based on examples.

- Neural Networks: Building models that act like a brain to find patterns in data, which I used for my recommendation system.

- Recommendation Systems: Learning how to suggest items based on what users like or item details, like movie genres.

- Model Evaluation: Checking if the model's predictions are good using numbers like error rates.

- Web Deployment: Putting the model online so users can interact with it through a web page.

These topics gave me the skills to go from raw data to a working recommendation system, which was exciting to learn.

2.3 Weekly Work Summary

Here's what I did each week during the five-week course:

- Week 1: Learned the basics of machine learning, like what it is and how it works. Started using Python libraries like Pandas and NumPy. Looked at the MovieLens dataset to plan my project. This week was about getting comfortable with the tools and understanding the data I'd use for my recommendation system.

- Week 2: Studied supervised learning and neural networks. Practiced building simple models with PyTorch and understood how they learn from data. I started thinking about how to use neural networks for my movie recommendation project.

- Week 3: Focused on recommendation systems. Learned about collaborative filtering, which uses user ratings, and content-based filtering, which uses movie details. Started planning how to combine them to make my system better.

- Week 4: Worked on my project by writing code to clean data, build the hybrid model, and train it. Tested the model to see how well it predicts movie ratings. This was the busiest week because I was coding a lot and fixing problems.

- Week 5: Built a web interface with Flask so users can try the system. Added a graph to show results and wrote this report to explain everything. This week was about making the project user-friendly and wrapping it up.

Each week helped me build on what I learned before, turning ideas into a real, working project.

CHAPTER 3: PROJECT DETAILS

3.1 Title of the Project

Movie Recommendation System Using Hybrid Machine Learning

3.2 Problem Definition

There are so many movies and anime shows out there, it’s hard to find ones you’ll really like. Some recommendation systems only look at what you’ve rated, but they might miss movies that are similar in style or theme. Others only look at movie details but don’t consider your personal tastes. My project solves this by combining both approaches—user ratings and movie details—to give better, more personalized suggestions. It’s designed to help people like me, who love movies and anime, find new things to watch easily. For example, if I like an anime like "Spirited Away," I want the system to suggest similar anime or movies I’d enjoy based on my ratings.

3.3 Scope and Objectives

The project has these main goals:

- Build a system that combines collaborative filtering (based on user ratings) and content-based filtering (based on movie details like genres or descriptions).

- Let users get the top N movies tailored to their tastes by entering their user ID.

- Allow users to find the top N movies similar to a movie they like by entering its title.

- Make the system easy to use with a web page built using Flask.

The system should be accurate and helpful for discovering new movies or anime shows, making it fun and easy to find something to watch.

3.4 System Requirements

To run the project, you need:

- Hardware: A computer with at least 8 GB RAM. A GPU is helpful for faster training but not necessary.

- Software: Python 3.8 or higher, with libraries like PyTorch, Pandas, NumPy, Scikit-learn, Flask, and Matplotlib.

- Dataset: MovieLens dataset, which has files like movies.csv (movie titles and genres), ratings.csv (user ratings), and others with movie details like descriptions.

I built and tested the project on my computer, and it works well with these requirements. The dataset was key because it had a lot of information about movies and user ratings, which I needed for my system.

3.5 Architecture Diagram

The system has a few main parts:

- Data Preprocessing: Loads MovieLens data, cleans it, and gets it ready for the model by turning things like user IDs and movie details into numbers.

- Hybrid Model: A neural network that uses user ratings and movie details to predict what users might like.

- Web Interface: A Flask app where users can enter their ID or a movie title to get recommendations.

3.6 Data Flow Diagram

The data moves through the system like this:

1. Load MovieLens data files (movies.csv, ratings.csv, etc.).

2. Clean and prepare data (encode user and movie IDs, scale ratings, extract movie features using TF-IDF).

3. Train the hybrid model using user ratings and movie content.

4. Use the model to predict ratings and suggest movies through the Flask app.

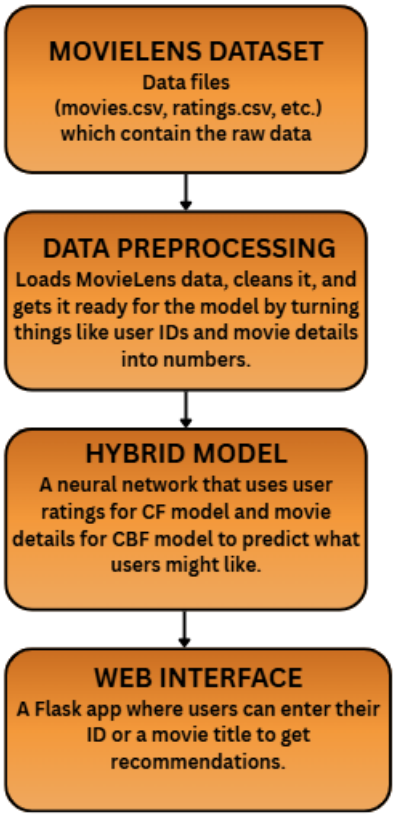

Figure 1: Data Flow Diagram for Recommendation Process

CHAPTER 4: IMPLEMENTATION

4.1 Tools Used

I used the tools listed in Chapter 2: Python, PyTorch, Pandas, NumPy, Scikit-learn, Flask, and Matplotlib. These helped me handle data, build the model, and create a web interface for users to try the system. Each tool had a specific job, like Pandas for data handling or Flask for the web page, and together they made the project possible.

4.2 Methodology

Here's how I built the project step by step:

1. Data Preprocessing: Loaded MovieLens data, filled in missing information (like empty movie descriptions), and turned user and movie IDs into numbers. Used TF-IDF to turn movie descriptions into numbers the model could use.

2. Model Development: Built a neural network with PyTorch that combines user ratings (collaborative filtering) and movie details (content-based filtering) to predict ratings.

3. Training: Trained the model for 10 rounds (called epochs) to make its predictions as close as possible to actual ratings, using a loss function called MSE.

4. Evaluation: Checked how good the model was using numbers like Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE).

5. Deployment: Added a web interface with Flask so users can enter their ID or a movie title and get recommendations.

This process took a lot of work, but it was exciting to see it come together.

4.3 Modules

The project has these main parts:

- Data Loading Module: Reads and prepares MovieLens data, like cleaning it and turning it into numbers.

- Model Module: The neural network that predicts ratings based on users and movies.

- Recommendation Module: Suggests personalized movies or ones similar to a given title.

- Web Interface Module: The Flask app where users can interact with the system.

Each module does a specific job, and they work together to make the system complete.

4.4 Screenshots

The Flask app has a simple web page where users can enter their ID or a movie title to get recommendations. It shows a form for input and lists the suggested movies with their predicted ratings or similarity scores.

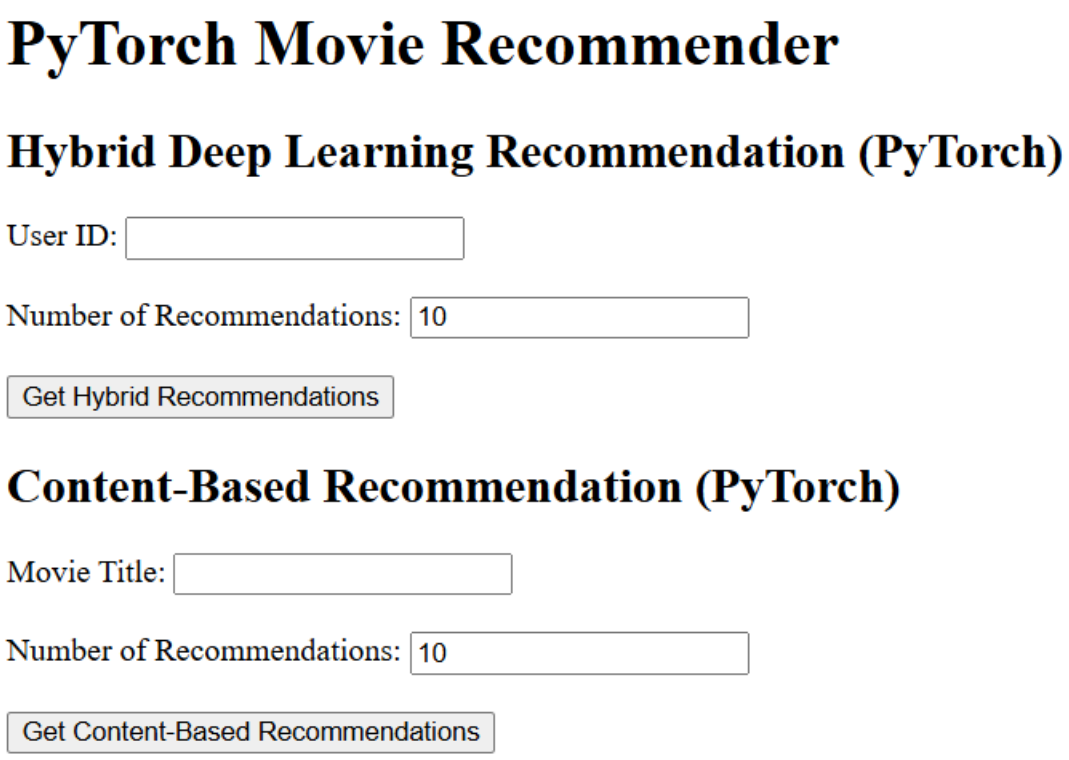

Figure 2: Flask Web Interface for Movie Recommendations

4.5 Code Snippets

Here are some key parts of my code. First, the model definition:

[Code Snippet 1: Hybrid Recommender Model]

```
class HybridRecommender(nn.Module):
    def __init__(self, n_users, n_movies, cf_embedding_size, cbf_feature_size, cbf_embedding_size):
        super().__init__()
        self.user_embedding = nn.Embedding(n_users, cf_embedding_size)
        self.movie_embedding_cf = nn.Embedding(n_movies, cf_embedding_size)
```

```
        self.cbf_fc1 = nn.Linear(cbf_feature_size, 256)
        self.cbf_dropout = nn.Dropout(0.3)
        self.cbf_embedding_output = nn.Linear(256, cbf_embedding_size)
        self.fc1 = nn.Linear(cf_embedding_size * 2 + cbf_embedding_size, 128)
        self.dropout1 = nn.Dropout(0.3)
        self.fc2 = nn.Linear(128, 64)
        self.output_layer = nn.Linear(64, 1)
```

Next, the function to recommend movies for a user:

[Code Snippet 2: User Recommendation Function]

```
def recommend_movies_hybrid_pytorch(user_id, n_recommendations, loaded_merged_ratings_df):
    try:
        user_encoded = loaded_user_enc_pytorch.transform([user_id])[0]
        rated_movie_ids = loaded_merged_ratings_df[loaded_merged_ratings_df['userId'] == user_id]['movieId'].unique()
        all_movie_ids = loaded_movies_df_pytorch['movieId'].unique()
        unrated_movie_ids = np.setdiff1d(all_movie_ids, rated_movie_ids)
        unrated_movie_ids_encoded = [loaded_movie_enc_pytorch.transform([movie_id])[0] for movie_id in unrated_movie_ids if movie_id in loaded_movie_enc_pytorch.classes_]
        user_ids_tensor = torch.tensor([user_encoded] * len(unrated_movie_ids_encoded), dtype=torch.long).to(device)
        movie_ids_tensor = torch.tensor(unrated_movie_ids_encoded, dtype=torch.long).to(device)
        with torch.no_grad():
            predicted_scaled_ratings_tensor = loaded_model_pytorch(user_ids_tensor, movie_ids_tensor, loaded_all_movie_tfidf_tensor_pytorch, loaded_encoded_movieid_to_feature_index)
        predicted_ratings = loaded_scaler_pytorch.inverse_transform(predicted_scaled_ratings.reshape(-1, 1)).flatten()
        predictions_df = pd.DataFrame({'movie_encoded': unrated_movie_ids_encoded, 'predicted_rating': predicted_ratings})
        predictions_df['movieId'] = predictions_df['movie_encoded'].map({encoded_id: original_id for original_id, encoded_id in zip(loaded_movie_enc_pytorch.classes_, range(len(loaded_movie_enc_pytorch.classes_)))})
```

```
        recommendations_df = predictions_df.sort_values(by='predicted_rating', ascending=False)
        recommended_movies_info = pd.merge(recommendations_df.head(n_recommendations), loaded_movies_df_pytorch[['movieId', 'title']], on='movieId', how='left')
        return recommended_movies_info['title'].tolist(), recommended_movies_info['predicted_rating'].tolist()
    except Exception as e:
        print(f"An error occurred: {e}")
        return [], []
```

Here's the function for content-based recommendations:

[Code Snippet 3: Content-Based Recommendation Function]

```
def recommend_similar_movies_cbf_pytorch(movie_title, n_recommendations=10):
    try:
        if movie_title not in loaded_movies_df_pytorch['title'].values:
            print(f"Movie '{movie_title}' not found in the dataset.")
            return [], []
        movie_id = loaded_movies_df_pytorch[loaded_movies_df_pytorch['title'] == movie_title]['movieId'].iloc[0]
        movie_index = loaded_movie_features_df_pytorch[loaded_movie_features_df_pytorch['movieId'] == movie_id].index[0]
        loaded_tfidf_tensor_for_sim = loaded_all_movie_tfidf_tensor_pytorch
        target_movie_vector = loaded_tfidf_tensor_for_sim[movie_index].unsqueeze(0)
        loaded_all_movie_tfidf_matrix_np = loaded_tfidf_tensor_for_sim.cpu().numpy()
        cosine_sim = cosine_similarity(loaded_all_movie_tfidf_matrix_np[movie_index].reshape(1, -1), loaded_all_movie_tfidf_matrix_np).flatten()
        sim_scores = list(enumerate(cosine_sim))
        sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
        sim_scores = sim_scores[1:n_recommendations+1]
        movie_indices = [i[0] for i in sim_scores]
        similarity_scores = [i[1] for i in sim_scores]
```

```
        recommended_movie_titles =
loaded_movie_features_df_pytorch['title'].iloc[movie_indices].tolist()
        return recommended_movie_titles, similarity_scores
    except Exception as e:
        print(f"An error occurred: {e}")
        return [], []
```

Finally, the Flask app setup:

[Code Snippet 4: Flask App Setup]

```
app = Flask(__name__)
port = 5002
HTML_TEMPLATE_PYTORCH = """
<!doctype html>
<html>
<head><title>PyTorch Movie Recommender</title></head>
<body>
    <h1>PyTorch Movie Recommender</h1>
    <h2>Hybrid Deep Learning Recommendation (PyTorch)</h2>
    <form action="/recommend_hybrid_pytorch" method="post">
        User ID: <input type="text" name="user_id"><br><br>
        Number of Recommendations: <input type="number" name="n_recommendations"
value="10"><br><br>
        <input type="submit" value="Get Hybrid Recommendations">
    </form>
    <h2>Content-Based Recommendation (PyTorch)</h2>
    <form action="/recommend_cbf_pytorch" method="post">
        Movie Title: <input type="text" name="movie_title"><br><br>
        Number of Recommendations: <input type="number" name="n_recommendations"
value="10"><br><br>
        <input type="submit" value="Get Content-Based Recommendations">
    </form>
```

```
    <div id="results"></div>
</body>
</html>
"""
@app.route('/')
def index_pytorch():
    return render_template_string(HTML_TEMPLATE_PYTORCH)
```

CHAPTER 5: RESULTS AND DISCUSSION

5.1 Output / Report

The system works great and does two things:

- Personalized Recommendations: Suggests the top N movies for a user based on their ratings. For example, if I enter my user ID, it gives me movies I might enjoy, like anime or sci-fi films.

Fig 3: Hybrid movie recommender using the CF model

Hybrid Recommendations (PyTorch) for User 25:

Predicted Ratings:

- 1. Enter the Void (2009) (Predicted Rating: 5.81)
- 2. Harlan County U.S.A. (1976) (Predicted Rating: 5.78)
- 3. The Jinx: The Life and Deaths of Robert Durst (2015) (Predicted Rating: 5.72)
- 4. Cheburashka (1971) (Predicted Rating: 5.67)
- 5. There Once Was a Dog (1982) (Predicted Rating: 5.63)
- 6. Mr. Skeffington (1944) (Predicted Rating: 5.63)
- 7. Going Places (Valseuses, Les) (1974) (Predicted Rating: 5.61)
- 8. Watermark (2014) (Predicted Rating: 5.60)
- 9. Mother (Madeo) (2009) (Predicted Rating: 5.58)
- 10. Best of Youth, The (La meglio gioventù) (2003) (Predicted Rating: 5.58)

Fig 4: Top 10 movies based on the individual user id

- Content-Based Recommendations: Finds movies similar to one I pick, like an anime movie, by looking at genres or descriptions.

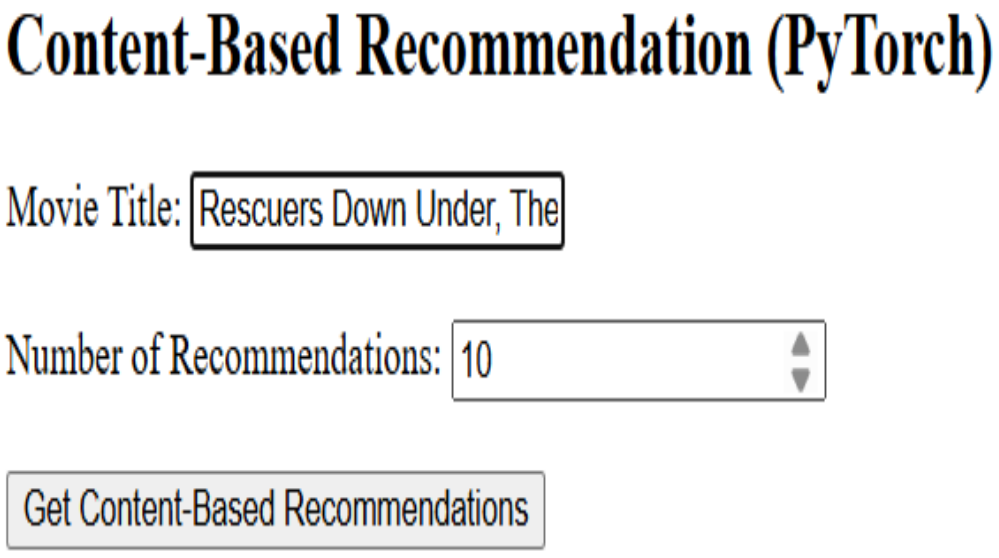

Fig 5: Content-Based Recommendation system using PyTorch

Content-Based Recommendations (PyTorch, similar to 'Rescuers Down Under, The (1990)'):

- 1. Rescuers Down Under, The (1990) (Similarity Score: 1.00)
- 2. Dinosaur (2000) (Similarity Score: 1.00)
- 3. Digimon: The Movie (2000) (Similarity Score: 1.00)
- 4. Return to Never Land (2002) (Similarity Score: 1.00)
- 5. Brother Bear (2003) (Similarity Score: 1.00)
- 6. Phineas and Ferb the Movie: Across the 2nd Dimension (2011) (Similarity Score: 1.00)
- 7. Pocahontas II: Journey to a New World (1998) (Similarity Score: 1.00)
- 8. Junior and Karlson (1968) (Similarity Score: 1.00)
- 9. Adventures of Mowgli: The Kidnapping (1968) (Similarity Score: 1.00)
- 10. Up (2009) (Similarity Score: 0.96)

Fig 6: Recommending the top 10 movies using the movie name

Here's a graph showing how close the model's predictions are to actual ratings:

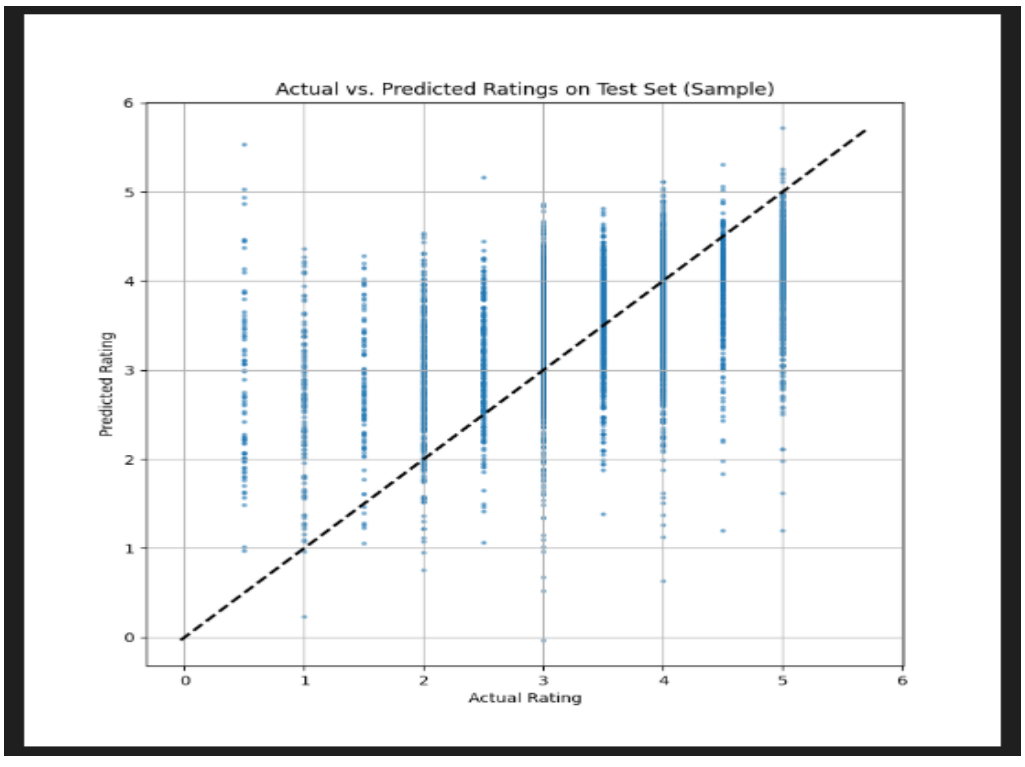

Fig 7: Shows actual ratings vs. predicted ratings from the test set

5.2 Challenges Faced

Building the project had some tough moments:

- Data Issues: Some movies in the MovieLens dataset were missing details, like descriptions. I had to fill these in carefully so the model wouldn't get confused. This took a lot of time to get right.

- Model Balance: Making the collaborative and content-based parts work together was hard. I had to adjust the model to use both parts effectively, which meant trying different settings and testing a lot.

- Slow Training: Training the model on my computer took a long time because I didn't have a GPU. I had to test small parts first to save time and make sure it worked.

These challenges taught me how to solve problems and be patient when things don't work right away.

5.3 Learnings

This project taught me a lot:

- How to build recommendation systems using machine learning, which is great for suggesting movies or anime.

- How to use PyTorch to create neural networks that predict things like movie ratings.

- How to make a web app with Flask so others can use my system easily.

- How to check if a model is good using numbers like MSE, MAE, and RMSE.

- How to handle data problems and finish a project from start to end.

It was exciting to build something that helps me find new movies and anime to watch, and I learned a ton about machine learning along the way.

CHAPTER 6: CONCLUSION

6.1 Summary

The summer course "Machine Learning Made Easy: From Basics to AI Applications" at Lovely Professional University was awesome. I learned how to use machine learning to solve real problems, and I built a movie recommendation system that suggests movies based on what users like or what's similar to other movies. The course taught me how to use Python, PyTorch, and other tools to make this happen. I'm proud of my project because it's useful for people like me who love movies and anime. The Flask web interface makes it easy to use, and the model gives good suggestions. This experience has made me more confident in machine learning, and I'm excited to work on more projects in the future. I feel like I can take what I learned and apply it to other fun ideas, like recommending anime series or even games.