Mehrdad Baradaran – Katayoun Kobraei

99222020 - 99222084

Final Project

Machine Learning Course 2023

The Movie Dataset

Recommender System

Outline

Abstract:

In this project we implemented a movie recommendation system using the collaborative filtering technique. It takes user input of three movies that they have watched and liked, along with their ratings. Based on this input, it generates a list of recommended movies similar to the user's preferences.

The code uses a dataset of movies that includes information such as movie titles, genres, production companies, cast, crew, and more. It preprocesses this data, including feature engineering and vectorization, to create a representation for each movie.

To recommend similar movies, the code calculates pairwise distances between the input movies and all other movies in the dataset using cosine similarity. It then ranks the movies based on similarity and selects the top recommendations.

The code also includes an evaluation function that measures the precision, recall, and F1 score of the recommendations using a test set. This allows for assessing the performance of the recommendation system.

The front-end part of the code is implemented using Streamlit, a Python library for building web applications. It provides an interactive interface where users can input their movie preferences and receive recommendations. The recommended movies are displayed with their details, including posters, genres, ratings, and more.

Introduction:

This is the process of developing and deploying our movie recommendation system with a user-friendly interface using Streamlit and deploying it to HuggingFace Spaces:

Step 1: Data Preparation

We have the movie metadata in CSV files (movies_metadata.csv, credits.csv, keywords.csv) so we added them. We preprocess the data to handle missing values, convert data types, and extract relevant features. This includes merging datasets, handling null values, and transforming textual data and generating additional features.

Step 2: Similarity Calculation and Recommendation Generation

We implemented a method for calculating similarity between movies. We used cosine similarity to measure the similarity between movie features. After that, we defined a function that takes a movie as input and returns a list of similar movies based on the calculated similarity.

Step 3: User-Friendly Interface with Streamlit

We created a user-friendly interface using Streamlit. Streamlit allows us to create interactive web applications with Python. We designed the interface to prompt the user to input their preferred movies and ratings. Then we display recommendations to the user, along with additional movie information such as title, release year, poster image, and other relevant details.

Step 4: Iterative Improvement and Evaluation

We collected user feedback and ratings to continuously improve the recommendation system. Then we used this feedback to refine the recommendation algorithm and make it more accurate. We evaluated the performance of our recommendation system using appropriate metrics such as precision, recall, or F1 score. This helped us assess the effectiveness of our system and make further improvements.

Step 5: Deployment to HuggingFace Spaces

We used the HuggingFace Spaces to deploy our Streamlit application to it. This involved configuring our environment, creating a Dockerfile, and pushing code to a Spaces repository. As it is deployed, our recommendation system is accessible through a web interface provided by HuggingFace Spaces.

- **Introducing Dataset**

What is The Movie Dataset ?

In this dataset files contain metadata for all 45,000 movies listed in the Full MovieLens Dataset. The dataset consists of movies released on or before July 2017. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages.

This dataset also has files containing 26 million ratings from 270,000 users for all 45,000 movies. Ratings are on a scale of 1-5 and have been obtained from the official GroupLens website.

This dataset consists of the following files:

movies_metadata.csv: The main Movies Metadata file. Contains information on 45,000 movies featured in the Full MovieLens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.

keywords.csv: Contains the movie plot keywords for our MovieLens movies. Available in the form of a stringified JSON Object.

credits.csv: Consists of Cast and Crew Information for all our movies. Available in the form of a stringified JSON Object.

links.csv: The file that contains the TMDB and IMDB IDs of all the movies featured in the Full MovieLens dataset.

links_small.csv: Contains the TMDB and IMDB IDs of a small subset of 9,000 movies of the Full Dataset.

ratings_small.csv: The subset of 100,000 ratings from 700 users on 9,000 movies.

This dataset is an ensemble of data collected from TMDB and GroupLens.

The Movie Details, Credits and Keywords have been collected from the TMDB Open API. This product uses the TMDb API but is not endorsed or certified by TMDb. Their API also provides access to data on many additional movies, actors and actresses, crew members, and TV shows. You can try it for yourself here.

The Movie Links and Ratings have been obtained from the Official GroupLens website.

- **Data Preprocessing**

We performed several preprocessing steps on movie data before using it for our recommendation system. Here are the preprocessing steps:

1. Reading in data: The code reads in three CSV files, namely "movies_metadata.csv", "credits.csv", and "keywords.csv", using the pandas read_csv() method.
2. Data slicing: The code slices the movies_meta dataframe to keep only the first 10,000 rows of data. It is a common technique to work with a smaller subset of data as it reduces the time and computational resources required for analysis. Also we could use other techniques in data sampling such as cluster sampling or stratified sampling.
3. Handling missing values: The code identifies missing values in the production_companies column of the movies_meta dataframe and removes the corresponding rows using the dropna() method with the subset parameter. It also fills missing values in the status, original_language, and btc_name columns with empty strings using the fillna() method.
4. Data type conversion: The code converts the id column of the movies_meta dataframe from a string type to an integer type using the astype() method.
5. Merging dataframes: The code merges the keywords and credits dataframes with the movies_meta dataframe using the merge() method to use just movies_meta later.
6. Data cleaning: The code removes duplicate rows from the movies_meta dataframe using the drop_duplicates() method.
7. Text preprocessing: The code defines two functions, btc_function() and get_values(), to preprocess text data in the belongs_to_collection, genres, production_companies, production_countries, spoken_languages, keywords, cast, and crew columns of the movies_meta dataframe. The first function extracts the name of the collection from the JSON format, while the second function extracts the name of each item in the list format and converts it into a string.
8. Vectorization: The code vectorizes the text data in the status, original_language, genres, production_companies, production_countries, spoken_languages, keywords, cast, crew, and btc_name columns of the movies_meta dataframe using the CountVectorizer() method from scikit-learn. The vectorization is performed separately for each column, and the resulting sparse matrices are concatenated into a single dataframe.
9. Feature engineering: The code creates several new features, such as release_year, vote_average_group, popularity_group, runtime_group, budget_group, revenue_group, vote_count_group, release_year_group, and title_new, based on the existing features in

the movies_meta dataframe. These features are created using pandas methods such as apply(), fillna(), astype(), qcut(), and concat().

10. Data normalization: The code normalizes the video and is_homepage columns of the movies_meta dataframe by converting their boolean values to integers using the astype() method.
11. Garbage collection: The code performs garbage collection using the gc.collect() method to free up memory space.

• Explanation of the model

Feature Engineering part: Two vectorization functions, vector_values, are defined to convert categorical columns into binary feature vectors. The vector_values function takes a DataFrame (df), a list of columns (columns), and a minimum document frequency value (min_df_value). Inside the function, a CountVectorizer is initialized with the specified min_df value. Each column in the columns list is processed using the CountVectorizer to convert it into a binary feature matrix. The resulting matrices are concatenated and returned as a new DataFrame.

Feature Selection part: The vector_values function is applied to the movies_meta DataFrame to create two additional DataFrames, movies_meta_addon_1 and movies_meta_addon_2. The 'belongs_to_collection' column is dropped from the movies_meta DataFrame, along with several other columns specified in the col list. The 'video' and 'is_homepage' columns are converted to Boolean values and then to integers.

Additional Data Processing: A custom function get_year is defined to extract the year from the 'release_date' column. Several columns ('popularity', 'budget', 'vote_average', 'runtime', 'revenue', 'vote_count') are converted to appropriate data types (float) for further analysis. The numerical columns 'vote_average', 'popularity', 'runtime', 'budget', 'revenue', 'vote_count', and 'release_year' are grouped into bins using the pd.qcut function. The 'release_year' column is filled with empty strings, converted to float, and then grouped into bins. A new column 'title_new' is created by concatenating the 'title' and 'release_date' columns.

One-Hot Encoding: The categorical columns ('vote_average_group', 'popularity_group', 'runtime_group', 'budget_group', 'revenue_group', 'vote_count_group', 'release_year_group') in the movies_meta DataFrame are one-hot encoded using the pd.get_dummies function. The resulting one-hot encoded DataFrames are stored in movies_meta_addon_3.

Final Data Preparation: All the generated DataFrames (movies_meta_addon_1, movies_meta_addon_2, movies_meta_addon_3) and selected columns from the movies_meta DataFrame ('video', 'is_homepage') are concatenated along the columns axis to create the final training dataset, movies_meta_train. The index of movies_meta_train is set to the 'title_new' column.

Similarity Calculation: The get_similar_movies function is defined to find similar movies given a movie title. It calculates the cosine similarity between the selected movie's feature vector and all other movies' feature vectors using the pairwise_distances function .The results are returned as a DataFrame sorted by similarity score.

The model creates a movie recommendation system by leveraging movie metadata and features. It uses similarity calculations based on cosine similarity to suggest movies that are similar to a given movie. The recommendation is made by comparing the feature vectors of movies and finding the most similar ones based on their attributes.

- ## Validation Part And Results

In this part we evaluates the performance of the movie recommendation system by comparing the recommendations generated by the system with a given test set of user-movie interactions.

The evaluate_recommendations function takes three input arguments: test_set, a dictionary of user-movie interactions where each key represents a user and the corresponding value is a list of movies that the user has interacted with; get_similar_movies_func, a function that generates movie recommendations for a given user; and num_rec, the number of recommended movies to evaluate.

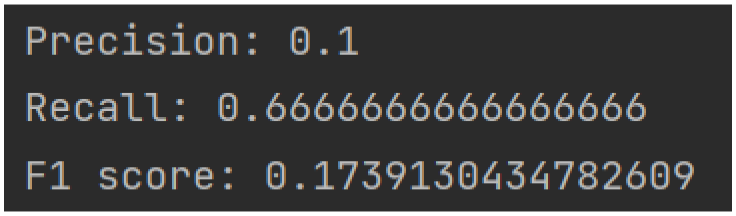
Within the function, the true positives, false positives, and false negatives are calculated for each user in the test set. The true positives are the number of recommended movies that the user has actually interacted with (i.e., the recommended movie is in the user's list of relevant items). The false positives are the number of recommended movies that the user has not interacted with, and the false negatives are the number of relevant items that were not recommended to the user.

The function then computes the precision, recall, and F1 score of the recommendation system based on these values. The precision is the ratio of true positives to the total number of recommended items. The recall is the ratio of true positives to the total number of relevant items. The F1 score is the harmonic mean of precision and recall.

Finally, the function returns the precision, recall, and F1 score of the recommendation system.

In the provided example, the test_set is a dictionary of user-movie interactions, and the evaluate_recommendations function is called with the get_similar_movies function, which generates movie recommendations for a given user. The precision, recall, and F1 score of the recommendation system are then printed to the console.

For our model we reach this results:



```
Precision: 0.1
Recall: 0.6666666666666666
F1 score: 0.1739130434782609
```

As we know that our model is simple these results are pretty good. The precision of 0.1 means that out of all the recommended movies, only 10% were actually relevant to the user. The recall of 0.6666666666666666 means that out of all the relevant movies, only 66.67% were recommended by the system. The F1 score is the harmonic mean of precision and recall, and it provides a single value that balances both precision and recall. In this case, the F1 score is 0.1739130434782609, which is relatively low, indicating that the recommendation system is not performing well.

However, it is important to note that this evaluation is based on a small test set with only a few users and movies. The performance of the recommendation system may improve with a larger and more diverse dataset.

- **Web-Application Using Streamlit Library**

The first function in this part is get_base64_of_bin_file that takes a binary file, reads its content, encodes it as a base64 string, and returns the decoded string. This function is cached using the @st.cache_data() decorator to optimize the performance of the app by avoiding redundant function calls.

The add_bg_from_url() function sets the background image of the web app using a URL. The st.markdown() function is used to insert HTML and CSS code to style the app. The st.set_page_config() function sets the title, icon, and theme of the web app. The select_list variable is a list of movie titles that are displayed in a dropdown menu using the st.selectbox() function. The user is prompted to enter three movies they have watched and rated using the st.slider() function.

After that we defined a button style using CSS and displays it using the st.markdown() function. The button is used to trigger the recommendation algorithm once the user inputs their movie choices. Then we retrieved the input movie names from the user and used them to get similar movies using the get_similar_movies() function. The number of movies recommended for each input movie is determined based on the user's rating using a formula. The recommended movies are stored in a recommendation_list.

The code then loops through the recommendation_list and retrieves the movie metadata, including the movie title, release year, and poster image, using the The Movie Database (TMDb) API. The poster image is displayed using the st.image() function.

In the next step the code appears to be responsible for displaying additional information about a selected movie using the st.expander() function in Streamlit. The code retrieves various metadata about the movie from the TMDb API and displays it in a formatted manner using HTML and CSS. The metadata includes the movie's runtime, IMDb rating, popularity, genres, release year, and vote count. It also displays the movie's tagline and overview. The code extracts the top five actors and their corresponding characters from the movie's cast list and displays them using the actors_line variable. It also extracts the director's name from the crew list and displays it using the director variable. The code replaces spaces with the HTML entity for non-breaking space (&nbsp;) to ensure that the formatting is preserved. Overall, the code provides a rich and informative user interface for the movie recommendation system by displaying various movie details in a formatted and visually appealing manner.

• Deployment on HuggingFace Space

In this part we explain how we deployed our model into HuggingFace.

First we created a HuggingFace account by signing up on their website. Click on "Spaces" on the top navigation bar, and then click "Create new space". Then we filled in the form with our space name and select "Streamlit" as the SDK. Once your space was created, we will see a command-line displayed on the page. Then we copied this command-line and ran it in our terminal. This will create a local copy of our space on our own machine. Then we moved our app files (including the code, any required dependencies, and the trained model) into this local copy of our space.

After that in our terminal, we navigated to the directory of our local copy of the space, and ran the following commands:

Copy

```
git add .

git commit -m "Initial commit"

git push
```

This pushed our app to Hugging Face and deploy it.

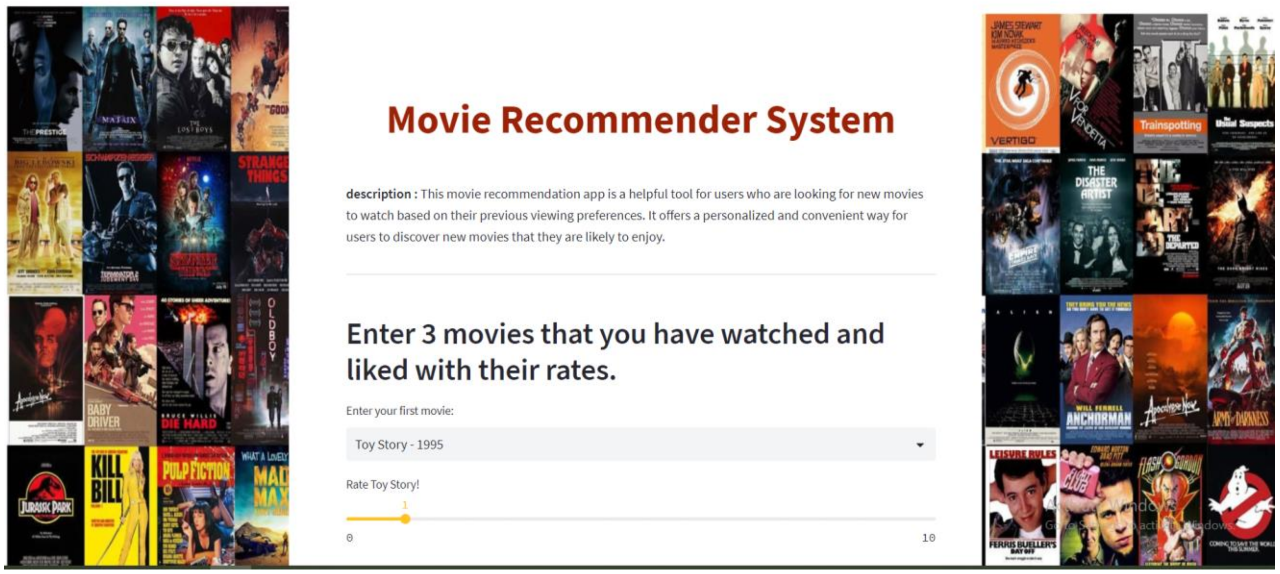
Once the operation it's done, we can access our app by going to our HuggingFace space page and clicking on the "Open app" button next to see the app.

This is the link of our app in HuggingFace:

https://huggingface.co/spaces/Mehrdadbn/Movie-recommender-system

Extra parts that we implemented are:

- In these method we used collaborative filtering not the regular methods such as clustering.
- We tried to make a more attractive interface and enhance the user experience with recommended movies, you have the option to utilize the following APIs that provide additional information about movies, including movie posters.



Movie Recommender System

description : This movie recommendation app is a helpful tool for users who are looking for new movies to watch based on their previous viewing preferences. It offers a personalized and convenient way for users to discover new movies that they are likely to enjoy.

Enter 3 movies that you have watched and liked with their rates.

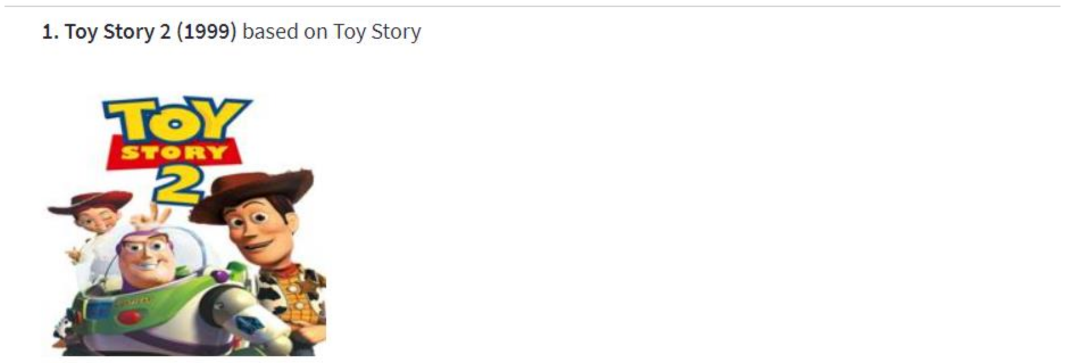
Enter your first movie:

Toy Story - 1995

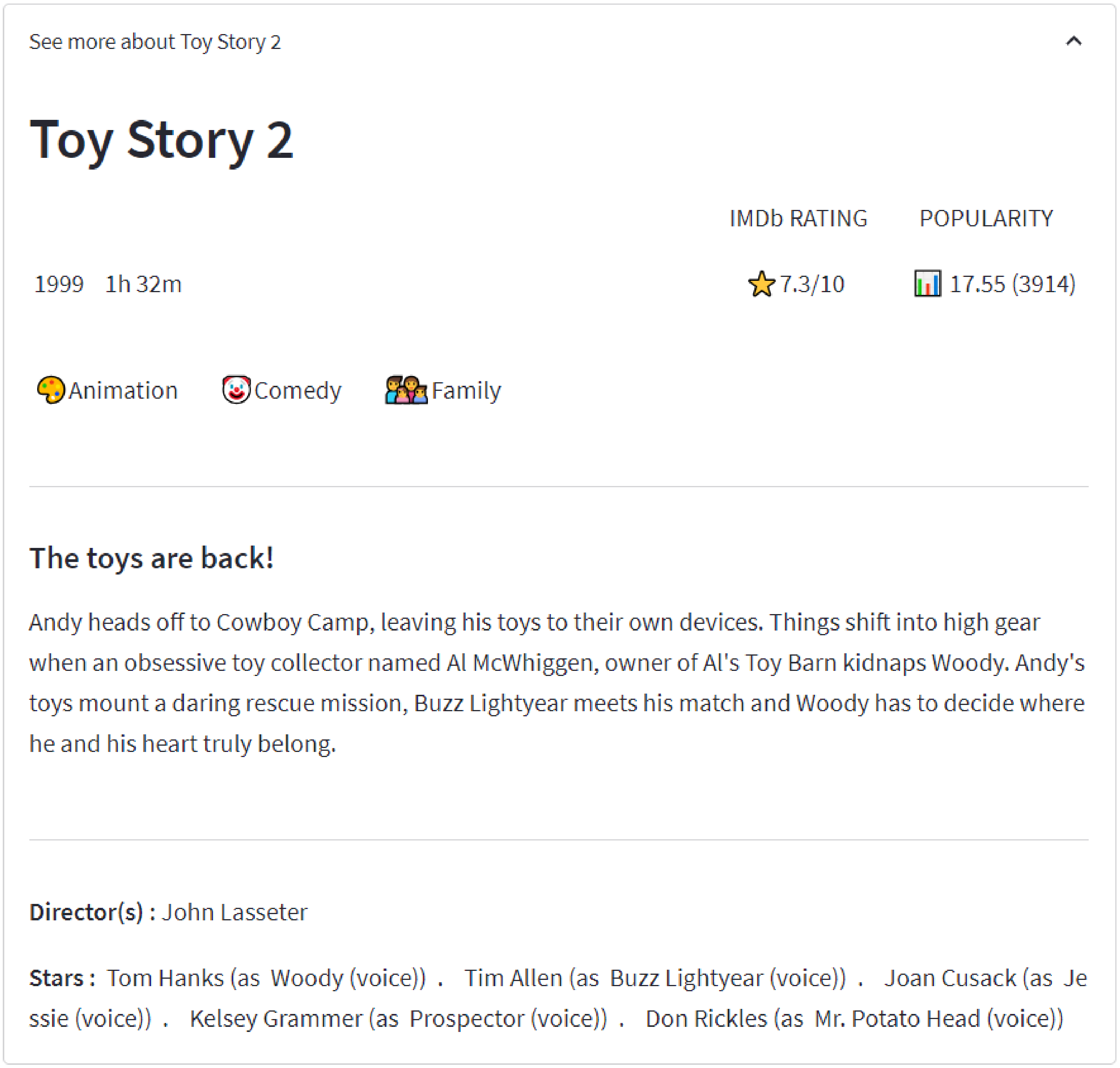
Rate Toy Story!

1

0 10



1. **Toy Story 2 (1999)** based on Toy Story


See more about Toy Story 2

Toy Story 2

	IMDb RATING	POPULARITY
1999 1h 32m	⭐7.3/10	📊 17.55 (3914)

Animation Comedy Family

The toys are back!

Andy heads off to Cowboy Camp, leaving his toys to their own devices. Things shift into high gear when an obsessive toy collector named Al McWhiggen, owner of Al's Toy Barn kidnaps Woody. Andy's toys mount a daring rescue mission, Buzz Lightyear meets his match and Woody has to decide where he and his heart truly belong.

Director(s) : John Lasseter

Stars : Tom Hanks (as Woody (voice)) . Tim Allen (as Buzz Lightyear (voice)) . Joan Cusack (as Je ssie (voice)) . Kelsey Grammer (as Prospector (voice)) . Don Rickles (as Mr. Potato Head (voice))

- We evaluated the proposed method’s performance using specific metrics for recommendation systems.