Movie Classification and Recommendation System

Christina Carbajal and Hitika Ghanani

Introduction

Movie making is one of the biggest entertainment industries that people enjoy. Whether it be action, horror, comedy, or any other genre, stories have been put on screen for generations. But how can we help directors determine if a movie will be a hit? In contrast, how can we tell if a movie will end up flopping? If there are millions of movies, how can we find the best movie for a specific person?

id	keywords	original_languag	original_title
19995	[{"id": 1463, "nan	en	Avatar
285	[{"id": 270, "nam	en	Pirates of the Ca
206647	[{"id": 470, "nam	en	Spectre
49026	[{"id": 849, "nam	en	The Dark Knight
49529	[{"id": 818, "nam	en	John Carter
559	[{"id": 851, "nam	en	Spider-Man 3
38757	[{"id": 1562, "nan	en	Tangled
99861	[{"id": 8828, "nan	en	Avengers: Age o
767	[{"id": 616, "nam	en	Harry Potter and
209112	[{"id": 849, "nam	en	Batman v Superı
1452	[{"id": 83, "name'	en	Superman Retur
10764	[{"id": 627, "nam	en	Quantum of Sola
58	[{"id": 616, "nam	en	Pirates of the Ca
57201	[{"id": 1556, "nan	en	The Lone Range
49521	[{"id": 83, "name'	en	Man of Steel
2454	[{"id": 818, "nam	en	The Chronicles c
24428	[{"id": 242, "nam	en	The Avengers
1865	[{"id": 658, "nam	en	Pirates of the Ca
41154	[{"id": 4379, "nan	en	Men in Black 3
122917	[{"id": 417, "nam	en	The Hobbit: The
1930	[{"id": 1872, "nan	en	The Amazing Sp
20662	[{"id": 4147, "nan	en	Robin Hood

Movies Dataset

- Dataset found off of Kaggle
- Consists of two files: movie credits and movie information (did not use credits for Part A)
- 5000 movies listed
- 20 columns, including title, genres, keywords, original language, original country, etc.

Data Preprocessing and Cleaning

1. Create a subset of columns we need
2. Remove missing and non-practical values in budget and revenue column
3. Parsing through columns with JSON format, convert it into count columns (such as genre and keywords)
4. Converting columns to datetime form and extracting only the year and replacing null values in runtime column with the median
5. **Converting categorical columns into numerical (such as original language and country)**

Movie Hit or Flop Classification

- Dropped movie title column to only have all numerical columns for model
- Created new hit_flop column to classify movies as a hit = 1, or flop = 0
- Function to classify hit or flop was based on revenue percentage, vote average, and popularity
- Logistic Model created
- Used GridSearch CV to find the best parameters
- Fit the model and calculated evaluation metrics

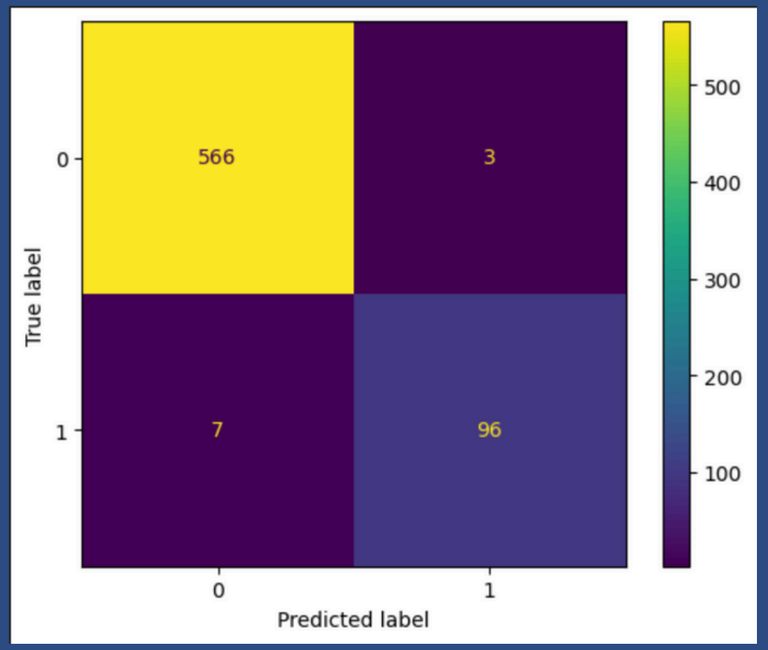
```
# Creating new column based on popularity, revenue, budget, and ratings
def hit_flop_func(row_entry):
    # Revenue percentage: if >= 100, the movie broke even or made profit
    revenue_pct = (row_entry['revenue'] / row_entry['budget']) * 100 if row_entry['budget'] > 0 else 0

    # vote_average range 1-10
    if (revenue_pct >= 100.0) \
       and (row_entry['vote_average'] >= 5.0) \
       and (row_entry['popularity'] >= 50.0): # average around 30
        return 1  # hit

    return 0  # flop

base_movies['hit_flop'] = base_movies.apply(hit_flop_func, axis=1)
base_movies.head() # result will have around 437 hits
```

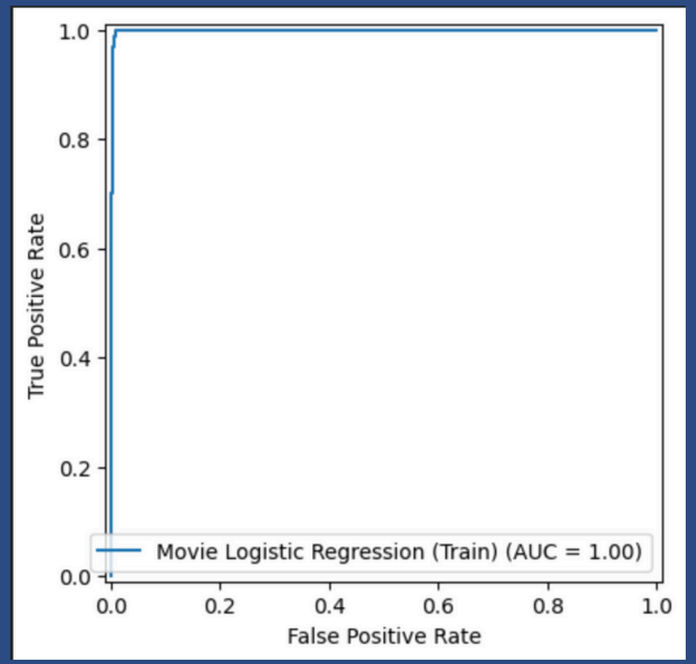
```
# Parameter hypertuning using GridSearchCV
log_movie = LogisticRegression()
param_grid = {'penalty': ['l2', 'l1'],
              'C': [1, 5, 10],
              'solver': ['liblinear', 'saga']}

log_grid = GridSearchCV(log_movie, param_grid)
log_grid.fit(X_train, Y_train)

print(log_grid.best_params_)
best_movie_log = log_grid.best_estimator_
```

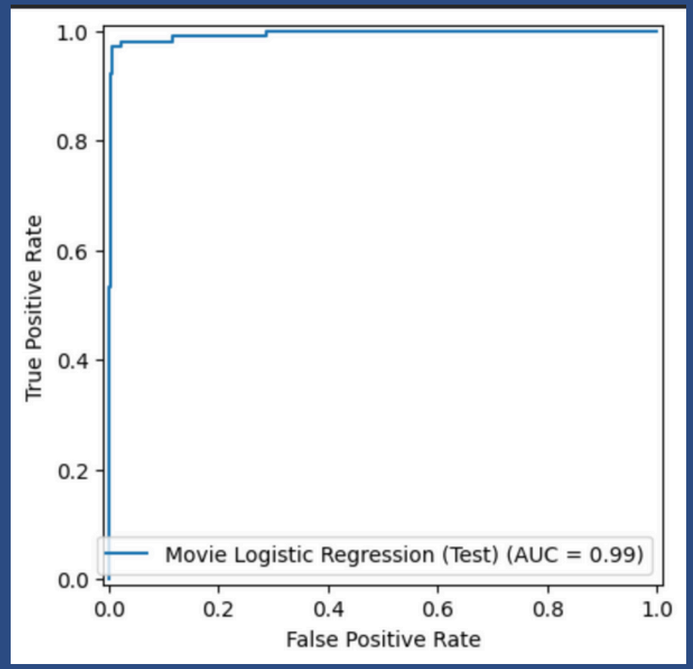
```
{'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
```

Classification Evaluation Metrics

- Accuracy: 0.985
- Precision: 0.970
- Recall: 0.932
- F1 Score: 0.950







Classification Results Discussion

- The model did extremely well in classifying both training and test sets, most likely due to the hit_flop function being a simple equation. Therefore, the logistic model was able to accurately identify which movies will perform better when released with high precision.

- For future applications, we can improve the hit_flop function by using columns that have the most impact (feature engineering) and use the logistic model to predict the success of movies being released. This will help directors determine if a movie is worth pursuing, what elements of the movie they need to change to improve success, and stop production if needed to save their own and crew's reputation.

Movie Recommendation System

HELPING YOU PICK THE PERFECT MOVIE, WITHOUT ASKING YOUR FRIENDS OR GOOGLE.

Because scrolling for 30 minutes to watch something for 10 minutes is not something that should be done.

We built a system that finds movies similar to the ones you like using movie data and text features. It suggests movies to a user based on the similarity of their past watch history, including genres, keywords, cast, crew, and descriptions.

Dataset Preperation

- Movies dataset → contains(title, id, genres, keywords, overview, rating, vote count, popularity, release date)
- Credits dataset → contains(movie id, title, cast details , crew details)
- Both datasets were merged using the id column
- Chose the required columns from the combined dataset

Feature Engineering

TRANSFORMING MOVIE DATA INTO USEFUL FEATURE

- Extracted genres: Converted genre lists into readable text.
- Extracted keywords: Captured important themes and topics of each movie.
- Selected top cast members: Used main actors (top 3) as key indicators of style & similarity.
- Pulled director information: Director often influences movie tone and genre.
- Cleaned movie overview text: Removed missing values and used overviews as natural descriptions.
- Created a soup of features: Combined genres, keywords, cast and director to build a single text representation for each movie.
- Converted text to numbers: Applied CountVectorizer to turn the soup into a numeric matrix.
- Reduced dimensionality: Applied TruncatedSVD to extract nearly 200 latent topics

- **Step 1: Building Movie Vectors**
- The combined “soup” of genres, keywords, cast and director is converted into numbers using CountVectorizer.

- **Step 2: Reduce Dimensions**
- Applied TruncatedSVD to turn high-dimensional text into ~200 latent factors and captures hidden themes like space adventure, romantic drama, etc.

- **Step 3: Compute Similarity**
- Used cosine similarity to measure how close two movies are in the latent feature space.

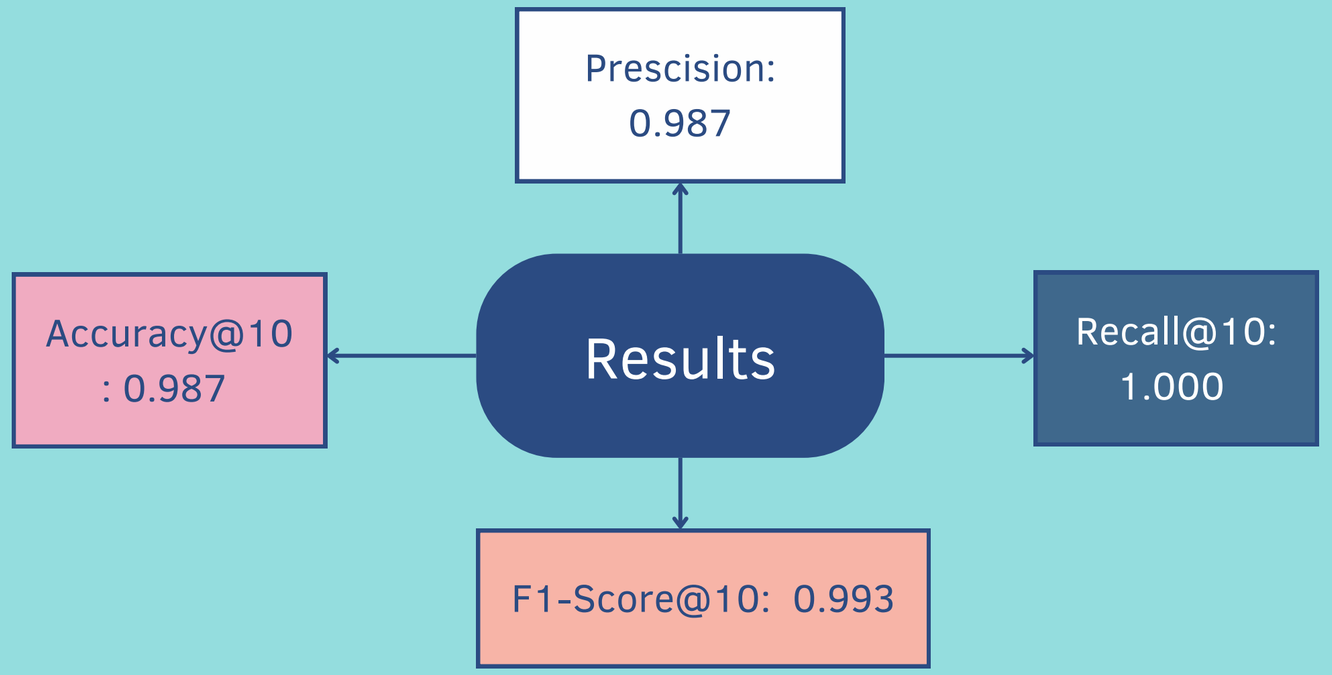
- **Step 4: Generate Recommendations**
- For any selected movie, the model returns the Top-K most similar movies based on these learned features.

Evaluation Metrics and Results

- We evaluated the quality of recommendations using a simple rule: A recommended movie is considered correct if it shares at least one genre with the input movie.
- For each movie, we looked at the Top-10 recommendations because:
- Users usually explore only the first few suggestions
- It keeps evaluation simple, consistent, and fast

Metrics Used

- Accuracy – How often the Top-10 recommendations were genre-correct
- Precision – Out of recommended movies, how many were relevant
- Recall – Out of all relevant movies, how many we captured
- F1-Score – Balance between precision and recall



Feels like the model is just too perfect right?

Very high scores show that the SVD-based similarity model consistently recommends movies with similar themes and genres, which matches the goal of a content based recommendation system.

Recommendation

```
get_recommendations('Avatar',cosine_sim2)
```

	title_x
2655	Dungeons & Dragons: Wrath of the Dragon God
786	The Monkey King 2
1438	Krull
20	The Amazing Spider-Man
71	The Mummy: Tomb of the Dragon Emperor
315	The Mummy Returns
1192	Spawn
1686	The Borrowers
715	The Scorpion King
379	Conan the Barbarian

dtype: object

```
get_recommendations('The Matrix', cosine_sim2)
```

	title_x
123	The Matrix Revolutions
125	The Matrix Reloaded
93	Terminator 3: Rise of the Machines
43	Terminator Salvation
108	Terminator Genisys
266	I, Robot
3439	The Terminator
582	Battle: Los Angeles
487	Red Planet
1008	Chappie

Conclusion and Learnings

From a student's perspective, this project was a really good way to understand how recommendation systems work in practice. Even though we didn't use any complex deep learning models, the combination of feature engineering, SVD, and cosine similarity gave us a model that performed surprisingly well. The recommendations were genuinely relevant, and the evaluation metrics showed that the system is able to capture similarities between movies quite accurately.

What we learned the most from this project is how powerful good feature representation can be. Cleaning the data, choosing the right features, and reducing the dimensionality had a much bigger impact on the results than we initially expected. We also learned how similarity measures work, why cosine similarity is used so often, and how even simple models can perform strongly as long as the data is prepared properly. Overall, this project helped us build intuition around real-world recommender systems and gave us hands on experience with the full workflow, from raw data to meaningful predictions.

Do you have any questions?