Movie Recommender

Priya, Insiya & Rachel

Project Outline

What is it?

A Recommender System refers to a system that is capable of predicting the future preference of a set of items for a user, and recommend the top items.

Why is it important?

Recommender systems attempt to solve the problem of *information overload* along with providing *personalizations* that help users make *optimized decisions* .

Objectives

Create an ML model that recommends movies to a user based on previous movies they have enjoyed

Create an ML model that will predict the rating of a movie based on past ratings

Agenda

1. Methodology
    a. Data Acquisition
    b. Data Cleaning
    c. Data Visualization
2. Machine Learning & Model Creation
    a. Model Development
    b. Model Optimization
3. Web Development & Deployment
4. Model Demonstration

1. Methodology

Approach

Our thought process from model creation to deployment

Gather data to create and train models that recommend movies

Once data is gathered, we cleaned it and began to create different models and find which works best / most accurately

Later on found different data and then used the *surprise python scikit* with this data

Used Manual & GridSearchCV optimization to decide on the best hyperparameters

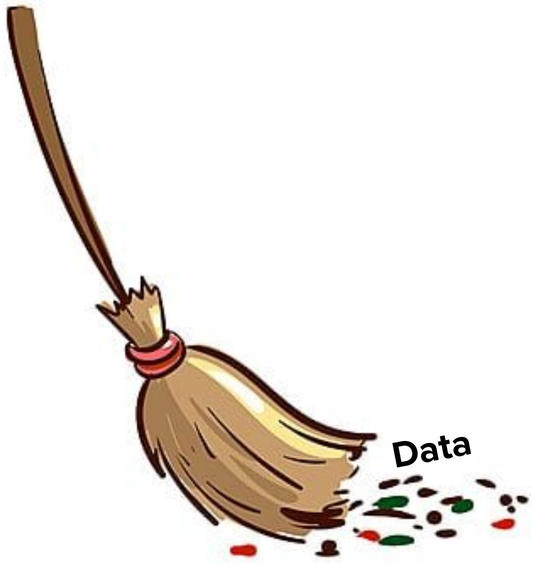
Data Acquisition

- Data was extracted in CSV format from the IMDB database
- The first sets of data we used for visualization creation and the second set of data we used for model creation & training
- These CSV files contain:
  - Movie ratings
  - Movie titles
  - Actor names & production companies
  - Genres
  - User ratings

Data Cleaning

Using Jupyter Notebook & Pandas:

- Removed null values
- Dropped non essential columns
- Split genre column into 3 columns
- Separated year and month into separate columns



Data Preview

Ratings.CSV

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Movies.CSV

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

Links.CSV

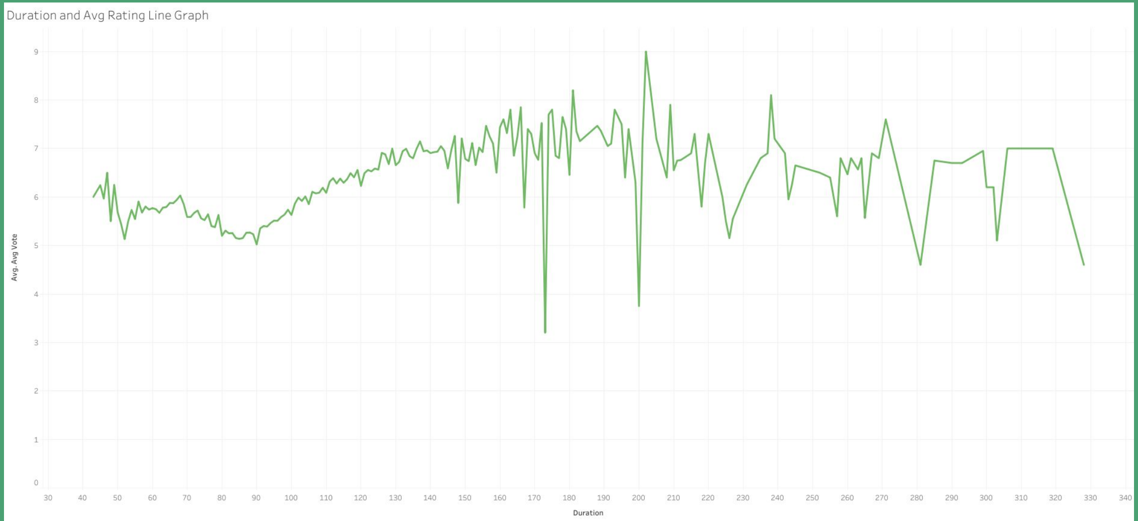
	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0

Data Visualization

Visuals were created to further explore the data we were using and are displayed on our webpage

Graphs were made using **Tableau** to display:

- Highest rated movies on average
- The correlation between ratings and duration of movie
- Most popular genres



2. Machine Learning Model Creation

Recommender Systems: Approaches

- Content Based Filtering (get movie recommendations)
  - Using Genre
  - Using Movie Description

- Unsupervised Learning (get movie recommendations)
  - KNN Movie Recommender

- Supervised Learning (get predicted movie rating)
  - Singular Value Decomposition Matrix Factorization (SVD MF); using surprise python scikit
  - XGBoost Algorithm (XGBoost is an implementation of gradient boosted decision trees designed for speed and performance)

Content Based Filtering: Using Genre

1. Converted Genre column to an array

2. Using **cosine_similarity()** created a matrix where each row corresponds to similarity of a movie with other movies

3. Joined Genre Array DF with movies

1

```
#convert genre column to array
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
X = cv.fit_transform(genre_data["genres"]).toarray()
X
```

```
array([[0, 1, 1, ..., 0, 0, 0],
       [0, 1, 0, ..., 0, 0, 0],
```

2

```
#Each row corresponds is a different movie
from sklearn.metrics.pairwise import cosine_similarity

similarities = cosine_similarity(X)
#Each row of matrix coressponds to similarity of a movi
similarities
```

```
array([[1.        , 0.77459667, 0.31622777, ..., 0.
        0.4472136 ],
       [0.77459667, 1.        , 0.        , ..., 0.
        0.        ],
```

3

```
output = data.loc[:,['movieId','title']]
output = output.join(pd.DataFrame(X))
output
```

	movieId	title	0	1	2	3	4	5	6	7	...
0	1	Toy Story (1995)	0	1	1	1	1	0	0	0	...
1	2	Jumanji (1995)	0	1	0	1	0	0	0	0	...
2	3	Grumpier Old Men (1995)	0	0	0	0	1	0	0	0	...
3	4	Waiting to Exhale (1995)	0	0	0	0	1	0	0	1	...

Content Based Filtering: Using Genre

4. We can then use **timestamp to query last movie** watched by the user and extract similar movies from matrix

5. **Sorts the similar movies by max similarity and outputs top 15**

4

```
uid = 18 #For user 18 lets recommend movies based on his recent watched movie
time = ratings.loc[ratings["userId"]==uid,["movieId","timestamp"]]
latest_movieId_watched_by_user = time.sort_values(by="timestamp",ascending=False)["movieId"].values[0]
latest_movieId_watched_by_user
```

```
166015
```

```
movie_index = data.loc[data['movieId']==latest_movieId_watched_by_user,["title"]].index[0]
output.loc[output['movieId']==8798,:]
```

	movieId	title	0	1	2	3	4	5	6	7	...	14	15	16	17	18	19	20	21	22	23
5305	8798	Collateral (2004)	1	0	0	0	0	1	0	1	...	0	0	0	0	0	0	0	1	0	0

5

```
print("Since u watched --->",get_movie_by_id(latest_movieId_watched_by_user),"<--- We recommend you")
for i in range(15):
    print(get_movie_by_index(similar_movie_indexes[i]))
```

```
Since u watched ---> The African Doctor (2016) <--- We recommend you
Last Detail, The (1973)
Inkwell, The (1994)
California Split (1974)
Mrs. Doubtfire (1993)
Letter to Three Wives, A (1949)
Million Dollar Arm (2014)
Love and Death on Long Island (1997)
Tales of Manhattan (1942)
Birdman: Or (The Unexpected Virtue of Ignorance) (2014)
Primary Colors (1998)
Two Girls and a Guy (1997)
The Players Club (1998)
Angriest Man in Brooklyn, The (2014)
Last Days of Disco, The (1998)
The Hundred-Foot Journey (2014)
```

Content Based Filtering: Using Movie Description

1. Use **TfidfVectorizer** to create a tfidf matrix of the words in the description column

```
#create the matrix
tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(df['description'])
```

2. Using **cosine_similarity()** calculate sim_scores based on the frequency of words being mentioned in the description

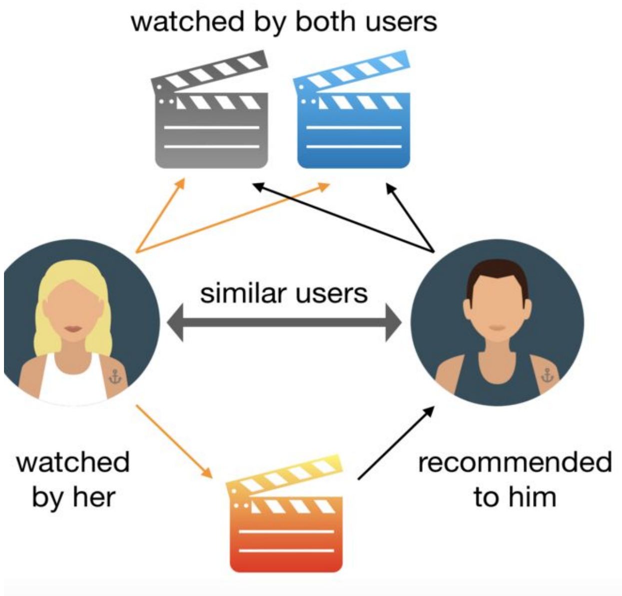
```
get_recommendations(movie_name).head(10)
```

```
21407              Avengers: Age of Ultron
23644                     Grandma's House
21704                         Wastelander
11000                          Alien Seed
18298                      Red Dirt Rising
21382         Dark Skies - Oscure presenze
20186        Evil Bong 3: The Wrath of Bong
17786         L'ultimo dominatore dell'aria
13493                        Lost in Space
9886                     Twice Upon a Time
Name: title, dtype: object
```

3. Using these scores we can recommend a movie to a user based off the description of a movie they already like

Collaborative Filtering: Nearest Neighbours

- Filters information by using the interactions and data collected by the system from other users.
- People who agreed in their evaluation of certain movies are likely to agree again in the future
- 2 Types: **User-based** and **Item-based** collaborative filtering
- User Preference is usually expressed by two categories. **Explicit Rating** and **Implicit Rating**



Collaborative Filtering: Nearest Neighbours

1. Each column represents unique userId and each row represents each unique movieId

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	0.0	0.0	0.0	4.0	0.0	4.5	0.0	0.0	0.0	...	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	0.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	0.0	0.0	...	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0	0.0
3	4.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

2. **Removing Noise from the data**

   To qualify a movie, a minimum of 10 users should have voted a movie.
   To qualify a user, a minimum of 50 movies should have voted by the user.

3. **Removing sparsity**

   Applied csr_matrix method to the dataset

4. Used **KNN algorithm** to compute similarity with cosine distance which is very fast and preferred over pearson coefficient. Found similar movies and sort them based on their distance from the input movie

Supervised Learning: SVD MF

- Singular Value Decomposition (SVD) is a widely used technique to **decompose a matrix into several component matrices**, exposing many of the useful and interesting properties of the original matrix.

- Matrix factorization is a class of collaborative filtering algorithms used in recommender systems. **Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices**.

1. Split Data into Training and Testing sets
2. Fit the model using SVD() and baseline parameters
3. Calculated RMSE and MAE
4. Optimized Model (nested for loop)
5. Refit the Model & got predictions
6. Calculated new RMSE and MAE
7. **Used predictions from SVD MF as an input into XGBoost Model**

Supervised Learning: XGBoost

- Prepared training and testing set using sparse matrix
- Added additional predictors:
  - Global rating average
  - User rating averages
  - Movie averages
  - Similar user ratings
  - Similar movie ratings
  - SVD MF predictions
- Fit the model using XGBoost algorithm
- Calculated RMSE and MAPE
- Optimized model using Gridsearch
- Refit the Model & got predictions
- Calculated new Error Values
- Readied the predictions for deployment

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

Supervised Learning: XGBoost

Testing set Preview:

GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	mf_svd
3.519977	2.0	5.0	4.0	4.0	4.5	3.0	4.0	3.0	5.0	5.0	4.366379	3.954545	4.306195
3.519977	4.0	5.0	4.0	4.0	5.0	4.0	3.0	3.0	5.0	5.0	3.636364	3.954545	3.941735
3.519977	4.0	4.0	5.0	4.5	4.0	4.5	4.5	5.0	4.0	3.0	3.230263	3.954545	4.543556
3.519977	5.0	3.0	4.0	4.0	4.0	3.5	3.0	5.0	3.0	3.0	3.448148	3.954545	4.677380
3.519977	4.0	5.0	4.0	4.0	4.5	4.0	4.5	5.0	5.0	5.0	4.209524	3.954545	4.810359

Ending Result:

movieId	title	genres	user	GAvg	sur1	sur2	sur3	sur4	sur5	smr1	smr2	smr3	smr4	smr5	UAvg	MAvg	rating	mf_svd	predicted rating
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	514	3.427881	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	2.5	5.0	3.311083	3.769231	4.0	3.425865	3.727467
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	517	3.427881	4.0	4.0	4.0	2.5	4.0	3.5	2.0	3.0	3.5	5.0	2.386250	3.769231	4.0	3.229036	3.052417
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	522	3.427881	5.0	4.0	4.0	4.0	3.0	3.5	4.0	5.0	5.0	5.0	3.830000	3.769231	3.0	3.026007	4.283056
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	524	3.427881	5.0	4.0	5.0	3.0	4.0	3.0	3.0	3.0	5.0	5.0	3.458015	3.769231	4.0	3.004834	3.441464
1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	525	3.427881	4.0	3.0	5.0	2.5	4.0	4.0	4.0	4.0	4.5	4.0	3.542000	3.769231	4.0	2.366412	3.921979

Model Optimization

Model Optimization: SVD MF

The starting RMSE for SVD MF was 0.65, and MAE is 0.50. **With optimization we were able to reduce it RMSE to 0.28 and MAE to 0.18.**

```
i=1
for factor in range(200,500,100):

    for learnrate in np.arange(0.005, 1, 0.1):

        svd = SVD(n_factors=factor, biased=True, random_state=15, lr_all=learnrate ,verbose=False)
        svd.fit(trainset)

        #getting predictions of train set
        train_preds = svd.test(trainset.build_testset())

        #adding results in list
        optimization_rmse.append(surprise.accuracy.rmse(train_preds, verbose=True))
        optimization_mae.append(surprise.accuracy.mae(train_preds, verbose=True))
        optimization_factor.append(factor)
        optimization_learnrate.append(learnrate)
        print(factor)
        print('Optimization #',i)
        i=i+1
        print ('----------------------')
```

Nested For Loop to optimize for min MAE and RMSE using *number of factors* and *learning rate*

```
optimization_grid = pd.DataFrame(
    {'RMSE': optimization_rmse,
     'MAE': optimization_mae,
     'Factors':optimization_factor,
     'Learning Rate' : optimization_learnrate
    })

optimization_grid.head()
```

	RMSE	MAE	Factors	Learning Rate
0	0.535439	0.417572	200	0.005
1	0.280285	0.184001	200	0.105
2	1.807258	1.480023	200	0.205
3	1.807258	1.480023	200	0.305
4	1.807258	1.480023	200	0.405

Model Optimization: XGBoost

The starting RMSE for XGBoost was 0.68, and MAPE is 20.6. **With optimization we were able to reduce it RMSE to 0.63 and MAPE to 18.7.**

```
from sklearn.model_selection import GridSearchCV
param_grid = dict(
    n_jobs=[16,20,25],
    n_estimators=[100,200,300,400],
    booster = ['gbtree'],
    learning_rate = [0.01,0.02,0.1,1])

xgb_model = xgb.XGBRegressor(silent=False, random_state=15)

grid_search = GridSearchCV(estimator=xgb_model,
                           param_grid=param_grid,
                           scoring='neg_root_mean_squared_error')

best_model = grid_search.fit(x_train, y_train)
print('Optimum parameters', best_model.best_params_)
```

Used GridSeachCV to identify best performing hyperparameters

This optimization did not result is significant improvement and took over 1 hour to complete.

3. Web Development & Deployment

Web Development & Deployment

HTML Front End

- Used an HTML code template that had the skeleton of the webpage
- Added drop down pages for visualizations and model type explanations
- Added a description of our project
- Added a place for our model to ask for a movie and give output

Back End

- Deployment Issues: used Local Host instead to deploy
- Originally wanted to deploy to Heroku
- Created an app.py, model.py and intidb.py file to hold our python code and connect it to the webpage

4. Model Demonstration

5. Next Steps

Model Improvements

- Ratings Predictor
  - Optimize on further hyperparameters
  - Optimize on features
  - Join additional features (genre, actors etc)
- Movie Recommender
  - Create a testing set and evaluate accuracy of the different models
  - Remove the movies already watched by the user, currently only removes the movie last watched/used to query
- Overall
  - Explore other models beyond KNN, XGBoost, SVD MF that could result in more accurate predictions
  - Some chunks took very long to run, optimize based on time

Questions?