MOVIE RECOMMENDATION SYSTEM

MSCA 31009 Machine Learning Final Project

Allen Dong | Jack Du | Kenneth Jin | Renyuan Liu

THE UNIVERSITY OF CHICAGO

AGENDA

1. Executive Summary
2. Problem Statement & Research Objective
3. EDA
4. Data Preparation & Feature Engineering
5. Hardware & Software Tools
6. Models
    a. Singular Value Decomposition Collaborative Filtering Recommendation System
    b. Neural Network Collaborative Filtering Recommendation System
    c. Content Based Movie Recommendation System
    d. Hybrid Recommendation System: Content + Collaborative
7. Findings & Conclusion
8. Lessons Learnt & Future Recommendation

Executive Summary

This project aims to build a movie recommendation system through a filtering process that is based on user preferences and ratings on their browsing history. It takes information from both users and movies as inputs and then implements the machine learning algorithms such as collaborative filtering, content-based filtering, and a hybrid model to create personalized movie recommendations. The dataset we used here consists 670+ unique user and 9000+ movie information sourcing from the MovieLens & IMDB website.

Problem Statement & Research Objective

With online streaming sites competing for user's attention, a feature that comes in handy in attracting customers and ensuring repeat business is content recommendation. In this project, given a set of users and their previous ratings on movies and movies general information, we **provide a mechanism to optimize user's streaming experience.**

01

Increase Customer Engagement

Achieve customer loyalty by providing relevant content and maximizing the time spent on the channel.

02

Drive Traffic

For movie distribution channels, our recommendation system potentially increases website traffic by sending out customized email notifications with targeted blasts.

03

Boost Revenues

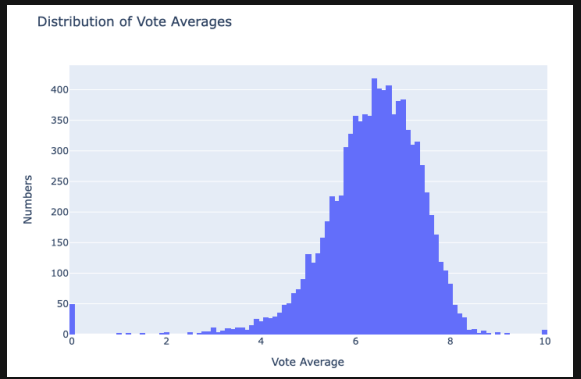
The recommendation system can significantly boost CTRs, conversions, and other important metrics which lead to generating more profits.

Data Summary

	Description	Data size	Row Numbers	Data Source
Ratings	Movies' rating by users	2.44 MB	440,016	MovieLens-100K from Kaggle
Movies	Movies' title and genres	458 KB	27,375	MovieLens-100K
Movies meta	Comprehensive information of each movie	34.4 MB	1,091,184	IMBD
Links	Movies' imbd_id and tmbd_id	183 KB	27,375	MovieLens-100
Tags	Movies' genres	41.9 KB	5,184	MovieLens-100K
Credits	Movies' cast group	189 MB	136,428	MovieLens-100K

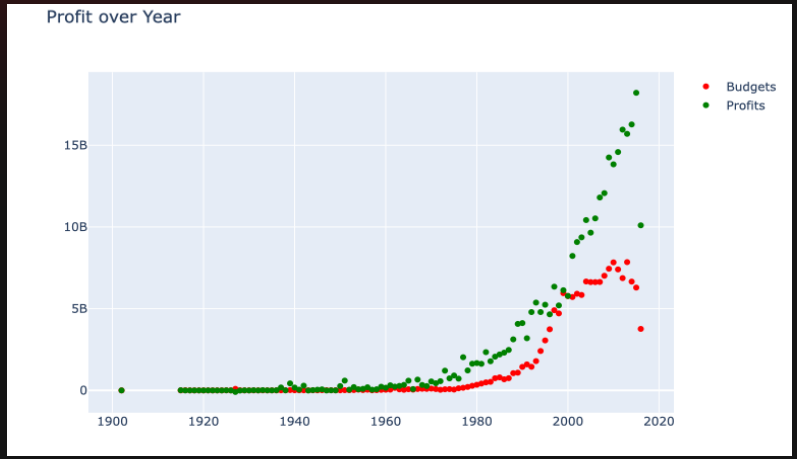
Rating EDA

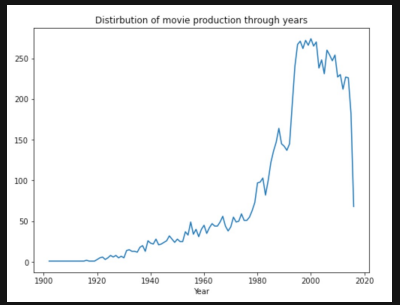
- The distribution of the average rating score indicates that most of the rating are between 3 to 5 and only less than 25 percent of the movie are below 3. The box plot shows the **average rating is around 3.5**.
- The voting average distribution shows that most movies get voting score **between 4 to 6**
- By the join the rating table with others , we **can rank movies** based on the rating scores or voting average
- The rating and voting score are the primary features we use to feed in the user – item matrix and build the recommendation system

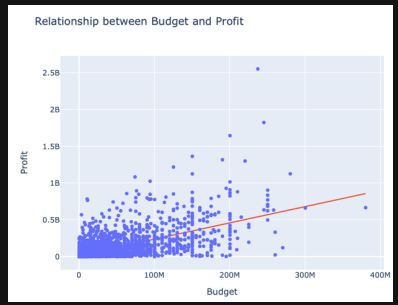


Movie EDA

- We created a new feature ‘profits’ using revenue minus budget.
- The production of movie jumped rapidly after 1980 and the profit increased along the way. Until 2000, the movie production started to making more profit than before, which may be affected by the new technology and more movie theatres.
- The graph indicates that profit achieved highest around year 2015 and it may continue grow up to now. In addition, there is a weak linear relationship between budgets and profit.

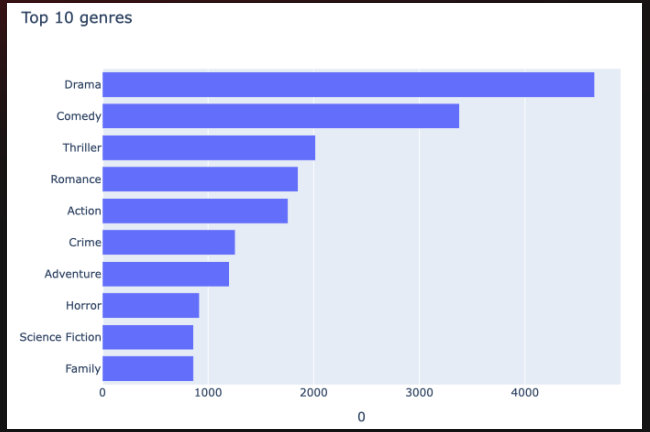


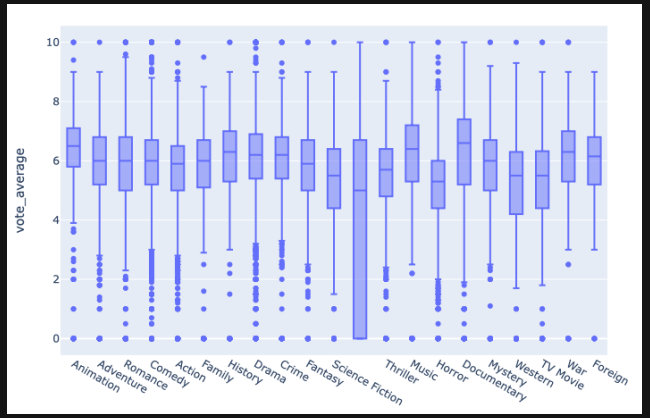




Genre EDA

- It is not surprise that Drama, Comedy and Thriller are the three most popular genres.
- The voting average across the top 10 genres are similar. Documentary and animation average score are sightly better than other genres.
- Comedy, action and drama are more likely to receive extreme voting score than other genres





Data Preparation & Feature Engineering

User Rating Data for Sparsity Check:

- # userId 671
- # movieId 9066
- Rating range: 0 - 10
- Sparsity: 0.984

Feature Engineering:

$$Weighted\ Rating(WR) = (\frac{v}{v+m}.R) + (\frac{m}{v+m}.C)$$

where,
v is the number of votes for the movie
m is the minimum votes required to be listed in the chart
R is the average rating of the movie
C is the mean vote across the whole report

IMBD library: Three additional features added for content-based RS

Storyline: movie's sequence of plots (Pulled through IMDB API)

```
from imdb import Cinemagoer
from tqdm import tqdm, tqdm_notebook

ia = Cinemagoer()

storyline = []

for i in tqdm(final_metadata['imdb_id'].str[2:]):
    #print(i)
    movie = ia.get_movie(i)
    if 'plot outline' not in movie.data.keys():
        storyline.append("NA")
        continue
    sl = movie.data['plot outline']
    storyline.append(sl)
```

Tagline: Movie's Slogans or catchphrases
Description: Tagline + Title + Storyline

HARDWARE & SOFTWARE TOOLS USED

Anaconda

Platform to run Python & PyTorch

ANACONDA.

Python

For Data Cleaning, EDA & Feature Engineering

PyTorch

For Running Deep Neural Network on GPU

PyTorch

CPU (AMD 5950X)

To run most Cleaning, Analysis, and Modeling

GPU (RTX 3080TI)

To run Deep Neural Network Model


HOW DO YOU KILL SOMETHING THAT CAN'T POSSIBLY BE ALIVE?
CHRISTINE
Tighten your seat belt.
You never had a trip like this before.
VANISHING POINT
A FILM BY RON HOWARD
RUSH
CINEMAS SEPTEMBER
EDGAR WRIGHT
BABY DRIVER
ALL YOU NEED IS ONE KILLER TRACK.
Walt Disney
HERBIE RIDES
THE ITALIAN JOB
American Graffiti
Where were you in '62?
VAL KILMER
HEAT
GONE IN 60 SECONDS
MATT DAMON CHRISTIAN BALE
FORD v FERRARI
CRUISE
Drive
THE FAST AND THE FURIOUS
WINNER SUNDANCE
"HE WAS THE BEST DRIVER WHO EVER LIVED"
SENNA
NO FEAR. NO LIMITS. NO EQUAL.
COMING SOON
STEVE McQUEEN
'BULLITT'
ROBERT VAUGHN
Disney PIXAR
Cars

Models

Collaborative Filtering Recommendation System

Referenced Link

Motivation:

To address the limitations of content-based recommendation such as computational cost and lack of serendipity, collaborative filtering can take into account of both implicit and explicit information from users, movies and their interactions to make recommendation.

Approach:

Model Based Collaborative filtering implemented by Singular Value Decomposition (SVD++).

SVD++ is an improved version of SVD which is a matrix factorization algorithm that can extract characteristics of the dataset's features. SVD++ can include not only explicit ratings, but also implicit ones. Implicit feedback are collected indirectly from user interactions, and they act as a proxy for user preference

Model:

Surprise Library

```
trainset = data.build_full_trainset()
```

```
algo = SVDpp()

algo.fit(trainset);
```

Define function to get top 10 recommendations for all users

```
def get_top_n(predictions, n=10):

    # First map the predictions to each user.
    top_n = defaultdict(list)
    for uid, iid, true_r, est, _ in predictions:
        top_n[uid].append((iid, est))

    # Then sort the predictions for each user and retrieve the k highest ones.
    for uid, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[uid] = user_ratings[:n]

    return top_n
```

Collaborative Filtering Recommendation System

Model Result:

Top 10 recommended movies for each user

RMSE: 0.5113

Disadvantages:

- Simple multiplication of latent features (inner product) may not be sufficient to capture the complex structure of user interaction data
- Matrix factorization often tends to recommend popular items to everyone which does not always reflect the specific user interests
- Collaborative Filtering model does not consider important side features such as movie's content

```
In [137]: for uid, user_ratings in top_n.items():
              print(uid, [iid for (iid, _) in user_ratings])

1 [318, 2542, 534, 3462, 905, 1217, 8132, 2019, 7502, 741]
2 [318, 905, 926, 3462, 969, 1217, 1172, 1945, 2064, 7502]
3 [2542, 4011, 116797, 1259, 3462, 56782, 922, 1172, 741, 2467]
4 [50, 527, 318, 778, 2318, 2858, 58559, 1221, 2692, 745]
5 [2542, 741, 969, 899, 1217, 1233, 1172, 3462, 1237, 318]
6 [905, 923, 7502, 2064, 1203, 318, 4993, 1228, 50, 134130]
7 [50, 593, 608, 858, 7502, 926, 296, 4993, 3462, 78499]
8 [1221, 8132, 7502, 898, 1299, 905, 3462, 2019, 1213, 1203]
9 [3462, 969, 926, 905, 7502, 858, 1217, 8132, 1203, 7153]
10 [608, 7502, 926, 2318, 2858, 4993, 858, 969, 905, 232]
11 [7153, 7502, 4993, 858, 318, 1221, 5952, 905, 527, 58559]
12 [5952, 1233, 2542, 4993, 50, 7153, 899, 1227, 79702, 1237]
13 [7153, 858, 905, 2542, 1221, 5952, 3462, 260, 2467, 8132]
14 [3462, 969, 318, 527, 7502, 926, 2571, 86882, 858, 2690]
15 [1305, 3037, 7502, 4444, 940, 2971, 3310, 1242, 3424, 5782]
16 [2542, 8132, 1221, 2571, 497, 2690, 905, 7502, 1066, 1957]
17 [534, 1203, 30749, 3030, 2395, 1201, 3310, 3000, 246, 1305]
18 [858, 318, 969, 6016, 1217, 527, 926, 50, 7502, 8132]
19 [926, 3462, 3310, 1945, 2858, 778, 1732, 6016, 8132, 7502]

In [139]: accuracy.rmse(predictionls)

          RMSE: 0.5113

Out[139]: 0.5113432948020294
```

NN Collaborative Filtering Recommendation System

Referenced Link

Motivation: To build a more complex collaborative filtering model by constructing user-item interaction with neural network on implicit feedback instead of explicit feedback and to present users with the highest interacted movies

Reasoning: A recommendation system that is built through implicit feedback can give recommendations real time, and doesn't need to rely on exterior user feedback data. Instead of predicting the rating that the user will give, it will now predict how likely the user will interact with the movie.

Difference between Explicit & Implicit Feedback:

- Explicit Feedback
  - Direct & Quantitative data collected from users
  - Amazon rating scale of 1-10
  - Youtube like button for each video
  - Instagram like button for each advertisement they announce
- Implicit Feedback
  - Indirect collection of interactions from users
  - User search history from Amazon
  - The length, genre, and type of Youtube video user is watching
  - Instagram click-through rate on each advertisement

NN Collaborative Filtering Recommendation System

Feature Conversion for Modeling: Convert dataset into implicit feedback dataset --> Binarize the rating dataset

1. Converting all of the ratings into 1's as it means that they have interacted with the movie
2. Generating samples of 4 to 1 ratio of negative to positive interacting data

Reasoning: We need to create negative samples, to train the model in understanding how users don't interact to certain movies.

```
train_ratings.loc[:, 'rating'] = 1
```

```
## Creating 4 negative samples ffor each row of the data
# Get a list of all movie IDs
all_movieIds = ratings['movieId'].unique()

# Placeholders that will hold the training data
users, items, labels = [], [], []

# This is the set of items that each user has interaction with
user_item_set = set(zip(train_ratings['userId'], train_ratings['movieId']))

# 4:1 ratio of negative to positive samples
num_negatives = 4

for (u, i) in user_item_set:
    users.append(u)
    items.append(i)
    labels.append(1) # items that the user has interacted with are positive
    for _ in range(num_negatives):
        # randomly select an item
        negative_item = np.random.choice(all_movieIds)
        # check that the user has not interacted with this item
        while (u, negative_item) in user_item_set:
            negative_item = np.random.choice(all_movieIds)
        users.append(u)
        items.append(negative_item)
        labels.append(0) # items not interacted with are negative
```

NN Collaborative Filtering Recommendation System

Model Architecture

User Embedding + Item Embedding
Concatenated Embedding
4 FC Layers
Sigmoid Output Layer

Adam Optimizer w/ 0.01 Learning Rate

50 Epochs
Trained on GPU
Ran on PyTorch & PyTorch Lightning

```
def __init__(self, num_users, num_items, ratings, all_movieIds):
    super().__init__()
    self.user_embedding = nn.Embedding(num_embeddings=num_users, embedding_dim=16)
    self.item_embedding = nn.Embedding(num_embeddings=num_items, embedding_dim=16)
    self.fc1 = nn.Linear(in_features=32, out_features=64)
    self.fc2 = nn.Linear(in_features=64, out_features=128)
    self.fc3 = nn.Linear(in_features=128, out_features=64)
    self.fc4 = nn.Linear(in_features=64, out_features=32)
    self.output = nn.Linear(in_features=32, out_features=1)
    self.ratings = ratings
    self.all_movieIds = all_movieIds

def forward(self, user_input, item_input):

    # Pass through embedding layers
    user_embedded = self.user_embedding(user_input)
    item_embedded = self.item_embedding(item_input)

    # Concat the two embedding layers
    vector = torch.cat([user_embedded, item_embedded], dim=-1)

    # Pass through dense layer
    vector = nn.ReLU()(self.fc1(vector))
    vector = nn.ReLU()(self.fc2(vector))
    vector = nn.ReLU()(self.fc3(vector))
    vector = nn.ReLU()(self.fc4(vector))

    # Output layer
    pred = nn.Sigmoid()(self.output(vector))

    return pred

def training_step(self, batch, batch_idx):
    user_input, item_input, labels = batch
    predicted_labels = self(user_input, item_input)
    loss = nn.BCELoss()(predicted_labels, labels.view(-1, 1).float())
    return loss

def configure_optimizers(self):
    return torch.optim.Adam(self.parameters(), lr = 0.01)

def train_dataloader(self):
    return DataLoader(MovieLensTrainDataset(self.ratings, self.all_movieIds),
                      batch_size=512, num_workers=0)
```

NN Collaborative Filtering Recommendation System

Evaluation: Creating the Hit Ratio @ 10

Steps & Reasoning:

1. Randomly select 99 items that the user has not selected and interacted with
2. Combine all of the 99 items with 1 of the test item (the one the user has interacted with)
3. Run the model on 100 of these items, and rank the top 10 items
4. If the test item is in the top 10 items we would call that a Hit
5. For each user that exist repeat the process, and the final number is the average hits

RMSE & Accuracy would be too simple of an evaluation for a complex model like this.

NN Collaborative Filtering Recommendation System

Evaluating Recommender Systems

```
# User-item pairs for testing
test_user_item_set = set(zip(test_ratings['userId'], test_ratings['movieId']))

# Dict of all items that are interacted with by each user
user_interacted_items = ratings.groupby('userId')['movieId'].apply(list).to_dict()

hits = []
for (u,i) in test_user_item_set:
    interacted_items = user_interacted_items[u]
    not_interacted_items = set(all_movieIds) - set(interacted_items)
    selected_not_interacted = list(np.random.choice(list(not_interacted_items), 99))
    test_items = selected_not_interacted + [i]

    predicted_labels = np.squeeze(model(torch.tensor([u]*100),
                                        torch.tensor(test_items)).detach().numpy())

    top10_items = [test_items[i] for i in np.argsort(predicted_labels)[::-1][0:10].tolist()]

    if i in top10_items:
        hits.append(1)
    else:
        hits.append(0)

print("The Hit Ratio @ 10 is {:.2f}".format(np.average(hits)))
```

```
The Hit Ratio @ 10 is 0.61
```

Model Results:

Hit Ratio @ 10: 61%

This means that 61% of the time the users were recommended items that they were actually interested in, based on the test item.

Disadvantages:

Collaborative Filtering model does not consider important side features such as movie's content

Content Based Recommendation System

Referenced

Motivation: Content based recommendation system can capture the words meaning and relationship of the product or service and quantify for further analysis. The data that is captured through this system can contain more information and often can provide more useful predictions compared to collaborative filtering-based recommendation.

Reasoning: Since this system is being built mainly to provide movie recommendations, movies can contain a lot of different meanings behind each storyline. Thus, by converting the storyline, tagline and the title of the movie to a quantifiable vector for recommendation, will provide a lot of additional meaning compared to just a collaborative filtering recommendation.

```
final_df['tagline'] = final_df['tagline'].fillna('')
final_df['title'] = final_df['title'].fillna('')
final_df['description'] = final_df['title'] +  ' ' + final_df['storyline'] + ' ' + final_df['tagline']
```

```
final_df['description'].head(5)
```

```
0    Toy Story A little boy named Andy loves to be ...
1    Jumanji Jumanji, one of the most unique--and d...
2    Grumpier Old Men Things don't seem to change m...
3    Waiting to Exhale This story based on the best...
4    Father of the Bride Part II In this sequel to ...
Name: description, dtype: object
```

Content Based Recommendation System

Feature Conversion for Modeling:

1. Vectorize the "Description" column that combines title, tagline, & storyline using TFIDF
2. Implement a cosine similarity function, to calculate the relationship between each content, and creating a cosine similarity pairwise matrix

Reasoning:

1. By using TFIDF we are able to better understand the relationship and relevancy of each word within a movies title, tagline, and storyline.
2. Cosine similarity is able to measure the similarity between two vectors of an inner product space, it is used to measure the document similarity of the text we are analyzing

```
tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(final_clean_df['description'])
```

```
tfidf_matrix
```

```
<8694x455664 sparse matrix of type '<class 'numpy.float64'>'
        with 925495 stored elements in Compressed Sparse Row format>
```

```
# Define similarity function and
from numpy import dot
from numpy.linalg import norm

def cosine_sim(list1, list2):
    result = dot(list1, list2)/(norm(list1)*norm(list2))
    return result
```

```
final_clean_df = final_clean_df.reset_index()
titles = final_clean_df['title']
indices = pd.Series(final_clean_df.index, index=final_clean_df['title'])
```

Content Based Recommendation System

Feature Creating for Modeling:

1. Creating a weighted rating column with algorithm listed on the right

Reasoning:

1. Weighted rating is required because after finding top 10 movies that are similar for recommendation. It needs to follow a certain ranking order for the user, and if it uses the normal rating scale it is biased as some ratings have 1-2 users and others have potentially 100+ users.

$$Weighted\ Rating(WR) = (\frac{v}{v+m}.R) + (\frac{m}{v+m}.C)$$

where,
v is the number of votes for the movie
m is the minimum votes required to be listed in the chart
R is the average rating of the movie
C is the mean vote across the whole report

```
#Defining Weighted Rating through Vote Counts & Average
def weighted_rating(x):
    C = vote_avg.mean()
    m = vote_cnt.quantile(0.60)
    v = x['vote_count']
    R = x['vote_average']
    return (v/(v+m) * R) + (m/(m+v) * C)
```

Content Based Recommendation System

Steps in Producing the Output:

1. Finding the cosine similarity score for this movie to all of the other movies
2. List out 30 of the movies that are similar based on the title, tagline, and storyline
3. Rank the top 10 movies through the weighted Rating

```python
#Content Based Similarity Recommendation TFIDF
def improved_recommendations_tfidf(title):
    idx = indices[title]
    #Find the cosine similarity score for this movie and
    sim_scores = list(enumerate(cosine_sim_tfidf[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:30]

    #Find the similar movies index
    movie_indices = [i[0] for i in sim_scores]
    #Get movies' info including voting
    movies = final_clean_df.iloc[movie_indices][['title', 'vote_count', 'vote_average', 'release_date']]

    #Create a qualified list with weighted rating
    vote_counts = movies[movies['vote_count'].notnull()]['vote_count'].astype('int')
    vote_averages = movies[movies['vote_average'].notnull()]['vote_average'].astype('int')
    #C = vote_avg.mean()
    #m = vote_cnt.quantile(0.60)
    qualified = movies[(movies['vote_count'] >= m) & (movies['vote_count'].notnull()) &
                       (movies['vote_average'].notnull())]
    qualified['vote_count'] = qualified['vote_count'].astype('int')
    qualified['vote_average'] = qualified['vote_average'].astype('int')
    qualified['wr'] = qualified.apply(weighted_rating, axis=1)
    qualified = qualified.sort_values('wr', ascending=False).head(10)
    return qualified
```

	title	vote_count	vote_average	release_date	wr
6142	Man with a Movie Camera	114	8	1929-01-08	7.196390
7797	Collateral	1476	7	2004-08-04	6.955555
3807	Brewster's Millions	144	6	1985-05-22	6.149201
4283	The Dead Pool	227	6	1988-07-12	6.112235
4672	Spy Game	592	6	2001-11-18	6.053712
5426	The 51st State	173	5	2001-12-07	5.518145
8	Sudden Death	174	5	1995-12-22	5.516307
6242	The League of Extraordinary Gentlemen	1181	5	2003-07-11	5.112955

Model Results & Assumptions:

1. For the movie Jumanji, Man with a Movie Camera was the highest recommended movie with a weighted rating of 7.196
2. In this specific recommendation system, we assume that the users have not watched all of the recommended movies listed

Hybrid Recommendation System

Referenced Link

Motivation:

A hybrid approach is a mixture of collaborative and content-based filtering methods while making suggestions; the film's context also considers. The user-to-item relation and the user-to-user relation also play a vital role at the time of the recommendation. This framework gives film recommendations as per the user's knowledge, provides unique recommendations, and solves a problem if the specific buyer ignores relevant data.

Approach:

Combine Content Based recommendation and Collaborative Filtering recommendation

Model:

```
def hybrid_recommendation_tfidf(title, user_id):
    #Get the recommendation movie list from the content based recommendation system
    idx = indices[title]
    #Find the cosine similarity score for this movie and
    sim_scores = list(enumerate(cosine_sim_tfidf[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:30]

    #Find the similar movies index
    movie_indices = [i[0] for i in sim_scores]
    #Get movies' info including voting
    movies = final_clean_df.iloc[movie_indices][['id', 'title', 'vote_count', 'vote_average', 'release_date']]

    movies = movies.dropna()

    movies_link = movies.merge(df_link, how='inner', left_on='id', right_on='tmdbId')

    raw_user_id = trainset.to_raw_uid(user_id)

    raw_movie_id = [trainset.to_raw_iid(int(x)) for x in movies_link['movieId']]

    movies['est_rate'] = [algo.predict(uid=raw_user_id, iid=x).est for x in raw_movie_id]

    movies = movies.sort_values('est_rate', ascending=False).head(10)

    return movies
```

Hybrid Recommendation System

Model Results:

- est_rate: estimated rating from Collaborative Filtering model

- Top 10 recommendation with most similar movies content-wise and highest estimated rating for userid 10 based on watched movie Jumanji

```
hybrid_output_tfidf = hybrid_recommendation_tfidf('Jumanji',  10)
```

```
hybrid_output_tfidf
```

	id	title	vote_count	vote_average	release_date	est_rate
3661	45905.0	The Opportunists	2.0	5.5	2000-05-17	4.382100
1382	638.0	Lost Highway	572.0	7.5	1997-01-15	4.244323
6659	48035.0	Ordet	66.0	7.8	1955-01-09	4.240881
7918	1435.0	Tarnation	22.0	7.5	2003-10-19	4.168065
8	9091.0	Sudden Death	174.0	5.5	1995-12-22	4.135744
2045	51942.0	I Married a Strange Person!	12.0	7.5	1998-08-28	4.069849
4180	26483.0	Beach Blanket Bingo	12.0	7.0	1965-04-14	4.021739
5679	13497.0	Drumline	118.0	6.2	2002-12-13	3.980511
4283	10651.0	The Dead Pool	227.0	6.3	1988-07-12	3.979477
6142	26317.0	Man with a Movie Camera	114.0	8.1	1929-01-08	3.974666

Evaluation

Evaluation by statistical metrics:

1. RMSE of predicted ratings --> Collaborative Filtering
2. Hit 10 Ratio

Evaluation by observation:

By looking at the recommendation generated by our models, we can check if the recommended movies make sense or not

Evaluation in real business context:

Measure the user engagement with metrics such as:

- Like : Unlike Ratio in our Recommendation
- Ratio of movie watched through Recommendation : Ratio of movie watched through Search

Findings & Conclusion

Based on our exploration over content-based model, collaborative filtering, neural network collaborative filtering and finally, hybrid model, we found that hybrid model gave us the most relevant recommendations based on the evaluation metrics and our observation. However, the model needs to be piloted and tested in the real-world business context. We have set up several key metrics to monitor our model's performance and we will keep tuning our model based on them.

Lessons Learnt & Future Recommendation

Lessons Learnt:

- The difference between implicit and explicit feedback
- The installation process of PyTorch and how to initiate GPU training on certain models and NLP process's
- The implementation of neural networks in creating a more complex collaborative filtering recommendation model and the final outputs of learning rate, activations, fc and output layers
- The usage of NLP tools like TFIDF, Count Vectorizer, and GloVe in content based recommendation systems
- The benefits of a hybrid recommendation system, that combines collaborative filtering and content based analysis

Recommendation:

- To try using the SVM model in creating a recommendation system
- Within the content based recommendation system, implement a function that detects if the user has watched the movie or not, and only recommend movies that the user haven't watched yet
- Hyper parameter tuning on additional layers and activation functions of the neural network collaborative filtering model

Thanks!

Any questions?

THE UNIVERSITY OF CHICAGO

References

All links are attached below the model title, the link to our data is listed as the link below:

https://www.kaggle.com/datasets/rajmehra03/movielens100k

Link to IMDB API Documentation:

https://buildmedia.readthedocs.org/media/pdf/imdbpy/latest/imdbpy.pdf

Link to TMDB API Documentation:

https://developers.themoviedb.org/3/getting-started/introduction