Movie Recommendation System

PYTHON

ASHIN K J

Abstract

The vast amount of data available on the Internet has led to the development of recommendation systems. This project proposes the use of soft computing techniques to develop recommendation systems. This report provides a detailed summary of the project “Movie Recommendation System” as part of fulfillment of the PGDS Project. The report includes a description of the topic, system architecture, and provides a detailed description of the work done till point. Included in the report are the detailed descriptions of the work done: snapshots of the implementations, various approaches, and tools used so far.

Index

Introduction

In this modern Era, entertainment plays a major role in our day to day life. As a human beings entertainment make us vibrant and positive mindset. Human brain requires to be in relaxing mood to become more productive. The companies like Amazon, Netflix, and YouTube etc. plays a major role in entertainment. Each of this company makes a huge profit through their recommendation system.

Recommendation is a term that means opening choices for the one who wanted. Every individuals have are their own entertainment, most of them likes movies, music, playing etc. We are focused on the movies, each of us have different taste like Action, Romantic, Thriller, Horror. Looking into someone history of watching or searching may predict the genre of movies like by him. There are certain peoples only watching their favorite actor's movies. Some of them are looking into proved directors or quality production house. A movie recommendation system fulfills the users needs and reduce the time of finding their taste.

Movie Recommendation Systems helps us to search our preferred movies among all of these different types of movies and hence reduce the trouble of spending a lot of time searching our favorable movies. So, it requires that the movie recommendation system should be very reliable and should provide us with the recommendation of movies which are exactly same or most matched with our preferences.

What is a recommender system?

A recommender system is a simple algorithm whose aim is to provide the most relevant information to a user by discovering patterns in a dataset. The algorithm shows the user the films that they would rate highly. An example of recommendation in action is when you visit Netflix and you notice that some films are being recommended to you and also used by Music streaming applications such as Spotify and Deezer to recommend music that you might like.

Different types of recommendation engines

The most common types of recommendation systems are **content-based** and **collaborative filtering** recommender systems. In collaborative filtering, the behavior of a group of users is used to make recommendations to other users. The recommendation is based on the preference of other users. A simple example would be recommending a movie to a user based on the fact that their friend liked the movie. There are two types of collaborative models **Memory-based** methods and **Model-based** methods. The advantage of memory-based techniques is that they are simple to implement and the resulting recommendations are often easy to explain. They are divided into two:

- **User-based collaborative filtering**: In this model, movies are recommended to a user based on the fact that the movies have been liked by users similar to the user. For example, if Derrick and Dennis like the same movies and a new movie come out that Derick like, then we can recommend that movie to Dennis because Derrick and Dennis seem to like the same movies.
- **Item-based collaborative filtering**: These systems identify similar items based on users' previous ratings. For example, if users A,B, and C gave a 5-star rating to movie name X and Y then when a user D watches movie Y they also get a recommendation to watch movie X because the system identifies movie X and Y as similar based on the ratings of users A,B, and C.

Model-based methods are based on Matrix Factorization and are better at dealing with sparsity. They are developed using data mining, machine learning algorithms to predict users' rating of unrated items. In this approach techniques such as dimensionality reduction are used to improve accuracy. Examples of such model-based methods include Decision trees, Rule-based Model, Bayesian Model, and latent factor models.

Content-based systems use metadata such as genre, producer, actor, musician to recommend items say movies. Such a recommendation would be for instance recommending Infinity War that featured Vin Diesel because someone watched and liked The Fate of the Furious. Content-based systems are based on the idea that if you liked a certain film you are most likely to like something that is similar to it.

Datasets to use for building recommender systems

In this tutorial, we are going to use the **TMDB5000 DATASET**. The dataset consists of 2 files, namely, tmdb_5000_credits.csv & tmdb_5000_movies.csv. There is another dataset called **THE MOVIES DATASET** which has more than a million movie reviews and ratings. However, I did not use it for 2 reasons.

1. The dataset is too large for the system & requires an estimate of 45–50GB RAM.

2. The machine learning model produced is also too large for Heroku. Heroku does not allow us to store more than 250MB on a free account.

Now I am not going to deploy the model in the Heroku, But for the feature sake.

Walkthrough of building a recommender system

a) Data processing

We load the 2 CSV files into df1 & df2 pandas data frames. Because The dataset consists of 2 files, namely, tmdb_5000_credits.csv and tmdb_5000_movies.csv.

	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe480092514l6c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

Figure 1: DataFrame 1 — tmdb_5000_credits.csv

production_companies	production_countries	release_date	revenue	runtime	spoken_languages	status	tagline	title	vote_average	vote_count
[{"name": "Ingenious Film Partners", "id": 289...	[{"iso_3166_1": "US", "name": "United States o...	2009-12-10	2787965087	162.0	[{"iso_639_1": "en", "name": "English"}, {"iso...	Released	Enter the World of Pandora.	Avatar	7.2	11800
[{"name": "Walt Disney Pictures", "id": 2}, {"...	[{"iso_3166_1": "US", "name": "United States o...	2007-05-19	961000000	169.0	[{"iso_639_1": "en", "name": "English"}]	Released	At the end of the world, the adventure begins.	Pirates of the Caribbean: At World's End	6.9	4500
[{"name": "Columbia Pictures", "id": 5}, {"nam...	[{"iso_3166_1": "GB", "name": "United Kingdom"...	2015-10-26	880674609	148.0	[{"iso_639_1": "fr", "name": "Fran\u00e7ais"},...	Released	A Plan No One Escapes	Spectre	6.3	4466

Figure 2: DataFrame 2 — tmdb_5000_movies.csv

Instead of handling both the data frames, I merged the data frames so that we have to work on a single data frame. The dataset thankfully does not have a large number of empty values. Let's handle them one by one. Here is an overview of all the columns.

homepage	3091
tagline	844
overview	3
runtime	2
release_date	1
production_companies	0
genres	0
id	0
keywords	0
original_language	0
original_title	0
popularity	0
crew	0
production_countries	0
cast	0
revenue	0
spoken_languages	0
status	0
title	0
vote_average	0
vote_count	0
tittle	0
budget	0

Figure 3: all columns view for NaN

Looking at the id column, which is unique for each movie, we do not need it because it will not contribute to the recommendations. Also, the tagline column should be eliminated because most of the movies have an overview and thus the tagline would result in more of a similar context. Dropping these 2 columns results in a data frame with 21 attributes.

There are multiple columns (see fig 3) where we have a string or node which contains a dictionary. We can use literal_eval from ast module to remove these strings or nodes and get the embedded dictionary. So we use literal_eval for attributes cast, keywords, crew, & genres. Now we have these attributes in the form of a dictionary, we can use these attributes and get important features

such as director names, a very important factor for our recommender system. Also for the cast, keywords, & genre attributes, we can return the top 3 names in each category in a list. Now we can create a single column which will a sum of all these 4 attributes, which are very dominant factors for our recommender system. Let's call this column "soup" (because it's like a soup/combination of 4 attributes).

Let's check out the dataset for NaN values now.

```
homepage                3091
runtime                    2
release_date               1
soup                       0
genres                     0
keywords                   0
original_language          0
original_title             0
overview                   0
popularity                 0
production_companies       0
production_countries       0
revenue                    0
director                   0
spoken_languages           0
status                     0
title                      0
vote_average               0
vote_count                 0
tittle                     0
cast                       0
crew                       0
budget                     0
dtype: int64
```

Figure 4: updated all columns view

Since our homepage has a lot of empty values, we have no other option but to drop it. We can also fill the runtime empty values with the mean value. Since we have one movie which is unreleased, we can drop that particular row, since the movie is unreleased. Now we have our final dataset which is ready for some machine learning modeling.

b) Building Movie Recommender Machine Learning Model

To build our model, we first create a count matrix that is created by the help of a **count vectorizer**. We create a count vector with English stopwords & fit and transform over the soup column we just created in the previous section. Scikit-learn has a very beautiful method called **cosine similarity**. It is simply a metric that is used to determine how similar documents are, irrespective of their size. After building the cosine similarity matrix for our dataset, we can now sort the results to find out the top 10 similar movies. We return the movie title & indexes to the user.

c) Creating Web Pages and connect it to Flask Rendering

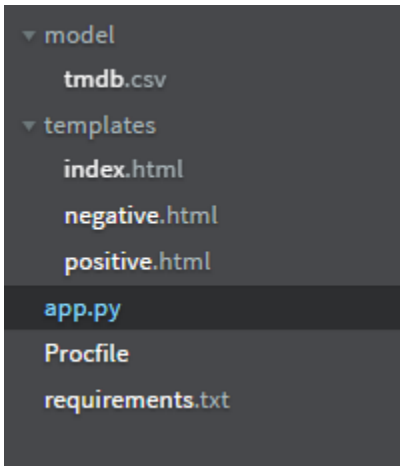


Figure 5: Directory setup

The webpage needs to look very simple yet informative enough for everyone to understand. Also, I love to design good UI effects so that the user feels very good.

I create a div with a class named movie and provide minimal information regarding the recommendation system. I provide the main input field inside a form where the user input's the movie name and the form is submitted by the action of the button. Upon submission, the movie name is captured at the backend and further processed and also provide good effects on the div and other CSS properties so that page looks great and very pleasant.

Now we have created the index.html web page, we need to connect it to the flask and render it when the link is initially opened. So let's jump to the basics of the flask web application framework to render it for the user.

We use the following code in app.py to simply render the page we just created.

```
import flaskapp = flask.Flask(__name__, template_folder='templates')

# Set up the main route
@app.route('/', methods=['GET', 'POST'])

def main():
   if flask.request.method == 'GET':
     return(flask.render_template('index.html'))
```

Now that we have our index.html rendered, let's hope that the user enters a movie name. Upon entering, the user clicks on the submit button and the form is submitted.

d) Create a complete interface and exception handling using python along with form validation

Now we have a movie name, which is submitted by the user in the form. Let's hold this name into the m_name variable in python. We accept the form submission using the post method.

```
if flask.request.method == 'POST':
m_name = flask.request.form['movie_name']
m_name = m_name.title()
```

We also convert the input movie name to the title format. The title form will simply convert every character of each word to upper case. Now we have 2 options:

- If the input movie name is misspelled or does not exist in the database.

  — If wrong, show error page & possible similar movie name based on the input.

- If a correct movie name is entered & present in the database, then show the recommendations.

Here is the code for doing the same.

```
if m_name not in all_titles:
  return(flask.render_template('negative.html',name=m_name))
else:
  result_final = get_recommendations(m_name)
  names = []
  dates = []
  for i in range(len(result_final)):
    names.append(result_final.iloc[i][0])
    dates.append(result_final.iloc[i][1])

return
flask.render_template('positive.html',movie_names=names,movie_date
=dates,search_name=m_name)
```

Let's carefully look into positive.html & negative.html.

1. **Negative.html**

negative.html is rendered if the input from the user does not match with all_titles list which contains all the movie names present in the database.

Negative.html page simply shows possible reason(s) for not able to find the searched movie. It also searches throughout the whole database and use of word search techniques likes (**difflib.get_close_matches() method** in python & **Levenshtein distance method**) to find the closest match and suggests the user about the movie names which are very similar to the one user entered. All this is done using javascript and finally rendered on the HTML page.

2. Positive.html

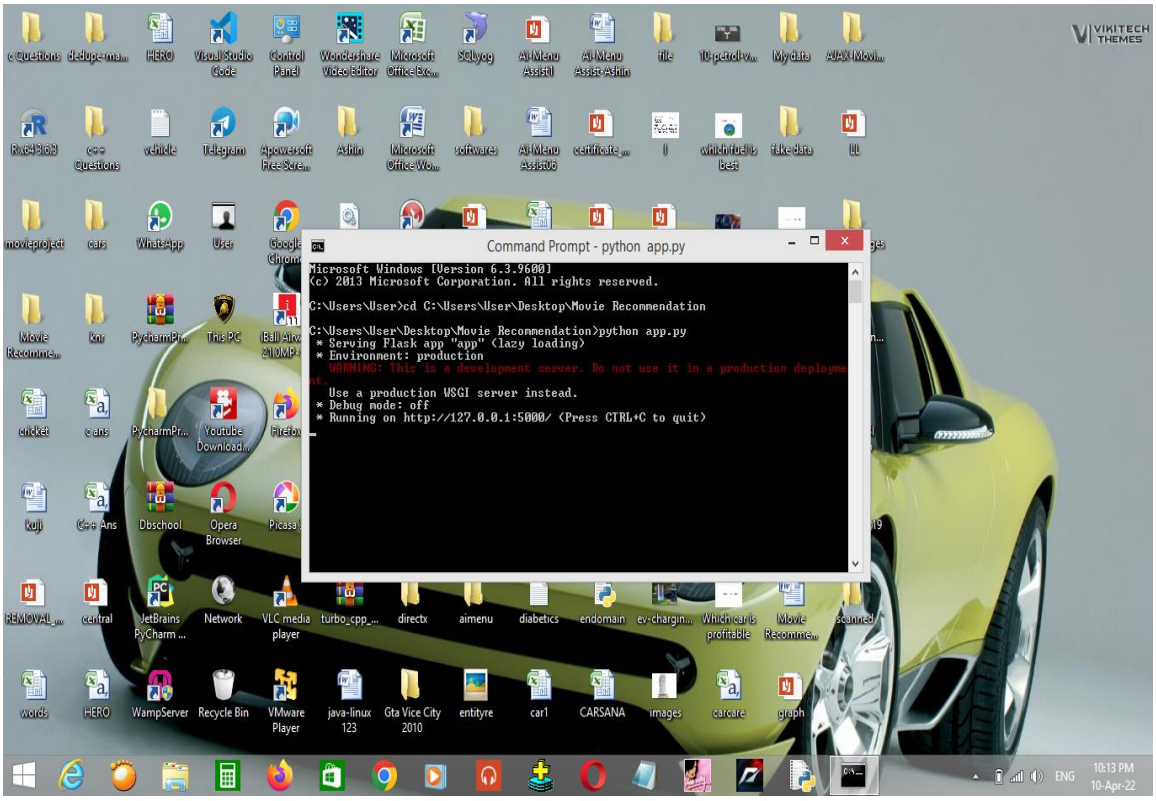
Positive.html is rendered if the input movie name matches with the database. If so, we call the get_recommendations function by passing the movie name. The get_recommendations function is the same as we have discussed in section 2. We take the movie name, calculate the cosine matrix with respect to the dataset and find the most similar movie to the input movie. We sort the results and return back top 10 results. We send similar movie names as well as their release date in a list to the positive.html. We create a tabular layout and print the 10 movies along with their release dates.

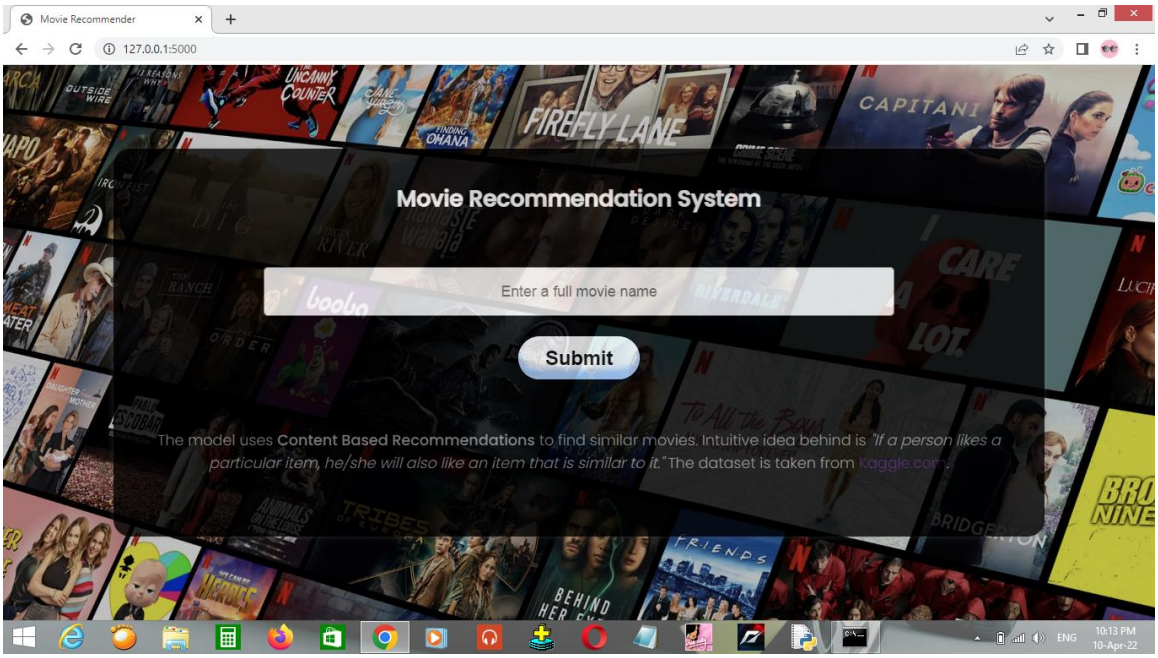
The UI part is purely on CSS & Javascript. It is based on my personal interest in making a great pleasant design website, however, it is completely optional and the app still works just fine.

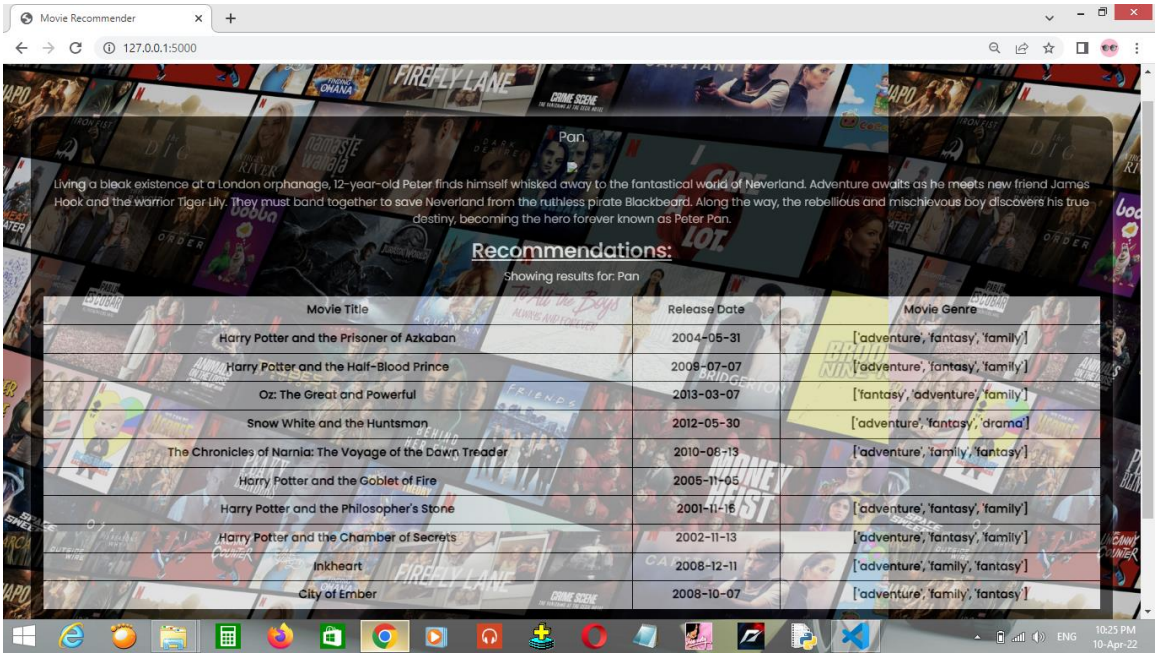
How to improve the recommendation system

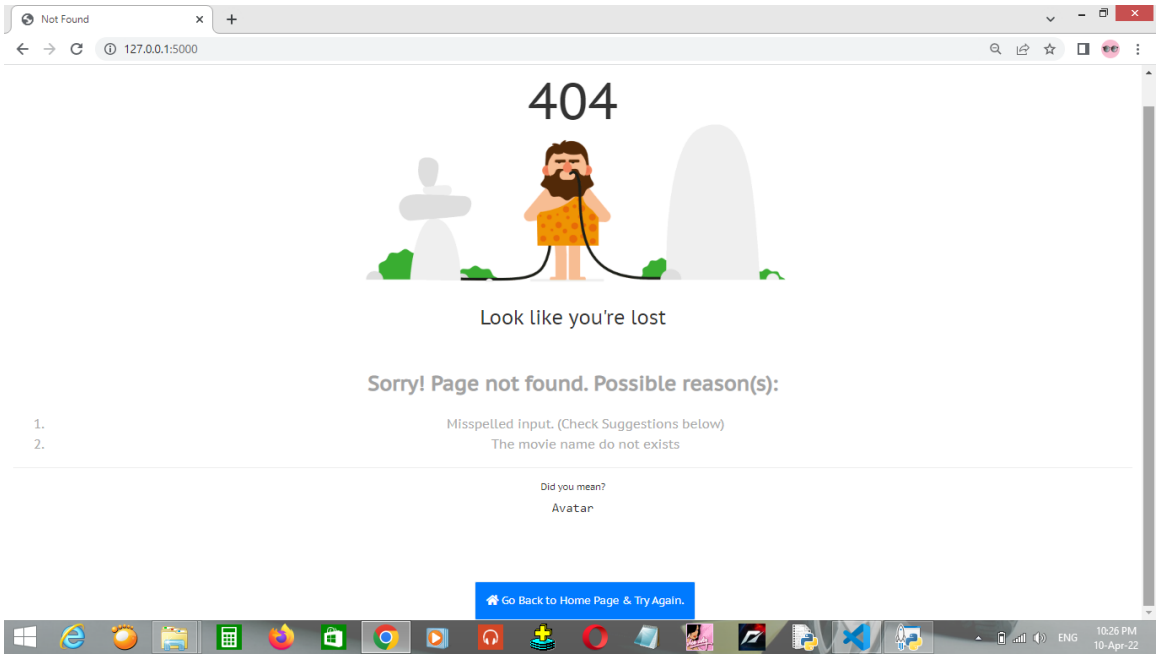
This system can be improved by building a Memory-Based Collaborative Filtering based system. In this case, we'd divide the data into a training set and a test set. We'd then use techniques such as cosine similarity to compute the similarity between the movies. An alternative is to build a Model-based Collaborative Filtering system. This is based on matrix factorization. Matrix factorization is good at dealing with scalability and sparsity than the former. You can then evaluate your model using techniques such as Root Mean Squared Error (RMSE).

Snapshots


Command Prompt - python app.py
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.
C:\Users\User>cd C:\Users\User\Desktop\Movie Recommendation
C:\Users\User\Desktop\Movie Recommendation>python app.py
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)


Movie Recommender
127.0.0.1:5000
Movie Recommendation System
Enter a full movie name
Submit
The model uses Content Based Recommendations to find similar movies. Intuitive idea behind is "If a person likes a particular item, he/she will also like an item that is similar to it." The dataset is taken from Kaggle.com.


Movie Recommender
127.0.0.1:5000
Pan
Living a bleak existence at a London orphanage, 12-year-old Peter finds himself whisked away to the fantastical world of Neverland. Adventure awaits as he meets new friend James Hook and the warrior Tiger Lily. They must band together to save Neverland from the ruthless pirate Blackbeard. Along the way, the rebellious and mischievous boy discovers his true destiny, becoming the hero forever known as Peter Pan.
Recommendations:
Showing results for: Pan
Movie Title | Release Date | Movie Genre
Harry Potter and the Prisoner of Azkaban | 2004-05-31 | ['adventure', 'fantasy', 'family']
Harry Potter and the Half-Blood Prince | 2008-07-07 | ['adventure', 'fantasy', 'family']
Oz: The Great and Powerful | 2013-03-07 | ['fantasy', 'adventure', 'family']
Snow White and the Huntsman | 2012-05-30 | ['adventure', 'fantasy', 'drama']
The Chronicles of Narnia: The Voyage of the Dawn Treader | 2010-08-13 | ['adventure', 'family', 'fantasy']
Harry Potter and the Goblet of Fire | 2005-11-05 | ['adventure', 'fantasy', 'family']
Harry Potter and the Philosopher's Stone | 2001-11-16 | ['adventure', 'fantasy', 'family']
Harry Potter and the Chamber of Secrets | 2002-11-13 | ['adventure', 'fantasy', 'family']
Inkheart | 2008-12-11 | ['adventure', 'family', 'fantasy']
City of Ember | 2008-10-07 | ['adventure', 'family', 'fantasy']


Not Found
127.0.0.1:5000
404
Look like you're lost
Sorry! Page not found. Possible reason(s):
1. Misspelled input. (Check Suggestions below)
2. The movie name do not exists
Did you mean?
Avatar
Go Back to Home Page & Try Again.
ENG
10:26 PM
10-Apr-22

Conclusion

This was a simplified explanation on how a recommendation system, similar to that used by YouTube or Netflix, actually works.

We also believe that recommender systems can democratize access to long-tail products, services, and information, because machines have a much better ability to learn from vastly bigger data pools than expert humans, thus can make useful predictions for areas in which human capacity simply is not adequate to have enough experience to generalize usefully at the tail.