IMDB_100

February 21, 2025

1 AI/Machine Learning Intern Challenge

1.1 Introduction

In this project, I will be building a content-based recommendation system.I want to help users find movies they would like to watch based on text description.

The goal of this system is to process a user query and provide the top 3–5 movie titles that closely match the description provided by the user. By using TF-IDF (Term Frequency-Inverse Document Frequency) and cosine similarity, the system will be able to identify which movies are most relevant to the user's preferences based on their title and description.

In this project I will begin with:

1. **Importing Necessary Libraries & Dataset**

To start, I will import the libraries, packages and dataset needed for this project. These include libraries for data manipulation, text processing, machine learning, and similarity calculations.

2. **Data Cleaning**

   The next step involves cleaning the movie dataset. I will:

   -Check for missing values and handle them appropriately (either by removing or filling them).

   -Identify and remove duplicate rows to ensure the dataset is unique and does not distort results.

3. **Data Preprocessing**

   Once the data is cleaned, I will move on to preprocessing the text data. This step is needed for NLP tasks. The following may be performed:

-Removing special characters and numbers from the movie descriptions can keep the text clean and relevant.

-Eliminating punctuation since it's not meaningful for many text analysis.

-Lowercasing which standardizes the text.

-Removing stop words, which are common words (e.g., "and," "the," "is") that don't add significant meaning to the analysis.

-Tokenizing the descriptions into individual words, which will allow for further analysis and vectorization.

-Applying stemming and lemmatization to reduce words to their root form (e.g., “running” becomes “run”) to standardize the text and improve matching.

These preprocessing steps are important to ensure that the text data is ready for use in building the recommendation system.

1.1.1 Import and Get to Know Your Data

[1]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import re

import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

# Download necessary NLTK resources
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt')  # Tokenizer
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/mariselamonterroza/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/mariselamonterroza/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     /Users/mariselamonterroza/nltk_data…
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/mariselamonterroza/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[1]:
```
True
```

[2]:
```
df = pd.read_csv("IMDB_100.csv")
```

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Name         100 non-null    object
```

```
 1   Genre         100 non-null    object
 2   Description   100 non-null    object
 3   Year          100 non-null    int64
 4   Runtime       100 non-null    object
dtypes: int64(1), object(4)
memory usage: 4.0+ KB
```

[4]:
```
columns_to_keep = ['Name', 'Description']
```

[5]:
```
df = df[columns_to_keep]
```

[6]:
```
print(df.head())
```

```
                          Name  \
0  Das Cabinet des Dr. Caligari
1                     Nosferatu
2                   Safety Last!
3                 The Gold Rush
4                   The General

                                         Description
0  Hypnotist Dr. Caligari uses a somnambulist, Ce…
1  Vampire Count Orlok expresses interest in a ne…
2  A boy leaves his small country town and heads …
3  A prospector goes to the Klondike during the 1…
4  After being rejected by the Confederate milita…
```

1.1.2 Clean The Data

[7]:
```
print(df.isnull().sum())  # Shows count of missing values per column
```

```
Name           0
Description    0
dtype: int64
```

[8]:
```
print(df.duplicated().sum()) # shows count of duplicates per column
```

```
0
```

[9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Name         100 non-null    object
 1   Description  100 non-null    object
dtypes: object(2)
memory usage: 1.7+ KB
```

1.2 Preprocess The Data

To preprocess the data, I will begin by focusing on the description column to ensure it is lemmatized before combining it with the name column.

Why do this?

will only lemmatize the description because:

1. Titles can contain proper names that shouldn't be changed
2. Descriptions may contain more context and words where lemmatization can reduce a word to its root form I will first clean and lemmatize the description and then combine it with the name column to create a unified field.

Additionally, I will use lemmatization instead of stemming.

Why lemmatization instead of stemming?

Stemming can alter meanings while lemmatization is more accurate. Since the description often references the title and vice versa, maintaining accuracy is crucial. For example, if the movie description says, "The man is fasting the entire month of May," stemming may reduce "fasting" to "fast," which changes the meaning. We don't want that. The lemmatizer will preserve "fasting," maintaining the intended meaning.

The preprocess I will clean the columns by lowercasing,looking at numbers, special characters and then tokenize.

To enhance the matching process, I will combine the title and description into a single text field.However,

Once the columns are combined, I will drop any unnecessary ones.

After these steps, my data will be ready for the actual user input process.

[10]:
```
# Function to clean text by removing special characters and lowercasing
def clean_text(text):
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text)  # Remove special characters
    text = text.lower()  # Convert to lowercase
    return text
```

[11]:
```
# Apply text cleaning( remove SC + Lowercase) to 'Name' and 'Description'␣
↪columns
df['Name_clean'] = df['Name'].apply(clean_text)
df['Description_clean'] = df['Description'].apply(clean_text)
```

[12]:
```
# Function to tokenize text
def tokenize(text):
    tokens = word_tokenize(text)
    # Add brackets around each token
    return ' '.join(f'[{token}]' for token in tokens)
```

[13]:
```
# Function to tokenize and lemmatize text
lemmatizer = WordNetLemmatizer()
```

```
def tokenize_and_lemmatize(text):
    tokens = word_tokenize(text)
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in tokens]
    # Add brackets around each token
    return ' '.join(f'[{token}]' for token in lemmatized_tokens)
```

[14]:
```
df['Name_tokenized'] = df['Name_clean'].apply(tokenize)
df['Description_tokenized'] = df['Description_clean'].
 ↪apply(tokenize_and_lemmatize)
```

[15]:
```
# Combine cleaned and tokenized Name and Description into one column
df['IMDB_100_output'] = df['Name_tokenized'] + " " + df['Description_tokenized']
```

[16]:
```
# Display rows that contain numbers in the 'Name' column
rows_with_numbers_name = df[df['Name_tokenized'].str.contains(r'\d',␣
 ↪regex=True)]
print("Rows with numbers in Name column:")
print(rows_with_numbers_name)
```

```
Rows with numbers in Name column:
                    Name                                        Description  \
26           42nd Street  When the leading lady of a Broadway musical br…
29  Gold Diggers of 1933  A wealthy composer rescues unemployed Broadway…

              Name_clean                                    Description_clean  \
26           42nd street  when the leading lady of a broadway musical br…
29  gold diggers of 1933  a wealthy composer rescues unemployed broadway…

                 Name_tokenized  \
26               [42nd] [street]
29  [gold] [diggers] [of] [1933]

                                Description_tokenized  \
26  [when] [the] [leading] [lady] [of] [a] [broadw…
29  [a] [wealthy] [composer] [rescue] [unemployed]…

                                      IMDB_100_output
26  [42nd] [street] [when] [the] [leading] [lady] …
29  [gold] [diggers] [of] [1933] [a] [wealthy] [co…
```

[17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 7 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Name                  100 non-null    object
```

```
 1   Description            100 non-null    object
 2   Name_clean             100 non-null    object
 3   Description_clean      100 non-null    object
 4   Name_tokenized         100 non-null    object
 5   Description_tokenized  100 non-null    object
 6   IMDB_100_output        100 non-null    object
dtypes: object(7)
memory usage: 5.6+ KB
```

[18]:
```
# Drop unnecessary columns after processing
df.drop(['Description_tokenized', 'Name_clean',␣
 ↪'Description_clean','Name_tokenized'], axis=1, inplace=True)
```

[20]:
```
# Save my cleaned data to a CSV file
df.to_csv("IMDB_100clean.csv", index=False)
```

1.3 Say It, Watch It: Movie Suggestions Made Simple!

This is the exciting part!

In this step, I will take the user's input and convert it into a format compatible with the movie titles and descriptions.The user's input will be transformed into a vector representation using TF-IDF. Afterward, I will calculate the cosine similarity between the user input and the movie data to recommend the top 5 movies that closely match the user's preferences based on their input. This allows for personalized and relevant movie suggestions based on user queries like,

"I really like scary movies with ghosts"

Basically think of it as a personalized movie genie granting recommendations!

[21]:
```
vectorizer = TfidfVectorizer(stop_words='english',ngram_range=(1,␣
 ↪2),max_features=500)#transforms text data->numerical
```

[22]:
```
#Step 2: Vectorize the 'IMDB_100_output' column (combined text)
tfidf_matrix = vectorizer.fit_transform(df['IMDB_100_output'])
```

[23]:
```
# Function to recommend movies
def recommend_movies(user_input, tfidf_matrix, vectorizer, df, top_n=5):
    user_input_vector = vectorizer.transform([user_input])  # Vectorize user␣
 ↪input
    cosine_sim = cosine_similarity(user_input_vector, tfidf_matrix)  #␣
 ↪Calculate similarity

    # Get indices of top N most similar movies
    top_indices = cosine_sim[0].argsort()[-top_n:][::-1]

    # Prepare recommendations
    recommendations = [
        {
```

```
            "Movie": df.iloc[idx]['Name'],
            "Description": df.iloc[idx]['Description'],
            "Similarity Score": cosine_sim[0][idx]
        }
        for idx in top_indices
    ]

    return recommendations
```

[24]:
```
# Example user inputs
user_inputs = [
    "Crime movies with detectives",
    "Musicals about Broadway",
    "I enjoy watching the hero save the day"
]
```

[25]:
```
# Loop through each user input and get recommendations
for user_input in user_inputs:
    print(f"\nRecommendations for: {user_input}\n")
    top_movies = recommend_movies(user_input, tfidf_matrix, vectorizer, df,␣
 ↪top_n=5)
    for movie in top_movies:
        print(f"Movie: {movie['Movie']}")
        print(f"Description: {movie['Description']}")
        print(f"Similarity Score: {movie['Similarity Score']:.4f}")
        print("-" * 50)
```

```
Recommendations for: Crime movies with detectives

Movie: Rope
Description: Two men attempt to prove they committed the perfect crime by
hosting a dinner party after strangling their former classmate to death.
Similarity Score: 0.2881
--------------------------------------------------
Movie: Gun Crazy
Description: Two disturbed young people release their fascination with guns
through a crime spree.
Similarity Score: 0.2870
--------------------------------------------------
Movie: The Public Enemy
Description: An Irish-American street punk tries to make it big in the world of
organized crime.
Similarity Score: 0.2807
--------------------------------------------------
Movie: Ace in the Hole
Description: A frustrated former big-city journalist now stuck working for an
Albuquerque newspaper exploits a story about a man trapped in a cave to rekindle
```

his career, but the situation quickly escalates into an out-of-control circus.
Similarity Score: 0.0000

Movie: The Scarlet Empress
Description: A German noblewoman enters into a loveless marriage with the dim-witted, unstable heir to the Russian throne, then plots to oust him from power.
Similarity Score: 0.0000

Recommendations for: Musicals about Broadway

Movie: Dames
Description: A multimillionaire decides to boycott "filthy" forms of entertainment such as Broadway shows.
Similarity Score: 0.6461

Movie: All About Eve
Description: A seemingly timid but secretly ruthless ingénue insinuates herself into the lives of an aging Broadway star and her circle of theater friends.
Similarity Score: 0.5066

Movie: 42nd Street
Description: When the leading lady of a Broadway musical breaks her ankle, she is replaced by a young unknown actress, who becomes the star of the show.
Similarity Score: 0.3849

Movie: Twentieth Century
Description: A flamboyant Broadway impresario who has fallen on hard times tries to get his former lover, now a Hollywood diva, to return and resurrect his failing career.
Similarity Score: 0.2843

Movie: Gold Diggers of 1933
Description: A wealthy composer rescues unemployed Broadway performers with a new play, but insists on remaining anonymous.
Similarity Score: 0.2663

Recommendations for: I enjoy watching the hero save the day

Movie: D.O.A.
Description: Frank Bigelow, told he's been poisoned and has only a few days to live, tries to find out who killed him and why.
Similarity Score: 0.3792

Movie: Arsenic and Old Lace
Description: A Brooklyn writer of books on the futility of marriage risks his reputation when he decides to tie the knot. Things get even more complicated

```
when he learns on his wedding day that his beloved maiden aunts are habitual
murderers.
Similarity Score: 0.3781
--------------------------------------------------
Movie: Jour de fête
Description: A village postman with no sense of humour delivers his mail via
bicycle on the day the travelling fair comes to town. He is disrupted by a short
film about US speed and efficiency and the playful teasing of the village folk.
Similarity Score: 0.2317
--------------------------------------------------
Movie: Dracula
Description: Transylvanian vampire Count Dracula bends a naive real estate agent
to his will, then takes up residence at a London estate where he sleeps in his
coffin by day and searches for potential victims by night.
Similarity Score: 0.1867
--------------------------------------------------
Movie: Ace in the Hole
Description: A frustrated former big-city journalist now stuck working for an
Albuquerque newspaper exploits a story about a man trapped in a cave to rekindle
his career, but the situation quickly escalates into an out-of-control circus.
Similarity Score: 0.0000
--------------------------------------------------
```

1.3.1 Let's try another one!

[26]:
```
user_inputs = [
    "i really like movies about cowboys and horses in the country"  # New input␣
 ↪here
]

# Run only this part again
for user_input in user_inputs:
    print(f"\nRecommendations for: {user_input}\n")
    top_movies = recommend_movies(user_input, tfidf_matrix, vectorizer, df,␣
 ↪top_n=5)
    for movie in top_movies:
        print(f"Movie: {movie['Movie']}")
        print(f"Description: {movie['Description']}")
        print(f"Similarity Score: {movie['Similarity Score']:.4f}")
        print("-" * 50)
```

```
Recommendations for: i really like movies about cowboys and horses in the
country

Movie: Dead of Night
Description: The guests invited to weekend in the country share their
supernatural stories, beginning with Walter Craig, who senses impending doom as
```

```
his half-remembered recurring dream turns into reality.
Similarity Score: 0.3283
--------------------------------------------------
Movie: Safety Last!
Description: A boy leaves his small country town and heads to the big city to
get a job. As soon as he makes it big his sweetheart will join him and marry
him. His enthusiasm to get ahead leads to some interesting adventures.
Similarity Score: 0.2468
--------------------------------------------------
Movie: Ace in the Hole
Description: A frustrated former big-city journalist now stuck working for an
Albuquerque newspaper exploits a story about a man trapped in a cave to rekindle
his career, but the situation quickly escalates into an out-of-control circus.
Similarity Score: 0.0000
--------------------------------------------------
Movie: Twentieth Century
Description: A flamboyant Broadway impresario who has fallen on hard times tries
to get his former lover, now a Hollywood diva, to return and resurrect his
failing career.
Similarity Score: 0.0000
--------------------------------------------------
Movie: Duck Soup
Description: Rufus T. Firefly is named the dictator of bankrupt Freedonia and
declares war on neighboring Sylvania over the love of his wealthy backer Mrs.
Teasdale, contending with two inept spies who can't seem to keep straight which
side they're on.
Similarity Score: 0.0000
--------------------------------------------------
```

[27]:
```
print("Movie recommendations generated successfully!")
```

```
Movie recommendations generated successfully!
```

[28]:
```
print("Salary Expectation Per Month:$1600")
```

```
Salary Expectation Per Month:$1600
```

[ ]: