Netflix Project

October 25, 2024

[38]:
```python
import pandas as pd

#Load the dataset
df = pd.read_csv('netflix_titles.csv', encoding='latin1')

#Remove 'Unnamed' columns
df = df.loc[:, ~df.columns.str.contains('^Unnamed')]

#Check the remaining columns to ensure cleanup
print(df.columns)
```

```
Index(['show_id', 'type', 'title', 'director', 'cast', 'country', 'date_added',
       'release_year', 'rating', 'duration', 'listed_in', 'description'],
      dtype='object')
```

[57]:
```
!pip install sklearn
!pip install gensim
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Collecting sklearn
  Downloading sklearn-0.0.post12.tar.gz (2.6 kB)
  Preparing metadata (setup.py) … error
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
   exit code: 1
   > [15 lines of output]
      The 'sklearn' PyPI package is deprecated, use 'scikit-learn'
      rather than 'sklearn' for pip commands.

      Here is how to fix this error in the main use cases:
      - use 'pip install scikit-learn' rather than 'pip install
sklearn'
      - replace 'sklearn' by 'scikit-learn' in your pip requirements
files
        (requirements.txt, setup.py, setup.cfg, Pipfile, etc …)
      - if the 'sklearn' package is used by one of your dependencies,
```

```
        it would be great if you take some time to track which package
uses
        'sklearn' instead of 'scikit-learn' and report it to their
issue tracker
      - as a last resort, set the environment variable
        SKLEARN_ALLOW_DEPRECATED_SKLEARN_PACKAGE_INSTALL=True to avoid
this error

      More information is available at
      https://github.com/scikit-learn/sklearn-pypi-package
      [end of output]

  note: This error originates from a subprocess, and is likely not a
problem with pip.
error: metadata-generation-failed

× Encountered error while generating package metadata.
 > See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
Defaulting to user installation because normal site-packages is not
writeable
Looking in links: /usr/share/pip-wheels
Requirement already satisfied: gensim in
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages (4.3.0)
Requirement already satisfied: numpy>=1.18.5 in
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages (from
gensim) (1.26.4)
Requirement already satisfied: scipy>=1.7.0 in
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages (from
gensim) (1.12.0)
Requirement already satisfied: smart-open>=1.8.1 in
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages (from
gensim) (5.2.1)
Collecting FuzzyTM>=0.4.0 (from gensim)
  Downloading FuzzyTM-2.0.9-py3-none-any.whl.metadata (7.9 kB)
Requirement already satisfied: pandas in
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages (from
FuzzyTM>=0.4.0->gensim) (2.1.4)
Collecting pyfume (from FuzzyTM>=0.4.0->gensim)
  Downloading pyFUME-0.3.4-py3-none-any.whl.metadata (9.7 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages (from
pandas->FuzzyTM>=0.4.0->gensim) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages (from
pandas->FuzzyTM>=0.4.0->gensim) (2023.3.post1)
```

```
Requirement already satisfied: tzdata>=2022.1 in
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages (from
pandas->FuzzyTM>=0.4.0->gensim) (2023.3)
Collecting scipy>=1.7.0 (from gensim)
  Downloading
scipy-1.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(58 kB)
                           58.9/58.9

kB 1.4 MB/s eta 0:00:00.9 MB/s eta 0:00:01
Collecting numpy>=1.18.5 (from gensim)
  Downloading
numpy-1.24.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(5.6 kB)
Collecting simpful==2.12.0 (from pyfume->FuzzyTM>=0.4.0->gensim)
  Downloading simpful-2.12.0-py3-none-any.whl.metadata (4.8 kB)
Collecting fst-pso==1.8.1 (from pyfume->FuzzyTM>=0.4.0->gensim)
  Downloading fst-pso-1.8.1.tar.gz (18 kB)
  Preparing metadata (setup.py) … done
Collecting pandas (from FuzzyTM>=0.4.0->gensim)
  Downloading
pandas-1.5.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(11 kB)
Collecting miniful (from fst-pso==1.8.1->pyfume->FuzzyTM>=0.4.0->gensim)
  Downloading miniful-0.0.6.tar.gz (2.8 kB)
  Preparing metadata (setup.py) … done
Requirement already satisfied: six>=1.5 in
/opt/conda/envs/anaconda-2024.02-py310/lib/python3.10/site-packages (from
python-dateutil>=2.8.2->pandas->FuzzyTM>=0.4.0->gensim) (1.16.0)
Downloading FuzzyTM-2.0.9-py3-none-any.whl (31 kB)
Downloading pyFUME-0.3.4-py3-none-any.whl (60 kB)
                         60.3/60.3

kB 1.3 MB/s eta 0:00:007.4 MB/s eta 0:00:01
Downloading
numpy-1.24.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.3
MB)
                         17.3/17.3

MB 22.6 MB/s eta 0:00:00 0:00:01[36m0:00:01
Downloading
scipy-1.10.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (34.4
MB)
                         34.4/34.4

MB 31.0 MB/s eta 0:00:00 0:00:01[36m0:00:01
Downloading
pandas-1.5.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.1
MB)
```

```
                          12.1/12.1

MB 56.7 MB/s eta 0:00:00 0:00:0136m0:00:01
Downloading simpful-2.12.0-py3-none-any.whl (24 kB)
Building wheels for collected packages: fst-pso, miniful
  Building wheel for fst-pso (setup.py) … done
  Created wheel for fst-pso: filename=fst_pso-1.8.1-py3-none-any.whl
size=20430
sha256=a0f5c5d0d49914af2b17c4855580ab75ddc84755614163d8201173feadce6be5
  Stored in directory: /home/973bca21-bf4b-487c-bf96-
64fcfb0d18bd/.cache/pip/wheels/2d/1b/42/88a19f6b3896c2230d5053832f208976cddf7062
5885201d06
  Building wheel for miniful (setup.py) … done
  Created wheel for miniful: filename=miniful-0.0.6-py3-none-any.whl
size=3507
sha256=05497db49f6348d430801eecd1e044f9f1a47cd903d87cf161bdbfdbb96ddb0d
  Stored in directory: /home/973bca21-bf4b-487c-bf96-
64fcfb0d18bd/.cache/pip/wheels/5b/86/8f/7bb7f6472e2c84de7addfc1a5cd7fd647f00d8fb
640da9ea9a
Successfully built fst-pso miniful
Installing collected packages: numpy, scipy, pandas, simpful, miniful, fst-pso,
pyfume, FuzzyTM
  WARNING: The scripts f2py, f2py3 and f2py3.10 are installed in

'/home/973bca21-bf4b-487c-bf96-64fcfb0d18bd/.local/bin' which is not on PATH.

  Consider adding this directory to PATH or, if you prefer to suppress this

warning, use --no-warn-script-location.

Successfully installed FuzzyTM-2.0.9 fst-pso-1.8.1 miniful-0.0.6
numpy-1.24.4 pandas-1.5.3 pyfume-0.3.4 scipy-1.10.1 simpful-2.12.0
```

[40]:
```python
#SENTIMENT ANALYSIS ON MOVIE DESCRIPTIONS USING TOPIC MODELING LDA
from sklearn.feature_extraction.text import CountVectorizer
from gensim import corpora
from gensim.models import LdaModel
import re

#Preprocess text: Clean the description, tokenize, and remove stop words
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS

#Enhanced cleaning function to handle unwanted characters
def clean_for_lda(text):
    text = text.encode('ascii', 'ignore').decode('ascii')  # Remove any␣
↪non-ascii characters
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    tokens = text.lower().split()  # Tokenize and convert to lowercase
    tokens = [word for word in tokens if word not in ENGLISH_STOP_WORDS]  #␣
↪Remove stop words
```

```
    return tokens

#Apply the enhanced clean function to the description column
df['cleaned_description'] = df['description'].apply(clean_for_lda)

#Recreate the dictionary and corpus for LDA
dictionary = corpora.Dictionary(df['cleaned_description'])
corpus = [dictionary.doc2bow(text) for text in df['cleaned_description']]

#Train the LDA model again
lda_model = LdaModel(corpus, num_topics=5, id2word=dictionary, passes=10)

#Display the updated topics
topics = lda_model.print_topics(num_words=10)
for topic in topics:
    print(f"Topic {topic[0]}: {topic[1]}")
```

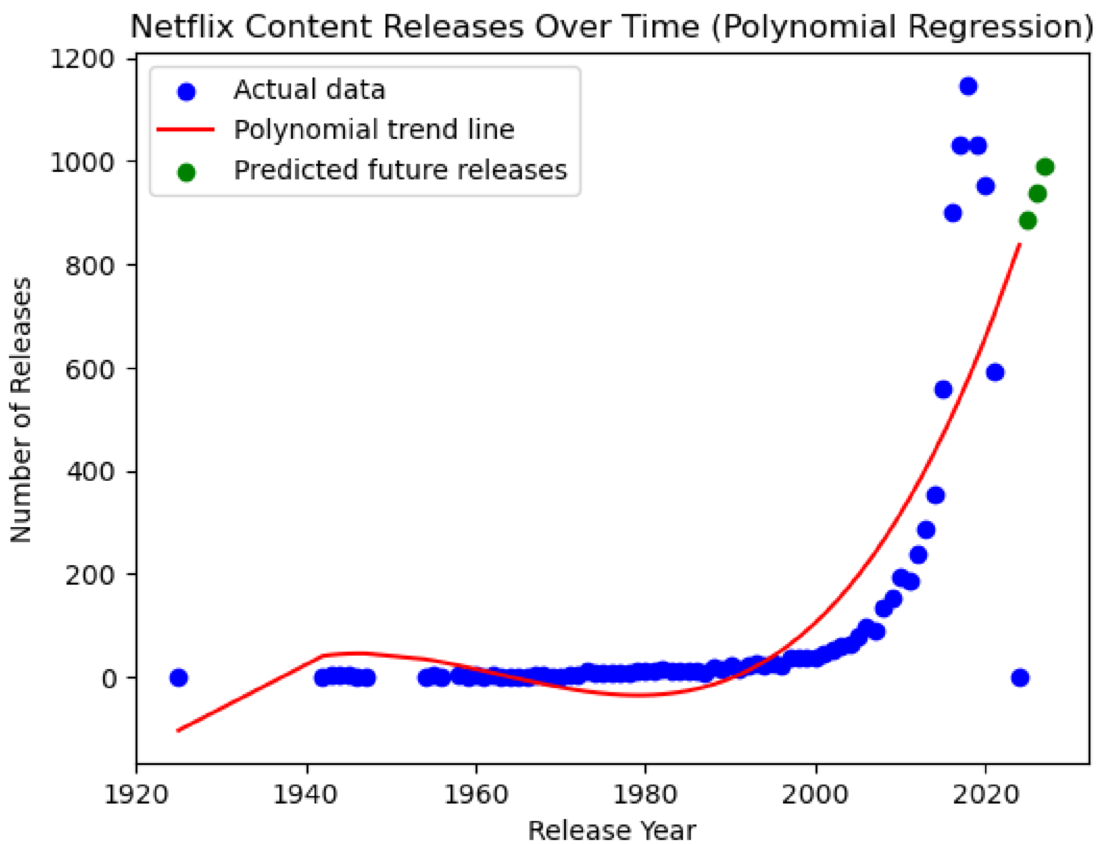
```
Topic 0: 0.007*"life" + 0.004*"series" + 0.003*"documentary" + 0.003*"special" +
0.003*"family" + 0.003*"takes" + 0.003*"new" + 0.003*"years" + 0.003*"history" +
0.003*"standup"
Topic 1: 0.008*"new" + 0.008*"young" + 0.007*"school" + 0.007*"friends" +
0.006*"family" + 0.005*"woman" + 0.005*"high" + 0.005*"man" + 0.005*"help" +
0.004*"love"
Topic 2: 0.012*"young" + 0.006*"man" + 0.006*"world" + 0.005*"woman" +
0.005*"new" + 0.004*"murder" + 0.004*"death" + 0.003*"war" + 0.003*"family" +
0.003*"love"
Topic 3: 0.009*"life" + 0.005*"new" + 0.004*"family" + 0.004*"love" +
0.004*"documentary" + 0.004*"series" + 0.003*"young" + 0.003*"women" +
0.003*"years" + 0.003*"man"
Topic 4: 0.008*"documentary" + 0.006*"world" + 0.006*"series" + 0.005*"life" +
0.004*"war" + 0.003*"new" + 0.003*"takes" + 0.003*"lives" + 0.002*"love" +
0.002*"film"
```

[46]:

```
#GRAPH NETFLIX RELEASES OVER TIME USING LINEAR REGRESSION W/ POLYNOMIAL␣
↪PROCESSING

#Group data by release year and count the number of releases for each year
content_by_year = df.groupby('release_year').size().
↪reset_index(name='num_releases')

#Polynomial regression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt

#Prepare the data for polynomial regression
```

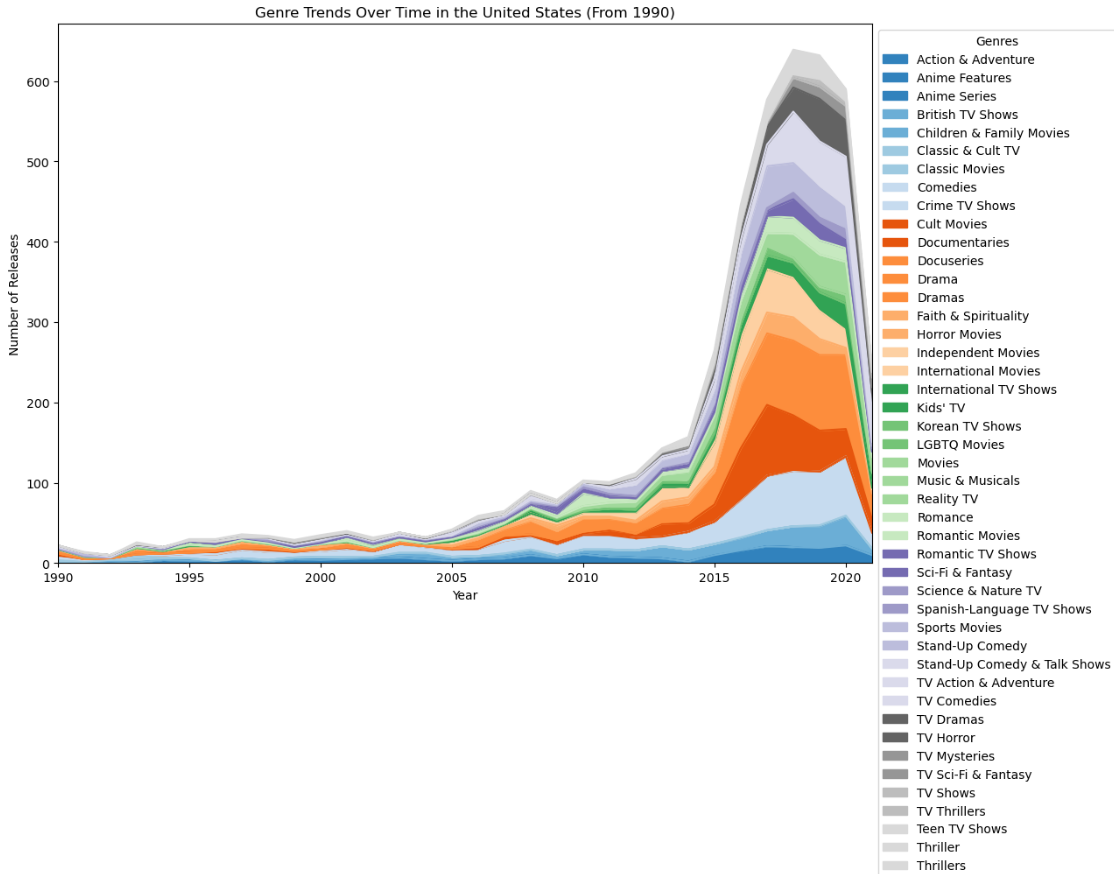
```python
X = np.array(content_by_year['release_year']).reshape(-1, 1)  # Release year
y = np.array(content_by_year['num_releases'])  # Number of releases

#Apply polynomial preprocessing (let's start with degree 3)
poly = PolynomialFeatures(degree=3)
X_poly = poly.fit_transform(X)

#Fit the polynomial regression model
poly_reg = LinearRegression()
poly_reg.fit(X_poly, y)

#Predict future releases for 2025, 2026, and 2027
X_future = np.array([[2025], [2026], [2027]])
X_future_poly = poly.transform(X_future)
predictions = poly_reg.predict(X_future_poly)

#Print predictions
print(f"Predicted releases for 2025: {predictions[0]}")
print(f"Predicted releases for 2026: {predictions[1]}")
print(f"Predicted releases for 2027: {predictions[2]}")

#Plot actual data, polynomial trend, and future predictions
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X, poly_reg.predict(X_poly), color='red', label='Polynomial trend␣
 ↪line')
plt.scatter(X_future, predictions, color='green', label='Predicted future␣
 ↪releases')
plt.xlabel('Release Year')
plt.ylabel('Number of Releases')
plt.title('Netflix Content Releases Over Time (Polynomial Regression)')
plt.legend()
plt.show()
```

```
Predicted releases for 2025: 886.5815372765064
Predicted releases for 2026: 937.3420465737581
Predicted releases for 2027: 989.8526874780655
```



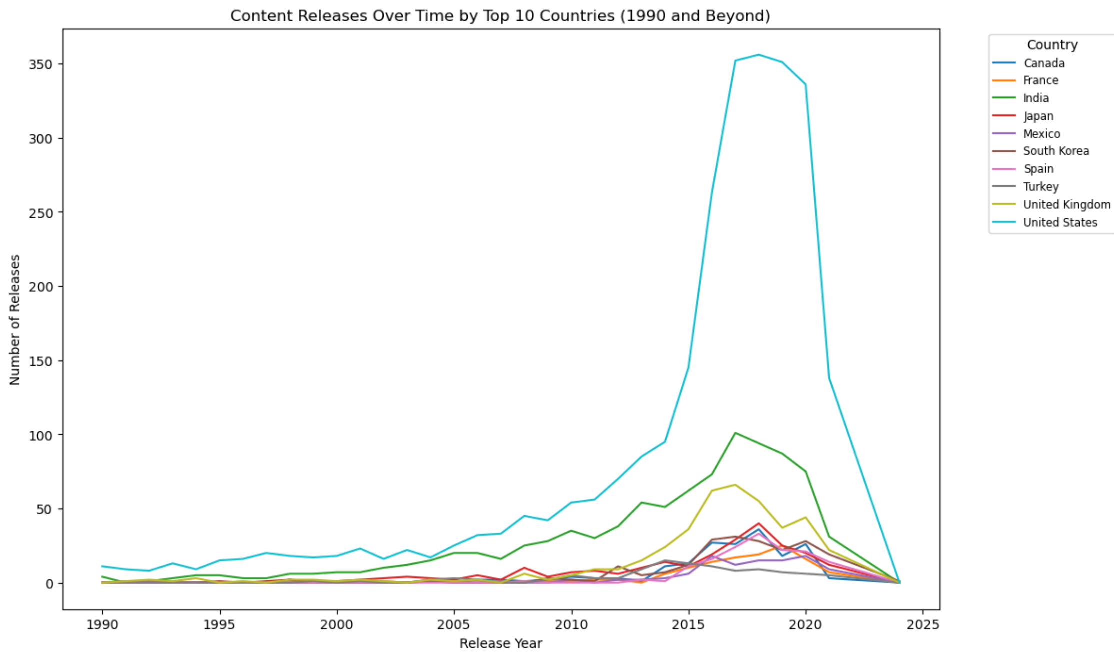
[54]:
```
import seaborn as sns
import matplotlib.pyplot as plt

#Filter the grouped data to only include releases from 1990 onwards
grouped_data_filtered = grouped_data[grouped_data['release_year'] >= 1990]

#Further filter for a specific country (e.g., 'United States')
country_data = grouped_data_filtered[grouped_data_filtered['country'] ==␣
↪'United States']

#Pivot the data for a stacked area plot (genre-wise releases over time)
pivot_data = country_data.pivot_table(index='release_year',␣
↪columns='genre_list', values='count', aggfunc='sum').fillna(0)

#Plot a stacked area chart starting from 1990
pivot_data.plot(kind='area', stacked=True, figsize=(12, 8), colormap='tab20c')
plt.title('Genre Trends Over Time in the United States (From 1990)')
plt.ylabel('Number of Releases')
plt.xlabel('Year')
plt.xlim(1990, pivot_data.index.max())  # Ensure the x-axis starts at 1990
```

```
plt.legend(title='Genres', loc='upper left', bbox_to_anchor=(1.0, 1.0))
plt.show()
```



[84]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Filter the data to only include releases from 1990 onward
df_filtered = df[df['release_year'] >= 1990]

# Count the number of releases per country and identify the top 10 countries
top_countries = df_filtered['country'].value_counts().head(10).index.tolist()

# Filter the data to only include the top 10 countries
df_top_countries = df_filtered[df_filtered['country'].isin(top_countries)]

# Group by release year and country to track content releases over time
region_trends = df_top_countries.groupby(['release_year', 'country']).size().
 ↪unstack().fillna(0)
```

```
# Plot the trends for the top 10 countries with the most releases
region_trends.plot(kind='line', figsize=(12, 8), colormap='tab10')
plt.title('Content Releases Over Time by Top 10 Countries (1990 and Beyond)')
plt.xlabel('Release Year')
plt.ylabel('Number of Releases')
plt.legend(title='Country', loc='upper left', bbox_to_anchor=(1.05, 1),␣
↪fontsize='small')
plt.show()
```



[92]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder

#Step 1: Load the datasets
df_netflix = pd.read_csv('netflix_titles.csv', encoding='ISO-8859-1')  # Adjust␣
↪encoding if needed
df_imdb = pd.read_csv('Netflix IMDB Ratings.csv', encoding='ISO-8859-1')  #␣
↪Adjust encoding if needed

#Step 2: Merge the datasets on 'title', now using 'imdb_score'
df_merged = pd.merge(df_netflix, df_imdb[['title', 'imdb_score']], on='title',␣
↪how='inner')
```

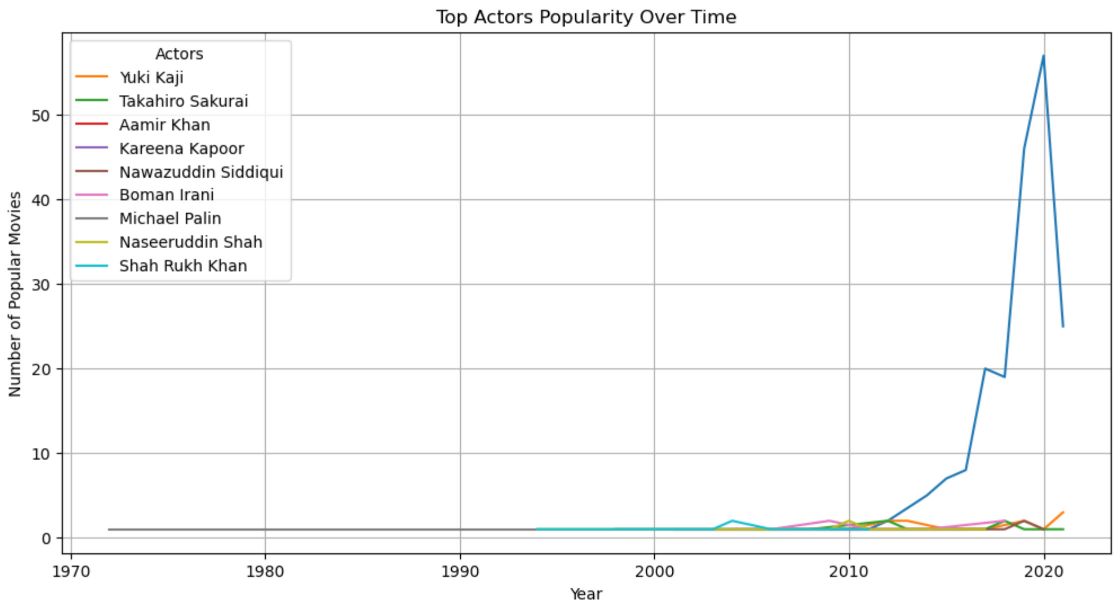
```python
#Step 3: Separate Movies and TV Shows for handling 'duration'
df_merged['is_movie'] = df_merged['type'] == 'Movie'

#For Movies, extract the number of minutes from the 'duration' column
df_merged.loc[df_merged['is_movie'], 'duration'] = df_merged.loc[df_merged['is_movie'], 'duration'].str.replace(' min', '').astype(float)

#For TV Shows, extract the number of seasons
df_merged.loc[~df_merged['is_movie'], 'duration'] = df_merged.loc[~df_merged['is_movie'], 'duration'].str.replace(' Season', '').str.replace('s', '').astype(float)

#Step 4: Prepare features and target
le_type = LabelEncoder()
df_merged['type_encoded'] = le_type.fit_transform(df_merged['type'])  # Encode 'type' (Movie/TV Show)

#Define features (e.g., release_year, duration, type_encoded)
X = df_merged[['release_year', 'duration', 'type_encoded']]

#Define the target variable (IMDB score)
y = df_merged['imdb_score']

#Step 5: Drop rows with missing values in X or y
X_y_combined = pd.concat([X, y], axis=1).dropna()
X_cleaned = X_y_combined.iloc[:, :-1]  # All columns except the last (y)
y_cleaned = X_y_combined.iloc[:, -1]   # The last column is the target (y)

#Split the cleaned data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_cleaned, y_cleaned, test_size=0.2, random_state=42)

#Step 6: Train the MLP Regressor
mlp = MLPRegressor(hidden_layer_sizes=(100,), max_iter=500, random_state=42)
mlp.fit(X_train, y_train)

#Step 7: Make predictions on the test set
y_pred = mlp.predict(X_test)

#Step 8: Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'MLP Regressor Mean Squared Error: {mse}')

#Optional: Inspect the predictions
df_predictions = pd.DataFrame({'Actual Score': y_test, 'Predicted Score': y_pred})
print(df_predictions.head())
```

```
MLP Regressor Mean Squared Error: 1.1345233586528307
      Actual Score  Predicted Score
1582           5.3         6.775695
3538           5.9         6.229175
263            4.8         6.318976
2474           6.5         7.245180
3140           6.9         7.222330
```

[94]:

```python
import pandas as pd
import matplotlib.pyplot as plt

#df is the merged Netflix-IMDb dataset

# Step 1: Split the 'cast' column into individual actors
df['cast'] = df['cast'].fillna('')  # Handle missing values
df['cast_list'] = df['cast'].str.split(', ')  # Split actors by comma and space

# Step 2: Explode the 'cast_list' to have one actor per row
df_actors = df.explode('cast_list')

# Step 3: Filter only popular movies (popularity == 1)
popular_actors = df_actors[df_actors['popularity'] == 1]

# Step 4: Group by actor and release year, then count number of popular movies
actor_popularity_over_time = popular_actors.groupby(['cast_list',␣
 ↪'release_year']).size().reset_index(name='popular_movies')

# Step 5: Find the top 10 actors by total number of popular movies over time
top_actors = actor_popularity_over_time.groupby('cast_list')['popular_movies'].
 ↪sum().nlargest(10).index

# Step 6: Filter the data to keep only the top actors
top_actors_data =␣
 ↪actor_popularity_over_time[actor_popularity_over_time['cast_list'].
 ↪isin(top_actors)]

# Step 7: Create a line plot showing popular movies for top actors over time
plt.figure(figsize=(12, 6))
for actor in top_actors:
    actor_data = top_actors_data[top_actors_data['cast_list'] == actor]
    plt.plot(actor_data['release_year'], actor_data['popular_movies'],␣
 ↪label=actor)

plt.xlabel('Year')
plt.ylabel('Number of Popular Movies')
plt.title('Top Actors Popularity Over Time')
plt.legend(title='Actors')
```

```
plt.grid(True)
plt.show()
```

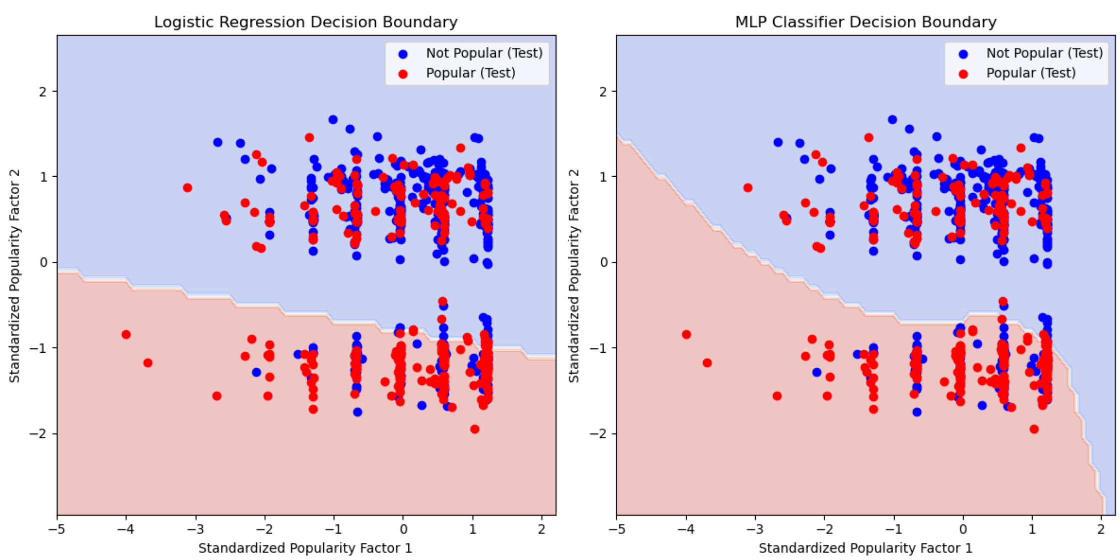


[90]:

```
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier

# Adjust pipeline to use TruncatedSVD and then Standardize the SVD components
model_pipeline_log_reg = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('scaler', StandardScaler(with_mean=False)),  # Standardize the original␣
↪data before SVD
    ('svd', TruncatedSVD(n_components=2))  # Use TruncatedSVD for␣
↪dimensionality reduction
])

model_pipeline_mlp = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('scaler', StandardScaler(with_mean=False)),  # Standardize the original␣
↪data before SVD
    ('svd', TruncatedSVD(n_components=2))  # Use TruncatedSVD for␣
↪dimensionality reduction
```

```
])

# Select features and target variable
X = df[features]
y = df['popularity']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Fit the preprocessing and dimensionality reduction pipeline
X_train_reduced = model_pipeline_log_reg.fit_transform(X_train)
X_test_reduced = model_pipeline_log_reg.transform(X_test)

# Standardize the reduced data (SVD-transformed components)
svd_scaler = StandardScaler()
X_train_reduced_std = svd_scaler.fit_transform(X_train_reduced)
X_test_reduced_std = svd_scaler.transform(X_test_reduced)

# Train Logistic Regression on standardized SVD-reduced data
log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train_reduced_std, y_train)

# Train MLP Classifier on standardized SVD-reduced data
mlp_clf = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500,␣
↪random_state=42)
mlp_clf.fit(X_train_reduced_std, y_train)

# Generate grid for plotting decision boundaries
x_min, x_max = X_test_reduced_std[:, 0].min() - 1, X_test_reduced_std[:, 0].
↪max() + 1
y_min, y_max = X_test_reduced_std[:, 1].min() - 1, X_test_reduced_std[:, 1].
↪max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

# Predictions for logistic regression and MLP on the grid
Z_log_reg = log_reg.predict(np.c_[xx.ravel(), yy.ravel()])
Z_mlp = mlp_clf.predict(np.c_[xx.ravel(), yy.ravel()])

# Reshape predictions to match the grid shape
Z_log_reg = Z_log_reg.reshape(xx.shape)
Z_mlp = Z_mlp.reshape(xx.shape)

# Plot the decision boundary for Logistic Regression
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
```

```
plt.contourf(xx, yy, Z_log_reg, alpha=0.3, cmap=plt.cm.coolwarm)
plt.scatter(X_test_reduced_std[:, 0][y_test == 0], X_test_reduced_std[:,␣
↪1][y_test == 0], color='blue', label='Not Popular (Test)')
plt.scatter(X_test_reduced_std[:, 0][y_test == 1], X_test_reduced_std[:,␣
↪1][y_test == 1], color='red', label='Popular (Test)')
plt.xlabel('Standardized Popularity Factor 1')
plt.ylabel('Standardized Popularity Factor 2')
plt.title('Logistic Regression Decision Boundary')
plt.legend()

# Plot the decision boundary for MLP Classifier
plt.subplot(1, 2, 2)
plt.contourf(xx, yy, Z_mlp, alpha=0.3, cmap=plt.cm.coolwarm)
plt.scatter(X_test_reduced_std[:, 0][y_test == 0], X_test_reduced_std[:,␣
↪1][y_test == 0], color='blue', label='Not Popular (Test)')
plt.scatter(X_test_reduced_std[:, 0][y_test == 1], X_test_reduced_std[:,␣
↪1][y_test == 1], color='red', label='Popular (Test)')
plt.xlabel('Standardized Popularity Factor 1')
plt.ylabel('Standardized Popularity Factor 2')
plt.title('MLP Classifier Decision Boundary')
plt.legend()

plt.tight_layout()
plt.show()
```



[ ]: