Recommendation System Case Study: Netflix Dataset

Ryan Gomberg

May 18, 2025

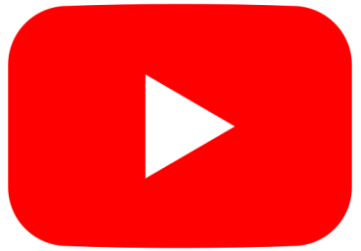
All diagrams and plots are my own.

Contents

When applicable, I will provide my rationale from a mathematical and practical approach.

The Need for Recommendation Systems

- The rapid growth of data collection has given rise to a new methodology of analyzing information and developing more efficient systems.
- As a result, recommendation systems have become increasingly prominent in filtering data, improving the quality of search results, and providing items that are more relevant to the user.
- Generally, recommendation systems are designed to *predict a rating or preference* that a user would associate with an item.
- Large consumer dependent companies – Amazon, Spotify, Netflix, YouTube – all require effective recommendation systems for successful business.
- In this project, we will provide two different approaches of recommendation systems to a dataset featuring thousands of movies available on Netflix (as of 2021) alongside other important features such as *text descriptions, genres, and IMDb scoring*.



Introduction to the Dataset, Goals

- We are using the netflix_db.csv and netflix_description.csv datasets from Kaggle.
- In total, there are 5967 unique entries, 5897 unique movies, and 14 different features.
- While this is not an exhaustive list, some features are
  - *Description*: offers a text description of each movie.
  - *Genres*: provides 1-3 categories for each movie (i.e. Comedies, Drama, Horror).
  - *IMDb Rating*: the movie's IMDb rating.
  - Other miscellaneous features: *Cast, Production Country, Release Date, Duration.*
- The general goals of this project are to:
  - Apply **exploratory data analysis** to identify and distributions and conclusions.
  - Use **cosine similarity** to recommend the top 3 movies based on movie description.
  - Build a **recurrent neural network** to recommend the top 3 movies based on movie history.

Data Preprocessing

The following implementations were made to the original dataset to prepare ourselves for exploratory data analysis and deploying our recommendation system models.

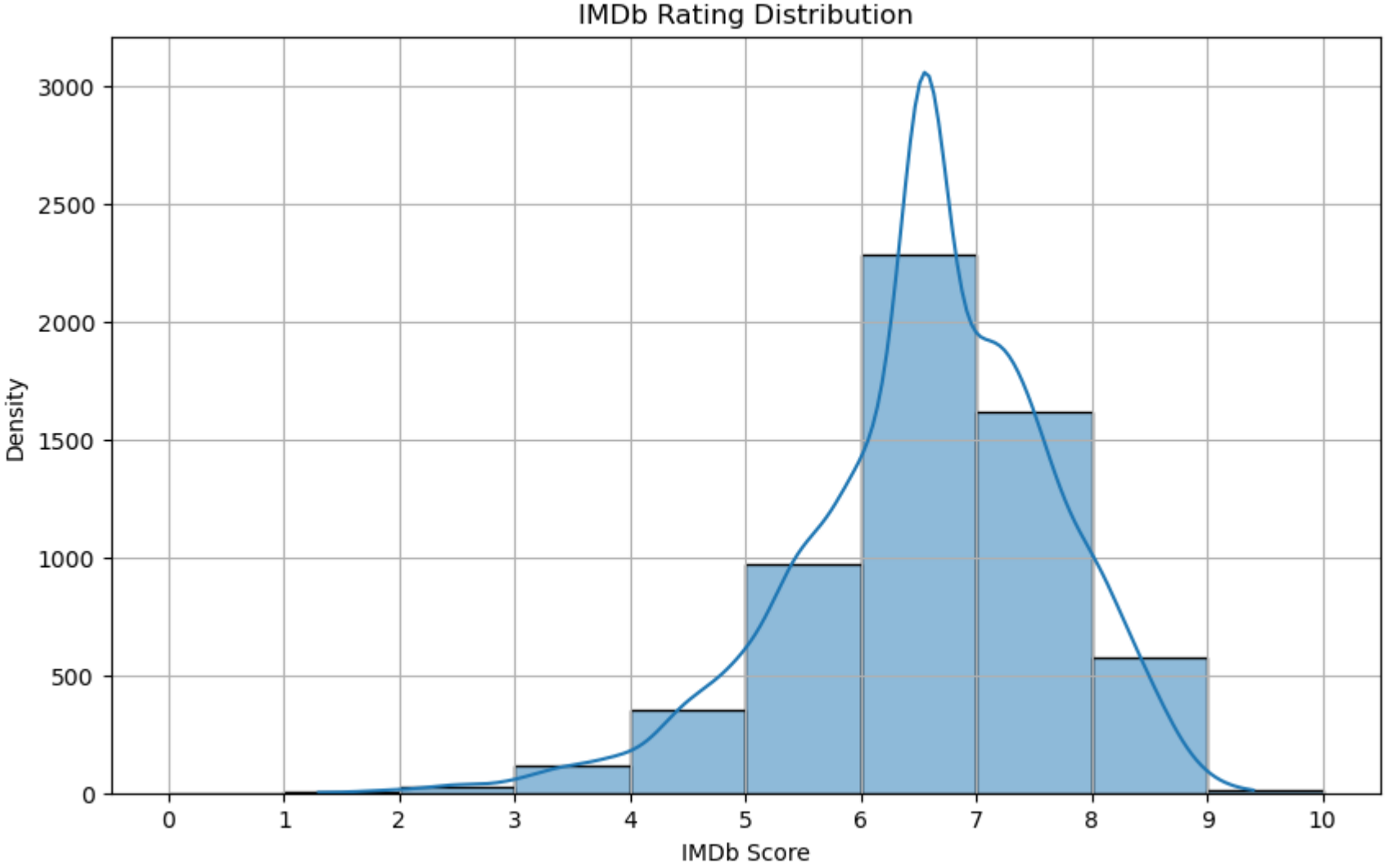
1. Merged netflix_db and netflix_description to a single dataset.
2. Rewrite IMDb scores as floats and fill missing values with the average score.
3. Drop missing values in the 'Title' and 'Date Added' columns.
4. Sort titles chronologically in the 'Date Added' column.

df2 is an expanded database (additional columns) used in the *RNN model*.

```
# Rewriting IMDb Scores as floats
def score_to_float(x):
    if x == '10.0/10':
        return 10.0
    else:
        return(float(str(x)[0:3]))

df['Imdb Score'] = df['Imdb Score'].apply(score_to_float)

# Set missing values to the mean score
df['Imdb Score'] = df['Imdb Score'].fillna(df['Imdb Score'].mean())


df2 = df.dropna(subset=['Title', 'Date Added'])
df2['Date Added'] = pd.to_datetime(df2['Date Added'])
df2 = df2.sort_values(['Title', 'Date Added'])
```

Exploratory Data Analysis

We will explore 3 graphs within our aggregated data. The first of which is the distribution of IMDb ratings. The submodules **seaborn** and **matplotlib** were used for visualization.

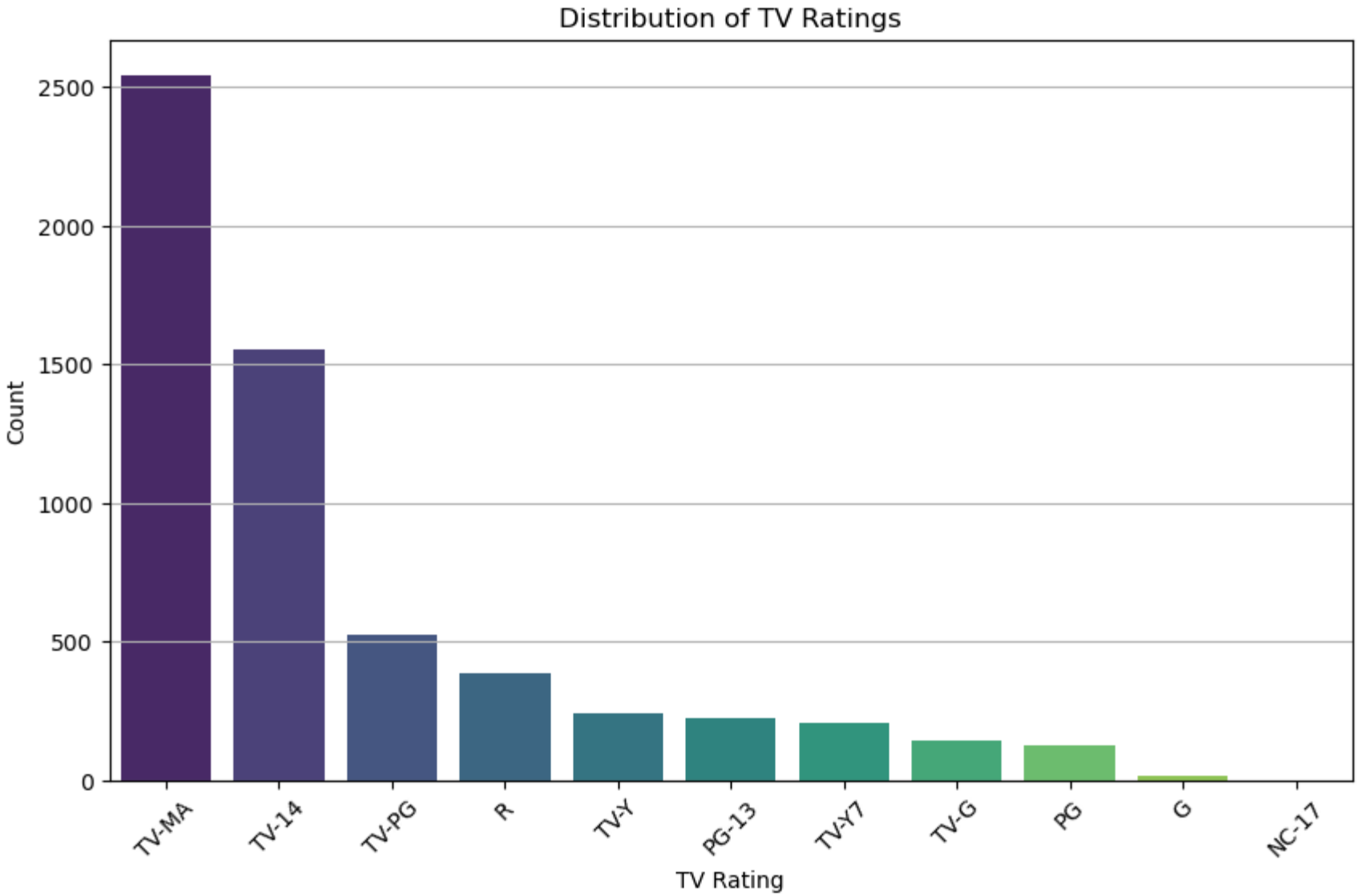


The majority of IMDb ratings fall within 4-9, with

- 6-7 taking up roughly 30% of our data.
- 5-8 taking up roughly 85% of our data.
- Ratings of 0-3 and 9-10 being exceptionally rare.

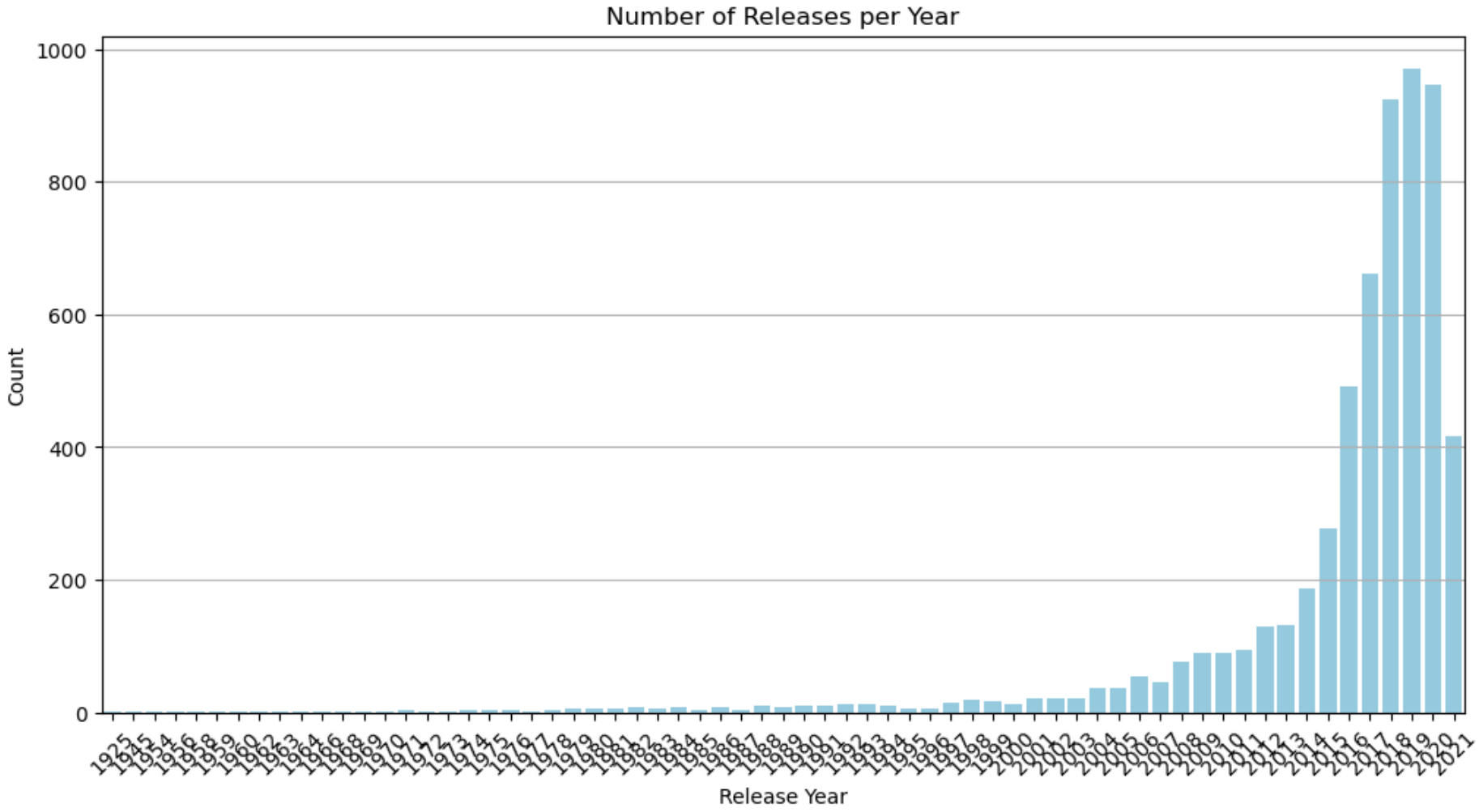
This reflects the standards of most IMDB ratings, with 7/10 as the “most average” score.

Exploratory Data Analysis



- Almost half of the movies are rated for mature audiences only.
- Roughly 4300 movies have an age restriction of 13+.
- It is likely that this dataset is not tailored towards family/children's films (i.e. animated films).

Exploratory Data Analysis

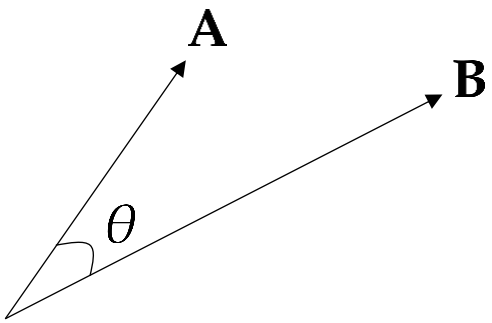


- As expected, almost all films are released after 1995, with roughly 80% of them released after 2014.
- Almost half of the movies in this dataset were released between 2018-2020.

Model 1: Cosine Similarity

Given two vectors **A, B**, the *similarity* between them is given by the cosine of the angle separating them:

$$\cos\theta = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||||\mathbf{B}||}$$



Since $-1 \leq \cos\theta \leq 1$, similarity is measured by -1 to 1, where 1 indicates a perfect correlation and 0 indicates no correlation. This is one way of implementing recommendation systems: given a piece of data **A**, compute the cosine similarity between candidate data entries **B** and output the three that are closest to 1.

In context of our dataset, we will use cosine similarity based on the following factor:

- **Content-based filtering**: Given a movie description, output the top 3 movie recommendations whose description closely matches the inputted one.

Model 1: Cosine Similarity

We will be importing the **TfidfVectorizer** and **cosine_similarity** submodules from scikit-learn.

- **TfidfVectorizer** converts strings to vectors stored in data.
- **cosine_similarity** constructs a symmetric, similarity matrix containing the similarity index between all candidate movies. For instance, the top 3 movies could have a similarity matrix of the form:

$$\begin{pmatrix} 1 & 1 & 0.978 \\ 1 & 1 & 0.992 \\ 0.978 & 0.992 & 1 \end{pmatrix}$$

- We will use our newfound recommendation system to find 3 movies whose description is most similar to *Zoo*.

```python
# TF-IDF Vectorization of Descriptions
tfidf = TfidfVectorizer(stop_words='english', max_features=300)
tfidf_matrix = tfidf.fit_transform(df['Description'])

# Compute cosine similarity matrix of Descriptions
cos_sim = cosine_similarity(tfidf_matrix)

# Recommendation function - input title of movie in dataset
# and outputs top 3 movies based on the given cos_sim matrix
def recommend_cosine(title, top_k = 3):
    # Find the index of the movie with the given title
    idx = df[df['Title'] == title].index
    if len(idx) == 0:
        return f"Title '{title}' not found."
    idx = idx[0]

    # Looking out for indexing errors
    if idx >= len(cos_sim):
        return f"Index {idx} is out of bounds for similarity matrix
                of size {len(cos_sim)}"

    # Rank movies by similarity
    sim_scores = list(enumerate(cos_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Pick top 3 movies (excluding itself)
    top_indices = [i for i, _ in sim_scores[1:top_k+1]]
    return df.iloc[top_indices][['Title', 'Genres', 'Imdb Score']]

print("Cosine recommendations:")
print(recommend_cosine("Zoo"))
```

Model 1: Cosine Similarity

Input

Zoo description: A drug dealer starts having doubts about his trade as his brother, his client, and two rappers from the slums each battle their own secret addictions.

Recommendations

Being Napoleon description: On the 200th anniversary of the Battle of Waterloo, thousands of enthusiast reenact the epic clash. But there can only be one Napoleon.

Ghosts of War description: Five, battle-worn Allied soldiers guarding a chateau previously occupied by Nazis start experiencing unexplained and terrifying supernatural horrors.

El señor de los Cielos description: Only Aurelio Casillas can fill Pablo Escobar's shoes and become Mexico's biggest drug trafficker of the '90s.

```
Cosine recommendations:
                        Title  \
592           Being Napoleon
1806           Ghosts of War
1472  El señor de los Cielos

                                                Genres  Imdb Score
592                                        Documentaries         5.5
1806                                       Horror Movies         5.4
1472  Crime TV Shows, International TV Shows, Spanis...         6.9
```

Model 1: Cosine Similarity

Overall, it appears that the recommendation system is partially successful.

The movie *El señor de los Cielos* is very similar to *Zoo*, mainly thematically and with respect to the plot.

Ghosts of War can be acquainted with *Zoo* in the sense of psychological trauma.

Being Napoleon is entirely unrelated with *Zoo*, but might have been recommended as a false positive due to some syntax overlap (i.e. battle, clash).

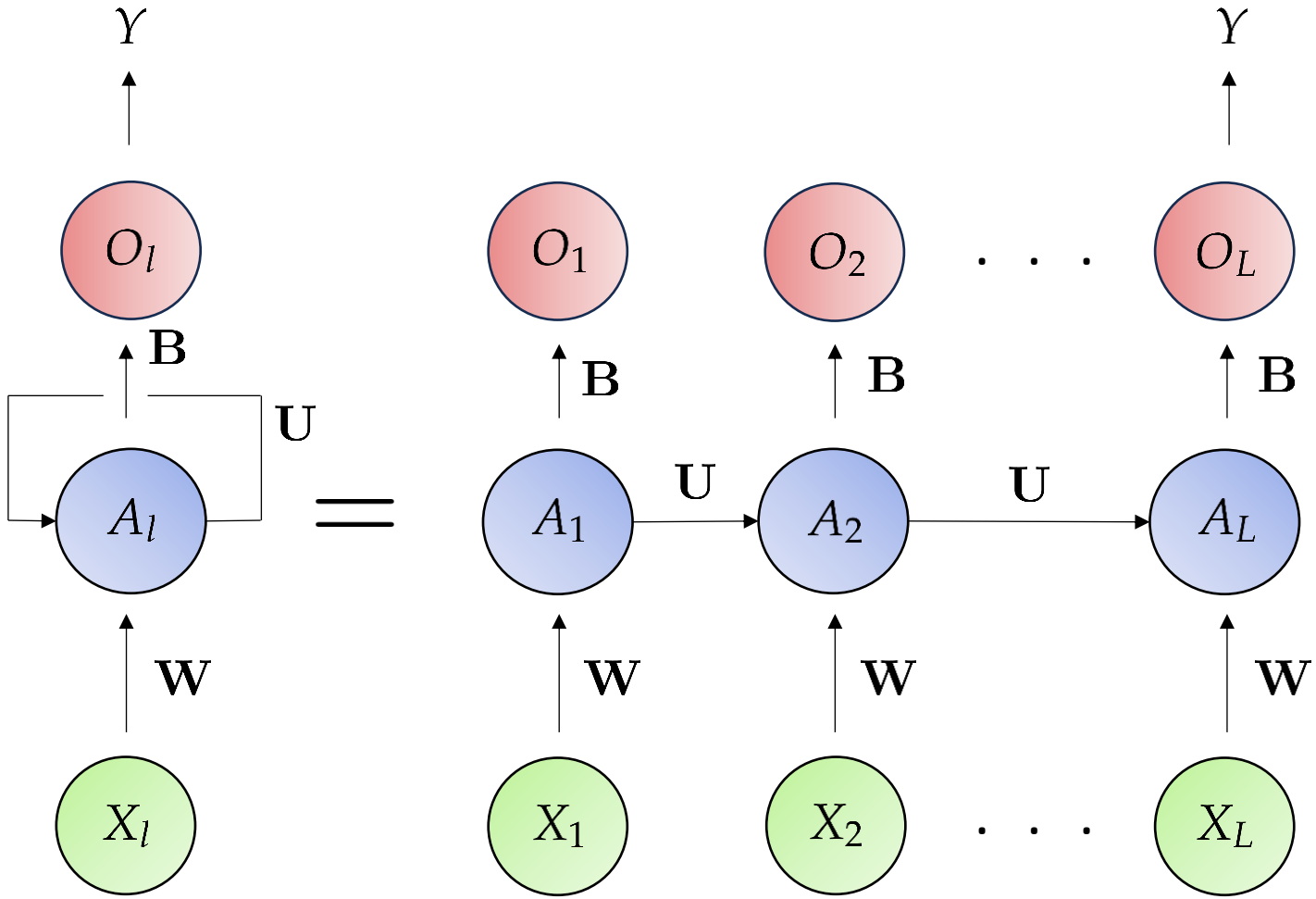
Model 2: Recurrent Neural Networks (RNNs)

Recurrent neural networks are neural networks well-suited for *sequential data*, storing data from previous time-steps and learning from temporal patterns.

The structure is almost identical to a typical neural network, with the exception of a recurrent weight **U**, repeating activation functions while using the same weights and biases **W, B** (over multiple time steps) in each hidden layer.

Additionally, we choose the number of *epochs*, or number of iterations of forward-feeding and backpropagation, to balance computational cost while minimizing our loss function.

For out dataset, RNNs are appropriate if we select a "history" of movies the user has watched. Through this, the neural network will output 3 movies to recommend with these optimized weights and biases.



Model 2: Recurrent Neural Networks (RNNs)

We propose the following RNN structure:

- **Input layer**: 5 neurons, or the last 5 movies the user has watched.
- **Embedding layer**: 32 neurons, mapping categorical data into a vector of 32 integers.
- **Hidden layer**: 64 neurons, GRU (Gated Recurrent Unit). Uses *sigmoid* and *tanh* activation functions as transitions.
- **Output layer**: 3 neurons, uses *softmax (logits)* as the activation function, predicting the 3 most likely movie to recommend.
- Since we are dealing with a multi-classification problem, we impose a *cross-entropy loss function.*
- We run over 5 sequences (as implied by input layer) and 30 epochs to achieve a desirable, minimal loss function.
- In total, there are 67,124 parameters!

Relevant submodules: **LabelEncoder**, **torch**, **nn**, **TensorFlow**, **defaultdict**.

```
# Neural Network Architecture: Size of layers
vocab_size = len(le.classes_)
embed_dim = 32
hidden_dim = 64

# Neural Network Architecture: Defining Activation Functions
embedding = nn.Embedding(vocab_size, embed_dim)
rnn = nn.GRU(embed_dim, hidden_dim, batch_first=True)
fc = nn.Linear(hidden_dim, vocab_size)

# Training Setup
params = list(embedding.parameters()) + list(rnn.parameters()) + list(fc.parameters())
optimizer = torch.optim.Adam(params, lr=0.01)
loss_fn = nn.CrossEntropyLoss()

# Training model over 30 epochs
for epoch in range(30):
    optimizer.zero_grad()
    x_embed = embedding(inputs)
    _, h = rnn(x_embed)
    logits = fc(h.squeeze(0))
    loss = loss_fn(logits, targets)
    loss.backward()
    optimizer.step()
    print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")
```

Model 2: Recurrent Neural Networks (RNNs)

For simplicity, we select our viewing history as the first 5 movies in the dataset.

- As expected, the loss function starts off large over the first few iterations of the neural network.
- As we repeatedly run through and backpropagate, the loss function appears to be steadily converging to some minimum.
- The viewing history primarily falls into international movies, documentaries, and comedies.
- The recommended movies share some of the genres in our viewing history. While we did not empirically measure the accuracy of our model, we can say there is some success in recommending new movies.

Viewing History

	Title
0	(Un)Well
1	#Alive
2	#AnneFrank - Parallel Stories
3	#blackAF
4	#cats_the_mewvie

Loss Function over 30 epochs

```
Epoch 1, Loss: 8.3258    Epoch 28, Loss: 0.2461
Epoch 2, Loss: 8.1931    Epoch 29, Loss: 0.1817
Epoch 3, Loss: 8.0521    Epoch 30, Loss: 0.1358
```

Output of Neural Network

	Title	Imdb Score
0	48 Christmas Wishes	3.8
1	March Comes in Like a Lion	8.4
2	Naruto	8.3

	Genres	Imdb Score
0	Children & Family Movies, Comedies	3.8
1	Anime Series, International TV Shows, Teen TV Shows	8.4
2	Anime Series, International TV Shows	8.3

Conclusion

Throughout this presentation, we have:

- *Motivated* the need for recommendation systems.
- Introduced the dataset and necessary *preprocessing* techniques.
- Performed *exploratory data analysis* on different features in our data.
- Used *cosine similarity* to recommend movies with similar descriptions.
- Deployed a *recurring neural network* to recommend movies given a user's viewing history (retrospective technique).

If time allowed, I would have liked to:

- Learn more about the general dataset through visualization techniques.
- Explore ways to improve the *cosine similarity* recommendation system (i.e. advanced word embeddings such as Word2Vec).
- Apply *cosine similarity* based on different features in the dataset (i.e. Rating, Genres).
- Find ways to test accuracy in our *recurrent neural network* and do more tinkering with the hidden/output layers.

Miscellaneous

The dataset can be found through the website:
https://www.kaggle.com/datasets/satpreetmakhija/netflix-movies-and-tv-shows-2021/data

The .ipynb file (Jupyter Notebook), containing the code to generate everything in this presentation, will eventually be accessible through my GitHub:

https://github.com/ryangomberg?tab=repositories

The recurrent neural network diagram was mostly inspired by *An Introduction to Statistical Learning*.