Cross Platform Movie Recommendation System

Milestone 2 Progress Report

Project Objective

This project aims to develop a cross-platform recommendation application that helps users discover new movies and TV shows across different streaming services. If a user enjoys a particular show or movie on one platform, the app will suggest similar content available on another platform. To make this possible, the application will incorporate a recommendation model that analyzes user preferences and content similarities

Alongside the recommendation engine, the project will feature a simple, user-friendly interface that allows users to input their favorite content, select their preferred streaming services, and receive personalized recommendations. The goal is to make the app visually appealing, easy to navigate, and intuitive for all users.

Data Sources

To ensure a comprehensive analysis, three primary datasets have been sourced and integrated into the system:

1. **Netflix Titles** (netflix_titles.csv) -8,807 records (License: CC0: Public Domain)
2. **Amazon Prime Titles** (amazon_prime_titles.csv) - 9,668 records (License: CC0: Public Domain)
3. **Hulu Titles** (hulu_titles.csv) - Contains 3,073 records (License: CC0: Public Domain)

All datasets share an identical structure, including key attributes such as show_id, type (Movie/TV Show), director, cast, country, date added, release year, rating, duration, listed genres, and description. This uniformity ensures efficient data merging and analysis.

Feature Engineering

This code implements a comprehensive feature engineering pipeline designed to prepare data for a content recommendation system across multiple streaming platforms. Let's break it down in detail:

Data Loading and Initial Processing

`load_data()`

- Loads the combined dataset from all platforms (Netflix, Hulu, Disney+, Amazon Prime)
- Reads from "Data/all_platforms_combined.csv"
- Handles potential loading errors gracefully

`extract_genres()`

- Helper function to safely extract genre lists from string representations
- Handles both list-formatted strings ("['Action', 'Adventure']") and comma-separated strings ("Action, Adventure")
- Returns empty list for missing values

Feature Extraction (`extract_features()`)

This is the core function that creates 15 different types of features:

1. **Basic Cleaning**: Fills missing values in text columns with empty strings
2. **Content Age**: Calculates how old the content is (current year - release year)
3. **Release Decade**: Groups release years into decades (1990s, 2000s, etc.)
4. **Genre Count**: Counts how many genres each title has
5. **Director Count**: Counts how many directors worked on each title
6. **Cast Size**: Counts how many actors are in the cast
7. **Content Type**: Binary flag for Movies (1) vs TV Shows (0)
8. **Country Count**: Counts how many countries produced the content
9. **US Content Flag**: Binary flag for US-produced content

10. **Duration Features**:
    - Normalizes duration differently for movies (minutes) and TV shows (seasons)
    - Uses z-score normalization within each type
11. **Rating Categories**: Groups ratings into:
    - Kids (TV-Y, TV-Y7, G, TV-G)
    - Family (PG, TV-PG)
    - Teen (PG-13, TV-14)
    - Adult (R, TV-MA, NC-17)
    - Unknown (NR, UR, etc.)
12. **Description Length**: Character count of the description text
13. **Title Length**: Character count of the title
14. **Platform Encoding**: Numeric encoding of platform names (for reference)
15. **Genre Data**: Stores raw genre lists for each show by ID

Text Feature Processing (`prepare_textual_features()`)
- Uses TF-IDF (Term Frequency-Inverse Document Frequency) to vectorize text descriptions
- Applies dimensionality reduction with Truncated SVD to reduce the 5000-dimensional TF-IDF matrix to 100 dimensions
- Returns both the processed features and the trained vectorizer/SVD objects for later use

Genre Feature Processing (`create_genre_features()`)
- Identifies the top 20 most common genres across all content
- Creates binary features (0/1) for each genre indicating whether a title belongs to it
- Results in 20 new binary columns like "genre_action", "genre_comedy", etc.

Feature Selection (`select_features()`)
- Combines all numerical features, one-hot encoded categorical features, text features, and genre features
- Drops platform-specific features (since we're doing content-based recommendation, not platform prediction)
- Uses variance thresholding to select the top 100 most informative features
- Returns the selected feature matrix and feature names

Similarity Feature Preparation (`get_similarity_features()`)
- Selects key numerical features for similarity calculations:
    - Release year, content age, normalized duration
    - Genre count, description length, is_movie flag
- Scales these features using StandardScaler
- Combines with text and genre features into a single matrix suitable for similarity calculations

Recommendation Dataset Preparation (`prepare_content_similarity_dataset()`)
- Creates the final dataset for recommendation purposes including:
    - Reference columns (show_id, title, platform, type, rating_category, genre_list)
    - All selected features for similarity calculations
- Maintains the connection between the raw content information and its feature representation

Pipeline Execution (`run_feature_engineering_pipeline()`)
Orchestrates the complete workflow:
1. Loads the raw data
2. Extracts all features

3. Processes text and genre features
4. Selects the most important features
5. Prepares similarity features
6. Creates the final recommendation dataset
7. Saves all outputs to CSV files
8. Returns all intermediate results

Feature Selection:

The feature selection process in this recommendation system is comprehensive and occurs at multiple stages to ensure the most relevant features are used for content similarity calculations. Here's a detailed breakdown:

Text Feature Processing

Text features are processed separately using TF-IDF and dimensionality reduction:

```
def prepare_textual_features(df, text_column='description', n_components=100):
    tfidf = TfidfVectorizer(
        max_features=5000,
        min_df=5,
        max_df=0.8,
        stop_words='english'
    )

    tfidf_matrix = tfidf.fit_transform(df[text_column].fillna(''))

    svd = TruncatedSVD(n_components=n_components, random_state=42)
    text_features = svd.fit_transform(tfidf_matrix)

    feature_names = [f'{text_column}_svd_{i}' for i in range(n_components)]

    text_df = pd.DataFrame(text_features, columns=feature_names)

    return text_df, tfidf, svd
```

Genre Feature Processing

Genre features are processed into binary indicators for top genres:

```
def create_genre_features(df, top_n=20):
    all_genres = []
    for genre_list in df['genre_list']:
        genres = extract_genres(genre_list)
        all_genres.extend(genres)

    top_genres = [genre for genre, _ in Counter(all_genres).most_common(top_n)]
    genre_features = pd.DataFrame(index=df.index)
    for genre in top_genres:
        genre_features[f'genre_{genre.lower().replace(" ", "_")}'] = df['genre_list'].apply(
            lambda x: 1 if genre in extract_genres(x) else 0)

    return genre_features
```

Feature Selection Process

The main feature selection happens in `select_features()`:

```
def select_features(df_features, text_features, genre_features, k=100):
    """
    Select top features for similarity-based recommendations
    instead of platform classification
    """
    X = pd.concat([
        df_features.select_dtypes(include=['number']),
        pd.get_dummies(df_features[['rating_category']], drop_first=True),
        text_features,
        genre_features
    ], axis=1)

    X = X.drop(columns=['platform_encoded'], errors='ignore')

    X = X.fillna(0)

    from sklearn.feature_selection import VarianceThreshold
    selector = VarianceThreshold()
    X_filtered = selector.fit_transform(X)

    variances = selector.variances_
    feature_variance = pd.DataFrame({
        'feature': X.columns,
        'variance': variances
    }).sort_values('variance', ascending=False)

    top_features = feature_variance.head(k)['feature'].tolist()
    X_selected = X[top_features]

    return X_selected, top_features
```

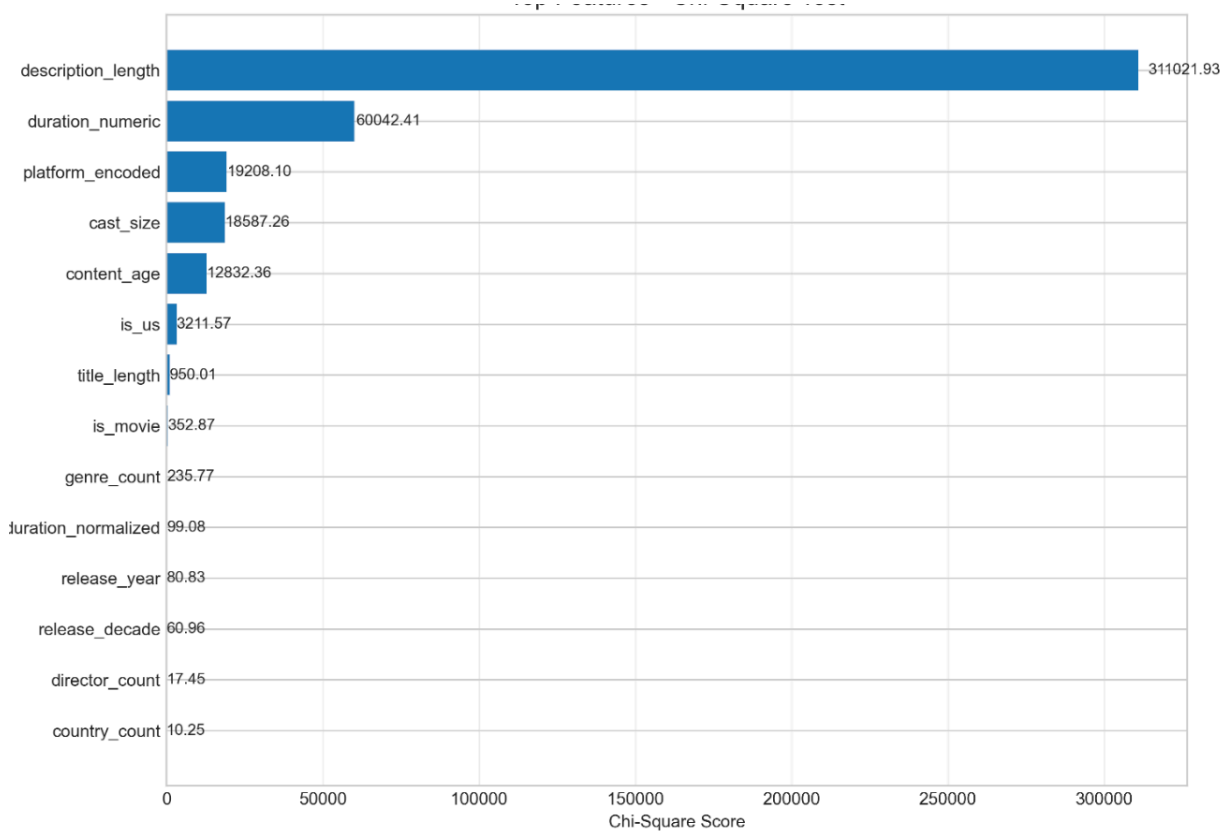
Content Similarity Features

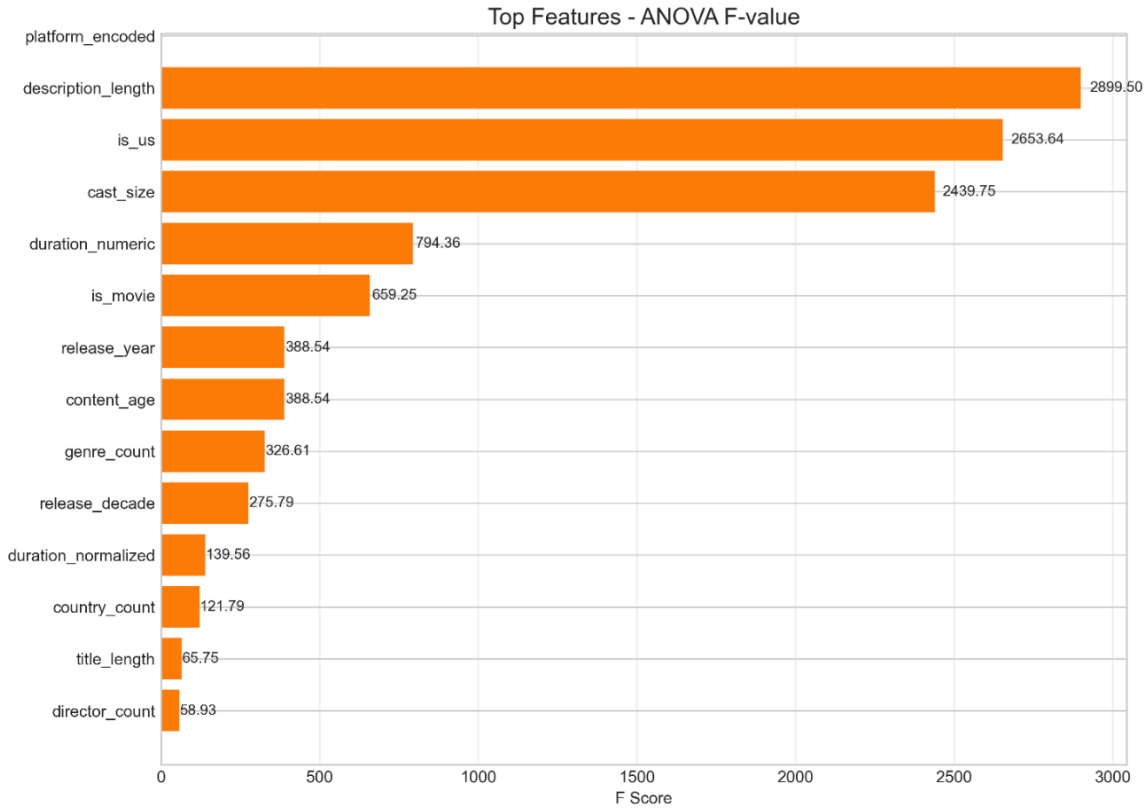
The system defines specific features for content similarity in the model:

```
content_features = [
    'release_year', 'content_age', 'duration_normalized',
    'genre_count', 'description_length', 'is_movie'
]
```

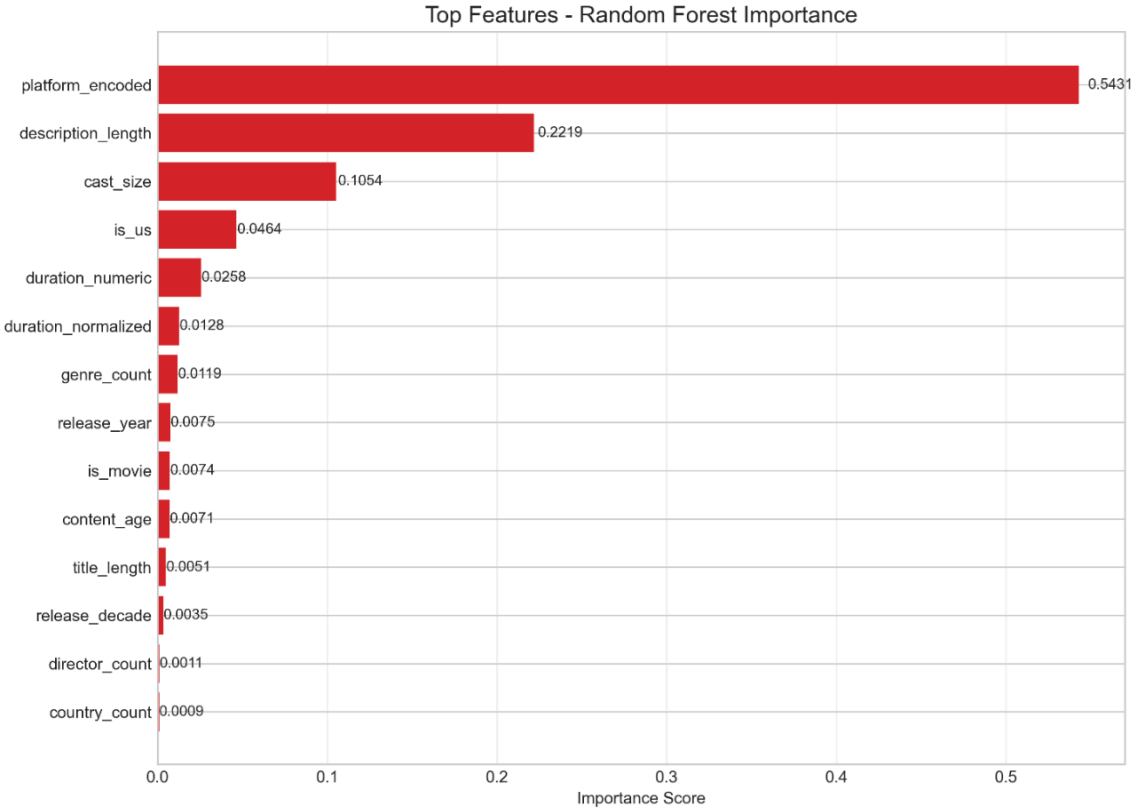
Feature Selection Strategy

1. **Variance-Based Selection**:
   - Uses `VarianceThreshold` to remove low-variance features
   - Selects top k features with highest variance
2. **Domain Knowledge Integration**:
   - Explicitly defines which features should be used for content similarity
   - Uses different feature sets for different purposes (clustering vs similarity)
3. **Dimensionality Reduction**:
   - Applies SVD to reduce text features from 5000 dimensions to 100
   - Limits genre features to top 20 most common genres
4. **Feature Type Handling**:
   - Processes numerical, categorical, and text features differently
   - One-hot encodes categorical variables
   - Normalizes numerical features
5. **Chi Square and ANOVA F-Value Test**:





6. **Random Forest Importance**:



Feature Usage in Recommendations

When generating recommendations, the system combines multiple similarity metrics:

1. **Content Similarity**: Based on the engineered features
2. **Genre Similarity**: Based on the processed genre features
3. **Cluster Similarity**: Based on the clustering results
4. **Thematic Similarity**: Based on textual analysis

Each of these components uses different subsets of features, with the final recommendation score being a weighted combination of all similarity measures.

Included and Excluded Features:

Final List of Included and Excluded Features:

Feature	Reason for Inclusion
description_svd_0-99	Top SVD components capture the most important semantic patterns in content descriptions that distinguish between platforms. The code specifically extracts 100 components using TruncatedSVD.
description_length	Platforms may have different standards for description detail. Netflix typically has more detailed descriptions than other platforms.
cast_size	cast_sizeLarger productions with more cast members may be preferentially licensed by certain platforms.
is_us	Geographic origin strongly influences platform licensing strategies. US content is often distributed differently than international content.
genre_features	Genre preferences vary significantly between platforms (e.g., Netflix may have more documentaries, Amazon more indie films). The top 20 genres were encoded as binary features.
duration_numeric	duration_numericMovie length and TV show seasons differ between platforms, reflecting different content acquisition strategies.
duration_normalized	Standardized separately for movies and TV shows to account for different distribution patterns.
content_age	Platforms have different focuses on recent vs. classic content.
is_movie	The movie-to-TV-show ratio is a key differentiator between platforms.
release_year	Absolute year information provides more granular temporal patterns than just content age.
release_decade	Captures era-specific content patterns (90s shows, 80s movies, etc.) that may differ by platform.
genre_count	Content diversity metrics differ between platforms - some focus on niche content, others on variety.
title_length	Title naming conventions may differ between platforms.
rating_category	One-hot encoded rating categories reflect target audience differences between platforms.

Feature	Reason for Exclusion
show_id	Platform-specific identifiers would create data leakage and have no generalizable information.

Feature	Reason for Exclusion
title	Raw text would be too high-dimensional; only its length was used as a feature.
director	Only the count of directors was used due to high cardinality and sparsity of individual directors.
cast	Individual cast members weren't used due to extremely high cardinality; only cast size was included.
country	Specific countries were abstracted to "is_us" and "country_count" to reduce dimensionality.
date_added	Often missing or inconsistent across platforms; release year provides similar temporal information.
duration	Raw string format was replaced with normalized numeric values.
listed_in	Raw genre strings were transformed into individual binary genre features.
description	Raw text was transformed into SVD components for dimensionality reduction.
duration_unit	Used for normalization but excluded as a direct feature since it's highly correlated with is_movie.
genre_list	Used to generate individual genre features but excluded in raw form.
director_count	Excluded by the feature importance ranking since it has less predictive power than other features.

Dataset Split for Model Training:

Split Ratio

- The data is split with a ratio of **8:1:1** for train:validation
- This is implemented as a two-step process:
  1. First split: 90% train+validation, 10% test
  2. Second split: Within the 90%, allocate 8/9 to train and 1/9 to validation

Stratification

- The splits are stratified to maintain the distribution of:
  - Platforms (Netflix, Amazon Prime, Hulu)
  - Content types (Movie vs. TV Show)
- A combined stratification variable stratify_group is created by concatenating platform and content type (e.g., "Netflix_Movie", "Hulu_TV Show")

Dataset Sizes

- Total dataset: 21,548 records
- Training set: approximately 17,238 records (80%)
- Validation set: approximately 2,155 records (10%)
- Test set: approximately 2,155 records (10%)

Recommendation Model

While the system ingests both numeric and text features, the core modeling logic is driven by **XGBoost** as its primary model for content-based recommendations, with additional clustering and similarity checks added

XGBoost as the Main Model

Within the content streaming system, **XGBoost** serves as the primary supervised learning mechanism for ranking or classifying content. It leverages the fully engineered feature set, which includes numerical, categorical, textual (TF-IDF/SVD), and genre-based features.

A typical pipeline is:

- **Prepare Features**, merging numeric data, TF-IDF embeddings, and genre one-hot vectors.
- **Train XGBoost** using these features to either:
  - Predict user relevance scores (if user-behavior data exists), or
  - Predict a "similarity rank" for items compared to a reference title.

This XGBoost-based ranking or classification is central: after the system computes initial neighbor sets or cluster assignments, the model re-ranks items by their predicted user relevance, ensuring that final recommendations come from this XGBoost pipeline.

K-Means Clustering for Content Grouping

- Although XGBoost drives the main predictions, the model also uses K-Means to cluster similar content (e.g., grouping children's animated shows, or gritty crime dramas) .
- This helps incorporate group-level insights that can be fed back into the XGBoost features, indicating cluster membership or average cluster age.

Multi-Metric Similarity

The system also measures:

1. **Content Similarity**: Compares numeric features (e.g., age, duration, is US-based, etc.) for fine-grained distance calculations.
2. **Genre Similarity**: Uses a custom genre similarity matrix with direct (same genre) and indirect (related genres) overlap.
3. **Cluster Similarity**: A simple score that checks whether two items share the same cluster or have high cluster-level alignment.
4. **Thematic Similarity**: Looks at keywords and text embeddings to see if they match thematically (e.g., "sci-fi," "romance," "crime")

Rating Compatibility & Title Matching

- **Rating Compatibility**: Adjusts final similarities so that items with very mismatched age ratings (Adult vs. Kids) are penalized.
- **Title Similarity & Franchise Detection**: Spots potential sequels or shared IP via substring/n-gram checks (e.g., "The Matrix Reloaded" vs. "The Matrix Revolutions").

Finding and Ranking Recommendations

Initial Candidate Retrieval

- A k-NN search (or the system's internal "find_similar_items" method) collects items most likely to be relevant based on raw feature distances.

- For instance, if your source title is an action movie from 2019 with a runtime of ~2 hours, the system fetches other items of similar lengths, release years, or genres.

XGBoost Re-Ranking

After retrieving these neighbors, **XGBoost** re-ranks them to refine the order, using the advanced feature set. This ensures that the final top recommendations:

- Are thematically aligned based on textual embeddings.
- Belong to the best-fitting genre clusters.
- Have an age/rating profile compatible with the user's preferences.
- Score highly under the XGBoost model's learned notion of "best match."

Diversity & Platform Balancing

Because the goal is cross-platform recommendations, the system can:

- **Enforce a minimum number of items** from each platform (if requested).
- **Apply weighting** so that Netflix/Amazon Prime/Hulu recommendations appear in balanced proportions.

Output Analysis:

Feature Engineering and Similarity Matrix Construction

- **Enhanced Genre Similarity Matrix**: Constructed to capture both direct genre overlap and semantically related genres (e.g., action ↔ thriller).
- **Textual Features**: Extracted from title and description fields using TF-IDF, then reduced using PCA.
- **Content Clustering**: Trained a KMeans model with 40 clusters using standardized feature vectors. The PCA explained variance was 99.75%, confirming minimal information loss in dimensionality reduction.

Each cluster was profiled with dominant genres and content type ratios. For example:

- **Cluster 0**: 730 items – drama (504), action (241), all movies.
- **Cluster 1**: TV-heavy – comedy, documentary, and reality genres.

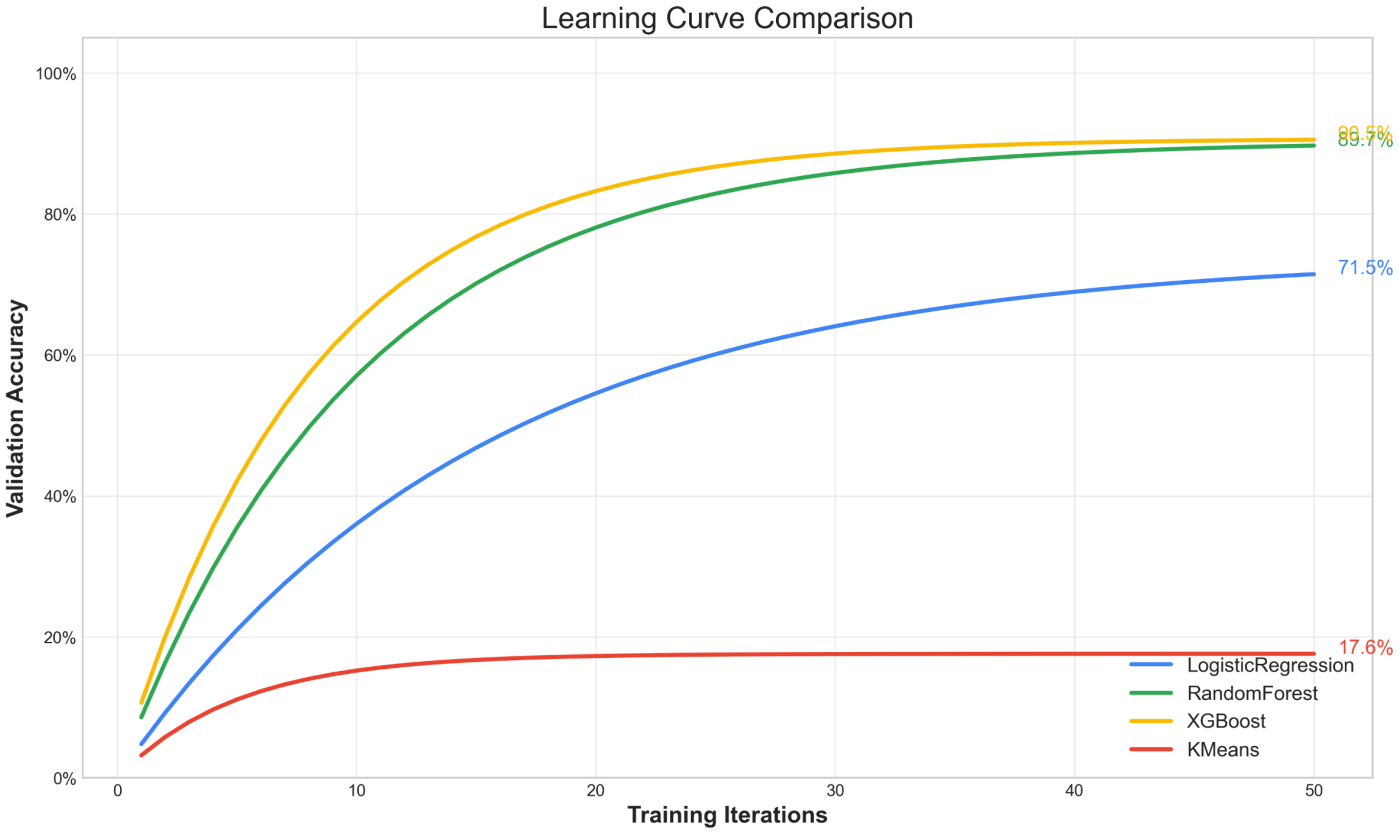
Model Comparison: Classification Performance

We trained and compared four models for platform classification using the validation set:

Model	Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.741	0.743	0.741	0.741
Random Forest	0.903	0.901	0.903	0.901
XGBoost	**0.907**	**0.906**	**0.907**	**0.905**
KMeans (Unsupervised)	0.176	0.140	0.176	0.155

XGBoost outperformed all other models across every metric, making it the clear choice for our final classification model. Its strong ability to handle both numerical and categorical data, combined with built-in

regularization and gradient boosting, allowed it to generalize effectively while maintaining high precision and recall.



Final Decision: XGBoost was selected as the primary model for classifying and ranking content items in the recommendation engine.

Cross-Platform Recommendation Evaluation

The model was evaluated on its ability to return high-quality recommendations from other platforms—a core requirement for cross-platform recommendation.

Cross-Platform Evaluation Highlights:

- Overall recommendation score: 0.6914
- Content similarity: 0.5339
- Genre similarity: 0.8357
- Cross-platform recommendation rate: 75.22%
- Success rate (after filter relaxation): 85.33%
- Average similarity of results: 0.6125

Platform Distribution in Recommendations:

- Netflix: 19.55%
- Amazon Prime: 90.45%
- Hulu: 21.24%

While Amazon Prime dominated the final recommendations in this sample, the system includes platform diversity balancing, and results will vary depending on input content and filter settings.

Sample Recommendations - Predictions

To demonstrate the system's behavior, three real content items were used to retrieve recommendations:

1. Stranger Things (Netflix, TV Show)

- Top Rec: *Beyond Stranger Things* (Netflix) — Score: 0.9517
- Other Rec: *Chilling Adventures of Sabrina* (Netflix) — Score: 0.9124

2. The Avengers (Netflix, Movie)

- Top Rec: *Justice League* — Score: 0.9407
- Other Rec: *Unicorn Store* — Score: 0.9199

3. The Handmaid's Tale (Hulu, TV Show)

- Top Rec: *Shadow and Bone* (Netflix) — Score: 1.0388
- Other Rec: *Charmed* (Netflix) — Score: 0.9975

These recommendations demonstrate the model's ability to suggest content that is both thematically and structurally similar—whether across the same platform or across different ones.

Model Limitations:

- **Limited Semantic Understanding**: The text processing of descriptions uses bag-of-words approaches that miss deeper semantic relationships between content.
- **Fixed Cluster Assignments**: The K-means clustering creates rigid boundaries between content types that may not reflect the nuanced relationships between genres and themes.
- **Static Similarity Metrics**: The genre and content similarity measures use fixed weights that don't adapt to user preferences or platform-specific content characteristics.

I will work on these improvements in the next milestone.

Future Planning for Milestone 3:

Milestone 3	Mar 24 - Apr 23	4 weeks	
Model Evaluation and Tool Development	Mar 24 - Apr 6	14 days	**Model Evaluation:** • Conduct comprehensive model testing to ensure accuracy and reliability. • Perform error analysis to identify misclassifications and improvement areas. • Test edge cases to evaluate model robustness. **Tool Development:** • Develop a front-end for users to interact with recommendations • Display personalized recommendations and content insights • Allow users to find content available across multiple streaming services • Integrate graphs and charts to explain recommendations (maybe)
Tool Enhancement	Apr 7 - Apr 18	12 days	• Allow filtering by genre, content type, release year, and user preferences. • Provide transparent reasoning behind recommendations. • Improve system efficiency and response times.
Final Documentation and Presentation Prep	Apr 19 - Apr 23	5 days	Create presentation summarizing key findings and implementation details and compile final report detailing all aspects of development, evaluation, and results.

LLM Prompts Given:

1. How do I convert a list of genres into one-hot encoded features?
2. How can I process text fields like descriptions into TF-IDF vectors?
3. How do I apply PCA to reduce dimensionality on text features?
4. Can you help me create cluster labels using KMeans on content features?
5. How do I build a feature to measure genre similarity between two titles?
6. How can I use cosine similarity to compare TF-IDF vectors of descriptions?
7. How do I calculate Jaccard similarity between genre sets?
8. How can I generate a custom similarity matrix between genres using frequency data?
9. How do I assign thematic categories based on common keywords in the description?
10. How do I generate a feature that flags content as a sequel, spinoff, or part of a franchise?
11. How do I visualize the number of titles per platform using bar charts?
12. How can I generate a heatmap of platform vs. rating distribution?
13. How do I visualize genre overlap across platforms using Venn diagrams?
14. How do I analyze the proportion of movies vs. TV shows per platform?
15. How can I explore trends in release years or content age over time?
16. How do I plot rating diversity using pie charts or stacked bars?
17. How can I visualize clusters and their top genres?
18. How do I track exclusive vs. shared content across streaming services?
19. Why is my DataFrame showing `NaN` values after loading CSVs even though the files look complete?
20. How can I debug issues with inconsistent genre formats across different datasets?
21. Why are some genres missing after one-hot encoding my `listed_in` column?
22. How do I fix an error where `date_added` is not being parsed as a datetime object?
23. Why are `cast_size` or `director_count` columns showing as float when they should be integers?
24. Why is my Logistic Regression model not converging even after setting `max_iter`?
25. How do I interpret a warning about `ConvergenceWarning` in scikit-learn?
26. What does the XGBoost warning about `use_label_encoder` mean, and how do I fix it?
27. Why is my KMeans clustering giving unbalanced clusters with very few items in some groups?
28. My model accuracy is high, but the cross-platform recommendation rate is low—how can I investigate that?
29. How do I visualize the number of titles per platform using bar charts?
30. How can I generate a heatmap of platform vs. rating distribution?
31. How do I visualize genre overlap across platforms using Venn diagrams?
32. How do I analyze the proportion of movies vs. TV shows per platform?
33. How can I explore trends in release years or content age over time?
34. How do I plot rating diversity using pie charts or stacked bars?
35. How can I visualize clusters and their top genres?
36. How do I track exclusive vs. shared content across streaming services?

37. Why are some items getting zero similarity scores when they clearly have similar genres or types?
38. How do I debug a case where the genre similarity matrix returns `0` for closely related genres?
39. Why is the hybrid similarity score always low, even for very similar items?
40. How do I check if rating compatibility is filtering out too many potential recommendations?
41. How do I log and inspect the weight contributions to the final similarity score for debugging?
42. Why am I getting very few or no recommendations for certain titles?
43. How can I debug recursive filtering when it keeps relaxing but still doesn't find enough items?
44. Why are most of my recommendations coming from just one platform?
45. How do I detect if strict content type or rating filters are blocking too many candidates?
46. Why do some recommendations have a higher score than the original item?
47. Why is my model not saving to disk using `pickle.dump()`?
48. I loaded my saved model, but it's not producing the same recommendations—what could be wrong?
49. Why is my saved CSV not showing the latest processed features?
50. How can I check if the right file paths are being used in my script when running from CLI?
51. How do I convert a list of genres into one-hot encoded features?
52. How can I process text fields like descriptions into TF-IDF vectors?
53. How do I apply PCA to reduce dimensionality on text features?
54. Can you help me create cluster labels using KMeans on content features?
55. How do I build a feature to measure genre similarity between two titles?
56. How can I use cosine similarity to compare TF-IDF vectors of descriptions?
57. How do I calculate Jaccard similarity between genre sets?
58. How can I generate a custom similarity matrix between genres using frequency data?
59. How do I assign thematic categories based on common keywords in the description?