Steam Game Recommender

Brian Woo

Table of contents

01
Business Problem

02
Exploratory Data Analysis

03
Modeling

04
Conclusion

05
Next Steps

01 Business Problem

Steam wants to improve their recommendation system, to attract more users to use their platform

Our tasks:

1. Analyze current gaming trends through ratings
2. Create models that will predict what users will rate a game they don't already own

02 Data Analysis

Dataset consists of over 40 million rows of data

- Games.csv
  - Title, Positive Ratio, Price, …
- Recommendations.csv
  - Helpful, funny, is_recommended, …
- Games_metadata.json
  - Description and Tags
- Users.csv
  - Products, Reviews

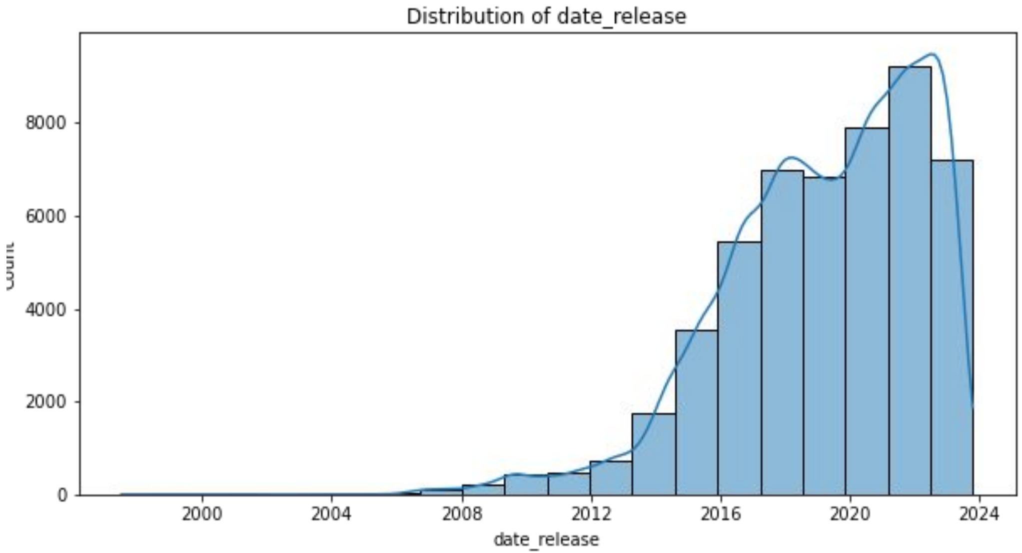
Stratified sample of ~100,000 rows

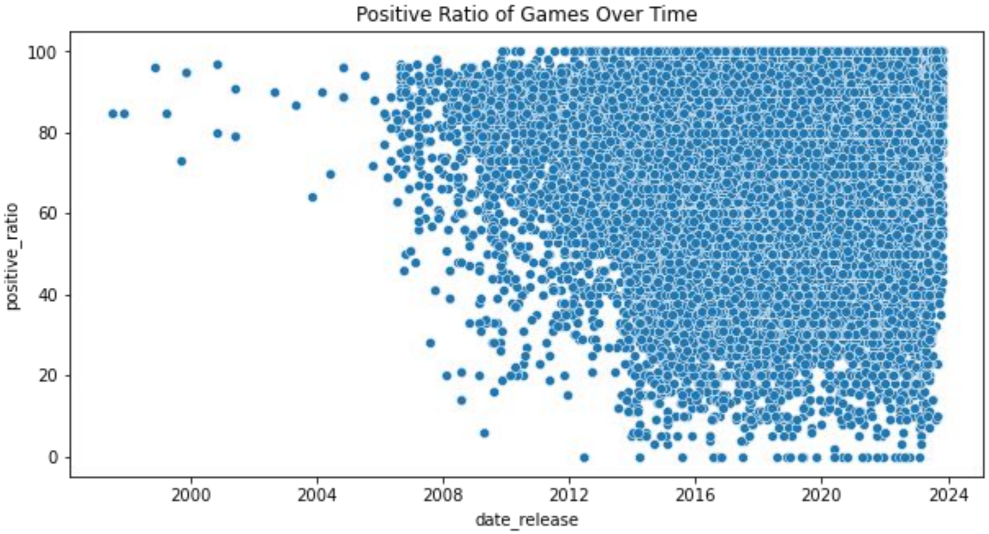
Source: Steam Game Recommendation

Data Filtering

Notes:
- More games released lately
- Highly rated games are more recent

1. Filtered dataset from 2010 - Present



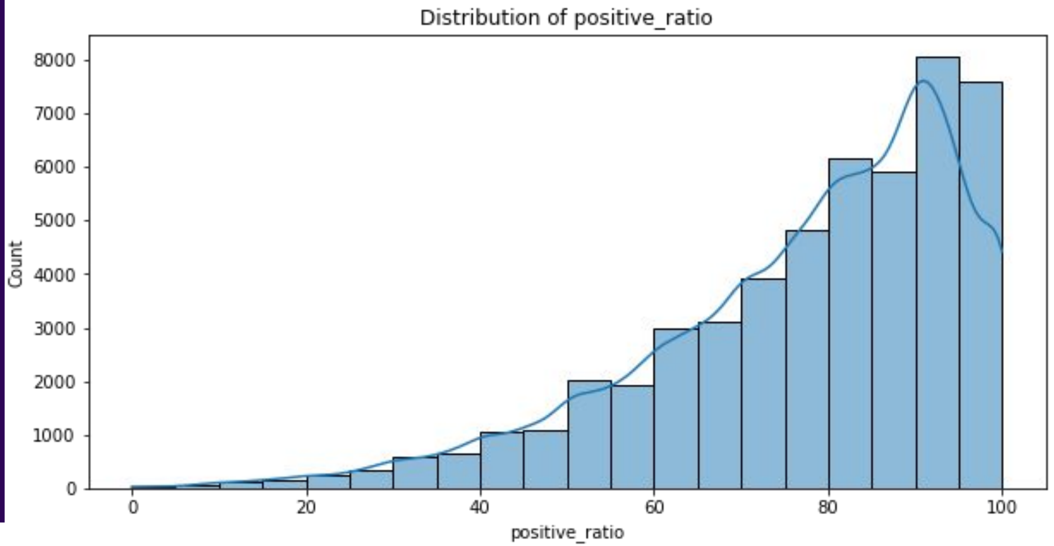


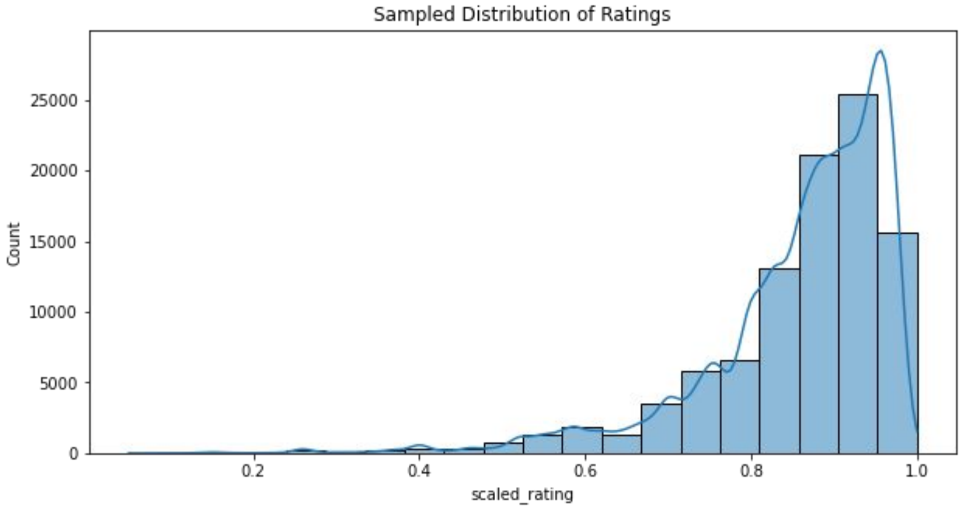
Data Filtering

Sampled dataset:

- ~ 100,000 rows
- Decent representation of data
- Majority of ratings are higher

1. Train test split using “stratify”
2. Train test split (1) for models





Target Variable: Positive Ratio

03 Modeling

Goal: Predict ratings of games users don't already have

Metrics:

- **Main Metric:** Root Mean Squared Error (RMSE)
  - How far predictions are from the actual values
- Mean Absolute Error (MAE)
  - Difference between the predicted and actual ratings
- Precision at *k* (P@k)
  - Proportion of recommended games in the top k that are relevant
- Recall at *k* (R@k)
  - Shows how well the recommender system is at capturing all relevant items within a limited list of recommendations

Collaborative-based Filtering Pipeline

1. Combine Data: Recommendations.csv & Games.csv
   - Columns: user_id, app_id, positive_ratio (scaled → 0 - 1)
   - ~ 40 million rows of data

2. Train Test Split the data to get ~100,000 rows of the data

3. Used Surprise library to Train Test Split
   a. Baseline and SVD: directly train/tune model
   b. KNN: Train Test Split again because of insufficient memory ~30,000 rows

4. Grid search to find best parameters

5. Evaluate the models
   - RSME, MAE, P@k, R@k
   - Sorted the ratings from best to worst

Collaborative-based Filtering Models

Baseline Model (not tuned)

- Acts as a baseline model to compare other models to
- Uses ALS as base model

Singular Value Decomposition (“n_factors”: 10, “n_epochs”: 40, “lr_all”: 0.02, “reg_all”: 0.1)

- Matrix Factorization technique
- Reduces the number of features of a dataset by reducing the space dimension

K-Nearest Neighbors (“k”: 10, “min_k”: 1, “sim_options”: {“name”: cosine, “user_based”: True})

- Finds the K nearest neighbors to a given data point based on distance metrics
    - Cosine, Pearson, Mean Squared Difference
- To run KNN I had to take a sample of the sampled dataset, which was ~ 30,000 rows

Collaborative-based Filtering Metrics

Baseline Model (not tuned)

Train RMSE: 0.052	Test RMSE: 0.060	Ave. Train P@10: 1.0	Ave. Test P@10: 1.0
Train MAE: 0.024	Test MAE: 0.028	Ave. Train R@10: 0.89	Ave. Test R@10: 0.88

Singular Value Decomposition (“n_factors”: 10, “n_epochs”: 40, “lr_all”: 0.02, “reg_all”: 0.1)

Train RMSE: 0.013	Test RMSE: 0.048	Ave. Train P@10: 0.99	Ave. Test P@10: 0.99
Train MAE: 0.008	Test MAE: 0.023	Ave. Train R@10: 0.96	Ave. Test R@10: 0.93

K-Nearest Neighbors (“k”: 80, “min_k”: 8, “sim_options”: {“name”: cosine, “user_based”: True})

Train RMSE: 0.076	Test RMSE: 0.114	Ave. Train P@10: 0.99	Ave. Test P@10: 0.99
Train MAE: 0.031	Test MAE: 0.083	Ave. Train R@10: 0.99	Ave. Test R@10: 0.93

Content-based Filtering Pipeline

1. Create Dataset: Games.csv & Games_metadata.csv
    - Columns: app_id, title, description, tags
    - ~ 50,000 rows of data

2. Combine all text data into one column

3. Preprocess the combined features using NLTK

4. Sklearn Train Test Split

5. Vectorize combined_features (TF-IDF)

6. Calculate Cosine Similarity through batches, because of memory issue

7. Evaluate the model
    - RSME, MAE, P@k, R@k
    - Sorted the similarity scores from best to worst

Content-based Filtering Model

NLTK & TF-IDF & Cosine Similarity

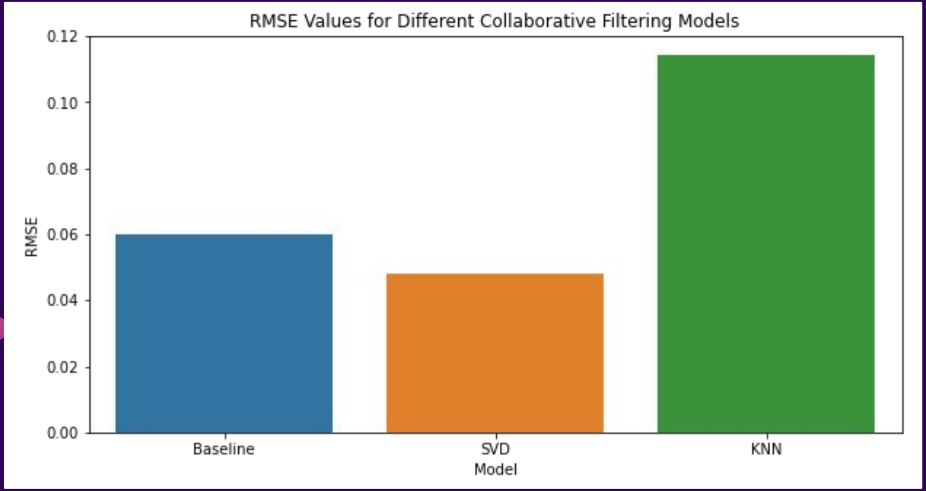
- NLTK (Natural Language Toolkit)
  - Tokenization, Lemmatization, Stopwords
- TF-IDF (Term Frequency-Inverse Document Frequency)
  - measures how important a word is to a document in a collection or corpus of documents
- Cosine Similarity determines the distance between the neighbors

Content-based Filtering Model

NLTK & TF-IDF & Cosine Similarity

Train RMSE: 0.084	Test RMSE: 0.090	Ave. Train P@10: 0.72	Ave. Test P@10: 0.73
Train MAE: 0.068	Test MAE: 0.067	Ave. Train R@10: 0.91	Ave. Test R@10: 0.89

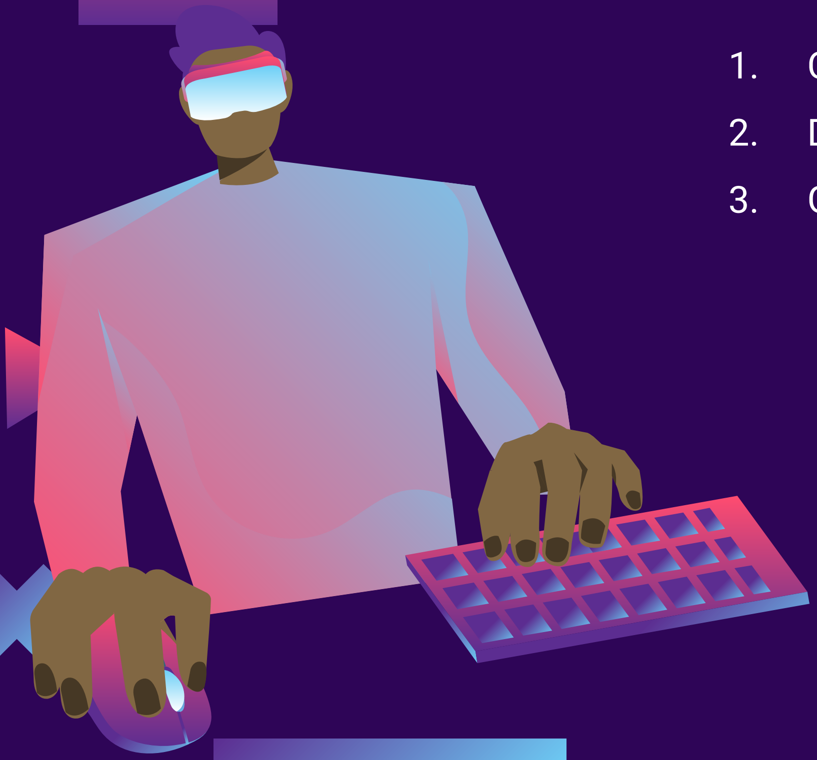
04 Conclusion



Suggestion: **SVD Collaborative Model**

- Lowest RMSE (main metric)
- All other metrics scored well
- Was able to run the most number of rows (more variability)

05 Next Steps



1. Create a Hybrid Models / Explore other models
2. Deploy and test on the web for real users
3. Get more RAM!!

Thanks!

Do you have any questions?

brianhwwoo@gmail.com