

Rotten Tomato Score Predictor

Project Definition

The goal of this project is to forecast the Rotten Tomatoes critic score by leveraging audience sentiment represented in YouTube trailer comments. The strategic components of this involve collecting, aligning, and curating disparate data sources, YouTube comments and Rotten Tomatoes scores, into a coherent dataset for predictive analysis. This also closely relates to the topics of lectures on data curation, cleaning, and integration, and the application of machine learning models in data-driven decision-making. It also highlights the potential value that can be obtained from user-generated content in predictive analytics.

Importance

Our project is important as it demonstrates how real-life, unstructured data generated by the public, such as YouTube trailer comments, can be transformed into valuable insights. It provides an accessible metric for studios, marketers, and audiences to gauge potential reception trends before a movie's release, even ahead of official critic reviews, highlighting how data science can reveal connections between public sentiment and expert assessments. While similar work has been done in this field, our approach stands out by offering a scalable way for studios to anticipate public reactions without the need for human professionals. This method also enables studios to produce smaller-scale test trailers to evaluate audience interest, helping them make informed decisions about pursuing full-scale productions.

Data Curation

- First, we identified what data we would require for this project, since we are looking to use trailer comments sentiment to predict rotten tomatoes scores, it was clear we would need to get both a list of comments and correlating scores.

Rotten Tomato Scores Scraping:

- To get the scores we started by searching through the website, to determine if there is any easy pattern we could use to scrape the data. After further investigation, we noticed that the website would populate the information by requesting a public API to get the data and then dynamically load that information into the site.
- After inspecting this API request we saw that the return data was in JSON format, which would be easy to parse through.
- By copying the request in cURL format, we could translate it into Python code and create a function to pull all of the required data.
- Since we were unsure which movie trailers we were going to get, we decided to cast a wide net and get as many Rotten Tomato scores as possible.
 - In doing so we would cache the data that we have received into a .json file in case there were any problems with the API requests during the function call.
- In addition to this, we also would clean the data returned by the API call in case any problems occurred.

```

# Click to add a breakpoint is the rotten tomato page and pulls all the names of the movies
# and their audience score and audience score
# It does this by making a question to the public API url that is used to load the page, and then
# read the json.

# Since we are working with a large data set it we made it so that the data was constantly dumped into a .json file
# incase anything went wrong along the way.
fetch_movies(limit=10) # limited to 10 for this example
✓ 0s

Movie Count: 1
Title: An Evening with Dua Lipa
Audience Score:
Critics Score:

Movie Count: 2
Title: Juror #2
Audience Score: 91%
Critics Score: 93%


Movie Count: 3
Title: Conclave
Audience Score: 86%
Critics Score: 93%
-----
Movie Count: 4
Title: The Substance
Audience Score: 75%
Critics Score: 98%
-----
Movie Count: 5
Title: Joy
Audience Score: 88%
Critics Score: 89%
-----
...
Title: AfrId
Audience Score: 51%
Critics Score: 22%

```

YouTube Comments Scraping:

- Much like the Rotten Tomato scores, we wanted to get as big of a sample size as possible, so we decided to find YouTube channels that posted a lot of trailers, and then filter out for videos that had trailers. This we found to be the best way as we did not need to handpick and find all of these movies.
- Similar to the Rotten Tomato scores, we thought it would be easier for us to use some sort of API to get the data instead of trying to scrape the YouTube interface.
- To do this, we generated a Google API key with authorization to the YouTube client.
- We then created the YouTube class of functions which would include the following functions:
 - 1) **get_channel_id**: a function that would retrieve the YouTube ID for the channel based on the channel name. (helper function)
 - 2) **get_video_names_and_ids**: this would receive all the IDs and titles of the videos in a channel based on a channel name.
 - 3) **get_video_comments**: this would retrieve the first n number of comments from a video and return a list.
 - a) **Clean_comments**: helper function to ensure data is clean.

Final Web-Scraping Plan:

- 1) We decided on a set of channels that would be used for our data set and pulled all of the video names and IDs in that channel.
 - In this step, we filtered for video titles that had the word “trailer” in them to ensure it was a trailer for a movie.
- 2) We imported our Rotten Tomato scores and began trying to match up those with our movie trailers.
 - This step was fairly tricky; we were looking for a substring (the movie title from Rotten Tomatoes) to a full string that contained noise (the YouTube trailer title). Because of this, we ran into issues where movies with simple names would often get picked up as matches for multiple movies.
 - To combat this, we generated an “ignore” list of movie names that were common and had the program ignore those
 - In addition to this, we also replaced some problematic words in the trailer titles with empty strings to ensure the matches were still good.
 - Once all of the above was done, we were left with about 150-200 movie “matches”, however when looking through them at a glance, we didn’t feel good about the match accuracy. Since it was a relatively small data set, we quickly did some manual corrections, comparing the video titles with the rotten tomato scores and ensuring the matches were accurate.
- 3) Now that we had our video matches, all we had left to do was generate the dataset and include the comments/sentiments.
 - We originally began by trying to use the Pandas library to complete this step, however, we continued to run into formatting issues.
 - Instead, we decided to use the CSV library to generate the file line by line.

- To do this we created the column header rows (including the 100 comments and 100 sentiment scores).
- Then using the YouTube function class and an additional Sentiment class that we created to generate the scores, we were able to append to the CSV line by line.
- Finally, once all of that was complete we saved the data to the CSV and validated that there were no errors that occurred while generating the data. If an error occurred we would skip that line and it would generate an empty list.

```

Success count: 1
Success count: 2
Success count: 3
Success count: 4
Success count: 5
Youtube Problem Comments for Paddington 2: 100
Success count: 6
Success count: 7
Success count: 8
Success count: 9
Success count: 10
Success count: 11
Success count: 12
Success count: 13
Success count: 14
Success count: 15
Success count: 16
Success count: 17
Youtube Problem Comments for Doctor Strange in the Multiverse of Madness: 100
Success count: 18
Success count: 19
Success count: 20
Success count: 21
Success count: 22

```

This is an image of the validation process as we appended rows to the CSV, notice that we were able to track when a problem occurred when generating the YouTube comments, as well as validate that the sentiment class did not have any errors.

```

#check that all worked fine
sentimeAdmin.countBad()

0

```

Data Validation:

- Validated sentiment columns (comment_sentiment_1 to comment_sentiment_100) to ensure all required data was present, dropping rows with missing values and converting invalid values to 0.
- Filtered out rows where more than 43 sentiment columns had a value of 0 to reduce noise in the dataset.
- Assigned sentiment columns as features (X) and critics scores as target values (y)
- Dropped rows with missing or invalid target values to ensure a clean dataset

Splitting Data:

- Split the cleaned dataset into an 80/20 ratio for training and testing
- Out of 70 valid rows, 44 were left after reducing noise leaving us with 35 in the training set and 9 in the testing set

Final Model:

- Implemented XGBoost regression model with hyperparameters for regularization
- Computed weights based on score distributions to balance any skewness.
- Trained the model on the training set with weighted samples for better generalization.
- Evaluated the model using metrics such as Mean Absolute Error (MAE), Root Mean Square Error (RMSE), R², and percent accuracy to assess prediction performance.

```

# Computes weights inversely proportional to the frequency of scores to balance skewed distribution
def compute_weights(critic_scores):
    bins = [0, 20, 40, 60, 80, 100]
    counts, _ = np.histogram(critic_scores, bins=bins)
    weights = 1 / np.sqrt(counts + 1e-6)
    bin_indices = np.digitize(critic_scores, bins) - 1
    bin_indices = np.clip(bin_indices, 0, len(weights) - 1)
    return weights[bin_indices]

# Initialize model with regularization
model = xgb.XGBRegressor(
    learning_rate=0.1,
    n_estimators=200,
    max_depth=4,
    subsample=0.8,           # Use 80% of the data per tree
    colsample_bytree=0.8,     # Use 80% of the features per tree
    reg_alpha=0.1,           # L1 regularization term
    reg_lambda=1.0,           # L2 regularization term
    random_state=42
)

# Train model with weights
model.fit(
    X_train,
    y_train,
    sample_weight=compute_weights(y_train)
)

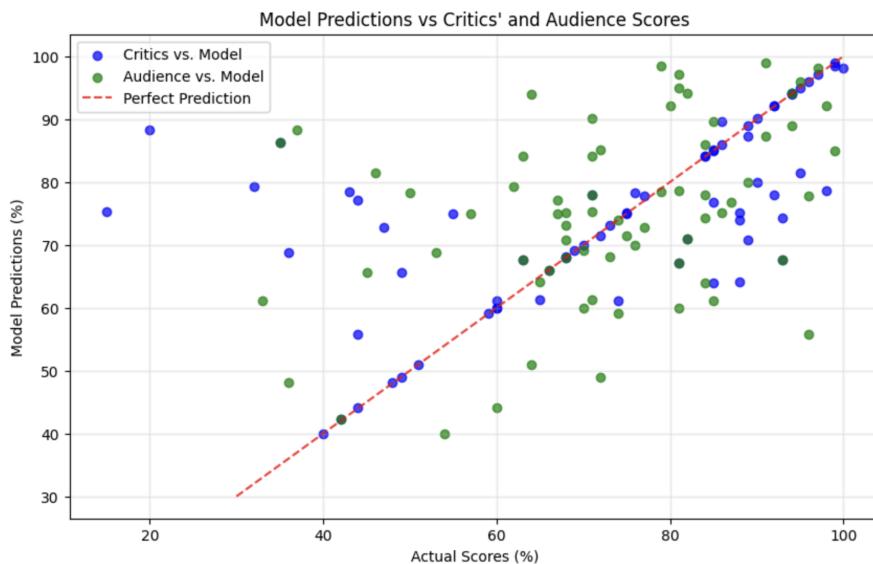
```

Prediction Storage:

- Created an SQLite database table (PredictedScores) with columns for MovieID, Title, AudienceScore, CriticScore, and PredictedScore.
- Iteratively predicted scores for each movie out of the 70 valid rows using the trained model and stored the results in the database.

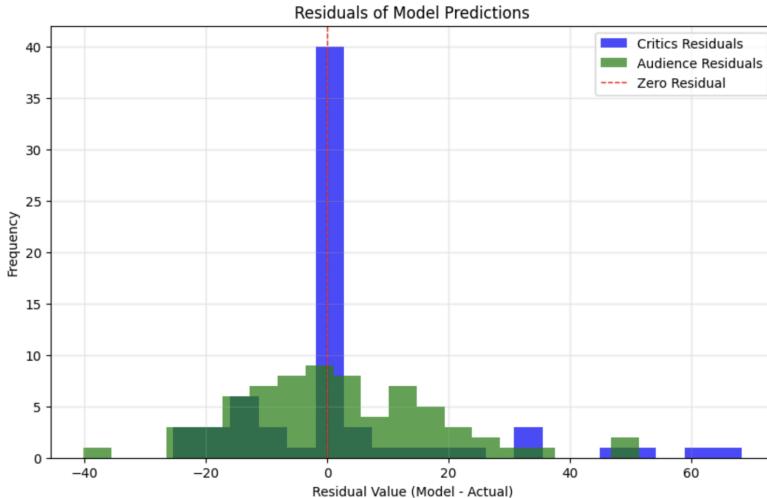
Visualization and Analysis:

- Plotted comparisons between predicted scores, critic scores, and audience scores using scatter plots, histograms, box plots, and bar charts to assess model accuracy.
- Our model was trained to predict critic scores, but it was trained using audience sentiment, so we included both Rotten Tomatoes scores because we thought it would be interesting to analyze how well it predicted the audience score as well.
- **Scatter Plot:**
 - Compared predicted scores against critic and audience scores, with a "perfect prediction" line for reference.
 - **Purpose:** Showed the accuracy and distribution of predictions relative to actual scores, highlighting patterns and outliers.



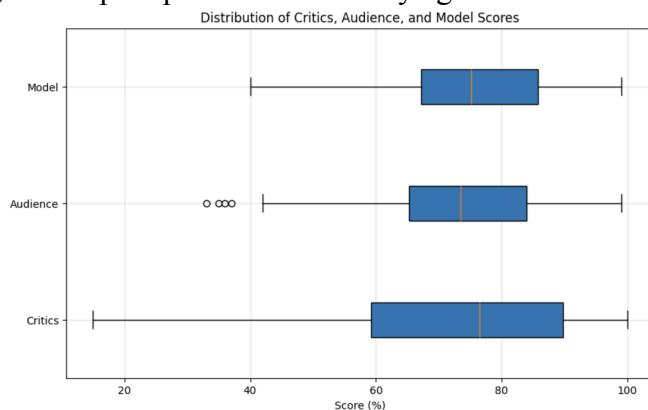
- **Residual Histogram:**

- Visualized the differences between predicted scores and actual scores for both critics and audience.
- **Purpose:** Assessed the spread and symmetry of residuals to evaluate model variance.



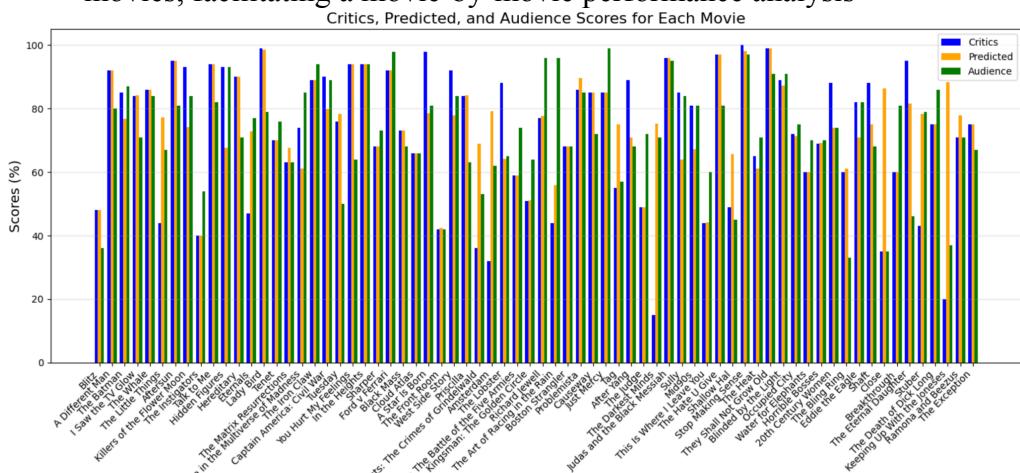
- **Boxplot:**

- Showed the distribution of critics, audience, and predicted scores.
- **Purpose:** Provided a summary of the data spread, including medians and potential outliers, to compare prediction accuracy against actual scores.

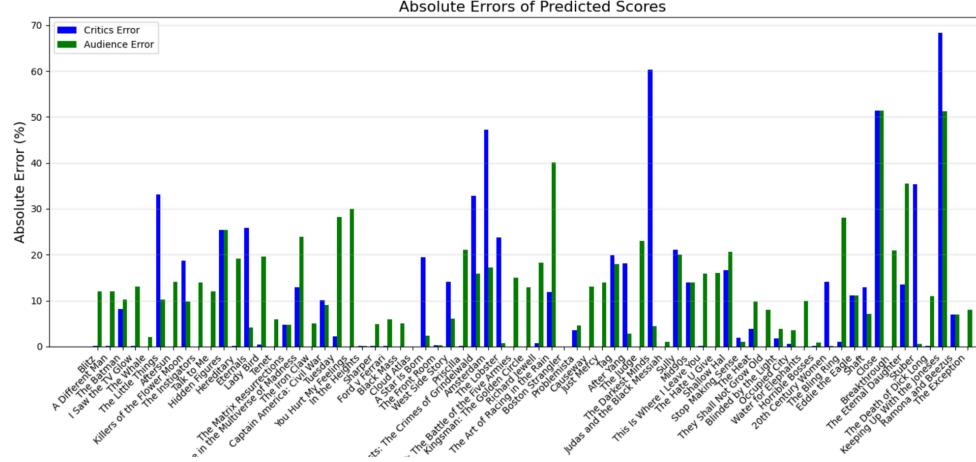


- **Score Bar Chart:**

- Displayed critic, audience, and predicted scores side-by-side for each movie.
- **Purpose:** Illustrated the differences and similarities in scores across individual movies, facilitating a movie-by-movie performance analysis

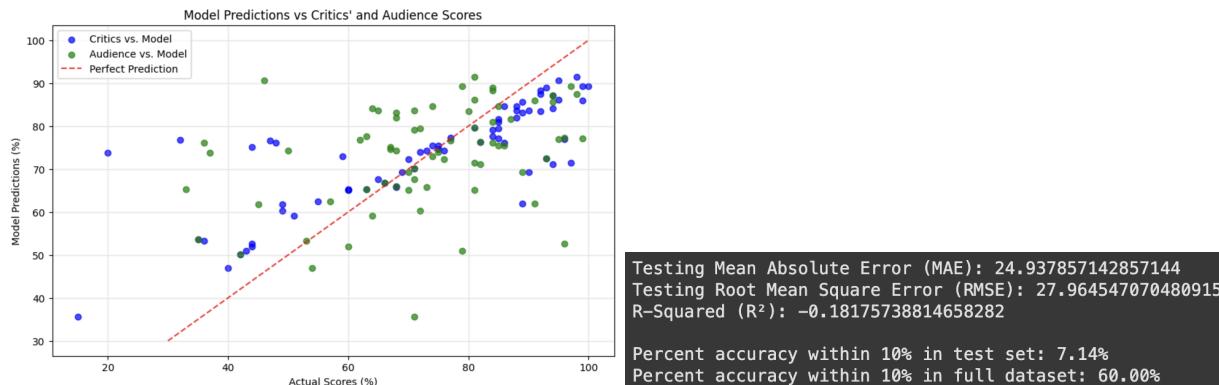


- **Error Bar Chart:**
 - Highlighted absolute errors between predicted and actual scores for both critics and audience.
 - **Purpose:** Quantified the model's prediction errors, showing which movies had larger deviations and providing a focus for improvement.



Model Optimization Steps:

Initial Model Metrics-



Model Selection-

- Experimented with various models including RandomForestRegressor, LinearRegression, Ridge, GradientBoostingRegressor, and finally XGBRegressor.

Stacking Models-

- Implemented a Stacking Regressor combining Random Forest, Gradient Boosting, and Ridge Regression as the final estimator, this did not improve model performance.

Hyperparameter Tuning

- Conducted grid search optimization to fine-tune parameters like n_estimators, max_depth, and min_samples_split.
 - Used cross-validation to ensure robust parameter selection and evaluated the tuned model's performance on the test set, but this did not improve the model either.

Principal Component Analysis (PCA)

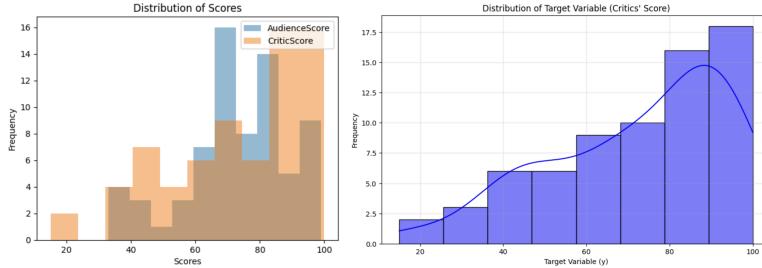
- Applied PCA to reduce the dataset's dimensionality and tested various numbers of components, which initially improved the model's performance significantly.
 - Observed a trade-off between lower variance and loss of information, leading to the decision to remove PCA for the final model when the dataset was less noisy.

Noise Removal-

- Dropped rows with excessive zeros to ensure sentiment columns contained useful data.
- Tested different thresholds for the percentage of zero sentiments to retain important reactions while reducing noise.

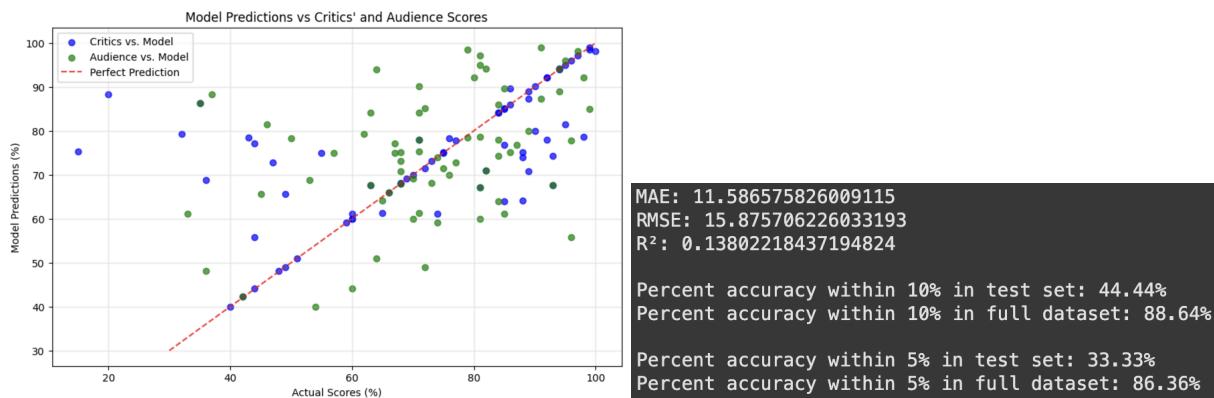
Target Variable (y) Skewness-

- Analyzed the skewness of the target variable (y) using histograms and statistical metrics:



- Applied transformations (logarithmic and exponential), but both did not improve model accuracy or skewness.
- Created a weight loss function to adjust weights based on the frequency of score ranges to balance skewed distribution, which significantly improved the model.

Final Model Metrics-



Key Findings

The key findings from the project demonstrated that audience sentiment from YouTube trailer comments can reasonably predict Rotten Tomatoes critic scores, validating the original hypothesis. The XGBoost regression model achieved predictions that aligned with critic scores, although challenges such as the dataset's limited size and variability introduced some constraints on accuracy. Evaluation metrics like MAE, RMSE, and R^2 were used together with visualizations of scatter plots, histograms, and error charts to show strengths as well as areas for improvement in the model. Results confirm the viability of sentiment-based predictions to approximate critic scores. This demonstrates a correlation between audience sentiment and critical evaluations, showing that public opinion can serve as a predictive tool for professional assessments.

Differences

The main differences between the original proposal and the final execution of the project are as follows: While the proposal intended to use BERT for sentiment analysis, storing comments in a PostgreSQL database, the actual implementation made use of a far simpler version of sentiment analysis, TextBlob, storing YouTube comments in a CSV and the scores and predictions in an SQLite database. The use of Rotten Tomatoes and YouTube APIs for data collection imposed practical limitations, such as the need for manual corrections in matching movie titles, which were not envisioned prior. These changes reflect necessary adjustments made to address the challenges encountered during data collection and processing.