Churn Prediciton Model

Data processing

Data Sources and Preparation

- **Game 1 Data:**
  - Structured as `.jsonl` files under `src/data/processed/game1_player_events.jsonl` (both eval and train sets), generated using `data_preparation.py`.
- **Churn Labeling:**
  - Churn prediction is based on an 80% threshold over a churn prediction period of 10 days, using `dataset_creation.py`.
- **Feature Engineering:**
  - Features are prepared for both games using `feature_engineering.py` (we use the reward in game 2 as the score):
    - **Game 1 Features:** `playCount`, `bestScore`, `meanScore`, `worstScore`, `sdScore`, `bestScoreIndex`, `bestSubMeanCount`, `bestSubMeanRatio`, `activeDuration`, `consecutivePlayRatio`
    - **Game 2 Features:** `playCount`, `bestScore`, `meanScore`, `worstScore`, `sdScore`, `bestScoreIndex`, `bestSubMeanCount`, `bestSubMeanRatio`, `activeDuration`, `consecutivePlayRatio`, `purchaseCount`, `highestPrice`

Models Used

Three different classification models were implemented using model_training.py:

- **Decision Tree Classifier**
- **Random Forest Classifier**
- **Logistic Regression**

Metrics Comparison

Game 1

- **Best Performing:** Logistic Regression
  - Highest accuracy (**92.91%**)
  - Best AUC-ROC score (**0.778**)
  - Excellent recall (**99.19%**)
- **Close Second:** Random Forest
  - Very similar accuracy (**92.82%**)
  - Good AUC-ROC score (**0.758**)

    - High recall (**98.98%**)
- **Worst Performing:** Decision Tree
    - Lower accuracy (**89.62%**)
    - Poor AUC-ROC score (**0.536**)

Game 2

- **Best Performing:** Logistic Regression
    - Highest accuracy (**78.15%**)
    - Best AUC-ROC score (**0.731**)
    - Highest recall (**95.83%**)
- **Close Second:** Random Forest
    - Good accuracy (**77.83%**)
    - Decent AUC-ROC score (**0.708**)
- **Worst Performing:** Decision Tree
    - Much lower accuracy (**68.53%**)
    - Poor AUC-ROC score (**0.593**)

Key Observations

- All models performed significantly better on **Game 1** than **Game 2** because game2 involve more cases and event types.
- **Logistic Regression** consistently performed best across both games.
- **Decision Trees** performed worst, suggesting the data may be too complex for simple tree-based decisions.
- The performance gap between models is larger in **Game 2** because it is more complex than **Game 1**.

Model Training Process and Code Insights

The model training process is implemented in Python using scikit-learn and pandas, as detailed in the `model_training.py` script. The workflow for both games is as follows:

1. **Data Loading:**

    - Processed feature datasets for each game and split (train/test) are loaded from CSV files.

2. **Model Training:**

    - Three classifiers are initialized: Decision Tree, Random Forest (with 100 estimators), and Logistic Regression.
    - Each model is trained on the training data. Logistic Regression uses the scaled features, while tree-based models use the raw features.

3. **Evaluation:**

    - Each trained model is evaluated on the test set using accuracy, precision, recall, F1 score, and AUC-ROC.
    - A comparison plot of all model metrics is also generated for each game.

4. **Results:**

    - All evaluation metrics are saved as JSON files for further analysis and reporting under resutlts folder.

Code Structure Highlights:

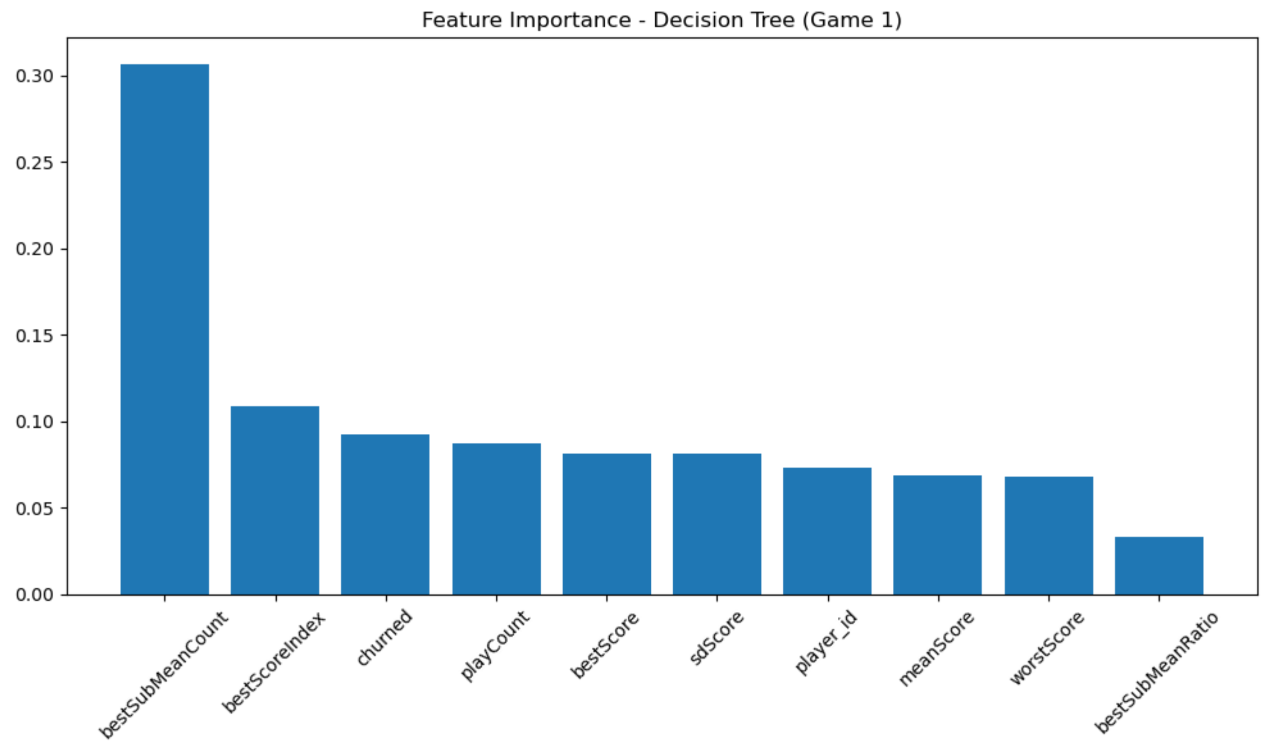
- The code is modular for each task we have a module containing functions for loading data, preparing features, training models, evaluating, and plotting.
- The main loop iterates over both games.
- The use of scikit-learn's API allows for easy extension or modification of models and metrics.

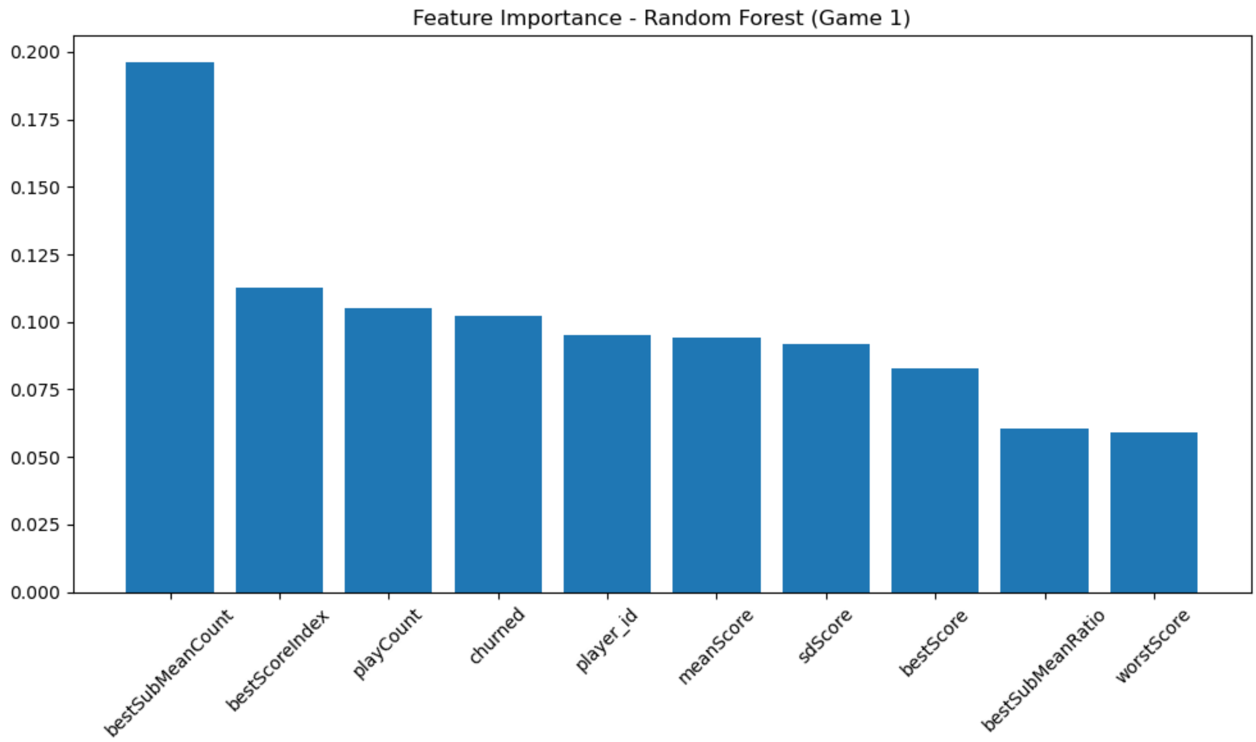
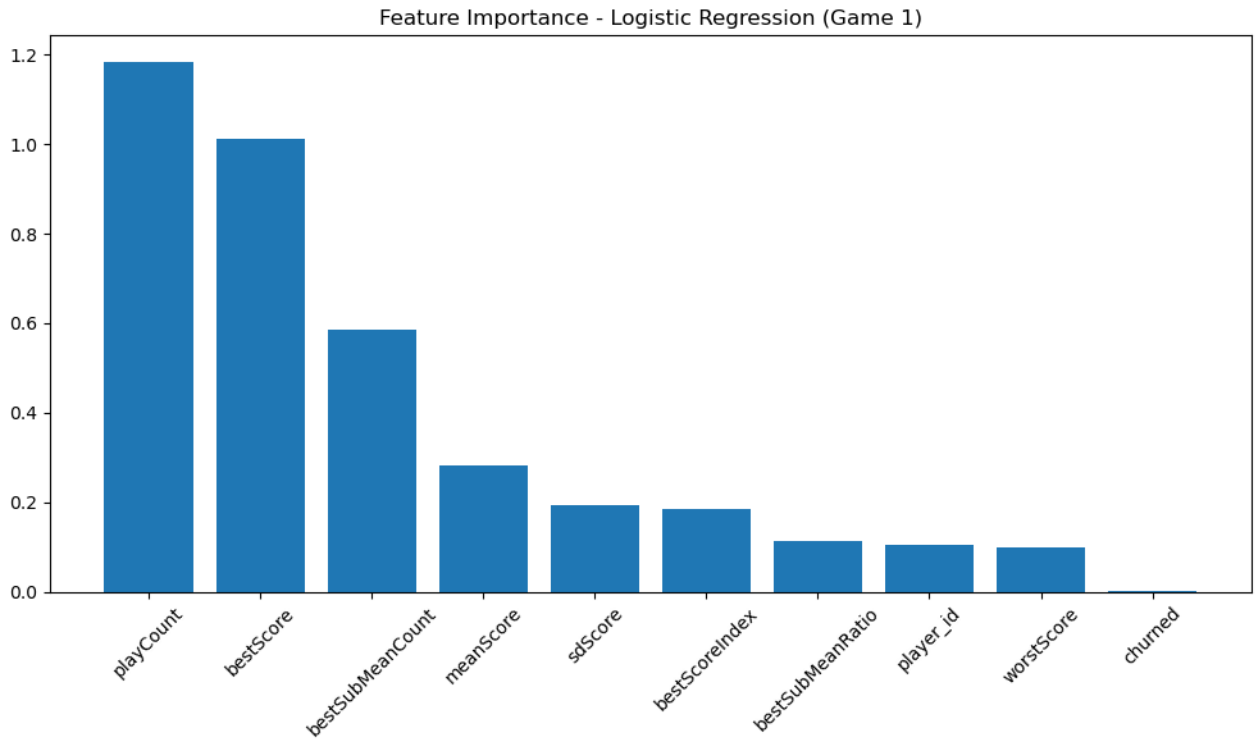
This structured approach ensures reproducibility and clarity in the model evaluation process, making it easier to compare model performance across different games and datasets.

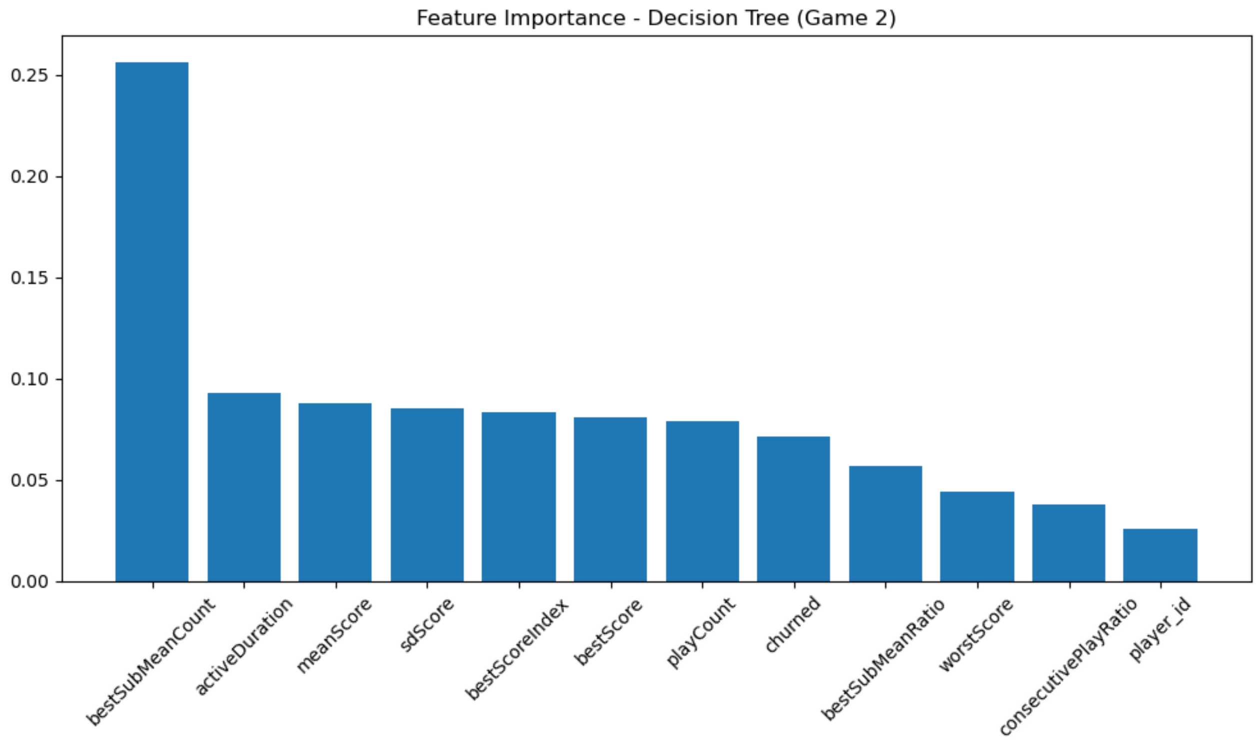
Plots and Models Comparsion

To further support the evaluation, the following visualizations are included:

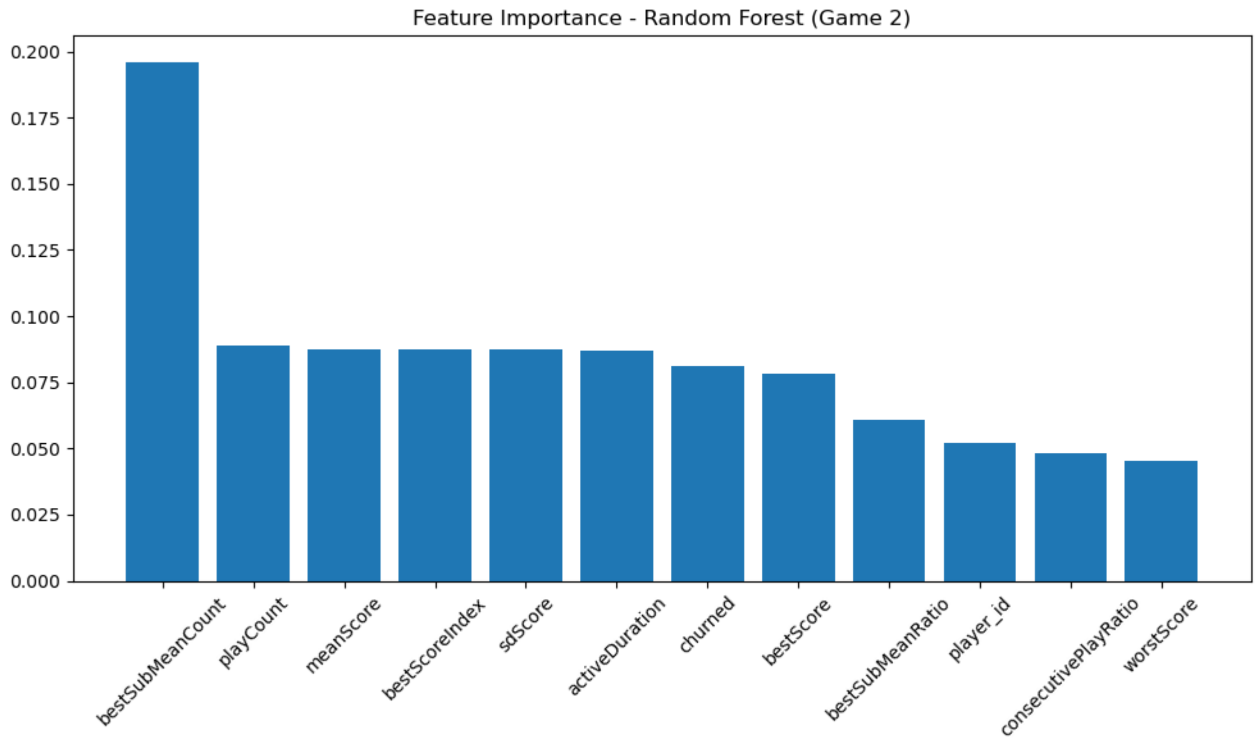
Feature Importance Plots

- 

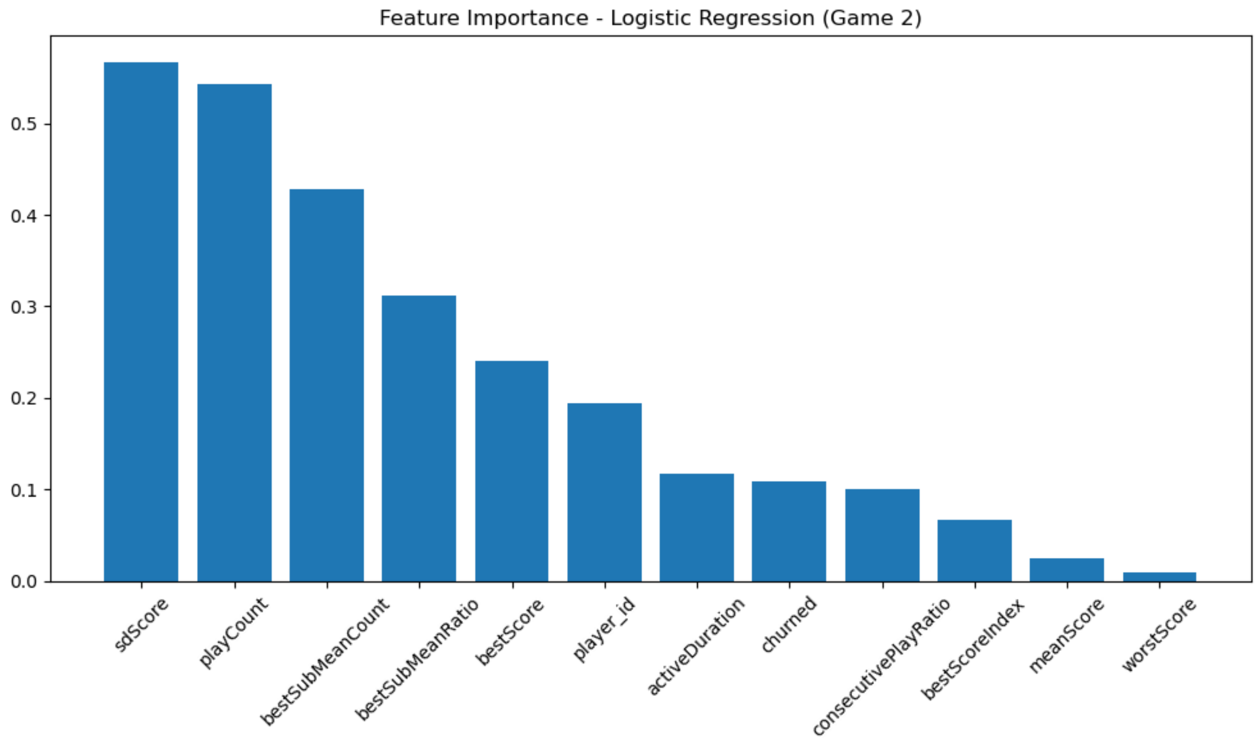
- 
Feature Importance - Random Forest (Game 1)
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
bestSubMeanCount
bestScoreIndex
playCount
churned
player_id
meanScore
sdScore
bestScore
bestSubMeanRatio
worstScore
- 
Feature Importance - Logistic Regression (Game 1)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
playCount
bestScore
bestSubMeanCount
meanScore
sdScore
bestScoreIndex
bestSubMeanRatio
player_id
worstScore
churned
-

- 

Feature Importance - Decision Tree (Game 2)
0.25
0.20
0.15
0.10
0.05
0.00
bestSubMeanCount
activeDuration
meanScore
sdScore
bestScoreIndex
bestScore
playCount
churned
bestSubMeanRatio
worstScore
consecutivePlayRatio
player_id

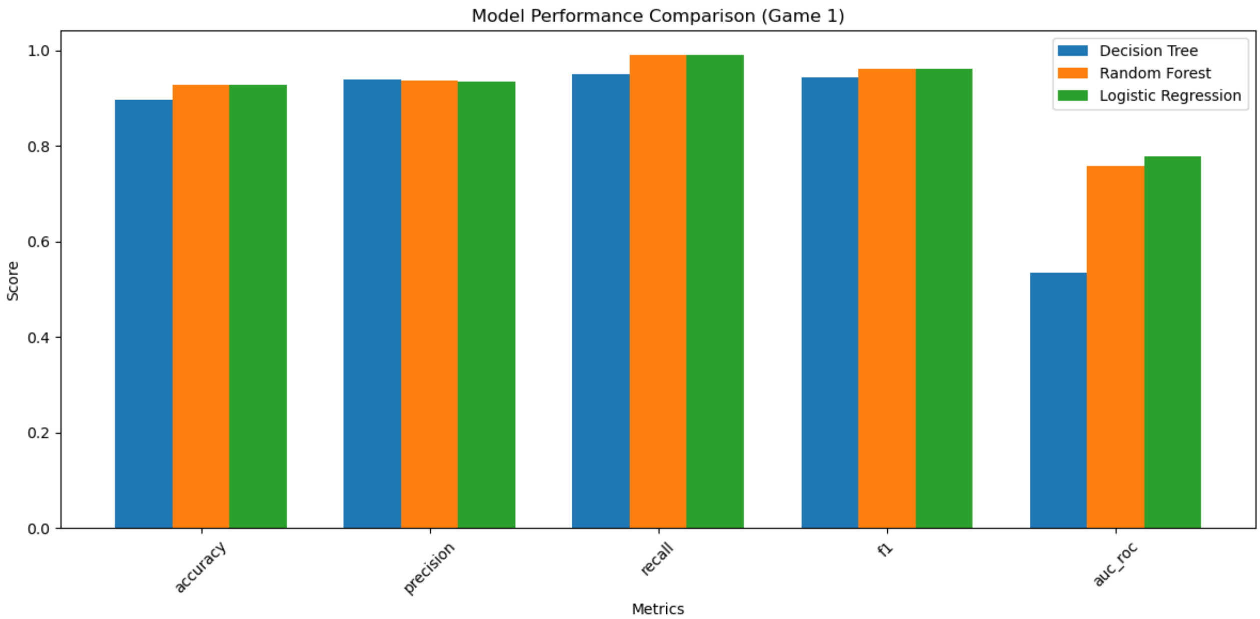
- 

Feature Importance - Random Forest (Game 2)
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
bestSubMeanCount
playCount
meanScore
bestScoreIndex
sdScore
activeDuration
churned
bestScore
bestSubMeanRatio
player_id
consecutivePlayRatio
worstScore

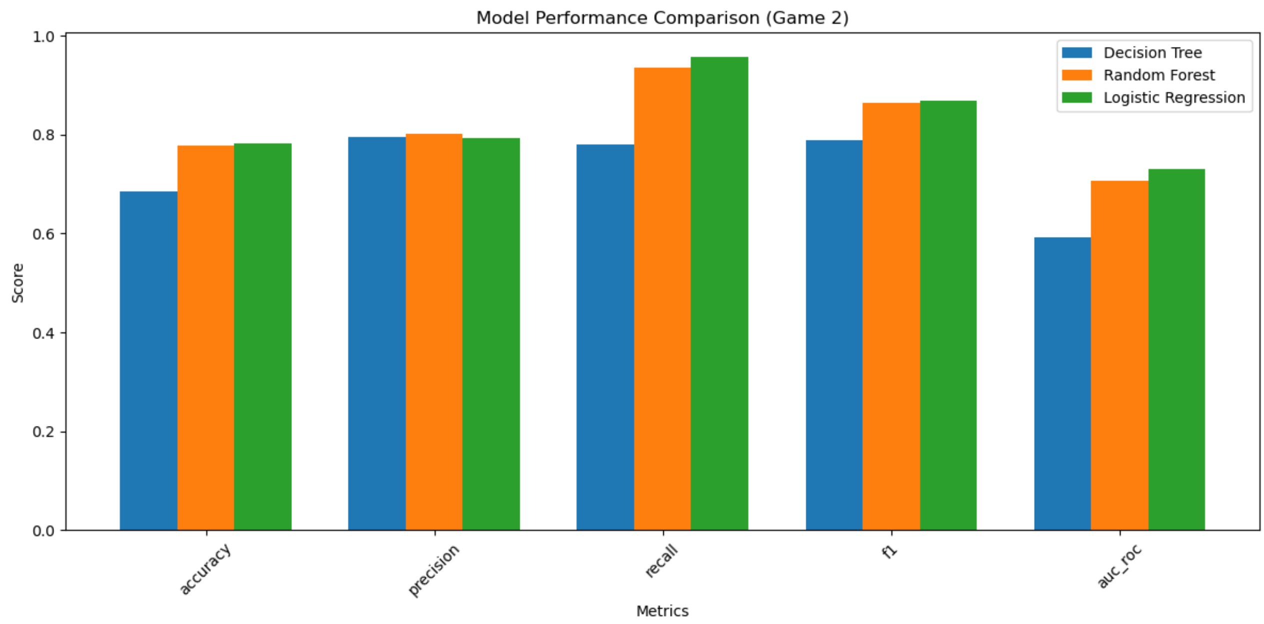


-

Model Metrics Comparison



-



These images provide a visual summary of feature importance for each model and a direct comparison of model performance metrics for both games.

LLM-based Churn Prediction or Analysis

In addition to traditional machine learning models, we implemented and evaluated a **DistilBERT-based** approach for churn prediction. This transformer-based model was tested on both games to explore the potential of natural language processing techniques for player behavior analysis.

DistilBERT Model Performance

Game 1 Results

- **Accuracy:** 50.00%
- **Precision:** 50.00%
- **Recall:** 100.00%
- **F1 Score:** 66.67%
- **AUC-ROC:** 0.00

Game 2 Results

- **Accuracy:** 50.00%
- **Precision:** 50.00%
- **Recall:** 100.00%
- **F1 Score:** 66.67%
- **AUC-ROC:** 0.00

Analysis of LLM Results

The DistilBERT model showed **significantly poorer performance** compared to the traditional machine learning approaches:

1. **Random Performance**: The 50% accuracy across both games suggests the model is performing at random chance level, indicating it has not learned meaningful patterns from the data.

2. **Perfect Recall, Poor Precision**: The 100% recall with 50% precision indicates the model is classifying all or most players as churned, which is not useful for practical churn prediction.

3. **Zero AUC-ROC**: An AUC-ROC score of 0.00 suggests the model has no discriminative ability between churned and non-churned players.

4. **Consistent Poor Performance**: The identical metrics across both games suggest systematic issues with either:

    - Data preprocessing for the LLM model
    - Feature representation incompatibility with transformer architectures
    - Model configuration or training issues

Comparison with Traditional ML Models

Model Type	Game 1 Accuracy	Game 2 Accuracy	Best Traditional ML
DistilBERT	50.00%	50.00%	Poor
Logistic Regression	92.91%	78.15%	Excellent
Random Forest	92.82%	77.83%	Very Good
Decision Tree	89.62%	68.53%	Good

The traditional machine learning models significantly outperformed the LLM-based approach, with **Logistic Regression achieving ~43-28 percentage points higher accuracy** than DistilBERT.

Recommendations for LLM Approaches

1. **Feature Engineering**: The current numerical features may not be suitable for transformer models. Consider creating textual representations of player behavior patterns.

2. **Model Architecture**: Explore specialized architectures designed for tabular data or time series, rather than text-focused transformers.

3. **Data Preprocessing**: Investigate different ways to represent player behavioral data that align better with LLM input expectations.

4. **Hybrid Approaches**: Consider combining traditional ML features with LLM-generated embeddings or insights.

For this churn prediction task, **traditional machine learning models remain the superior choice**, particularly Logistic Regression and Random Forest, which demonstrate both high accuracy and practical interpretability.

Project Structure Overview

Below is the key modules involved in the pipeline:

```
project-root/
├── src/
│   ├── data_processing/
│   │   ├── data_preparation.py      # Data loading, cleaning, and splitting
│   │   ├── dataset_creation.py      # Observation/churn period labeling
│   │   ├── feature_engineering.py   # Feature extraction from player data
│   │   └── pipeline.py              # Orchestrates the data pipeline
│   ├── model_training.py            # Model training, evaluation, and reporting
│   ├── llm_prediction.py            # LLM-based prediction module
│   └── ...
├── results/
│   ├── features/                    # Extracted feature CSVs
│   ├── models/                      # Trained model files
│   └── ...
├── plots/                           # Visualizations
└── ...
```

- **data_preparation.py**: Handles raw data conversion, cleaning, and splitting into train/eval sets.
- **feature_engineering.py**: Extracts behavioral features from player event logs for ML.
- **model_training.py**: Trains and evaluates machine learning models, generates metrics and plots.
- **llm_prediction.py**: (Optional) Module for LLM-based churn prediction or analysis.

This modular structure ensures each stage of the pipeline is clear, maintainable, and extensible for future work.