Steam Review Radar

A Pipeline for Helpfulness Prediction, Reviewer Characterization, and Spam Detection

Students: Cody Jiang (zj2247) • Andrew Wang (aw5574) • Zhiding Zhou (zz10140)

Course: FA 25 Big Data CS-GY 6513, Section E

Project Abstract

Gaming platforms such as Steam host hundreds of thousands of games, creating a discovery problem where users can struggle to find the ones that they might enjoy. User reviews server as an important signal for both Steam's recommendation algorithm and the users' purchasing decisions. Due to this value, there are bound to be bad actors attempting to undermine the integrity of these signals.

The goal of this project is to build a compact, production-oriented pipeline using a historical Steam user-review dataset to (1) **predict helpfulness** of newly posted reviews in real time, (2) **categorize reviewers** into archetypes based on differences in writing style and behavior, and (3) detect and down-weight **template and near-duplicate** reviews that are attempting to inflate metrics or distort downstream model accuracy. This system will employ Apache Spark for distributed ETL processing for the Steam user-review dataset, lightweight ML for **count prediction** (Negative-Binomial / XGBoost), **unsupervised clustering** for characterizations, and **MinHash/Isolation-Forest** for spam detection. Deliverables will include reproducible Jupyter notebooks, a metrics report (RMSE, MAE, Spearman, calibration), and characterization results.

Problem Statement and Objectives

Problem

Steam relies on “helpful” votes to highlight quality reviews, but these votes only appear over time and are influenced by factors like playtime, purchase status, and visibility. This delay makes it hard to know which new reviews deserve attention right away. At the same time, duplicated or template reviews (“copypasta”) and inconsistent reviewer behavior can distort helpfulness signals and bias models.

Objectives

Our goal is to build a real time helpfulness predictor that gives us a prediction as soon as a review is posted, while also identifying reviewer types and filtering out spam or near-duplicate content—all within a scalable, practical big-data workflow.

1. **Helpfulness Prediction (counts):** Predict helpful upvotes (or the Steam helpfulness score) within a fixed future window (e.g., 14 days) from features available at post time. Evaluate **RMSE/MAE/Poisson deviance**, **Spearman ρ**, and **calibration** (binned observed vs. predicted).

2. **Reviewer Archetypes (unsupervised):** Cluster authors by behavior (games owned, reviews count, playtime, purchase ratio, average sentiment, etc.) using **MiniBatch K-Means + UMAP**; develop persona cards and compare their typical helpfulness/positivity.

3. **Spam and Template Detection:** Detect near-duplicate/copypasta reviews via **MinHash LSH (char 5-grams)** plus style-based **Isolation Forest**. Produce global duplicate rate and show the impact on prediction when we drop or down-weight flagged reviews.

4. **Systems Pragmatism:** Store intermediate tables in **Apache Parquet**; keep the pipeline reproducible, modular, and small—consistent with the sample proposals’ emphasis on practical pipelines and clear evaluation.

Data Source

Dataset: *100 Million+ Steam Reviews* (Kaggle)
Link: https://www.kaggle.com/datasets/kieranpoc/steam-reviews/data
Full scale: ~113,883,717 reviews (42.49GB).

Course-sized subset: we will take a **time-based slice** (e.g., most recent **2–5 million** reviews) to keep compute/storage within course limits while preserving real-world scale. We will also take a **1–5% size sample** of the subset for rapid iteration before scaling.

Dataset Information

- Author (steamid, games_owned, reviews_count, playtime_all, playtime_recent, playtime_at_review, last_played)
- Language
- Timestamp of created and updated
- Review (positive/negative)
- Helpful votes, funny votes, comment count, purchased flag, free-key flag, early access flag, developer response with date (if any)

(If the raw export is partitioned by app/date, we will read partitions incrementally and persist to Parquet.)

Proposed Technologies & Programming Language

Programming Language: Python (primary)

Data Processing / Storage

- **PySpark / Apache Spark** for scalable ETL, joins, and aggregations; **Parquet** (Snappy) tables; optional Hive metastore for easy querying
- **Pandas / Polars** for quick EDA and sampling; **PyArrow** for efficient IO.

Models & libraries

- **Helpfulness prediction:**
    - **Negative-Binomial regression** (statsmodels GLM with log link) and/or **XGBoost** (Poisson or log1p-regression objective).
    - **Calibration** utilities (isotonic/temperature; reliability diagrams).
- **Reviewer archetypes: MiniBatchKMeans**, **UMAP** (visualization), silhouette / Calinski-Harabasz for stability.
- **Spam radar: datasketch** (MinHash LSH for char 5-grams) or FAISS cosine for candidate pairs; **scikit-learn IsolationForest** for style-anomaly scoring.

Visualization / reporting

- **Matplotlib/Altair/Plotly** charts (metrics, calibration, persona overlays), **Streamlit** (optional single-page dashboard).

(Optional) Streaming / ops

- If time permits: **Kafka** to simulate live reviews → a tiny Spark Structured Streaming job to score and emit alerts—kept minimal and aligned with the systems tone in the samples.

Notes on Scope & Rigor

- **Time-based train/validation/test** split to avoid leakage; define a **future window** (e.g., +14 days) for helpfulness labels.
- **Ablations**: base features vs. +personas vs. +spam flags; **before/after** filtering vs. **down-weight** flagged rows.
- **Governance**: we will report how spam hygiene changes accuracy and calibration, not just raw scores—mirroring the “evaluation & operations” emphasis in the samples.