HelpfulLens: Modeling Yelp Review Helpfulness

A Minimal, Reproducible Pipeline from Ingest to Models

Team HelpfulLens
Data Science Project

December 14, 2025

Abstract

We built HelpfulLens to study and predict the "helpfulness" of Yelp reviews (useful votes) with a lightweight, reproducible pipeline. The project ingests the public Yelp JSON dumps, cleans and aligns review/user/business tables, assembles train/evaluation datasets, engineers text and sentiment features, and trains a small set of transparent models (constant baseline, linear regression on log1p counts, Poisson, gradient boosting, and an optional hurdle classifier+regressor). Outputs include feature caches, metrics, plots, and model artifacts, all driven by a single YAML config and bash entrypoint. This report describes the data, preprocessing, feature engineering, modeling choices, and how to reproduce the results. Metrics populate after running the training script; figures shown come from the EDA sweep on $\sim$7M reviews.

Contents

1 Introduction

Yelp reviews carry "useful" votes that reflect community judgment of review helpfulness. Predicting these counts can surface better content, inform ranking, and guide review-writing feedback. Our goal is a minimal, config-driven pipeline—no heavy MLOps—that converts raw Yelp dumps into modeling-ready features and trains baseline-to-strong models with transparent evaluation and artifacts.

1.1 Team Overview

We split work into ingest/clean, feature engineering, and modeling/reporting. The data engineering track handled parquet caching and schema cleaning; feature engineering covered text stats, sentiment, and categorical encodings; modeling implemented baselines and a hurdle variant plus plotting/reporting. Documentation and reproducibility (config, scripts, README) were maintained jointly.

1.2 Report Structure

Section **??** details data, preprocessing, and modeling choices. Section **??** summarises EDA and describes how to interpret the model outputs. Sections **??** and **??** discuss implications, limitations, and future work.

2 Data and Methods

2.1 Data Sources

We use the Yelp Open Dataset (JSON format). A full pass in our EDA processed 6,990,124 reviews, 150,346 businesses, and 1,987,925 users. Raw files live under `data/raw/`; ingestion caches parquet shards under `data/raw/parquet/`.

2.2 Data Preparation

- **Ingestion** (`src/data/ingest_raw.py`): auto-discovers JSON/JSON.GZ shards, supports chunked reads and limits, writes dataset-specific parquet.
- **Cleaning** (`src/data/clean_yelp.py`): enforces schemas, drops NAs, coerces numerics, derives date parts, parses elite flags, and saves `reviews_clean.parquet`, `users_clean.parquet`, `business_clean.parquet`.
- **Dataset assembly** (`src/data/make_dataset.py`): joins review/user/business, engineers targets (`target_useful_votes`, smoothed rate, total votes), and splits into train/eval parquet bundles.
- **Optional filters**: The modeling config allows restricting to reviews with at least `min_total_votes` (useful+funny+cool) to focus on well-exposed content.

2.3 Feature Engineering

- **Numeric base**: vote totals, smoothed useful rate, text length (chars/words), punctuation counts, caps ratio, temporal features (week-of-year, weekend flag), business and user stats (stars, review counts, fans).

- **Categorical encodings**: top cities, states, and business categories one-hot encoded with caps set in config.
- **Sentiment/text**: optional merge of VADER or transformer sentiment and text stats from `scripts/feature_engineering_sentiment.py`; optional TF-IDF fitted on train only with configurable vocab size and n-grams.
- **Schema**: `src/features/build_features.py` outputs aligned `X_train/X_eval`, `y_train/y_eval` plus `feature_schema.json` (feature order, dtypes, one-hot maps, imputations).

Category hygiene: before one-hotting, business categories are split, de-duplicated per business, and generic parents (Restaurants, Food, Hotels & Travel, Nightlife, etc.) are dropped so the resulting indicators focus on meaningful subcategories (e.g., Mexican, Pizza, Nail Salons).

2.4 Data Selection and Leakage Controls

To focus on exposed content and avoid label leakage:

- We optionally filter to reviews with at least `min_total_votes` (useful+funny+cool) via the modeling config.
- Target-bearing columns (`useful`, `total_votes`, smoothed targets) are dropped from features; schema checks enforce no duplicates and aligned train/eval columns.
- TF-IDF is fit on train only and applied to eval; sentiment merges are optional and key-validated.

2.5 Modeling Approach

Implemented in `src/models/train_and_evaluate.py`:

- **Baseline mean**: constant prediction of global mean useful.
- **Linear (log space)**: linear regression on $\log(1 + \text{useful})$ with a small numeric set.
- **Poisson**: `PoissonRegressor` on count targets.
- **Tree**: `HistGradientBoostingRegressor` on $\log(1 + \text{useful})$.
- **Hurdle (optional)**: logistic classifier for useful> 0 + regressor for positive counts; combines probability and conditional expectation.

2.6 Evaluation Metrics

Regression: MAE and RMSE on $\log(1 + \text{useful})$, and Spearman correlation between predictions and true useful. Classification (hurdle): ROC-AUC and PR-AUC for useful> 0. Plots: true vs. predicted scatter (log scale), residuals vs. length, histograms of $\log(1 + \text{useful})$, and if applicable PR + calibration curves.

2.7 Reproducible Workflow

The main entrypoint `./scripts/run_data_pipeline.sh` chains ingest → clean → dataset assembly → feature build → modeling. Toggles in `src/config/config.yaml` control sentiment, TF-IDF, category caps, min vote filter, and model list. Artifacts (metrics, plots, model.joblib, preds) are written to `artifacts/<run_id>/`.

3 Results

3.1 EDA Highlights

We examine how *helpful* votes relate to popularity signals (cool/funny, author reputation), content signals (length/category), and context (city/time). Figures come from the 7M-review sweep in `reports/eda/figures`.

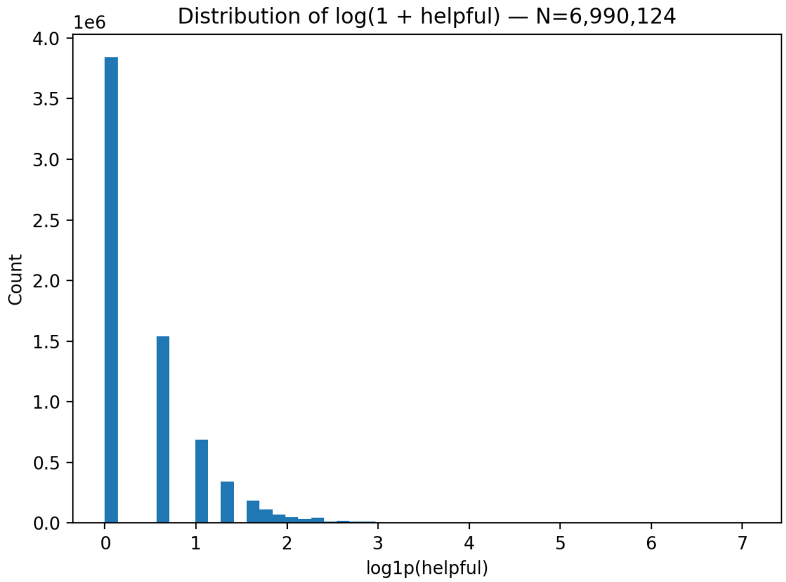

Figure 1: Distribution of $\log(1 + \text{helpful})$.

Zero inflation and tails. Most reviews have zero helpful votes; a small tail reaches 1,000+. This heavy skew motivates log transforms and the hurdle formulation.

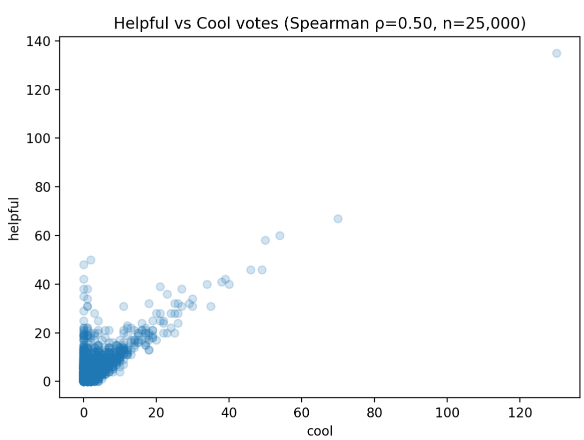

Figure 2: Helpful vs. cool votes (sample).

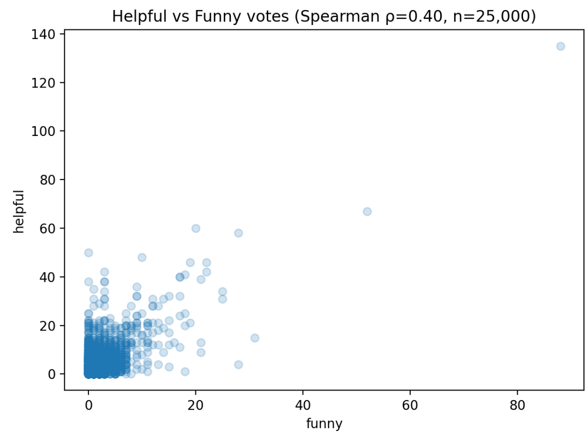

Figure 3: Helpful vs. funny votes (sample).

Popularity vs. helpfulness. Helpful correlates with cool/funny (shared popularity). These are treated as exposure controls rather than targets.

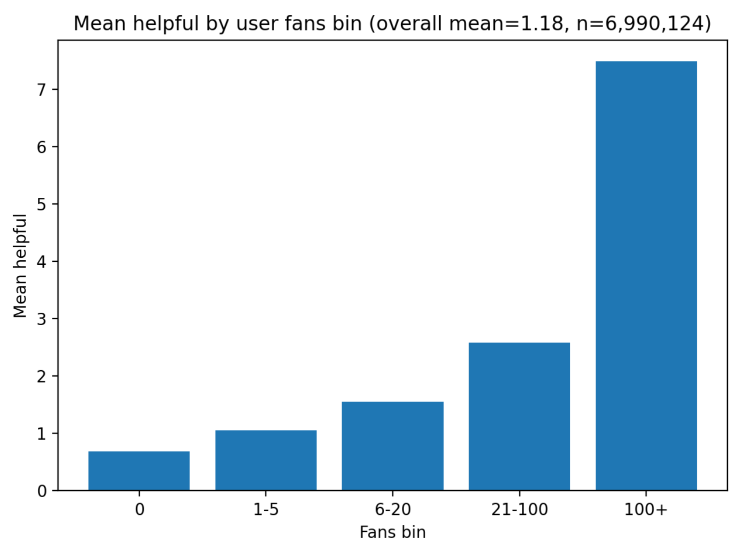

Figure 4: Mean helpful by user fan bins.

Author reputation. High-fan authors get substantially more helpful votes, highlighting visibility bias; fan count is kept as a confounder feature.

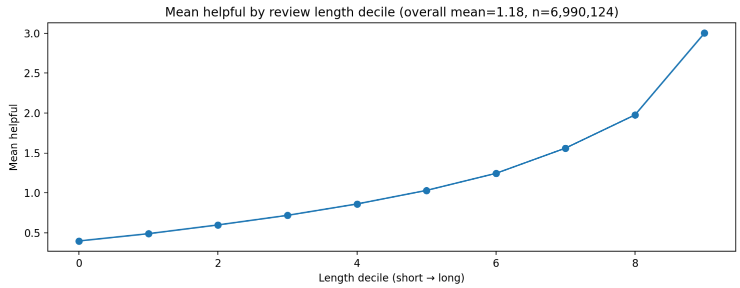

Figure 5: Mean helpful by review length decile.

Content signals. Longer reviews earn more helpful votes (monotonic, accelerating in top deciles), so length features are useful but not sufficient on their own.

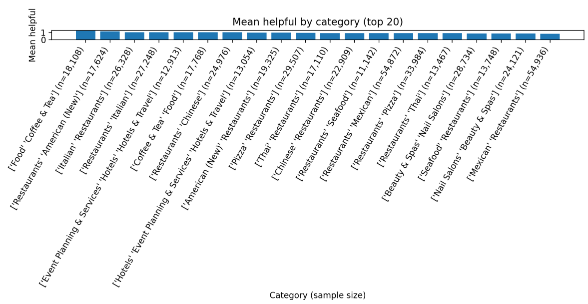

Figure 6: Mean helpful by category (top 20).

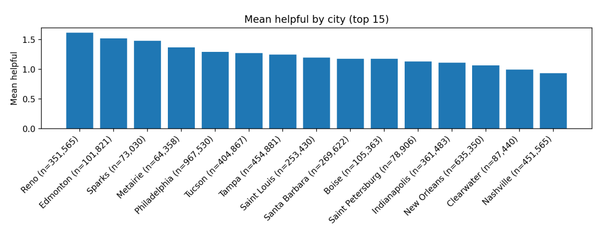

Figure 7: Mean helpful by city (top 15).

Place and category. Certain cuisines and locales attract more helpful votes, suggesting audience density effects.

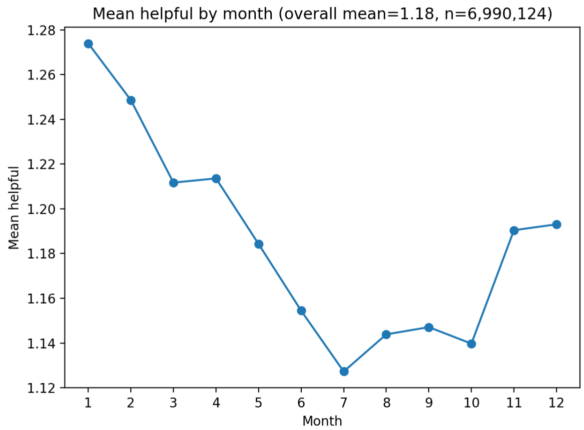

Figure 8: Mean helpful by month.

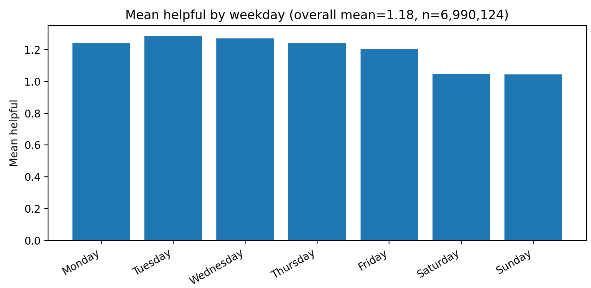

Figure 9: Mean helpful by weekday.

Temporal effects. Seasonality and weekday effects are mild; we keep them as controls.

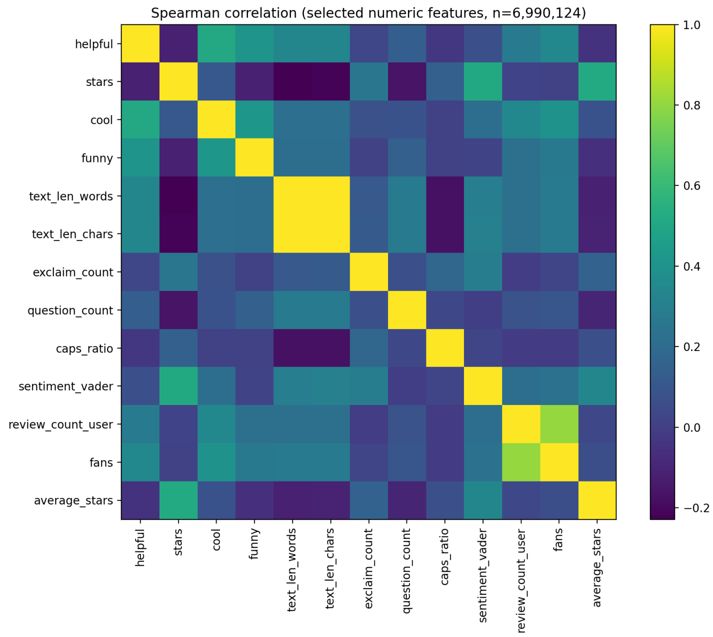

Figure 10: Spearman correlation among numeric features.

Correlations. Length measures correlate moderately with helpful; star ratings are weak; sentiment (when included) aligns more with rating than helpfulness. No single feature dominates, reinforcing multi-signal models.

3.2 Model Outputs

Run `python -m src.models.train_and_evaluate --config src/config/config.yaml` to populate metrics and plots. A typical artifact bundle includes:

- `metrics.json`: MAE/RMSE (log1p) and Spearman for each model; ROC/PR for hurdle.
- `preds_eval.parquet`: true vs. predicted useful counts (and hurdle probabilities).
- Figures: true vs. predicted scatter, residuals vs. length, histogram overlay; PR & calibration if hurdle enabled.

3.2.1 Current best practices (post-leakage fix)

- Drop `useful` and `total_votes` from features to avoid leakage.
- Consider `min_total_votes` $\geq$ 1–3 to train on reviews with real exposure.
- If the hurdle classifier warns about convergence, raise `max_iter` in `train_and_evaluate.py` or standardize numeric features.

3.2.2 Latest run (leakage fixed)

Run ID: `20251214_010535_b35a`. Features include city/state/category one-hots and basic numeric stats (no TF-IDF or sentiment). Target-bearing columns have been dropped.

Table 1: Model comparison (run `20251214_010535_b35a`). Lower RMSE/MAE is better; higher Spearman/ROC/PR is better.

Model	$\mathrm{MAE}_{\log 1p}$	$\mathrm{RMSE}_{\log 1p}$	Spearman	ROC-AUC	PR-AUC
Baseline mean	0.611	0.710	n/a	n/a	n/a
Linear (log)	0.536	0.645	0.118	n/a	n/a
Poisson	0.611	0.710	n/a	n/a	n/a
HGB	0.339	0.446	0.637	n/a	n/a
Hurdle	0.439	0.520	0.545	0.783	0.786

Warnings observed: (i) Deprecation on `datetime.utcnow()` (harmless); (ii) `RuntimeWarning` from covariance/standardization on sparse data; (iii) logistic hurdle classifier hit iteration limit (increase `max_iter` or standardize); (iv) benign multiprocessing resource-tracker cleanup noise. Next improvements: add text length/sentiment/TF-IDF features and retune models.

4 Discussion

Length, sentiment, and user reputation are informative, but the heavy zero inflation suggests hurdle or zero-inflated approaches are appropriate. TF-IDF and transformer sentiment can boost performance but increase runtime; the config keeps them optional. Filtering to reviews with sufficient total votes focuses learning on content with real exposure and mitigates noise from never-seen reviews.

4.1 Practical Implications

The pipeline can rank recent reviews by predicted helpfulness, surface candidates for featuring, or give authors feedback on likely impact. Lightweight dependencies and parquet caching make it easy to refresh when new dumps arrive.

4.2 Future Work

Add calibrated uncertainty estimates, experiment with gradient-boosted trees on raw counts (Tweedie), and integrate a simple serving notebook for batch scoring. More granular temporal features and business-level priors (e.g., hierarchical smoothing) could reduce variance for sparse entities.

5 Conclusion

HelpfulLens delivers an end-to-end, minimal pipeline for Yelp helpfulness modeling: raw data ingestion, cleaning, dataset assembly, feature building, and baseline-to-strong models with clear artifacts. The config-first design keeps runs reproducible and easy to extend with richer features or alternative models.

Acknowledgements

Thanks to the Yelp Open Dataset for providing the data, and to the team members who split data engineering, feature design, and modeling/reporting duties. Open-source libraries (pandas, numpy, scikit-learn, matplotlib) underpin the entire pipeline.

References

[1] Yelp Open Dataset. `https://www.yelp.com/dataset`.

[2] Pedregosa, F. et al. *Scikit-learn: Machine Learning in Python*. JMLR 12, 2825–2830, 2011.