Book Popularity Prediction for an Online Bookstore

Author: ISTIAK ALAM
Affiliation: DataCamp Real World Project

Executive Summary
This project builds a text-and-metadata based classification pipeline to predict whether a book will be popular or not, meeting a target accuracy of at least 70%. Using the bookstore's historical catalog (price, authors, categories, and rich review text fields), I engineered features from both structured attributes and unstructured reviews and trained a supervised learning model. The final approach combines bag-of-words signals from review/summary text with numeric features such as price and review helpfulness to accurately flag popular titles, enabling better inventory planning, marketing focus, and homepage curation.

Objectives

- Prepare an analysis-ready dataset from books.csv.
- Engineer predictive features from text (reviews) and tabular attributes (price, helpfulness, categories/authors).
- Train and evaluate a classification model to predict popularity.
- Produce actionable insights for merchandising and stock control.

Data
Source: data/books.csv provided by the bookstore.
Key fields:

- price (float)

- popularity (target; Popular/Unpopular)
- review/summary (short text)
- review/text (long text)
- review/helpfulness (e.g., “17/19” upvotes)
- authors (string; possibly multiple)
- categories (string; single top-level category)
- title and description (auxiliary text)

Initial Exploration

- Dataset contains product-level rows with at least one review per title.
- review/helpfulness is a ratio string requiring parsing into numeric upvotes and votes.
- Text fields (summary, review/text, description) are long and suitable for bag-of-words representation.
- Target popularity is imbalanced toward Unpopular in the preview; handling imbalance is considered in evaluation.

Methodology

1. Data Cleaning and Parsing

- Parsed review/helpfulness like “17/19” into two integers: helpful_yes=17, helpful_total=19 and derived helpful_ratio=17/19 (guarding against division by zero).
- Standardized price to numeric and clipped extreme outliers if needed.

- Normalized authors and categories (lowercased, stripped). Extracted simple indicators: has_multiple_authors, category token.

2. Feature Engineering

- Text features:
    - Combined review/summary and review/text into a single text field to maximize signal density.
    - Used CountVectorizer to produce a sparse bag-of-words representation; limited vocabulary to frequent terms to reduce noise.
- Numeric features:
    - price (scaled later if using linear models; tree models are scale-agnostic).
    - helpful_yes, helpful_total, helpful_ratio.
- Categorical encodings:
    - Target-agnostic one-hot for major categories (top K by frequency) and a “other” bucket.
    - Optional: binary has_multiple_authors.

3. Modeling

- Split: train/test with stratification on popularity to preserve class balance.
- Baseline: RandomForestClassifier (good default with mixed sparse/dense inputs when stacked with text).
- Alternative: Linear models (Logistic Regression with liblinear/saga) using only text or text+numeric features.

- Evaluation: accuracy as the primary metric (target ≥ 70%), with precision/recall on Popular class to judge marketing usefulness.

4. Training and Validation

- Trained on train split; evaluated on held-out test set.
- Checked confusion matrix to verify Popular recall (reduces missed bestsellers) and precision (controls overstocking).

Results

Model Performance

- The combined feature approach (review text + summary + price + helpfulness) yielded a robust classifier.
- Accuracy met or exceeded the 70% target on held-out data.
- Text signals (adjectives/phrases in summaries and reviews) contributed the largest lift; helpful_ratio and price provided additional separation.

Key Drivers and Signals

- Strong positive indicators in summaries (e.g., words reflecting quality, engagement, recommendation) aligned with Popular predictions.
- Higher review helpfulness ratios correlated with Popular.
- Certain categories showed different base rates (e.g., History/Religion vs. Fiction-like narratives), suggesting curation effects.

Business Impact

- Inventory planning: Pre-release or early-review titles flagged as Popular can be stocked deeper.

- Marketing: Prioritize Popular-probability items in homepage carousels and email campaigns.
- Review operations: Encourage early detailed reviews; helpful early reviews improve predictive certainty.

Limitations

- Popularity label definition: If based on sales thresholds or ratings counts, ensure consistent labeling across time.
- Review leakage risk: If reviews reflect post-launch popularity, use only information available at decision time (e.g., early reviews or summaries) for a fair prospective model.
- Imbalance: If Popular is much smaller than Unpopular, consider class_weight or focal loss; track precision/recall trade-off via decision threshold.

Recommendations and Next Steps

- Add n-gram features (bi-grams) with a capped vocabulary to capture short phrases from summaries.
- Try TF-IDF weighting in place of raw counts; compare with Logistic Regression and Linear SVM for sparse text.
- Calibrate probabilities (Platt scaling or isotonic) and choose an operating threshold that balances stock risk.
- Add simple description/title signals and entity features (author popularity prior, category priors).

- Set up periodic retraining (e.g., monthly) and drift monitoring (vocabulary shifts, category mix).
- Create a lightweight inference API to score new titles as soon as summaries and first reviews arrive.

Reproducible Workflow Outline (reflecting the notebook)

- Load
    - books = pd.read_csv("data/books.csv")
- Parse helpfulness
    - Extract helpful_yes, helpful_total from review/helpfulness; helpful_ratio = helpful_yes / max(helpful_total, 1)
- Text preparation
    - text = review_summary + " " + review_text
    - Vectorize with CountVectorizer(min_df, max_features, stop_words='english')
- Assemble features
    - Numeric: price, helpful_yes, helpful_total, helpful_ratio
    - Categorical: topK category one-hots
    - Stack with sparse text matrix
- Split and train
    - train_test_split with stratify=popularity
    - Fit RandomForest or LogisticRegression

- Evaluate accuracy and confusion matrix; tune if needed

Conclusion

Using the bookstore's reviews and basic product metadata, I built a practical, accurate popularity predictor that meets the 70% accuracy goal. The model is operationally useful for demand forecasting and merchandising. With modest enhancements (TF-IDF, calibrated thresholds, and additional author/category priors), it can become a reliable upstream decision tool for stocking and promotion.