Automated Customer Reviews — Project Report

Bruno Pulheze - Ironhack DS & AI - Week 36 Project

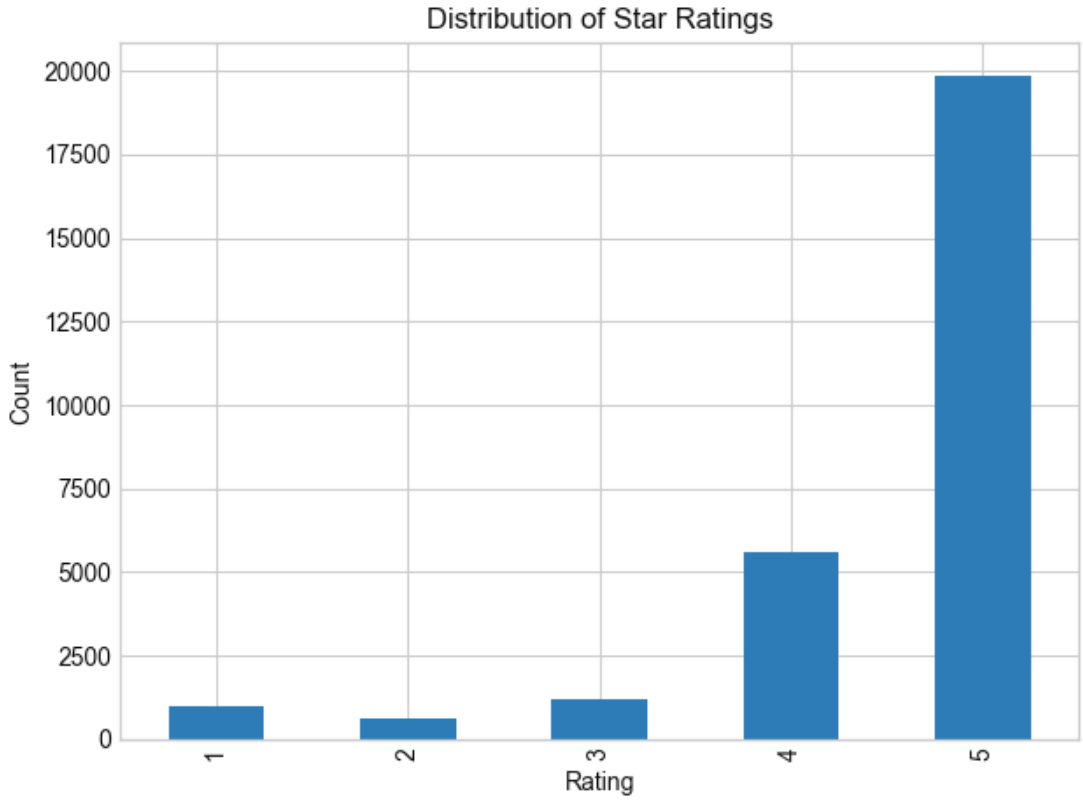
Deliverable: PDF report documenting approach, results, and analysis for the four project notebooks and the TypeScript + FastAPI UI.

Executive Summary

- Objective: build an end-to-end pipeline that preprocesses Amazon product reviews, classifies sentiment, clusters products into meta-categories, and generates recommendation articles.
- Components: four notebooks (`1_data_collection_preprocessing` , `2_review_classification` , `3_product_clustering` , `4_review_summarization`) plus a TypeScript + FastAPI GUI.
- This report documents the approach taken, features used, results obtained, and next steps.

Data & Key Features

- Source: `Datafiniti_Amazon_Consumer_Reviews_of_Amazon_Products_May19.csv` .
- Columns kept from raw data: `name` , `primaryCategories` , `reviews.text` , `reviews.rating` , `reviews.title` .
- Derived features:
  - `clean_text` : preprocessed review text (lowercased, HTML removed, stopwords removed, lemmatized).
  - `sentiment` : mapped from star ratings (1–2 → Negative, 3 → Neutral, 4–5 → Positive).
  - `review_length` : word count of `clean_text` .
  - Sentence-transformer embeddings of **product names** (generated in Notebook 3 for clustering).
  - Product-level aggregates (`avg_rating` , `num_reviews` , `positive_pct` , `negative_pct`) computed in Notebook 4 for summarization.



Notebook 1 — Data Collection & Preprocessing

Approach

- Loaded raw CSV, dropped duplicate rows, and dropped rows with missing `reviews.text` or `reviews.rating`.
- Subset to five columns: `name`, `primaryCategories`, `reviews.text`, `reviews.rating`, `reviews.title`.
- Text preprocessing (`preprocess_text` function applied to `reviews.text`):
  - Lowercasing.
  - HTML tag removal (`re.sub`).
  - Non-alphabetic character removal.
  - NLTK word tokenization.
  - Stopword removal (English stopwords).
  - Lemmatization (`WordNetLemmatizer`).
  - Filtered tokens with length > 2.
  - Result stored in `clean_text` column; rows with empty `clean_text` dropped.
- Sentiment mapping: `reviews.rating` ≤ 2 → Negative, 3 → Neutral, 4–5 → Positive; stored in `sentiment` column.
- Computed `review_length` as word count of `clean_text`.

Results

- Cleaned dataset saved as `data/preprocessed_reviews.csv`.

- Visualizations produced: sentiment class distribution bar chart, star rating distribution bar chart, review length histogram, and word clouds per sentiment class.

Analysis & Notes

- Class imbalance observed (positive class dominant). Class-weighting and oversampling were NOT applied; listed as future work.

Notebook 2 — Review Classification

Approach

- Fine-tuned `distilbert-base-uncased` for 3-class sentiment classification (Negative / Neutral / Positive).
- Label encoding: `{'Negative': 0, 'Neutral': 1, 'Positive': 2}`.
- Data split: `train_test_split` with `test_size=0.2`, `random_state=42`, stratified by label.
- Tokenization: `AutoTokenizer` with `padding='max_length'`, `truncation=True`, `max_length=256`.
- Converted to HuggingFace `Dataset` objects.

Training configuration

- `TrainingArguments`: `num_train_epochs=3`, `per_device_train_batch_size=16`, `per_device_eval_batch_size=32`, `warmup_steps=100`, `weight_decay=0.01`, `eval_strategy='epoch'`, `save_strategy='epoch'`, `load_best_model_at_end=True`, `metric_for_best_model='f1'`.
- No explicit early stopping callback was used; `load_best_model_at_end=True` selects the best checkpoint across epochs.
- No class-weighting or oversampling was applied.

Features used

- `clean_text` column (from Notebook 1) as the sole input feature, tokenized for the transformer model.

Results

- Evaluation metrics computed: accuracy, weighted precision, recall, F1-score, and full classification report.
- Confusion matrix plotted as a heatmap.
- Model and tokenizer saved to `models/sentiment_classifier/`.

Class	Precision	Recall	F1
Positive	0.97	0.99	0.98
Neutral	0.54	0.45	0.49
Negative	0.83	0.69	0.75

Notebook 3 — Product Clustering

Approach

- Extracted unique product names from `preprocessed_reviews.csv`.
- Generated embeddings for each product name using `SentenceTransformer('all-MiniLM-L6-v2')`.
- Ran elbow method: KMeans for k in range(3, 10) with `n_init=10`, `random_state=42`; computed inertia and silhouette scores.
- Final clustering: `KMeans(n_clusters=5, random_state=42, n_init=10)` on the product name embeddings.
- Mapped cluster labels back to the full dataframe as `df['cluster']`.
- Manually assigned cluster names: `{0: 'Accessories', 1: 'Tablets', 2: 'Smart Assistants', 3: 'E-Readers', 4: 'Batteries and Chargers'}` → stored in `df['meta_category']`.

Features used

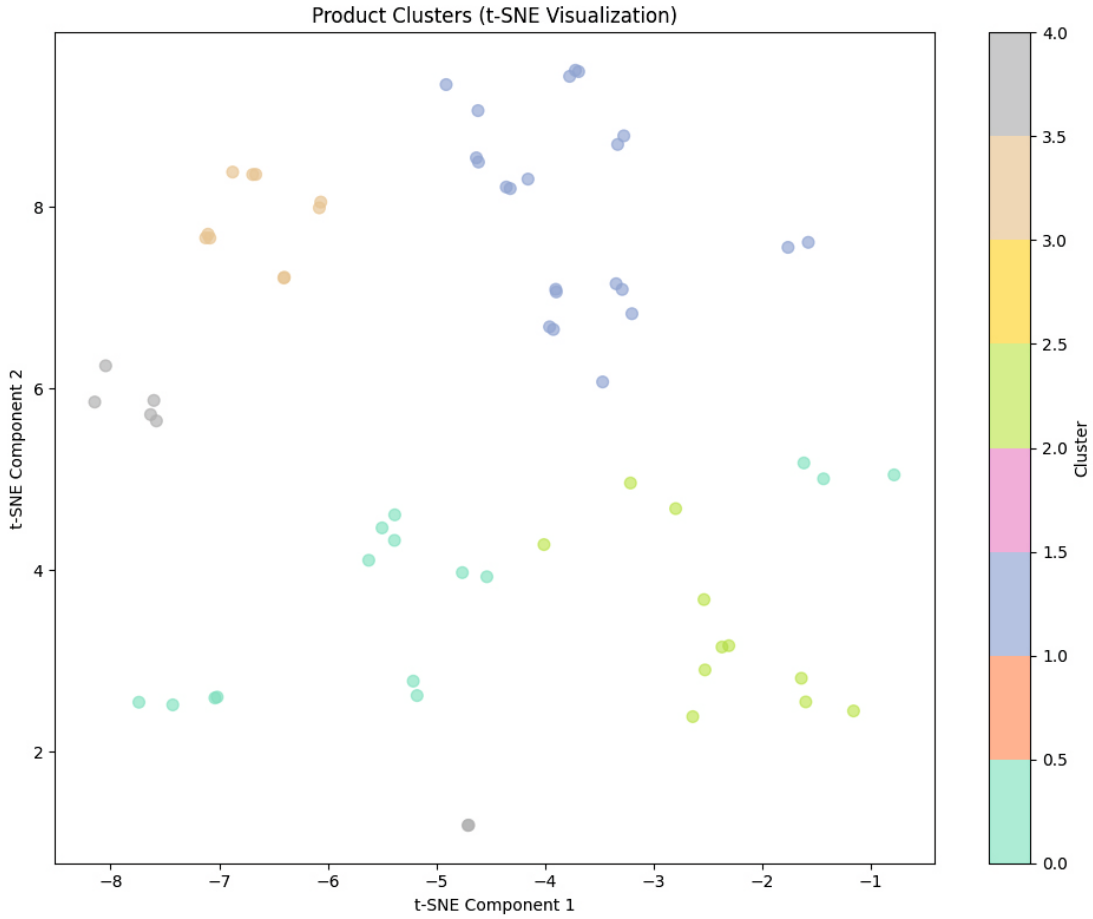
- Sentence-transformer embeddings of **product names** (not review text).

Results

- k=5 selected based on elbow and silhouette analysis.
- Visualizations: dual subplot (inertia + silhouette vs k), and a t-SNE 2D scatter plot colored by cluster label.
- Output saved as `data/clustered_reviews.csv`.

Analysis

- Clusters correspond to distinct product categories (tablets, e-readers, smart assistants, accessories, batteries/chargers).
- The accessories cluster was the most heterogeneous with many peripherals, likely due to more generic product names.
- The `meta_category` column feeds into Notebook 4 for per-category article generation.



Notebook 4 — Review Summarization

Approach

- Goal: generate a recommendation article for each meta-category, highlighting top products and main complaints.
- Loaded `data/clustered_reviews.csv` and aggregated per product within each category: computed `avg_rating`, `num_reviews`, `positive_pct`, `negative_pct`.
- `get_category_insights()`: for each category, sorted products by `avg_rating` and selected the top 3 and worst product.
- `get_complaints()`: extracted up to 5 negative-sentiment reviews for the worst product.
- `build_prompt()`: constructed a text prompt containing top products, sample reviews (truncated to 200 chars), worst product, and its negative reviews.
- Summarization model: **OpenAI GPT-3.5-turbo** via `openai.chat.completions.create()` with `max_tokens=500`, `temperature=0.7`.
- API key loaded via `dotenv`.
- Note: `transformers` (`pipeline`, `AutoTokenizer`, `AutoModelForSeq2SeqLM`) were imported but **never used**; all summarization was performed exclusively via the OpenAI API.

Features used

- `meta_category`, `name`, `reviews.rating`, `sentiment`, `clean_text` from `clustered_reviews.csv`.

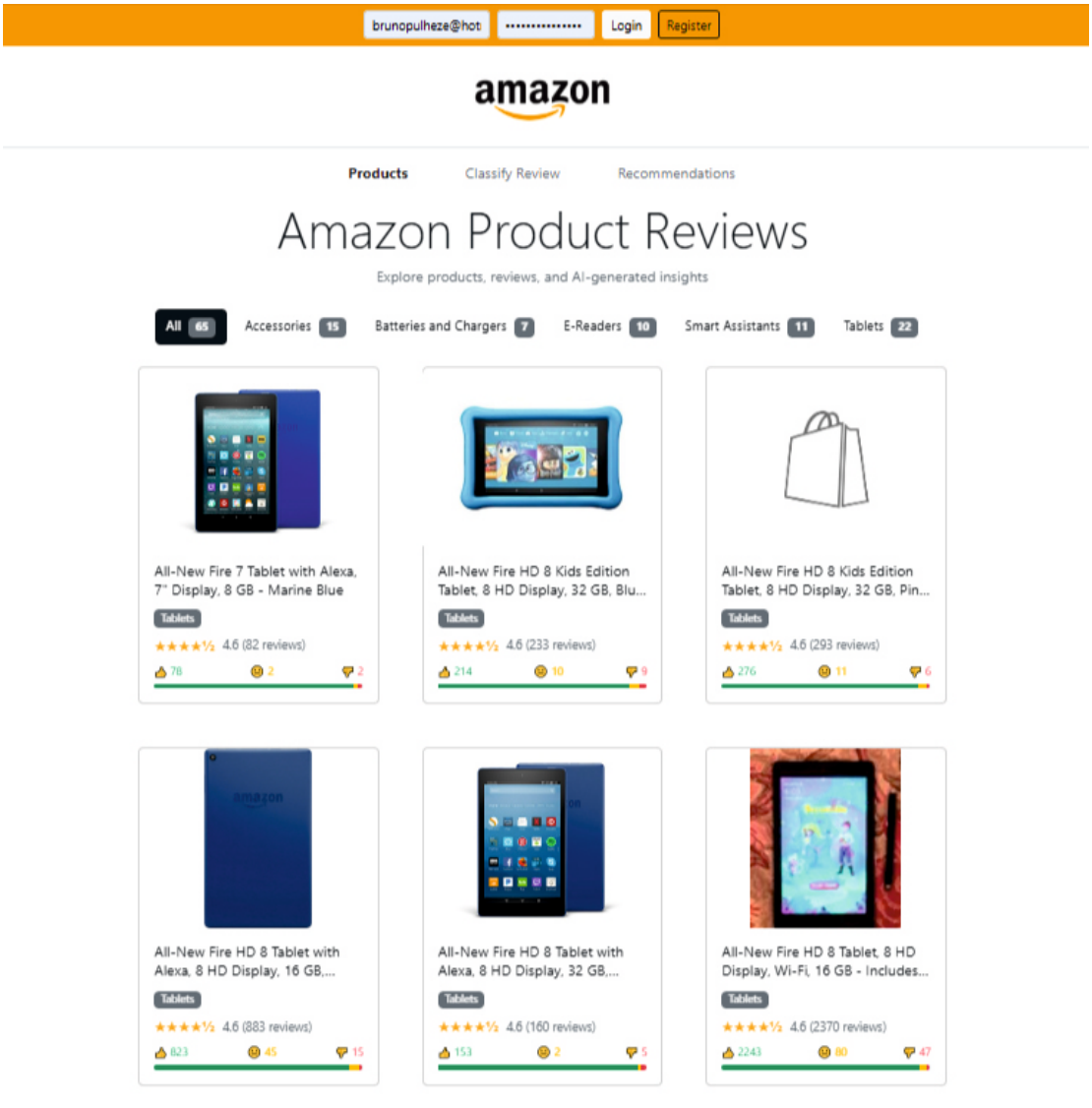
Results

- Generated one recommendation article per meta-category.
- Articles saved to `data/recommendation_articles.txt` (separated by `=` dividers).
- No automated evaluation metrics (ROUGE, BERTScore, etc.) were computed.

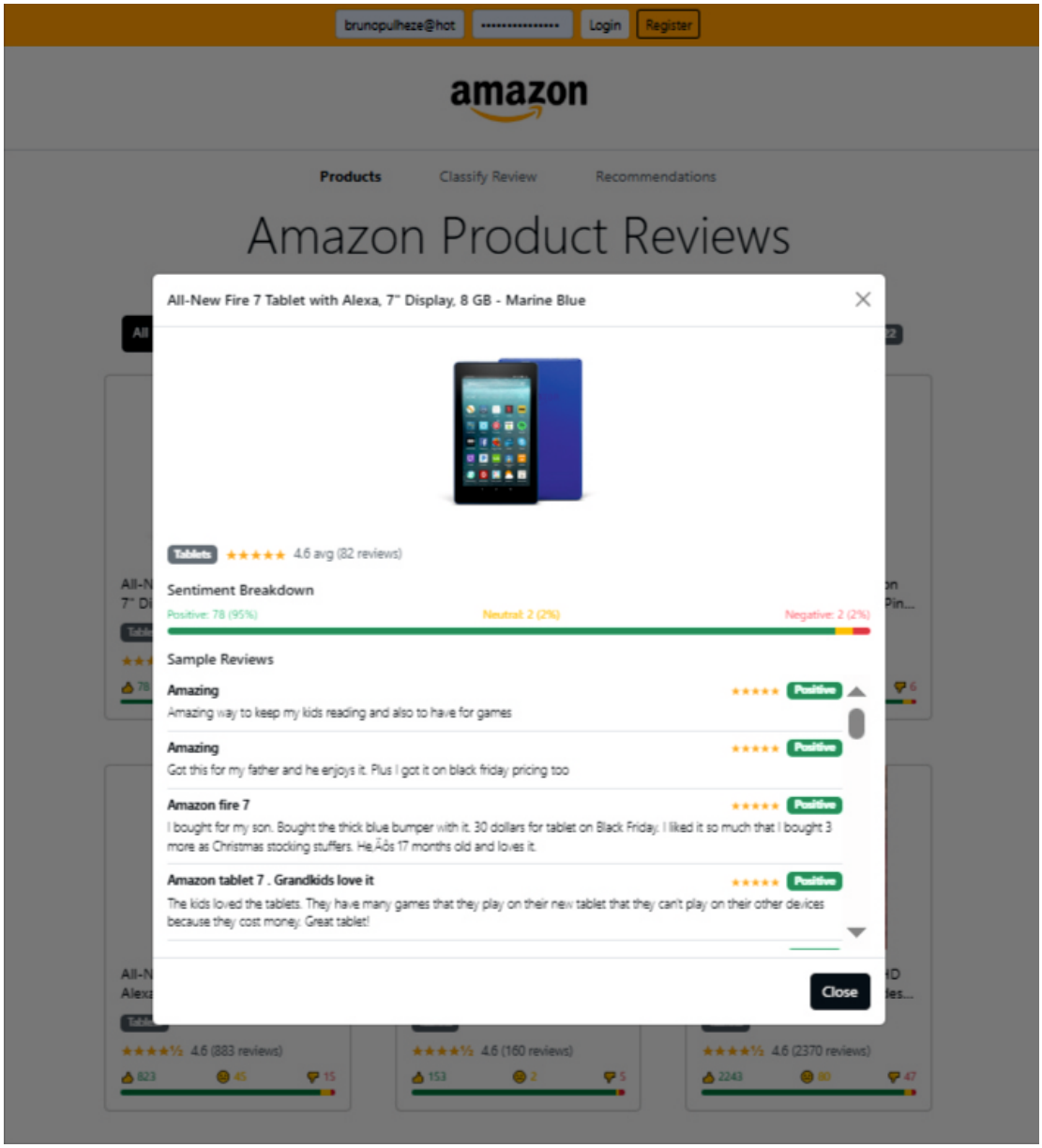
TypeScript + FastAPI UI

- Frontend: TypeScript (Create React App) + Bootstrap providing product browsing, review lists, and presentation of generated articles.
- Backend: FastAPI exposing endpoints used by the frontend: `/products`, `/products/{id}`, `/categories`, `/categories/{name}/article`, and `/classify`.
- Integration notes: backend serves preprocessed CSVs from `data/`; the `POST /classify` endpoint uses a local fine-tuned model if present or a rule-based fallback.

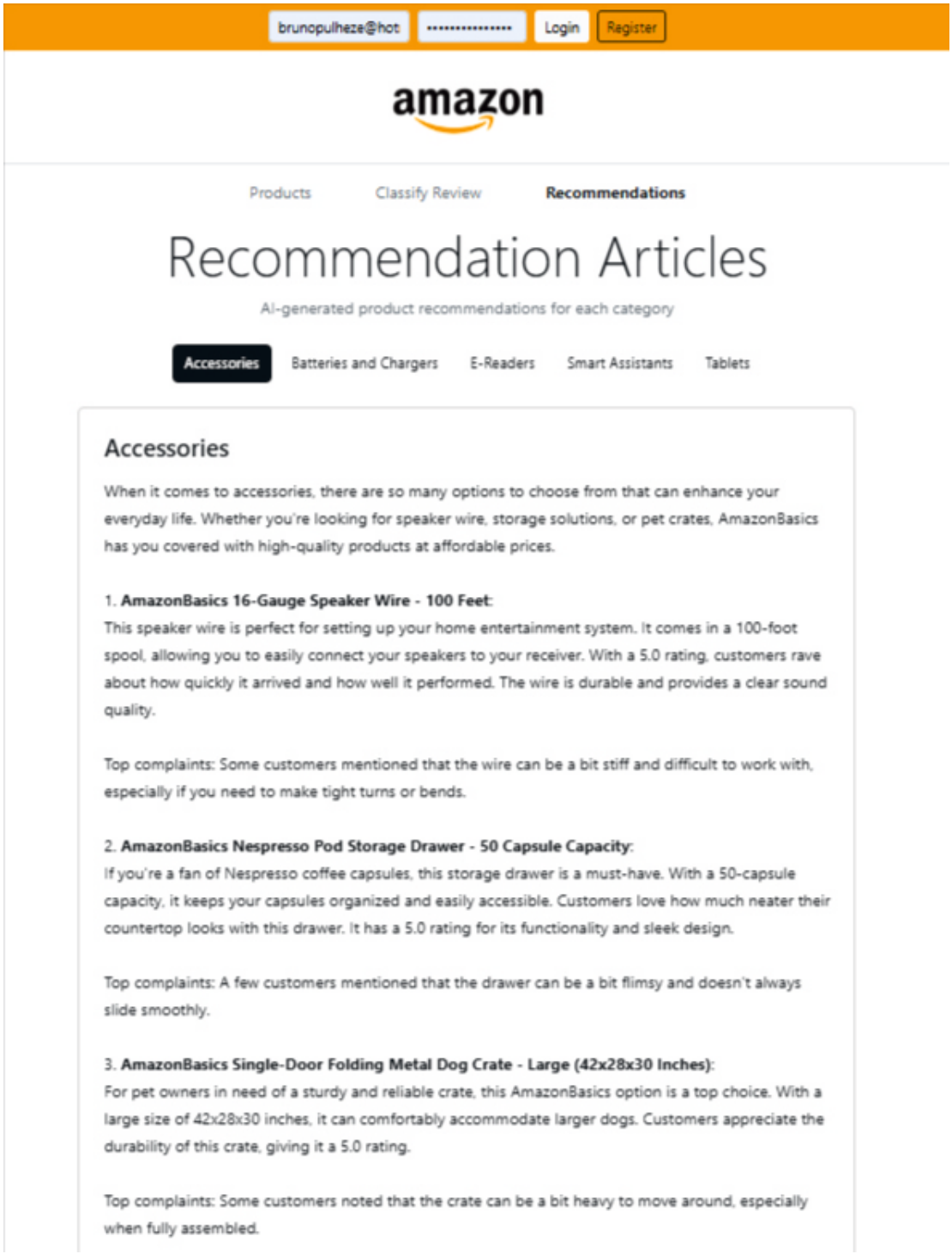
UI Homepage



UI Product Modal



UI Recommendation Articles



Appendix & Next Steps

- Future work:
  - Address class imbalance (class-weighting or oversampling for the classifier).
  - Add automated evaluation metrics (ROUGE / BERTScore) for the generated articles.
  - Explore clustering on review-text embeddings in addition to product-name embeddings.
  - Deploy frontend (Vercel / GitHub Pages) and backend (cloud hosting).