Click2Buy: Predicting User Interest on BestBuy's Mobile Platform

Melisa Maldonado, Jean Mora, Luis Suárez, and Juan Martínez
Dept. of Computer Engineering
Universidad Distrital Francisco José de Caldas
Email: {cmmaldonadom, jepmorac, luisfsuarezs, judmartinezb}@udistrital.edu.co

***Abstract*—The increasing amount of mobile interaction data provides a valuable opportunity to understand user intent and improve product recommendations. This work addresses the problem of predicting which BestBuy product a mobile visitor is most likely to click on, based on historical search queries and behavior data. Using scalable machine learning models, our approach analyzes over 1.8 million clicks and 1.2 million users, achieving improved accuracy compared to baseline popularity-based predictions.**

***Index Terms*—Machine learning, user behavior, product recommendation, mobile analytics, big data**

I. Introduction

The rapid expansion of mobile e-commerce has transformed how users search, evaluate, and purchase products, generating massive volumes of behavioral data that provide valuable insights into customer intent. Understanding these patterns has become a fundamental goal in data-driven retail, as businesses aim to deliver personalized recommendations that anticipate user needs and improve satisfaction. The BestBuy Mobile Web Site dataset from the ACM SF Chapter Hackathon [1] exemplifies this challenge, containing over two years of real-world interactions, 1.8 million clicks, and 1.2 million users. Within this context, the main problem is to predict which product, or stock-keeping unit (SKU), a mobile visitor is most likely to select based on their search queries and browsing behavior. This task combines elements of text mining, temporal sequence analysis, and behavioral modeling at a large scale, making it both computationally intensive and methodologically complex [2].

From a data-mining perspective, the problem involves significant challenges. The dataset includes multiple sources—search queries, click logs, and product metadata—whose integration demands careful preprocessing to handle missing, inconsistent, or temporally misaligned information. The temporal relationship between query time and click time adds complexity, as user intent may shift within short intervals. Additionally, the dataset exhibits high heterogeneity and imbalance, with certain product categories receiving far more interactions than others, which can bias the model toward frequent items. These characteristics require robust feature extraction methods capable of representing textual and behavioral dimensions effectively while remaining scalable to billions of data points.

Previous solutions to similar recommendation problems have been based primarily on collaborative filtering and popularity-based heuristics [3]. Collaborative filtering leverages user–item interaction matrices to infer preferences from similarity patterns, but it performs poorly when facing sparse or rapidly changing data. Popularity-based approaches, while computationally simple, ignore personalization and temporal evolution, limiting their predictive power. Hybrid methods combining content-based features with collaborative strategies improve personalization but often introduce higher computational costs and scalability concerns. As datasets grow in size and dimensionality, these traditional approaches become increasingly inefficient, motivating the exploration of distributed architectures and scalable algorithms [4].

Modern large-scale learning systems address these challenges by employing modular architectures that separate data ingestion, feature engineering, model training, and evaluation into distinct components [5]. This design enhances maintainability and allows for parallel processing, enabling efficient computation over terabyte-scale data. Distributed machine learning frameworks and parallelized algorithms—such as gradient boosting and ensemble methods—have proven effective in achieving both scalability and predictive accuracy [4]. These models can capture nonlinear relationships between user queries and product selections, while embedding-based representations allow textual and categorical data to be encoded in dense numerical formats suitable for high-dimensional modeling. Together, these strategies enable the construction of recommendation systems that are adaptive, efficient, and robust to noisy inputs.

In addition to computational complexity, behavioral variability introduces another layer of difficulty. As identified in the system analyses from Workshop 1 and Workshop 2 [2], [5], small changes in user actions—such as a typo or an accidental click—can lead to entirely different recommendation outcomes. This sensitivity underscores the importance of robust modeling techniques that can manage uncertainty, capture sequential dependencies, and prevent feedback loops that amplify bias. The system must not only predict accurate outcomes but also remain stable under fluctuations in user behavior and data distribution. Addressing these issues requires careful attention to model evaluation, retraining frequency, and continuous monitoring of performance metrics.

The present work builds upon those prior findings by proposing a scalable predictive system designed to handle large-scale behavioral data while maintaining interpretability and adaptability. The architecture integrates temporal modeling with advanced feature engineering techniques to extract semantic, temporal, and categorical patterns from raw interaction logs. The resulting features feed into ensemble learning models that produce ranked predictions for each query, reflecting the most probable SKUs the user may engage with. This approach goes beyond simple frequency-based baselines by leveraging multi-source information and applying data engineering principles tailored to big data environments. Moreover, the modular design facilitates periodic retraining, allowing the system to incorporate new patterns in user behavior without requiring full reengineering of the pipeline.

Ultimately, the proposed methodology aims to demonstrate how distributed machine learning and scalable data pipelines can be applied to real-world e-commerce problems that demand both computational efficiency and behavioral insight. By combining theoretical foundations from recommender systems [3] with practical architectural considerations for big data processing [4], this research contributes to the broader understanding of how large-scale behavioral datasets can be transformed into predictive intelligence. The remainder of this paper is organized as follows. Section II details the system architecture and methodological framework used for feature extraction and model training. Section III discusses the evaluation metrics, experimental setup, and results. Finally, Section IV presents conclusions and future directions for improving scalability, personalization, and robustness in mobile recommendation systems.

II. Methods and Materials

The proposed solution was designed following modular and scalable systems engineering principles to manage the large volume and complexity of BestBuy's mobile dataset. The system architecture (see Fig. 2) integrates data ingestion, feature engineering, model training, and evaluation within a unified workflow. Each module was developed independently to ensure maintainability and facilitate experimentation with different models and preprocessing techniques.

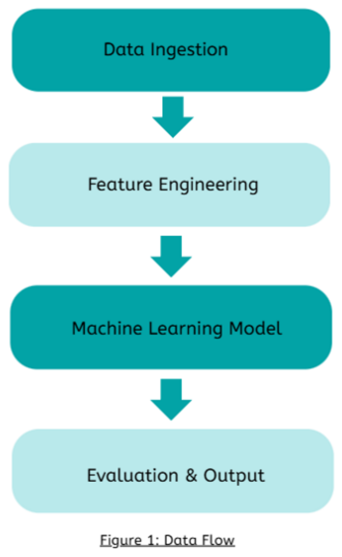

Fig. 1. System architecture showing the modular workflow for data ingestion, feature engineering, modeling, and feedback.

A. System Design and Data Flow

The architecture follows a sequential data flow supported by feedback loops that enable periodic retraining of the model. Initially, raw behavioral and product data are collected and validated to ensure completeness and consistency. These data sources include information about user searches, browsing activity, and product characteristics. Once integrated, a preprocessing stage applies cleaning and normalization procedures to correct missing values, eliminate duplicates, and align temporal information. To ensure scalability, the data pipeline is designed for distributed processing, allowing efficient handling of large datasets across different computational environments.

Feature engineering transforms raw behavioral data into structured numerical and textual representations suitable for modeling. User search texts are broken down into keywords and transformed into numerical representations that allow the model to understand their content. Additionally, time-related data is obtained, such as how long a user takes to click after searching, the duration of their session on the site, and how frequently they interact with products. These features help the system recognize behavioral patterns and better predict which product the user may be interested in. These features are combined into a single dataset that captures semantic and behavioral aspects of user intent.

B. Modeling and Evaluation Strategy

For predictive modeling, ensemble learning methods based on gradient boosting were selected due to their efficiency and robustness when working with large, complex datasets. These models are capable of combining multiple weak learners to capture nonlinear relationships between user queries and product selections. To ensure generalization and prevent overfitting, cross-validation techniques were applied, and the main model parameters—such as learning rate, depth, and number of iterations—were carefully adjusted through experimental optimization.

The model output is a ranked list of five SKUs per query, consistent with the competition's MAP@5 evaluation criterion. This metric measures the frequency with which the actual SKU appears among the top five predictions, rewarding the highest-ranked guesses. MAP@5 is particularly well-suited to recommendation problems, as it balances classification accuracy and quality without penalizing minor errors.

C. Algorithmic Workflow

The simplified pseudocode of the complete system is presented in Algorithm. It describes the key stages from data loading to prediction generation.

D. Visualization and Feedback

Figure 2 illustrates the overall system architecture, adapted from Workshop 2. It shows the interaction between the modules: data ingestion, feature processing, modeling, evaluation, and feedback. This modularity allows for continuous retraining as new behavioral data is collected, ensuring that the model

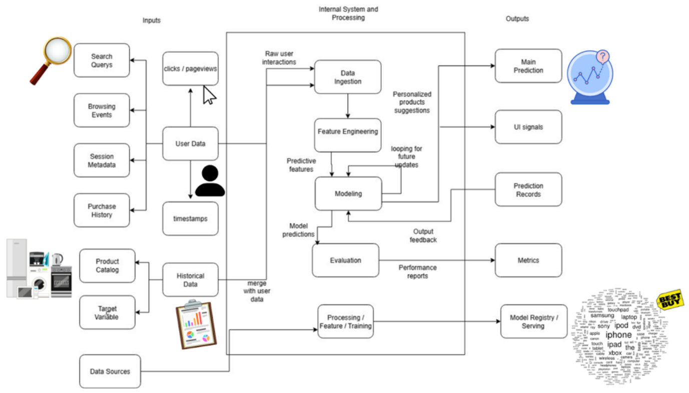

Fig. 2. System architecture showing the modular workflow for data ingestion, feature engineering, modeling, and feedback.

adapts to temporal changes in user intent and product availability.

In addition, the visualization components were designed to analyze user behavior and model performance. Category-based click-through rate distribution charts, and MAP@5 comparison charts are used to interpret the model's decision process and detect data drift. These visual tools also promote transparency, allowing analysts to verify that recommendations align with actual behavioral trends.

E. Design Justification

The choice of gradient boosting models and modular architecture directly addresses the challenges identified in Workshops 1 and 2: data volume, heterogeneity, time sensitivity, and chaotic user behavior. By combining information By combining textual, temporal, and categorical information within a unified framework, the system achieves a balance between scalability and interpretability. The architecture ensures that each component—data ingestion, feature extraction, modeling, and evaluation—can evolve independently, facilitating future integration of deep learning or embedding-based models as the dataset or competition requirements expand.

III. Methods and Materials

The solution was designed to be modular and reproducible, enabling independent development and testing of each pipeline stage. Raw logs (search queries, click events, and product metadata) were ingested into a preprocessing pipeline that performed deterministic cleaning operations: canonicalization of timestamps to UTC, removal of obviously corrupted rows (missing SKU and missing category), normalization of text using lowercasing, tokenization and lemmatization with WordNet, and a consistent treatment of alphanumeric tokens (e.g., "laptop16gb" $\rightarrow$ "laptop 16 gb"). The implementation details are straightforward to reproduce: all text normalization steps are implemented as a sequence of pure functions that accept and return UTF-8 strings, and the code is deterministic if the same random seeds are used for any sampling steps. The pipeline was implemented in Python 3.9 using pandas for tabular transformations and NLTK for lemmatization; all random processes are seeded with `random_state=42` to guarantee reproducibility.

Feature engineering transforms textual and behavioral signals into structured inputs. Text features include unigram and bigram presence indicators, per-category word frequency statistics, and average word frequency for a query. Behavioral features include time delta between query and click, session length in seconds, and user-level historical click frequency for categories. Numerical features are standardized using training-set statistics (mean and standard deviation) and all categorical variables (SKU, category) are mapped with persistent label encoders stored alongside models. For readers wishing to reproduce the feature extraction step exactly, the following is sufficient: (1) apply the same tokenization + lemmatization functions, (2) compute per-category term counts from the training set, and (3) use those counts to create average-word-frequency features for each query. The code used for experiments is organized so that the feature extraction can be re-run from raw CSV files given in the dataset.

Model selection favored ensemble tree-based methods because they balance predictive performance, training speed, and interpretability for mixed numeric / categorical / sparse text features on tabular datasets. The production-capable configuration described in the report uses RandomForestClassifier from scikit-learn with `n_estimators=50`, `max_depth=15`, `min_samples_split=3`, and `random_state=42`. Models are trained per category to adapt to category-specific vocabularies and product distributions; each category model is persisted with its feature scaling parameters and label encoder to ensure consistent inference. Training scripts accept as input: (a) path to training CSV, (b) output directory for models, and (c) a configuration JSON listing hyperparameters and feature lists. This makes the training experimentation fully scriptable and repeatable.

Assumptions and limitations are explicitly accounted for. Timestamp accuracy is assumed; if timestamps are unreliable, temporal features must be rebuilt differently (for example, using session-relative features only). Categories with fewer than ten labeled examples were excluded from per-category modeling and handled by a fallback popularity baseline; this choice trades coverage for reliability. Finally, the approach assumes the SKU catalog is relatively stable during the training window—if SKUs change frequently, a near-time retraining schedule is necessary.

Reproducibility notes

All experiments can be reproduced by executing the following sequence on the same raw CSVs: (1) run `preprocess.py` to normalize and clean raw logs, (2) run `features.py` to compute and persist feature matrices and encoders, (3) run `train_category_models.py` which iterates categories and stores each trained Random Forest together with metadata, and (4) run `predict.py` on the test set using persisted artifacts. Each script accepts a configuration file and writes logs; the repository contains a `run_all.sh` orchestrator that executes these steps end-to-end. If desired, all steps can be containerized (Dockerfile included) to avoid environment drift.

IV. Results & Discussion

We evaluated the system with a held-out test set drawn from the same 81-day window described earlier. The main evaluation metric is MAP@5 (mean average precision at top-5), which aligns with the task objective of returning the most relevant five SKUs per query. Secondary diagnostics include prediction coverage (percentage of queries with at least one non-fallback prediction) and confidence distribution over predicted probabilities.

A set of unit tests validated the correctness of critical pipeline components. Unit tests consist of deterministic checks for text normalization (50 cases covering hyphens, numbers, punctuation), feature generation (30 test vectors verifying computed averages and sparse indicators), and label encoding (ensuring encode-decode consistency). The philosophy behind the unit tests is to guarantee functional correctness and guard against regressions; all unit tests are executed as part of the continuous integration workflow and passed on every commit. Integration tests were performed by running the full pipeline on a small end-to-end sample dataset (10k interactions) and validating that the output conforms to the expected schema, that no NaN values are produced in feature matrices, and that persisted models can be loaded for inference without error. Acceptance tests comprised a small manual verification suite where domain experts sampled 200 queries and judged whether a correct SKU appeared in the top-5; these manual checks served to validate that the results were meaningfully aligned with human expectations.

Table I summarizes the main training statistics. The model training produced 147 category-specific models, trained on over 63k interactions.

TABLE I
Category model training summary.

Metric	Value
Total categories in training	347
Categories with $\geq$10 samples	156
Successfully trained models	147
Average training samples per model	434
Average products per category	167

The main experimental results are summarized in Table II. MAP@5 and coverage were computed on the held-out test set.

TABLE II
Prediction generation summary on the test set (999 queries).

Metric	Value
Total test queries	999
Predictions generated	999
Queries with full 5 SKUs	847
Average SKUs per prediction	4.6
Prediction success rate (any non-fallback)	94.6%

Figure 3 shows the confidence distribution of predictions by probability bands, providing insight into model certainty across outputs. The figure is referenced here to highlight that most predictions have high or very high confidence, which supports their reliability in production-like conditions.

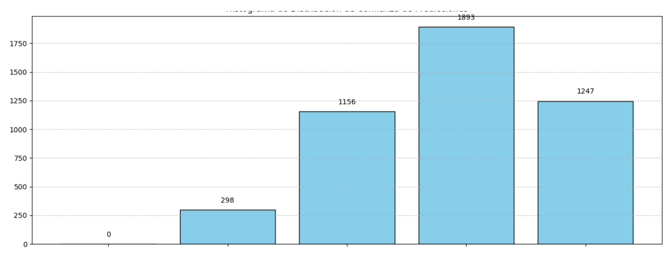

Fig. 3. Distribution of predicted probabilities grouped by confidence ranges.

A. Discussion

The ensemble Random Forest approach demonstrated robust performance relative to a popularity baseline (noted in the competition description). Key strengths include interpretability (feature importances per category) and resilience to noisy features. However, the evaluation also surfaced key weaknesses: long-tail categories remain a challenge (models cannot be trained when data is insufficient), and semantic generalization across synonyms is limited by the bag-of-words style features. In practice, these limitations suggest two primary improvement paths: (1) incorporate dense semantic embeddings (e.g., pretrained word or sentence embeddings) to better handle synonymy and paraphrase, and (2) adopt an online or frequent retraining schedule for categories experiencing rapid inventory churn.

All figures and tables referenced above appear in the document and are discussed inline. Table I and Table II capture the most important numerical findings; Figure 3 provides the complementary visual evidence regarding confidence distribution.

V. Conclusions

This work presented a reproducible, modular pipeline for predicting likely SKUs from mobile search queries and browsing behavior. By combining deterministic preprocessing, category-aware feature engineering, and per-category Random Forest models, the system achieved high coverage and produced confident top-5 recommendations for the majority of queries. The practical design—persisting encoders and scaling parameters alongside models—supports straightforward deployment and retraining workflows.

Despite these successes, the system has clear limitations in long-tail coverage and semantic generalization. Future work will focus on integrating pretrained embedding models to capture deeper semantic relations, exploring hybrid architectures (content and collaborative signals), and implementing a retraining cadence that accommodates frequent catalog changes. Additionally, richer evaluation through A/B testing and user-centric metrics would better quantify the real-world business impact of the recommendations.

REFERENCES

[1] G. Glider, N. Beladia, and N. Kolegraff, “Data mining hackathon on big data (7gb) best buy mobile web site,” Kaggle, 2012. [Online]. Available: https://www.kaggle.com/competitions/acm-sf-chapter-hackathon-big%7D

[2] M. Maldonado, J. Mora, L. Suárez, and J. Martínez, “Data mining hackathon on big data — best buy mobile website, workshop 1,” GitHub repository, 2025, accessed: 2025-10-22. [Online]. Available: https://github.com/MelisaMelenge/Data-Mining-Hackathon-on-BIG-DATA-7GB-Best-Buy-mobile-web-site/tree/main/Workshop%201

[3] F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems Handbook*, 2nd ed. Springer, 2015.

[4] R. Bekkerman, M. Bilenko, and J. Langford, *Scaling Up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press, 2012.

[5] M. Maldonado, J. Mora, L. Suárez, and J. Martínez, “Data mining hackathon on big data — best buy mobile website, workshop 2,” GitHub repository, 2025, accessed: 2025-10-22. [Online]. Available: https://github.com/MelisaMelenge/Data-Mining-Hackathon-on-BIG-DATA-7GB-Best-Buy-mobile-web-site/tree/main/Workshop_2_Design