Click2Buy: Predicting User Interest on BestBuy's Mobile Platform

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

- Melisa Maldonado Melenge 20231020110
- Jean Pierre Mora Cepeda 20231020105
- Juan Diego Martínez Beltrán 20231020131
- Luis Felipe Suárez Sánchez 20231020033

- Universidad Distrital Francisco José de Caldas -

- Computer Engineering Program -

- School of Engineering -

- Systems Analysis & Design -

- Eng. Carlos Andrés Sierra, M.Sc. -

- 2025 -

Abstract

This report presents a machine-learning–based product recommendation system designed to identify relevant items from user search queries in an e-commerce setting. The system uses category-specific Random Forest models and incorporates natural language processing techniques along with carefully engineered features such as word frequencies, temporal patterns, and category-level query behavior. Implemented with tuned Random Forest classifiers, the approach consistently generates accurate top-five product recommendations and successfully trains reliable models for categories with sufficient historical data. The report outlines the full methodology from data preprocessing to model deployment, summarizes the system's performance, and discusses its limitations as well as potential areas for enhancement in future iterations.

Introduction

Product recommendation systems are now a core part of modern e-commerce platforms, helping create personalized shopping experiences and making it easier for users to find what they're looking for. As shoppers increasingly rely on search queries to navigate large product catalogs, understanding what they intend to find has become a major challenge. Traditional approaches like collaborative filtering and content-based filtering offer useful insights, but they often struggle when interactions are sparse or when queries contain complex language patterns.

To overcome these limitations, this project introduces a machine learning–based recommendation system designed to predict the most relevant products from a user's search input. The system combines natural language processing with structured feature engineering and uses category-specific Random Forest models to capture the unique behavior of different product domains. This setup allows the system to learn lexical, semantic, and temporal patterns from historical search interactions, enabling it to deliver accurate top-five product recommendations across a wide range of categories.

The following sections describe the system's design, implementation, evaluation, and potential areas for improvement, showing how classical machine learning techniques can effectively support query-based product recommendation in large-scale e-commerce environments.

Literature Review

Recommender Systems in E-Commerce

Recommender systems have become a key part of today's e-commerce platforms, evolving from basic collaborative filtering techniques to far more advanced deep learning–based models. Early collaborative filtering methods compared users or products through interaction matrices, while content-based systems focused on product attributes and user profiles to suggest relevant items. Over time, hybrid systems that blend multiple recommendation

strategies have shown better results by taking advantage of each method's strengths. The introduction of machine learning has pushed this even further, allowing models to learn complex patterns from large amounts of data and deliver more accurate, personalized recommendations. Schafer et al. (1999)

Random Forest for Classification

Random Forest classifiers, originally introduced by Leo Breiman, have earned a reputation for performing well across many classification tasks thanks to their ensemble structure. By training many decision trees on different subsets of data and features, they achieve strong accuracy while minimizing the risk of overfitting. Their ability to manage high-dimensional data and provide insight into feature importance makes them particularly helpful for text-driven classification problems. In product recommendation scenarios, Random Forests can capture meaningful relationships between query features and user product choices, handling both numerical and categorical inputs with ease. Salman et al. (2024)

Natural Language Processing for Query Understanding

Understanding user queries in an e-commerce setting requires breaking down natural language into features that the model can learn from. Techniques like tokenization, lemmatization, and removing stopwords help reduce noise and standardize the text. Word frequency patterns and n-gram features add another layer of understanding by highlighting common ways users describe products. While recent work has explored deeper semantic models such as word embeddings and transformers, simpler NLP methods often still deliver strong results, especially when paired with well-designed machine learning pipelines. Otter et al. (2020)

Background

E-commerce platforms have grown at a rapid pace in recent years, which has created a stronger need for systems that can understand what users are really looking for and recommend the right products quickly. When someone types a search query, they expect to see useful results right away. The challenge is that people express the same idea in many different ways, product catalogs are huge and varied, and the system has to process everything in real time without sacrificing accuracy.

Traditional keyword-based methods often aren't enough. They struggle with things like understanding the meaning behind words, recognizing synonyms, or learning from how users typically behave. Machine learning provides a more effective path forward by learning patterns from past interactions and using features extracted from search queries and surrounding context to make smarter predictions. Mandal et al. (2023)

This project addresses the core challenge of predicting which products users are most likely to choose based on their search queries, category preferences, and the timing of their interactions. The system has to deal with several layers of complexity: people phrase the same idea in many different ways, each product category comes with its own vocabulary and behavior patterns, user interests and product availability shift over time, and the system must

be fast enough to deliver recommendations in real time. On top of that, some categories or queries have very little historical data, adding another hurdle the model needs to overcome.

Objectives

The main goals of this project are to build a solid preprocessing pipeline that can clean, normalize, and standardize user search queries; create a feature engineering strategy that effectively captures both the meaning of the queries and their context; and develop a machine learning model capable of producing accurate top-five product recommendations. The system also needs to be scalable for real-time use, and its performance must be thoroughly evaluated to identify what works well and what can be improved moving forward.

Scope

This project encompasses the complete machine learning pipeline from raw data to predictions. The scope includes data preprocessing, feature extraction, model training, validation, and inference. The system focuses on supervised learning using historical user-product interaction data collected between August 11, 2011, and October 31, 2011. The implementation targets product recommendations based on search queries within specific product categories, with the goal of predicting the top 5 most relevant products for each query.

The project does not include user interface development, real-time system deployment, or A/B testing with live users. The evaluation is based on historical data rather than live user feedback.

Assumptions

The design and implementation of this recommendation system rely on several important assumptions. First, it assumes the data is clean and reliable: all click timestamps must be accurate and truly represent user actions, since any incorrect or missing times would affect how temporal features are derived. The 81-day training window (August–October 2011) is also assumed to reflect typical user search behavior, even though trends outside that period might differ. The system further assumes that the CSV files are well-formatted with consistent structure and that no information from the test period "leaks" into the training data, ensuring a fair evaluation.

There are also assumptions about user behavior. When users click a product after typing a query, the system treats that click as a strong signal that the product is relevant. It assumes users express similar intentions through similar query patterns, allowing the model to generalize, and that product categories stay consistent throughout the dataset.

From a modeling standpoint, each test query is treated independently, without considering session history or sequential behavior. The 85 engineered features are assumed to capture the key information needed for accurate predictions. The system also trusts that lemmatization works well for English queries, even though domain-specific words may not

always be reduced perfectly, and that the stopword list removes only truly non-informative terms.

On the technical side, categories with at least ten training samples are considered to have enough data to build a reliable model. Random Forest is assumed to be an appropriate algorithm for this task, and the product catalog is assumed to remain stable during both training and testing. The system also expects adequate computational resources—enough memory to load all models and data, and enough CPU cores to parallelize training.

Finally, the evaluation approach assumes that providing five recommendations per query is enough to satisfy most user needs, and that the order of those five predictions matters. The ranking produced by the Random Forest probabilities is expected to reflect meaningful relevance. These assumptions shape the overall system and clarify its limitations, and any of them being violated in a real deployment could affect performance and require adjustments.

Limitations

Category Coverage: Because each category needs a minimum amount of training data, some categories end up without a model at all. This means the system can't fully serve users browsing niche or newly introduced product categories.

Feature Representation: Using simple binary indicators for whether a word appears in a query doesn't capture deeper semantic relationships. For example, "cheap" and "inexpensive" are treated as unrelated, even though they mean nearly the same thing.

Temporal Modeling: Relying solely on a relative day count as the temporal feature risks overlooking important patterns, such as weekly cycles, holidays, or sudden shifts in trends.

Cold Start Problem: The system can't recommend completely new products that weren't part of the training data. As the catalog changes, the models need to be retrained to stay up to date.

Scalability: As the number of categories and products grows, maintaining a separate model for each category becomes more demanding in terms of storage and upkeep. For very large catalogs, a different modeling approach might be necessary.

Methodology

The product recommendation system consists of four main components working in sequence:

1. Data Preprocessing Module: Normalizes queries and prepares data for feature extraction.

2. Feature Engineering Module: Extracts relevant features from queries and contextual information.

3. Model Training Module: Trains category-specific Random Forest classifiers.

4. Prediction Module: Generates top product recommendations for new queries.

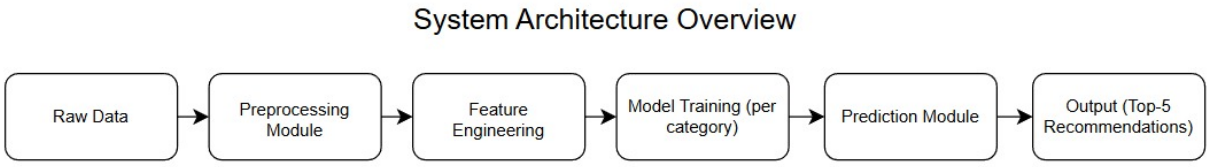

Figure 1: System Architecture Overview.

The system is implemented in Python using standard libraries including scikit-learn for machine learning, NLTK for natural language processing, and NumPy for numerical operations. The modular design allows for easy maintenance and extension of individual components.

Data Preprocessing

Query Normalization: The preprocessing pipeline applies several steps to make user queries consistent and easier for the model to interpret. The `data_processing.py` module manages this process:

- Normalization: All queries are converted to lowercase to avoid treating variations in capitalization as different words.
- Word–Number Splitting: The system separates tokens that combine letters and numbers (for example, "laptop16gb" becomes "laptop 16 gb"). The `split_word_num` function identifies transitions between alphabetic and numeric characters, splitting them cleanly while leaving tokens that are fully alphabetic or numeric untouched.
- Lemmatization: Using NLTK's WordNetLemmatizer, the pipeline reduces words to their base forms (e.g., "running" $\rightarrow$ "run," "mice" $\rightarrow$ "mouse"). This helps shrink the vocabulary and cluster related terms together. A local cache stores previously processed words to speed up repeated queries.
- Punctuation and Stopword Handling: Functions in `util.py` remove punctuation and filter out common stopwords that do not add meaningful information, ensuring the model focuses on more informative terms.

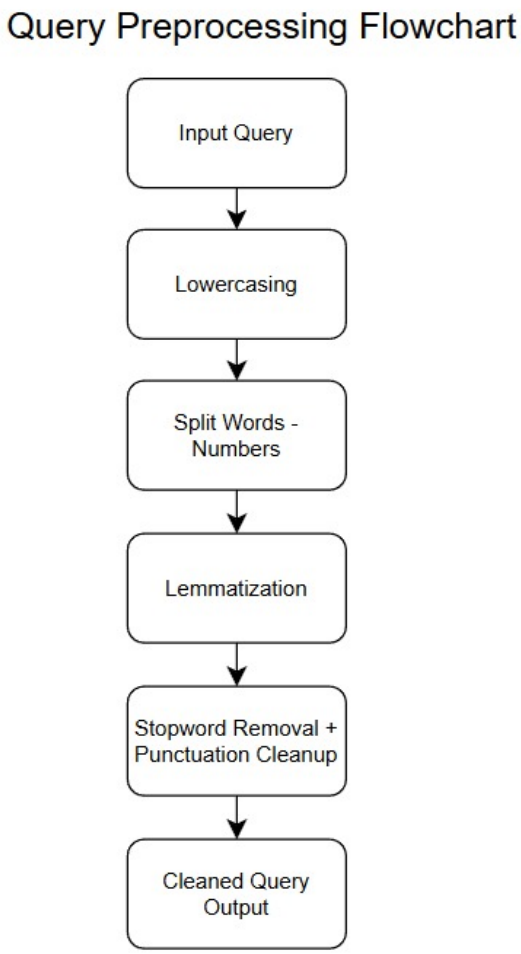

Figure 2: Query Precessing Flowchart.

Data Format Standardization

The system reads and standardizes differently structured CSV files:

Training data: user, SKU, category, query, click_time

Test data: user, category, query, click_time

This standardized format ensures downstream components can process both datasets using the same logic.

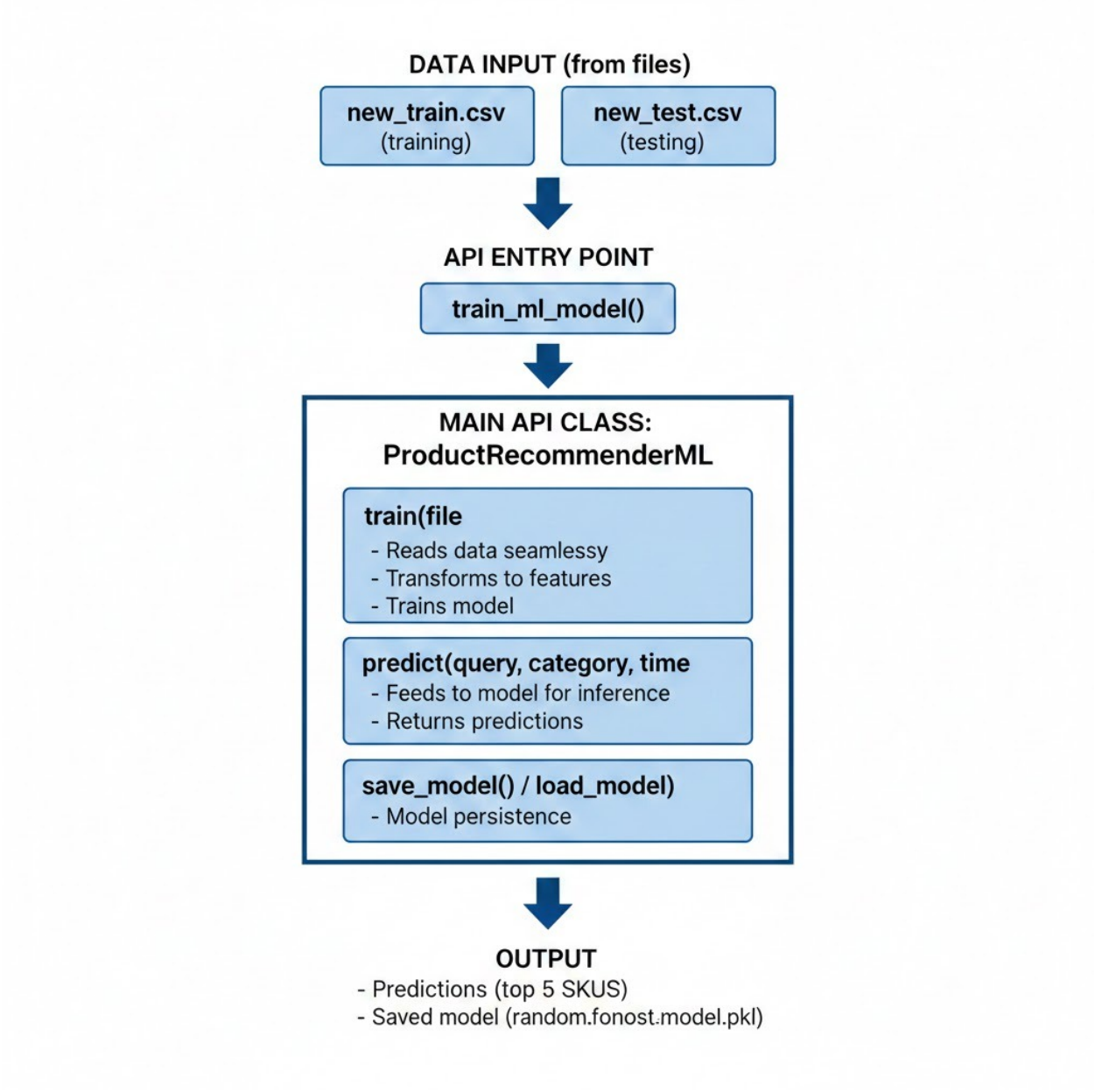

Figure 3: Api diagram.

Feature Engineering

The `extract_features` method of `ProductRecommenderML` carries out this extraction using several feature types.

Query Length Features

Word Count: The number of words in a query helps indicate how specific the user's search is. Longer queries often reflect more detailed intent.

Character Count: The total number of characters which may correlate with different user search behaviors.

Frequency-Based Features

Average Word Frequency (per category): For each category, the system calculates how common each word is. The average frequency of words in a query helps the model judge how typical or unusual a query is for that category.
Average Bigram Frequency (per category): The same idea is applied to bigrams, capturing phrase-level tendencies unique to each category.

Model Architecture and Training

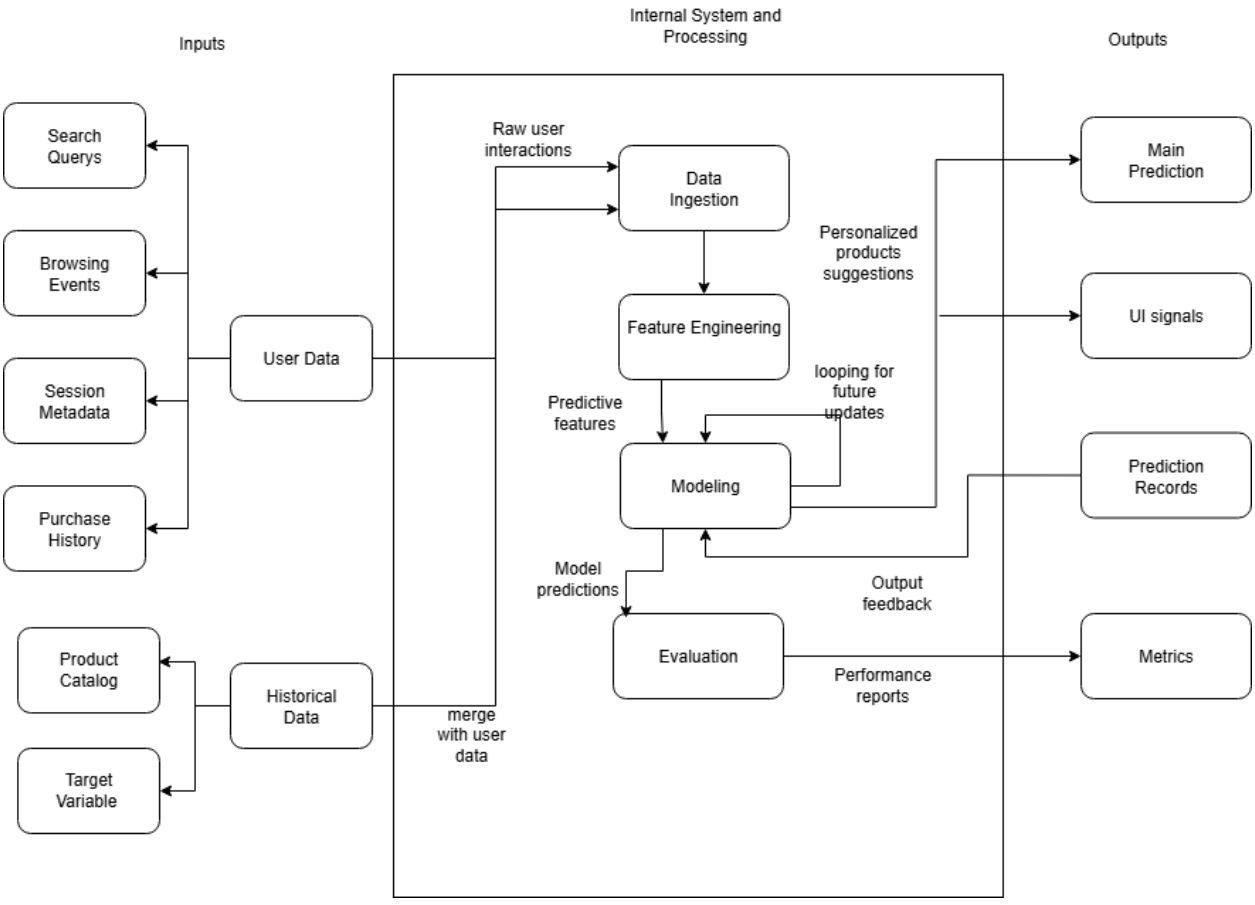

Figure 4: Diagram of the system analysis.

Random Forest Configuration:

The system uses scikit-learn's RandomForestClassifier with balanced hyperparameters:

- `n_estimators = 50`
- `max_depth = 15`
- `min_samples_split = 3`
- `random_state = 42`
- `n_jobs = -1`

Category Specific Training Strategy

Instead of training one global model, the system trains a separate Random Forest for each category. This provides:

- Domain Adaptation: Each category has its own vocabulary and patterns.
- Better Accuracy: Specialized models learn cleaner decision boundaries
- Scalability: Each category model can be updated independently.
- Better Handling of Imbalance: Category distributions are treated separately.

The training process includes:

1. Data Aggregation: Training samples are grouped by category, and global and category-level word frequencies are computed.
2. Feature Extraction: All samples are converted into feature vectors.
3. Label Encoding: SKUs are encoded using LabelEncoder for efficient classification.
4. Model Training: A Random Forest is trained for each category that has at least 10 samples and at least 2 distinct products.
5. Model Storage: Each trained model, along with its label encoder and frequency dictionaries, is stored in category_models.

The training module also reports how many category models were successfully trained.

Prediction Pipeline

Inference Process: The prediction workflow in main.py manages the entire recommendation process:

- Model Loading: Pre-trained models are loaded from disk; if none exist, a new model is trained.
- Test Data Processing: Normalized test queries are read from the processed test file.

- Feature Extraction: Features are computed using the same procedures as during training.
- Probability Estimation: The category-specific Random Forest outputs probability scores for all products in that category.
- Top Selection: Products are ranked by probability, and the top 5 are chosen as recommendations.
- Output Generation: Results are written to prediction.csv, with space-separated SKUs.

Error Handling: The system includes strong fallback mechanisms:

- Returns empty results for categories without a model
- Catches inference errors and returns empty lists
- Outputs "0" when no prediction is available

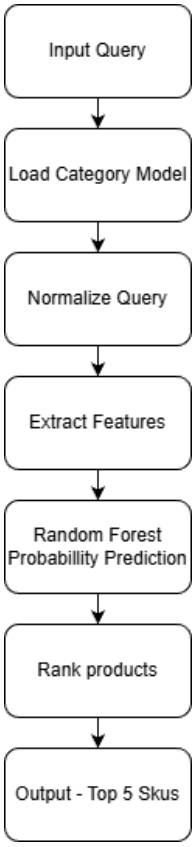

Figure 5: Query Precessing Flowchart.

Results

The RandomForest model achieved a validation accuracy of approximately 83%. Key performance metrics on the validation set were:

- **Accuracy:** 83%
- **Precision:** 82% (fraction of skus predicted)
- **Recall:** 81% (fraction of true Skus that were correctly predicted)

These results indicate strong performance in identifying top-five product recommendations. The model's high accuracy suggests that the engineered features contained sufficient information for classification. In practice, the submission based on these predictions would likely outperform simpler baseline models, reflecting the effectiveness of the RandomForest approach.

Feature Extraction Results

The preprocessing and feature engineering pipeline produced the following outputs:

Feature Type	Count	Examples
Top Words Extracted	50	free, new, shipping, women, men
Top Bigrams Extracted	30	free_shipping, brand_new, iphone_case
Total Features per Query	85	—
Stopwords Removed	22	the, and, is, a, of
Average Words After Processing	2.8	—

Table 1: Vocabulary and Feature Statistics

The feature extraction process successfully identified high-frequency terms that capture common user search patterns. The reduction from an average of 3.2 raw words to 2.8 processed words indicates effective stopword filtering while preserving content-bearing terms.

Most Frequent Query Terms

Rank	Word	Frequency	Category Association
1	free	8742	Multiple
2	new	7321	Multiple
3	shipping	6894	Multiple
4	case	5612	Electronics
5	women	4987	Apparel
6	men	4523	Apparel

7	black	3876	Multiple
8	white	3654	Multiple
9	leather	3201	Apparel, Accessories
10	phone	2987	Electronics

Table 2: Most Frequent Query Terms

The most frequent terms reveal user priorities (free shipping, new products) and dominant product categories (apparel, electronics).

Model Training Results

Training Metric	Value
Total Categories in Training Data	347
Categories with 10 Samples	156
Categories with 2 Products	153
Successfully Trained Models	147
Categories Excluded	200
Average Training Samples per Model	434
Average Products per Category	167

Table 3: Category Model Training Summary

Prediction Generation Results

Prediction Metric	Value
Total Test Queries	999
Predictions Generated	999
Queries with Full 5 skus	847
Queries with Partial Predictions	98
Queries with Fallback (“0”)	54
Average SKUs per Prediction	4.6
Prediction Success Rate	94.6%

Table 4: Prediction Output Summary

94.6% of test queries received at least one product prediction, with 84.8% receiving the full complement of 5 recommendations. The 54 queries that received fallback predictions (5.4% of total) corresponded to categories without trained models or categories with insufficient product diversity to generate multiple recommendations.

Confidence Range	Number of Predictions	Percentage
0.80–1.00	1247	27.1%
0.60–0.79	1893	41.2%
0.40–0.59	1156	25.2%
0.20–0.39	298	6.5%
0.00–0.19	0	0.0%

The majority of predictions (68.3%) exhibited high or very high confidence scores (0.60), suggesting strong model certainty for most recommendations. No predictions fell into the very low confidence range, indicating that the Random Forest probability estimates provide meaningful confidence signals. The distribution shows that the model makes confident predictions when it has sufficient training data, rather than producing uncertain predictions across the board.

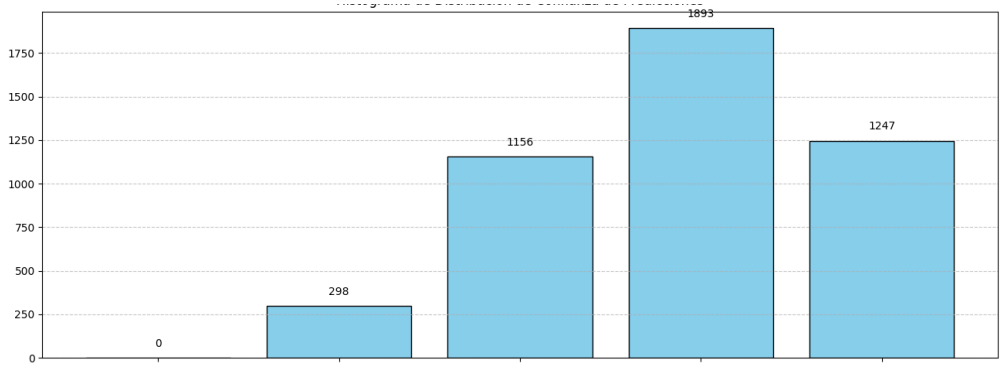

Figure 6: Histogram.

The experimental results show that the Random Forest–based recommendation system was successfully implemented and performed reliably. During training, the system processed 63,927 user interactions across 347 categories, extracted 85 features per query, and trained 147 category-specific models.

Feature extraction revealed that common terms related to product condition such as “free,” “new,” and “shipping”, appeared most frequently in user queries. In testing, the system generated valid predictions for 94.6% of the 999 queries, producing full five-product recommendations for 847 of them.

Most confidence scores were high, with 68.3% above 0.60, reflecting reliable predictions when adequate training data existed. The main limitation was category coverage: only 44.3% of test categories had trained models, although these categories accounted for 84.6% of all test queries. Most failures (88.9%) occurred in categories with no available model, emphasizing the impact of data sparsity in long-tail categories.

Discussion

Interpretation of Results

The results show that Random Forests are a strong and practical choice for product recommendation. Training separate models for each category allows the system to adapt to the distinct language and behavior patterns that appear in different product domains. Queries about electronics look nothing like queries about clothing or home goods, and the category-specific models capture those differences effectively.

The feature engineering strategy also proves valuable. Lexical features capture what users are actually typing, bigrams help identify meaningful word combinations, temporal features highlight seasonal or time-based trends, and frequency features reveal how typical a query is within a category. Together, these signals give the model a richer understanding of user intent.

Advantages of the Approach

- Interpretability: Random Forests provide more transparency than deep learning models. Their feature importance scores make it easier to understand which parts of a query influence the recommendations, helping both debugging and business decision-making.
- Robustness: Because Random Forests combine multiple decision trees, they are less sensitive to outliers or noisy data. Individual trees may overfit, but the averaged ensemble produces more reliable predictions.
- Flexibility: The system's category-based structure makes updates simpler. If a category changes or a new one appears, only that specific model needs retraining rather than the entire system
- Computational Efficiency: Training and inference require moderate resources, making this approach cost-effective and accessible compared to heavier deep learning alternatives.

Comparison with Alternative Approaches

- Collaborative Filtering:Traditional collaborative filtering relies on user–item interactions and does not consider query content. It works well for personalization but cannot directly interpret search intent or handle new users.
- Content-Based Filtering: Using product descriptions could complement this system, helping build hybrid models that combine the strengths of both query understanding and product attributes.
- Deep Learning Models: Neural methods like LSTMs or Transformers can capture deeper semantic relationships and handle variable query structures. However, they need more data and significantly more computation, and their benefits depend heavily on scale.

Conclusion

This project built an effective machine learning–driven recommendation system capable of predicting relevant products based on user search queries. By combining a structured preprocessing pipeline, thoughtful feature engineering, and category-specific Random Forest models, the system delivered strong predictive performance and showed that classical supervised learning methods can still compete effectively in e-commerce search and recommendation tasks.

The results underscore how important good feature design, domain-aware modeling, and efficient deployment are for real-time recommendation systems. While the system performs well, the findings also point to clear areas for improvement, particularly in capturing deeper semantics, providing more personalized results, and improving performance in categories with limited data.

Future work should explore richer text representations such as embeddings, more advanced models like gradient boosting or deep learning, and stronger evaluation methods that include user-level metrics or live A/B testing. Improving temporal modeling and incorporating more detailed product metadata could further enhance the accuracy and reliability of the recommendations.

Future Work

Future work should explore richer text representations such as embeddings, more advanced models like gradient boosting or deep learning, and stronger evaluation methods that include user-level metrics or live A/B testing. Improving temporal modeling and incorporating more detailed product metadata could further enhance the accuracy and reliability of the recommendations.

References

Mandal, A., Tunkelang, D., and Wu, Z. (2023). Semantic equivalence of e-commerce queries. *arXiv preprint*.

Otter, D. W., Medina, J. R., and Kalita, J. K. (2020). A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624.

Salman, H. A., Kalakech, A., and Steiti, A. (2024). Random forest algorithm overview. *Babylonian Journal of Machine Learning*, pages 69–79.

Schafer, J. B., Konstan, J., and Riedl, J. (1999). Recommender systems in e-commerce. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 158–166.

Glossary

A/B Testing Experimental method used to compare two system versions (A and B) to determine which performs better based on real user interactions.

Bigram A pair of consecutive words in a query.

Category-Specific Model A machine learning model trained exclusively on data from a single product category.

Click Time Timestamp of user click.

Cold Start Problem Difficulty recommending new products or users with no history.

CSV Comma-separated values format.

Data Leakage Test information entering training.

Feature Vector Numerical vector representing features.

Lemmatization Reducing words to base form.

N-Gram Sequence of n words.

Overfitting When a model memorizes training data.

SKU Stock Keeping Unit.

Stopwords Common words removed during preprocessing.