 School of Computing and Creative Technologies

**Predictive Modelling of Query-Product Relevance: A Case Study on Structured Search Optimisation**

Student Code: 24038341 | Module Code: UFMFJR-15-M | Module Title: Machine Learning

1. **Introduction**

In today's competitive e-commerce environment, the precision and relevance of search results directly shape consumer decisions and business profitability. With approximately 45% of e-commerce revenue driven by search functionalities, optimising query-product relevance is critical *(Constructor IO, 2025)*. Consumers increasingly demand faster and more accurate searches, with Accenture reporting that 75% of shoppers express frustration over difficulties finding suitable products quickly *(Crowley et al, 2024)*. Addressing this challenge through predictive modelling techniques can significantly reduce cognitive load, enhance customer satisfaction, and drive revenue growth. This study leverages machine learning methodologies to improve structured search optimization and provide user semantics insights, ultimately enhancing user experience and solidifying search as a strategic profit driver.

1. **Dataset Overview**

The datasets are originally sourced from Kaggle - Home Depot Product Search Relevance Competition

*Table I: Dataset information*

|  |  |  |
| --- | --- | --- |
| # | File | Description |
| 1 | attributes.csv | 2,038,381 records (after NaN removal) with fields:  product\_uid |name |value |
| 2 | product\_descriptions.csv | 124,428 records with fields:  product\_uid |product\_description |
| 3 | train.csv | 74,067 records with fields:  id|product\_uid |product\_title|search\_term|relevance |

During EDA, we removed 1.1% of the *attributes.csv* dataset due to missing values. Given the low proportion, listwise deletion was preferred over imputation, which could introduce bias—especially in the absence of evidence that such data is missing (completely) at random *(Schafer, J.L. and Graham, J.W., 2002)*. The small removal had minimal impact on statistical power, while avoiding distortion of distributions or artificial variance compression that imputation methods can cause.

Then, the id field was removed from *train.csv* dataset since it adds no meaningful information. Then *product\_description.csv* and *train.csv* was consolidated into *query\_df* dataset.

During the featurisation and enrichment state, the 40 top relevant attributes per *product\_uid* were added as columns and the remaining ones were concatenated and added as a description. The goal was to maximize the relationships between the *search term* and the different *product description* texts involved.

Query search terms

As shown in the figure, the top 10 queries range from 11 to 15 occurrences. These include precise product-related terms such as: *ceiling mounted lighting fixtures, metal sheet*, and *½ zip wall”*

*Figure I: Top query search terms*

A blue and white bar chart

AI-generated content may be incorrect.

This pattern suggests that even the most frequent queries are not dominant, and no single term overwhelms the dataset — reflecting a diverse and granular product search space. Such specificity in user queries is typical in e-commerce or hardware retail environments, where customers often search for exact models, dimensions, or materials.

From a modelling perspective, this justifies the need for:

* Robust text preprocessing (to normalize sparse variants)
* Query clustering or dimensionality reduction, and
* Intent-aware retrieval using semantic representations

Clustering Analysis

The clustering exercise on *dev\_df* / training dataset sought to uncover latent structure in users’ search‑term behaviour without relying on labels. After reducing the high‑dimensional sparse matrix with *Truncated SVD* and projecting the result into two dimensions through a tuned *t‑SNE*, we fitted *K‑Means* and obtained a global Silhouette coefficient of 0.61. In practical terms this means that, on average, every record lies noticeably closer to its own cluster centre than to the closest competing cluster, indicating both good cohesion and good separation. Scores above 0.6 are widely considered evidence of a strong underlying pattern, so the model is capturing real, non‑random groupings in the data rather than artefacts of the projection.

The *t‑SNE* map corroborates this numerical finding: most clusters appear as compact, clearly delimited clouds, while the boundaries between them are visibly sharp.

*Figure II: Search term clustering*

A colorful paint splatter on a graph

AI-generated content may be incorrect.

A preliminary linguistic inspection suggests that the clusters may reflect semantically coherent groupings. The table below presents a summary derived from the top 10 words per cluster and tentatively associates each set with an indicative intent, acknowledging that interpretation may vary without domain expert validation.

*Table II: Top 10 words per cluster*

|  |  |  |
| --- | --- | --- |
| **cluster\_id** | **Description** | **Possible Indicative theme** |
| 0 | |  | | --- | | A close-up of words  AI-generated content may be incorrect. | | Exterior fixtures & general home‐improvement hardware |
| 1 | A close-up of a sign  AI-generated content may be incorrect. | Interior home renovations: walls, showers, and doors |
| 2 | A close-up of words  AI-generated content may be incorrect. | Interior painting & lighting |
| 3 | A close-up of a sign  AI-generated content may be incorrect.A close up of words  AI-generated content may be incorrect.A close-up of a sign  AI-generated content may be incorrect. | Outdoor & kitchen fixtures: patio doors, faucets, grills |
| 4 | Plumbing hardware & fittings |
| 5 | Bathroom fixtures & vanities |
| 6 | |  | | --- | | A close up of a sign  AI-generated content may be incorrect. | | Flooring & cabinetry surfaces |
| 7 | A close up of a word  AI-generated content may be incorrect. | Kitchen cabinetry & hardware |

Potential interpretation:

Because each cluster might correspond to a concrete renovation task, the segmentation can directly inform personalised ranking, campaign targeting and content curation

1. **Problem Definition**

This study tackles a supervised regression problem focused on predicting a continuous relevance score (ranging from 1 to 3) for each *(search\_term, product)* pair in the Home Depot dataset. The inputs consist entirely of textual features—search term, product title, product description, and structured product attributes—processed using NLP techniques such as TF-IDF (token and character level), BM25 weighting, and numeric overlap/length-based statistics.

Since all rows sharing the same search term belong to a semantic group, we enforce group integrity to prevent leakage:

* Hold-out splitting ensures that queries in the test set are completely unseen during training.
* *GroupKFold* cross-validation ensures group-wise independence across folds during hyperparameter optimisation.

The task is computationally challenging due to the high dimensionality and sparsity of the feature space (e.g. TF-IDF vectors, BM25). To manage this:

* We use Bayesian optimisation (*Hyperopt*) to tune models efficiently with O(n log n) complexity, avoiding exhaustive grid search with O(n2).
* We apply sparse matrix operations, modular featurisation, and code optimisation to run on a non-GPU (CPU-only) environment, mitigating memory and compute constraints.

Additionally, we complement the regression pipeline with an unsupervised semantic analysis of the query space. To account for the non-linear nature of linguistic features—common in NLP—we use t-SNE instead of PCA for dimensionality reduction. This allows us to cluster search terms meaningfully via *KMeans*, surfacing dominant term patterns and potential query groupings for further expert analysis or feature development.

Algorithms selected

* XGBoost Regression, CatGBoost Regression and LightGBM Regression
* T-SNE since it properly addresses nonlinear data

Algorithms disregarded

* Linear Regression
* Principal Components Analysis (not effective in nonlinear features)

Evaluation Methodology

We use *RMSE* (same metric as the Kaggle competition) because it penalises large absolute errors—crucial when mis‑ranking highly relevant items hurts revenue. R² would ignore scale and is less interpretable for rank‑style targets.

Furthermore, for the unsupervised clustering approach, we use *Silhouette* score as a key metric to evaluate the quality of the clustering.

1. **Analysis and Evaluation**

The following pipeline was created to solve this problem:

*Table III: Overall workflow process*

|  |  |  |
| --- | --- | --- |
| Sequence | Title | Description |
| 1 | Data Cleaning, and Preparation | * Load datasets (*attributes\_df, query\_df, and product\_descriptions\_df*) * Null Removal in *attributes\_df* dataset * Drop *id* column in *query\_df* dataset * Left Join query\_df with *description\_df* into *data\_sample* |
| 2 | Data Enrichment | * Join *data\_sample* with *attributes\_df* with the top most frequent features as columns and the remaining features as a concatenated column description field. |
| 3 | Group Shuffle Split the dataset | * Split the *full* dataset into training (80%) and testing set (20%) with the *search\_term* group criterion. |
| 4 | Featurisation | A function that addresses the NLP approach was created with the following treatments:   * Description text clean * Numeric extraction measures from *product\_title*, *product\_description*, and the 40 most frequent product attributes. * Stem TF-IDF vectorisation of *search\_term* and *title\_clean* * Lemma TF-IDF vectorisation of *desc\_clean* + *attrs\_text* * N gram TF\_IDF vectorization of *search\_term* and *title\_clean* to capture non-linear relationships. * Scale the numeric features, if any * Text vectorization * Cosine similarity field creation amongst *cos\_t, cos\_l, bm25\_t, bm25\_l, cos\_char* * Sparse matrix creation with all those features. |
| 5 | Optimisation Pipeline | Inside the training set scope (pseudo code):  For each of the models:  Generate a Hyper opt Optimization (target function is the mean (RMSE) of a new 5 CV split) with no group leakage.  Apply the *featurise* function (Highly expensive)  The output is the optimised hyperparameters  For each of 5 Group K-Fold splits:  Apply *featurise* function (Highly expensive)  Record the RMSE obtained in each split with the optimized hyper parameters  End For  End For |
| 6 | Model Selection | Based on RMSE recordings and bootstrap analysis |
| 7 | Model Evaluation | Based on the selected model and best hyperparameters, we fit in the overall training set and predict in the testing set to obtain an RMSE. |

**Data enrichment:**

Every SKU (*product\_uid*) appears many times in *attributes.csv*. We join those rows so that each product row now owns *all* textual evidence that enrich the product description.

Naively expanding the 5401 unique attribute names into 5401 one‑hot columns would yield approximately (74 k × 5 k) 370 M cells—impossible to process on a laptop and prone to the “curse of dimensionality”. On the contrary, concatenating all the features in a text might make us lose signal of critical features that might impact in our target.

To get a feasible solution, we chose only the top 40 most frequent features based on domain knowledge and concatenate the remaining features into a text description column for each SKU.

**Group aware data split:**

We first allocate 20 % of *search‑term* groups to a hold‑out dev/test pair (outer split). Each unique query string therefore lives either in training or in test, never both, blocking lexical leakage. This follows best‑practice for learning‑to‑rank tasks with repeated queries.

**Featurisation:**

Short‑text block: 2–5‑gram TF‑IDF on *search term* and *title*. Short n‑grams catch typos and word order variants—character n‑grams are especially robust to noisy e‑commerce queries  
Long‑text block: lemmatised TF‑IDF on descriptions + attributes tail. Lemma normalises inflections; long‑form gives global context.   
Similarity & BM25: after L2‑normalising both query and doc vectors we compute cosine similarity and BM25 scores. BM25 re‑weights terms by document length and has been shown to outperform plain TF‑IDF in product‑search settings.

Numeric text statistics: token counts, overlap ratios, etc., are standardised and concatenated. All TF‑IDF / BM25 matrices stay scipy‑sparse float32—saves RAM and speeds up dot products.

**Model Selection**

**Hyper parameter optimisation:**

For every candidate model we run a 5‑fold *GroupKFold* on the *training/dev* split only. *Hyperopt’s* TPE sampler minimises mean RMSE across folds; each trial builds its own leakage‑free vector store. Nothing from the *test* portion is touched—so no information leakage occurs.

With the *best* hyper‑parameters frozen, we perform 5 shuffled *GroupKFold* evaluations. This yields a sampling distribution of *RMSE* for each model, later boot‑strapped for CIs. The procedure is lighter than a textbook nested CV because hyper‑parameters are not re‑tuned inside every outer split; instead, the (computationally expensive) tuning is run once per model. As Saito et al. note, “single‑shot” tuning followed by repeated test‑set sampling gives a nearly unbiased estimate at a fraction of the cost.

Cat Boost achieved the lowest μ RMSE and σ, edging LGB and XGB trees handle sparse‑high‑cardinality data efficiently and are less memory‑hungry than deep nets.

The final Cat Boost Regressor trained on the full dev set scores within the 95 % CI of the outer‑CV distribution, supporting the validity of the tuning protocol.

*Table IV: Summary of search spaces*

|  |  |
| --- | --- |
| Model | Search space |
| Cat Boost Regressor | *cat: {*  *"iterations": hp.quniform("cat\_iter", 500, 3000, 200),*  *"depth": hp.quniform("cat\_depth", 4, 14, 1),*  *"learning\_rate": hp.loguniform("cat\_lr", np.log(0.005), np.log(0.3)),*  *"l2\_leaf\_reg": hp.loguniform("cat\_l2", np.log(1e-3), np.log(50)),*  *"subsample": hp.uniform("cat\_sub", 0.5, 1.0),*  *"bagging\_temperature": hp.uniform("cat\_bt", 0, 1),*  *},* |
| LGB Regressor | lgb: {                  "num\_leaves": hp.quniform("lgb\_num\_leaves", 16, 512, 8),                  "max\_depth": hp.quniform("lgb\_max\_depth", 4, 50, 1),                  "learning\_rate": hp.loguniform("lgb\_lr", np.log(0.005), np.log(0.15)),                  "n\_estimators": hp.quniform("lgb\_n", 400, 1600, 100),                  "min\_child\_samples": hp.quniform("lgb\_mcs", 3, 80, 5),                  "boosting\_type": hp.choice("lgb\_boost", ["gbdt", "dart"]),                  "subsample": hp.uniform("lgb\_sub", 0.5, 1.0),                  "colsample\_bytree": hp.uniform("lgb\_col", 0.5, 1.0),                  "subsample\_freq": hp.quniform("lgb\_freq", 1, 7, 1),                  "reg\_alpha": hp.loguniform("lgb\_alpha", np.log(1e-3), np.log(50)),                  "reg\_lambda": hp.loguniform("lgb\_lambda", np.log(1e-3), np.log(50)),              } |
| XGB Regressor | xgb: {                  "max\_depth": hp.quniform("xgb\_depth", 3, 20, 1),                  "learning\_rate": hp.loguniform("xgb\_lr", np.log(0.005), np.log(0.3)),                  "n\_estimators": hp.quniform("xgb\_n", 400, 2000, 100),                  "subsample": hp.uniform("xgb\_sub", 0.5, 1.0),                  "colsample\_bytree": hp.uniform("xgb\_col", 0.5, 1.0),                  "gamma": hp.uniform("xgb\_gamma", 0, 5),                  "reg\_alpha": hp.loguniform("xgb\_alpha", np.log(1e-3), np.log(50)),                  "reg\_lambda": hp.loguniform("xgb\_lambda", np.log(1e-3), np.log(50)),              } |

*Table V: Summary of cross validation results*

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Hyper parameters | RMSE Mean | RMSE Std |
| Cat Boost Regressor | *cat: {*  *"bagging\_temperature": 0.11379626,*  *"depth": 12,*  *"iterations": 236,*  *"l2\_leaf\_reg": 14.24113813952691,*  *"learning\_rate": 0.0633337750854,*  *"subsample": 0.9068277038201439*  *}* | *0.497257* | *0.002881* |
| LGB Regressor | lgb: {      "boosting\_type": "gbdt",      "colsample\_bytree": 0.67093631858,      "learning\_rate": 0.006540905666557,      "max\_depth": 30,      "min\_child\_samples": 10,      "n\_estimators": 385,      "num\_leaves": 320,      "reg\_alpha": 0.04423500084098686,      "reg\_lambda": 14.389107387593821,      "subsample": 0.6502106773952542,      "subsample\_freq": 1    } | 0.498099 | 0.002948 |
| XGB Regressor | xgb: {      "colsample\_bytree": 0.94244098706,      "gamma": 2.8204426530143745,      "learning\_rate": 0.16975970710347,      "max\_depth": 4,      "n\_estimators": 70,      "reg\_alpha": 0.00106117767232505,      "reg\_lambda": 0.0019126625509608,      "subsample": 0.650210677395254    } | 0.498197 | 0.003169 |

*Figure III: Cross Validated Results – Real vs Bootstrapped*

*A diagram of a model

AI-generated content may be incorrect.*

**Bootstrapped approach**

To assess the reliability of our model evaluation strategy, we compared the observed RMSE scores from cross-validation against the empirical distribution obtained via bootstrapping. Specifically, for each model, we conducted a non-parametric hypothesis test:

Let the observed RMSE values be those computed on the five outer cross-validation splits. These serve as our limited-sample estimates of model performance. To understand whether these values accurately reflect the expected generalisation error, we constructed a bootstrap distribution by repeatedly resampling the five RMSE values and computing their mean. This generates an empirical approximation of the sampling distribution of the RMSE mean under the assumption that the observed values are representative.

For each model, the null hypothesis H0​ states that the observed mean RMSE from cross-validation is consistent with the true expected RMSE. We tested this by evaluating whether the observed mean lies within the central 95% of the bootstrap distribution, and computed a two-tailed p-value accordingly.

*Table VI: Bootstrap distribution – Statistical Significance*

A screenshot of a black screen

AI-generated content may be incorrect.

The results strongly failed to reject H0​ with a 95% confidence interval.

**Model Evaluation**

*Table VII: Summary of model evaluation*

|  |  |  |
| --- | --- | --- |
| Test/Train | Model | RSME |
| Train | CatBoost | 0.468948 |
| Test | CatBoost | 0.494044 |

The final CatBoost model achieved an RMSE of 0.469 on the training set and 0.494 on the test set. To assess if this test performance was expected, we compared it against a bootstrapped distribution of cross-validation RMSEs. The 95% confidence interval from the bootstrap ranged from 0.495 to 0.500, and the p-value of 0.997 indicates no statistically significant deviation between the test RMSE and expected values.

This bootstrap-based check helps confirm the calibration of our CV-based model selection process, highlighting that the gap between train and test performance is not due to data leakage or overfitting, but is instead attributable to natural sampling variability.

**Model Interpretation**

We use Shapley to exhibit the features importance

*Figure IV: Shapley Cat Boost Feature Importance*

A graph with blue and white stripes

AI-generated content may be incorrect.

In the *CatBoost* model the five highest‐ranking features are, in order:

1. *cos\_t* – cosine similarity between the query and the product *title* TF‑IDF vectors
2. *bm25\_l* – *BM25* similarity between the query and the *long* text blob (description + attributes)
3. *cos\_l* – cosine similarity on the same long text blob
4. *bm25\_t* – *BM25* similarity on the title
5. *cos\_char* – character‑level cosine similarity on the title

Together they account for just over 55% of the total Shapley importance mass.  
Their dominance confirms that what ultimately drives the relevance score is the *lexical overlap* between the user’s search phrase and the various textual fields that describe the product. BM25 scores matter almost as much as the simpler cosine measures, indicating that IDF‑down‑weighting of frequent terms and length‑normalisation contribute useful extra signal beyond plain TF‑IDF.

Surprisingly, brand information is almost absent from the top‑20 list. The Home Depot catalogue contains thousands of brands, many of which appear rarely in the training data; *CatBoost* therefore prefers more reliably distributed cues such as title similarity. In addition, brand names already occur verbatim in the product title for most listings, so the “brand” attribute yields little incremental gain once lexical similarity to the title has been accounted for.

The long tail of attribute‑level numeric features has only marginal impact. This supports our design choice to include only the 40 most frequent attributes explicitly and to fold the remainder into the “long” text field: their predictive power is largely absorbed by the BM25 and cosine features computed over that composite blob.

Overall, the Shapley profile confirms that the feature‑engineering pipeline is well aligned with the task:

* High‑value signal comes from *term‑matching* scores (cosine, BM25) between query and product texts.
* Supplementary signal comes from *text structure* (length and span statistics).
* Brand and rare attributes, though intuitively useful to humans, do not move the needle once lexical overlap is modelled.

1. **Reflection and further work**

The current pipeline demonstrates that a sparse, tree‑based approach can deliver solid relevance predictions under strict hardware limits and group‑aware evaluation. Several aspects nevertheless warrant further investigation.

Feature Engineering

* BM25 parameters. The standard *(k₁ = 1.2, b = 0.75)* complies with prior IR literature (Robertson, S. and Zaragoza, H.,2009); tuning them on the development set may yield small but measurable improvements.
* Dimensionality control. The TF‑IDF space is large. Techniques such as Truncated SVD could cut memory and training time while helping regularisation; their real impact should be verified experimentally.

Model Architecture

* Stacking / meta‑learning. Using out‑of‑fold predictions from LightGBM, XGBoost, and CatBoost as additional inputs to a simple neural combiner may improve robustness without radically changing the pipeline.
* Transformer embeddings. Pre‑trained sentence encoders capture semantics beyond term overlap. Incorporating these vectors—either alone or alongside existing features—remains an open line of work, especially if suitable compute is available.

Compute Considerations

Bayesian hyper‑parameter search with early stopping kept experiments feasible on CPU hardware but run‑times were still substantial. Porting critical steps (e.g. sparse linear algebra, boosted‑tree training) to GPU libraries could shorten iteration cycles; any speed‑up should be weighed against added engineering complexity.

Overall, this study is a practical baseline for query‑relevance regression under limited resources. Future work should focus on targeted feature refinements, cautious exploration of neural or hybrid models, and validation designs that balance statistical rigour with computational expense.

1. **References**

Anna Montoya, RG, and Will Cukierski. *Home Depot Product Search Relevance.* https://kaggle.com/competitions/home-depot-product-search-relevance, 2016. *Kaggle*.

Crowley, J., Wright, O., Standish, J., Blackburn, E. and Weiss, E. (2024) The Empowered Consumer. *Accenture* [[online]](https://www.accenture.com/us-en/insights/consulting/empowered-consumer).

Finkelshteyn, E. (2025) Don’t Stop at Search Relevance in Ecommerce. Constructor [[online]](https://constructor.com/blog/search-relevance-is-it-really-the-most-valuable-metric).

Ikezaki, T. and Kaneko, O. (2022). *Data-driven Estimated Iterative Feedback Tuning by One Shot Experimental Data.* Keisoku Jidō Seigyo Gakkai Ronbunshū, 58(2), pp.112–117. doi:10.9746/sicetr.58.112.

Robertson, S. and Zaragoza, H. (2009) *The Probabilistic Relevance Framework: Bm25 and Beyond*. *Foundations and Trends® in Information Retrieval* [online]. 3 (4), pp. 333-389.

Schafer, J.L. and Graham, J.W., 2002. *Missing data: Our view of the state of the art*. Psychological Methods, 7(2), pp.147–177. <https://doi.org/10.1037/1082-989X.7.2.147>

***Total net word count excluding Charts, Tables and references: 2198***