Learning to Rank in E-Commerce Marketplaces:

Why Target Design and Feature Engineering Beat Model Complexity

Anjan Goswami

Abstract

Replacing a hand-tuned heuristic ranking system with machine-learned ranking in a large-scale e-commerce marketplace is one of the hardest problems in applied search. The challenge is not model architecture—it is everything else: what the model should predict, how training data is constructed, how negative examples are sampled, and which features encode the domain knowledge that made the heuristic effective. This case study describes the multi-year effort to replace eBay's Best Match ranking system with a machine-learned model. We detail the journey through target variable design (where click prediction, sales prediction, and naive objectives each failed for different reasons), training data construction (including position-bias-aware negative sampling), and the critical insight that converting the heuristic's hand-crafted features into ML features was the key to finally winning. The eventual system—a gradient boosted tree model with real-time behavioral signals and price-normalized sales targets—succeeded where years of model-centric approaches had not.

1 The Problem: Beating Best Match

eBay's Best Match ranking system was a linear heuristic: a weighted combination of features designed by product managers and search engineers, tuned over years of experimentation. Some features were straightforward (price competitiveness, seller rating, shipping speed). Others encoded subtle domain knowledge—complex interaction features that captured marketplace dynamics difficult to articulate formally but easy for experienced product managers to recognize.

Best Match was the incumbent, and it was **hard to beat**. Multiple attempts to replace it with machine-learned ranking had failed. The failures were instructive: each revealed a different way that naive application of ML to ranking can go wrong.

The core challenge was that e-commerce ranking optimizes for a **business outcome** (revenue), not an information retrieval outcome (relevance). A search engine ranks documents by topical match. A marketplace ranks products by expected transaction value—a function of relevance, price, conversion likelihood, seller reliability, and inventory dynamics. Standard learning-to-rank methods, developed for web search, do not account for this distinction. Importing them directly produces models that optimize the wrong objective.

2 Training Data Construction

2.1 The Position Bias Problem

The most natural approach to building training data for learning-to-rank is to use historical click and purchase data from the current ranking system. This creates a fundamental bias: items ranked higher receive more impressions, more clicks, and more purchases *because* they are ranked higher, not necessarily because they are better. A model trained on this data learns to replicate the existing ranking, not to improve it.

We addressed position bias through **source-aware sampling**. Rather than sampling positive and negative examples from Best Match results, we drew examples from **price-sorted results**—a ranking determined entirely by price, independent of the relevance model. This provided a clean

signal: items purchased from price-sort were bought despite having no relevance-based ranking advantage, providing unbiased evidence of genuine demand.

2.2 Negative Sampling Design

For each positive example (an item sold in a session), we sampled negative examples (items shown but not sold). The sampling design required balancing several competing concerns:

- **Ratio**: We experimented with positive-to-negative ratios of 1:10, 1:30, 1:50, and 1:100. Negatives were sampled randomly from the top 100 results in the session. We found that **1:30 from the top 100** provided the best balance: enough negatives to give the model sufficient contrast, but not so many that training became dominated by easy negatives (items obviously irrelevant to the query).
- **Hard negative weighting**: We hypothesized that closely matched negatives—items similar to the purchased item but not sold in that session—would be particularly informative training examples. We implemented importance weighting that upweighted hard negatives based on their similarity to the positive. **This did not produce measurable improvements.** The model learned to distinguish sold from unsold items equally well with uniform negative sampling, suggesting that the feature representation was already capturing the relevant distinctions without needing curriculum-style training.
- **Pairwise vs. pointwise formulation**: We experimented with both pairwise ranking (learning preferences between sold and unsold items) and pointwise regression (predicting a binary or continuous target). Both formulations were viable; the pairwise approach had a slight edge in early experiments but the advantage disappeared as the feature set matured.

2.3 Key Insight: Negative Results Are Results

The hard negative weighting experiment produced a null result, but it was an *informative* null result. It told us that the bottleneck was not the training data distribution—it was the target variable and the feature set. This redirected months of effort from data engineering to the two problems that actually mattered.

3 The Target Variable Problem

This was the most time-consuming and consequential design decision in the entire project. The target variable defines what the model learns to predict, and in e-commerce ranking, **every natural choice is wrong in a different way.**

3.1 Click Prediction

The most common target in web search ranking is click-through rate (CTR). We tried it. The model optimized for clicks, which meant it promoted items with attractive images, compelling titles, and low prices—items that users clicked on but did not buy. Click prediction **reduced revenue and did not improve sales**. Users clicked more but purchased at the same or lower rate. The model had learned to maximize engagement, not transactions.

3.2 Sales Prediction

The next natural target was purchase probability: predict whether an item will be sold given a query. This produced a model that **systematically promoted cheap items**. The reason is straightforward: cheaper items have higher conversion rates (lower purchase friction), so a model trained on binary sales outcomes learns that low price is the strongest predictor of purchase.

The resulting ranking increases unit sales but decreases revenue—the opposite of the business objective.

3.3 Price-Normalized Sales

The solution was to normalize the sales target by price. We experimented with several normalization functions and found that dividing by the square root of price, with a cutoff to prevent extreme values, produced the best results:

$$y_i = \frac{1[\text{sold}_i]}{\sqrt{\min(p_i, p_{\text{cap}})}}$$

The square root provides a concave transformation that penalizes cheap items without completely eliminating the price signal. A linear normalization (y/p) over-penalized cheap items and promoted expensive items that rarely converted. The square root struck the right balance: the model still preferred items likely to sell, but not at the expense of promoting low-value transactions.

3.4 Weighted Multi-Objective

Even price-normalized sales was insufficient as a standalone target. We found that **weighting sales and add-to-cart signals together** produced the best overall ranking. Add-to-cart is a stronger intent signal than a click but occurs earlier in the funnel than a purchase, providing a denser training signal. The final target combined:

- Price-normalized sales (highest weight)
- Price-normalized add-to-cart events (moderate weight)
- Click signals were **excluded** from the target—including clicks degraded both revenue and sales, or at best matched the Best Match baseline

This target variable design was the single most important technical decision in the project. It took months of experimentation to reach, and there was no theoretical framework that predicted it. It was discovered empirically through systematic ablation.

4 The Feature Engineering That Won

4.1 Why the Heuristic Was Hard to Beat

Best Match had been tuned by product managers over years. Some of its features encoded marketplace knowledge that was not present in standard ML feature sets: seller behavior patterns, seasonal inventory dynamics, category-specific quality signals, and interaction effects between price, shipping, and seller reputation.

The breakthrough came when we **systematically reverse-engineered the heuristic features and converted them into ML features**. Rather than treating Best Match as a black box to be replaced, we treated it as a **feature engineering oracle**—a curated set of signals that domain experts had identified as predictive over years of observation. Each heuristic rule was decomposed into its constituent signals, and those signals were included in the ML feature vector.

This is a general principle: **when a heuristic is hard to beat, the heuristic contains information your model lacks.** The correct response is not to build a more powerful model—it is to extract the information from the heuristic and give it to the model.

4.2 Feature Categories

The final feature vector comprised signals at multiple granularities:

- **Query-level features**: Query frequency, query specificity (number of tokens, category entropy), historical conversion rate for the query.
- **Item-level features**: Price, price relative to category median, review count, review score, seller rating, listing age, image quality signals.
- **Seller-level features**: Historical sell-through rate, response time, return rate, feedback score distribution (not just average).
- **Category-level features**: Category-specific conversion baselines, seasonal demand patterns, average price and price variance within category.
- **NLP features**: Token-level matching between query and item title, attribute extraction, brand recognition.
- **Bayesian word-level models**: For each word appearing in item titles, we built a separate Bayesian model estimating the likelihood of a sale given that word's presence. These models captured the empirical association between specific vocabulary and purchase behavior—for example, the word "new" in certain categories was a strong positive signal, while in others it was neutral. These word-level posterior probabilities were fed as features into the main ranking model.
- **Real-time behavioral signals**: Current-session clicks, add-to-cart events, impressions, and sales aggregated at the item, query, category, and word level. These signals were computed from streaming data and updated continuously, providing the model with information about what was selling *right now*, not what sold last week.
- **Converted heuristic features**: The explicit rules and interaction features from Best Match, decomposed and included as structured features rather than as a competing ranking signal.

4.3 The Bayesian Word-Level Model

This component deserves additional detail because it illustrates a pattern that recurs across marketplace ranking systems. Standard NLP features capture whether a query term matches an item title. The Bayesian word-level model goes further: it estimates, for each word w in the title vocabulary:

$$P(\text{sale} \mid w) = \frac{P(w \mid \text{sale}) \cdot P(\text{sale})}{P(w)}$$

These posterior probabilities, computed offline and updated daily, converted raw text into a dense set of "commercial value" features. A title containing words with high $P(\text{sale} \mid w)$ received a boost independent of query matching. This captured a dimension of item quality—commercial viability—that relevance-based features alone could not express.

5 The Model

After extensive experimentation with logistic regression (with and without interaction terms), pairwise ranking models, and various ensemble methods, **gradient boosted trees (GBT)** produced the winning model. The GBT model's advantage was its ability to automatically discover nonlinear interactions between features—precisely the kind of complex feature interactions that Best Match's product managers had hand-coded.

The model took as input:

- Real-time behavioral signals (sales, clicks, add-to-cart, impressions, price)
- Aggregated statistics at item, query, category, and word level
- Bayesian word-level sales probabilities
- Converted heuristic features from Best Match
- Standard query-item matching features

The predicted scores were converted to a ranking, and the system was deployed behind an A/B testing framework that measured both sales and revenue impact simultaneously.

5.1 Production Serving: GBT Model Compilation to C

Training the GBT model at scale required specialized infrastructure. The training data comprised billions of impression-level records—every query–item–outcome tuple from months of search traffic, spanning clicks, purchases, add-to-cart events, price, and contextual signals. Standard machines could not hold the full training set in memory; we used large-memory clusters to fit the complete feature matrix during training.

The resulting GBT ensembles were large—thousands of trees with deep splits—because model expressiveness was critical. The non-linear interactions between price, behavioral signals, Bayesian word-level posteriors, and converted heuristic features required deep trees to capture. Smaller models or shallower ensembles consistently lost to Best Match in A/B tests; the interaction effects that made the ML model finally win were precisely the ones that required model complexity.

This created a production serving problem. The ranking system sat in the critical path of every search request: each query required scoring hundreds of candidate items in real time. Standard model serving introduced unacceptable latency through tree traversal, model deserialization, and interpretation overhead.

We built a **custom compiler that translated the trained GBT model into a native C program**. The compiler unrolled the entire tree ensemble into a sequence of optimized branching if-else statements, eliminating all interpretation overhead. The generated C code was compiled with aggressive optimization flags, producing a scoring function with predictable memory access patterns and favorable branch prediction characteristics. Key properties of this approach:

- **Bit-exact fidelity**: The compiled C code produced identical scores to the original model—no approximation, distillation, or quantization was involved. This was critical: any deviation between offline evaluation and production scoring would have invalidated A/B test results.
- **Sub-millisecond per-item scoring**: Eliminating interpretation overhead reduced scoring latency by an order of magnitude compared to standard model serving frameworks, enabling real-time ranking across the full candidate set.
- **Decoupled model complexity from serving constraints**: The science team could iterate on model quality—adding features, deepening trees, expanding training data—without being constrained by production latency budgets. This was essential because each iteration of the target variable and feature experiments required full production A/B testing to validate.

This infrastructure was not incidental to the project's success. The years of experimentation described above—target variable iterations, feature engineering, negative sampling experiments—each required deploying a new model to production and measuring live impact. A serving layer that could handle arbitrarily complex GBT models without degrading search latency was what made this iterative experimental cycle feasible at production scale.

6 What We Tried That Did Not Work

Negative results are as informative as positive results. The following approaches failed:

Approach	Why It Failed
Click prediction target	Promoted clickbait; increased clicks but reduced or maintained sales; reduced revenue
Raw sales prediction	Promoted cheap items; increased unit sales but decreased revenue
Hard negative weighting	No measurable improvement over uniform negative sampling; feature representation already sufficient
Pairwise ranking alone	Marginal initial advantage disappeared as feature set matured
Including clicks in target	Degraded both revenue and sales, or at best matched Best Match
Logistic regression with interaction terms	Could not capture the non-linear feature interactions that GBT discovered automatically
ML features without heuristic features	Lost to Best Match; the heuristic encoded domain knowledge the model could not rediscover from data

Table 1: Approaches that failed and the reasons for failure.

7 Generalizable Principles

1. Target design is the real problem in marketplace ranking. Model architecture receives disproportionate attention in the literature. In practice, the choice of what the model predicts—and how different objectives are weighted—determines success or failure. There is no theoretical framework for this; it must be discovered empirically through systematic experimentation. Budget months for it.

2. When a heuristic is hard to beat, the heuristic contains information your model lacks. The correct response is not a more powerful model—it is feature extraction from the heuristic. Decompose the heuristic into its constituent signals and include them in the ML feature vector. The model's job is to learn better weights and interactions, not to rediscover domain knowledge from scratch.

3. Position bias corrupts training data silently. Sampling from the existing ranking system's results creates a model that replicates the existing ranking. Alternative ranking surfaces (price-sort, random, time-based) provide less biased training signal at the cost of sparser data. The tradeoff is worth it.

4. Negative sampling ratios matter less than feature quality. We spent significant time optimizing the positive-to-negative ratio and hard negative weighting. The marginal value of these optimizations was small compared to the value of better features and better target design. When in doubt, use a reasonable ratio (1:30) and invest the saved time in features.

5. Clicks are dangerous as ranking targets. Click prediction is the default target in web search and a natural first choice in e-commerce. In marketplaces, it is actively harmful: it promotes engagement over transactions, and the resulting ranking degrades the metrics that matter (sales, revenue). Click data is useful as a *feature* (real-time click aggregates), not as a *target*.

6. Price normalization of sales targets is non-trivial. Linear normalization (y/p) overcorrects. No normalization promotes cheap items. The square root transform with a cap struck the right balance in our setting, but the optimal function depends on the price distribution and category structure of the marketplace. This should be treated as a hyperparameter search problem, not an engineering decision.

7. Real-time signals unlock performance that batch features cannot reach. The same

model with daily-updated features versus real-time features produces materially different results. What is selling *right now* is a stronger signal than what sold last week. Invest in the streaming pipeline.

8. Bayesian word-level models bridge NLP and commerce. Standard NLP features capture relevance; word-level sales posteriors capture commercial viability. These are orthogonal signals. The combination gives the model a richer understanding of both *what the user wants* and *what is likely to sell*.