Project brief: Fixing MXT.co’s product recommendations

Background

Mxt.co is a skincare products ecommerce site that recommends products once a user has completed a diagnostic quiz (see mxt.co - use password mixed4you). The tech stack is a Shopify Plus store with a custom-built quiz app. The quiz app is connected to an Airtable database that lists each question in the quiz and each possible answer and has product recommendations attached to each answer.

Current state: The selection process works this way: users take a quiz about their skin and lifestyle (the quiz has a total of 65 possible questions and 307 possible answers, but due to branching logic, the average users answers about 20). 2. About 500 quiz results have been linked to a selection of products (there are 18 total products) that are the best skin treatment for that quiz respondent. The selections are grouped into 5 categories: Primary 1, Primary 2, Primary 3, and Upsell 1, Upsell 2. Thus for each of these 500 quiz results (the "seed data") 5 products have been linked (recommended), one in each category. The categories depend on the role a product has in the skincare routine, Primary 1 is a moisturizer or cream, Primary 2 is always and only a cleanser, Primary 3 is usually a serum or cream. Upsell 1 and Upsell 2 can be any type of product. These quiz results are then linked to each quiz answer. ***The more times a product shows up as linked to an answer, the higher recommendation score it gets***. When a new user completes a quiz, ***the system counts the number of times each product name appears linked to a quiz answer, and recommends the highest scoring product in each of the five categories.***

As discussed below, this 1-1 voting system does not produce reliable results and needs to be revised so that new user outcomes (recommendations) more accurately match the seed data recommendations. It can be achieved either by replacing the vote counting system with algorithms that perform the calculations described below, or by creating an AI agent that performs the calculations on the fly.

Project Summary

Goal: Replace the current “1-answer = 1 vote” recommender with a system that reproduces seed recommendations more faithfully and scales to new users. Two viable paths:

- A deterministic statistical recommender using weighted signals and per-category prediction.
- An AI agent that reasons over nearest seed results with guardrails and outputs five products.

Success: Significantly higher agreement with seed data per category, clearer explanations, stable tie-breaking, and production-ready integration with Shopify + custom quiz + Airtable.

Background and current state

Stack: Shopify Plus, custom quiz app, Airtable (quiz graph, answer options, product mappings).
Quiz: 65 possible questions, 307 answers; typical user answers ~20 due to branching.
Products: 18 total. Five output slots per user: Primary 1 (moisturizer/cream), Primary 2 (cleanser only), Primary 3 (usually serum/cream), Upsell 1, Upsell 2 (any type).
Seed data: ~500 fully labeled quiz outcomes where a complete set of five products is curated.
Current algorithm: For a new user, count how often each product is linked to the answers they selected across Airtable; pick the highest count per category.

Diagnosis: Why the 1–1 vote system underperforms

Equal weighting: Every answer contributes equally even if some answers are ubiquitous or weak predictors; rare but highly predictive answers get drowned out.
No conditioning by category: Votes aren't learned per category context (e.g., what predicts cleansers vs moisturizers can differ).
Frequency bias and popularity bias: Answers linked to popular products inflate counts (rich-get-richer).
Ignores interactions: Important combinations (e.g., "very sensitive" + "retinoid use") aren't modeled; single-answer votes miss cross-effects.
Sparse overlap: With branching logic, many users share few answers with seeds; raw counts become noisy and unstable.
Poor calibration and tie-breaking: Ties and close scores lead to arbitrary picks; no uncertainty handling or fallback.
Drift and maintainability: As products or quiz logic change, manual link maintenance is brittle.

Objectives

Primary: Maximize agreement with seed labels for each category (P1, P2, P3, U1, U2).
Secondary: Provide interpretable "why" explanations for chosen products; deterministic, testable logic; easy to maintain and extend as seed data grows.
Non-goals (for now): Deep personalization beyond quiz inputs; collaborative filtering; long-horizon causal A/B uplift.

Data and constraints

Labeled examples: ~500 seed outcomes; limited sample size suggests regularized, simple, interpretable models.
Inputs: Binary indicators of answers selected, possibly question-level context; product taxonomy (product type, allowed categories).
Outputs: One product per category with category constraints (e.g., P2 must be cleanser).
Guardrails: Respect product-type constraints; allow business rules (e.g., contraindications if applicable), inventory availability, and price bands if required.

Proposed solution A: Deterministic statistical recommender High-level

Treat recommendation as five constrained multiclass predictions: one model per category chooses one product from allowed product set for that category.
Build three complementary components and ensemble them with calibrated weights:
- Weighted naive Bayes-like scorer with TF-IDF and Bayesian smoothing.
- Regularized multinomial logistic regression per category.
- Nearest-neighbor retrieval over seed outcomes with a sparse similarity metric.

Details

1. Weighted probabilistic scorer (baseline, interpretable)
   For category c and product p: Score_c(p | user U) = Prior_c(p) + Σ over answers a in U of w_a * log Odds_c(p | a)
   Weights:
   - w_a = idf-type weight = log((N + 1) / (n_a + 1)) to downweight common answers, scaled to [0,1].

   Odds estimation:
   - Use Laplace/Dirichlet smoothing: P_c(p | a) = (count of seeds where category c = p and a present + α) / (count of seeds where a present + α|P_c|)
   - Prior_c(p) = log((count of p in c + β) / (total in c + β|P_c|))

   Output: For each category, pick argmax Score_c with category-legal products only.
   Pros: Simple, fast, human-readable attribution via per-answer contributions.

   2. Regularized multinomial logistic regression per category

   Inputs: One-hot answers; optionally engineered interactions for a short list of clinically important pairs (e.g., oiliness x acne severity).
   Model: Softmax with L1/L2 regularization; class weights if imbalanced.
   Output: P(y_c = p | answers). Choose top-1 per category; calibrate with isotonic or Platt scaling.
   Pros: Captures interactions via coefficients; better calibration than counts.

   3. kNN over seed outcomes

   Represent each seed quiz as a sparse vector of answers; compute similarity to a new user:
   - Jaccard similarity (binary) or cosine on TF-IDF answer weights.

   Aggregate the top-K nearest seeds' labels per category with distance-weighted votes; apply smoothing.
   Pros: Strong when new users closely resemble existing seeds; robust fallback if models underfit.

Ensemble and tie-breaking

- Combine the three per-category scores with learned weights via cross-validation, or use a simple rank aggregation.
- Deterministic tie-breakers: product priority list per category; prefer higher calibrated probability; then business rules (inventory, margin); then stable alphanumeric.

Explanations

- Return per-category top reasons: top contributing answers and their weight contributions, seed neighbors used, and model confidence.

Edge cases

- Very few answers: fall back to prior + NB scorer.
- New answers/products: default to prior; easy to update with nightly retrains.
- Category constraints: enforce whitelist (e.g., only cleansers eligible for P2).

Proposed solution B: AI agent recommender with retrieval and guardrails High-level

- Retrieval-augmented agent that:
    - Encodes user answers and retrieves the K most similar seed quizzes.
    - Reads product taxonomy and any clinician rules.
    - Generates five product choices with justifications.
    - Validator enforces constraints and resolves IDs deterministically.

Architecture

- Vector store: TF-IDF or sentence embeddings over answer text; store seed quizzes with their five-product labels.
- LLM prompt:
    - Provide user answers, top-K seed exemplars with labels, product catalog and constraints.
    - Instruct to output JSON with five product IDs, per-category, and short rationale.
- Validator:
    - Check category legality, uniqueness, inventory; if invalid, correct using deterministic backup (kNN or NB score).
- Pros: Fast to iterate, rich explanations.
- Cons: Hallucination risk, non-determinism without strict constraints, heavier runtime. Best with a validator and caching.

Evaluation plan

- Offline backtesting on seed data with stratified K-fold CV:
    - Per-category top-1 accuracy.

    - Exact 5-of-5 set match rate.
    - Hamming loss across five slots.
    - Mean reciprocal rank of the seed product within model ranking (top-3, top-5 recall).
    - Calibration (Brier score) for probabilistic models.
- Business proxies:
    - Agreement uplift vs current system.
    - Stability: tie frequency, variance across runs.
- Online A/B (post-launch):
    - Add-to-cart rate for recommended products.
    - Conversion lift, AOV, return rate.
    - User satisfaction signals (clickthrough on "why we recommended this").

Data work and features

- Export clean seed dataset: binary answer matrix per user, five labeled products.
- Product taxonomy: map product IDs to allowed categories; mark exclusivities and contraindications if any.
- Feature engineering: answer idf weights; optional curated interaction pairs.
- Logging: store recommendation inputs, scores, chosen outputs, and explanation traces.

Implementation plan and timeline (indicative) Week 1

- Data audit and schema freeze; confirm allowed products per category and any medical/business rules.
- Baseline reproduction of current vote system and metrics.

Weeks 2–3

- Build NB scorer with smoothing + TF-IDF; per-category logistic regression; kNN retrieval.
- Cross-validated evaluation; select ensemble weights.
- Explanation layer and deterministic tie-breaking.

Week 4

- Package as a stateless inference API (FastAPI) with model artifact store; latency target <100 ms per request.
- Staging integration with quiz app; Airtable sync for taxonomy and priors.
- Optional: AI agent prototype behind a feature flag with validator.

Week 5

- QA, load test, and guardrail checks.
- Launch controlled A/B (10–20% traffic); monitoring dashboards.

Deliverables

- Model artifacts: per-category models, priors, idf weights, kNN index.
- Inference service API:
    - Input: user_id, list of answer_ids.
    - Output: products_by_category, per-product scores, top features, confidence.
- Integration shim for Shopify/quiz app; Airtable sync scripts.
- Documentation: data dictionary, training pipeline, troubleshooting, and model cards.
- Offline evaluation report; online experiment plan.
- Optional: AI agent + validator module and prompts.

Acceptance criteria

- Offline: +15–25% relative improvement in per-category accuracy vs current system and at least 10% absolute uplift in exact 5-of-5 match rate on seed CV.
- Deterministic, explainable outputs; tie frequency reduced by 80% vs baseline.
- P95 latency <200 ms; >99.9% availability.
- No category-constraint violations in 10k test cases.

Integration notes (Shopify/Airtable)

- Keep Airtable as the source of truth for:
    - Answer catalog and deprecations.
    - Product taxonomy and category eligibility.
    - Optional business rules and product priors.
- Nightly retrain job reads Airtable exports and seed labels from your DB; writes back summary stats (feature weights) for transparency.
- Quiz app calls the inference API after quiz completion; API returns the five products plus explanations to render on PDP/quiz results.

Risks and mitigations

- Small dataset: prefer regularized, interpretable models; use kNN to leverage seed similarity; add smoothing.
- Drift when quiz changes: version features and store mappings by quiz version; retrain on changes.
- LLM variability (if using agent): force JSON schema, use deterministic temperature, add validator and fallbacks.
- Business changes (inventory, new SKUs): keep priors and eligibility in Airtable for instant updates without retrain.

Open questions

Are there hard clinical rules (e.g., pregnancy-safe, avoid acids for high sensitivity) we must enforce?
How are products mapped to categories today, especially for Upsell slots?
Do we want diversity constraints (avoid recommending two near-duplicate moisturizers)?
Tie-breaking preferences: price band, margin, brand priorities?
What is the acceptable trade-off between exact seed match vs business objectives (e.g., margin)?
Any PII or compliance constraints for quiz data storage and logging?

Recommended path

Implement Solution A first (deterministic ensemble: NB + logistic regression + kNN). It will likely deliver strong accuracy gains and robust explanations with minimal runtime complexity. If richer narratives are desired on the results page, add Solution B’s AI agent as an explanation generator only, while the deterministic system selects the products; keep the validator in place for safety.