PERSONALIZED PROMOTIONS

Rona Fang

Manager: Doug Ngo

Mentor: Bryce Colson

BUSINESS PROBLEM

- Missing out on orders from potential customers who are already visitors
- Obtaining a store visitor is expensive: average Google cost per ad click > $2.
- Ecommerce average: 2-3% conversion rate
  - 1% conversion increase → 40% increase in orders
- A merchant's ads are scalable if they are profitable: increasing ROI on ad-spend leads to their exponential growth.
  - Increasing conversion rate = increasing profit per visitor = increasing return on per ad spend

Let's increase conversion rate for any BwP store, and make our merchants even more profitable.

WHY DOESN’T A CUSTOMER PURCHASE?

- No urgency: “I could buy this anytime”
- No desirable product: “I don’t want this enough”
- Attention lost: average visitor only spends ~45 seconds on a store

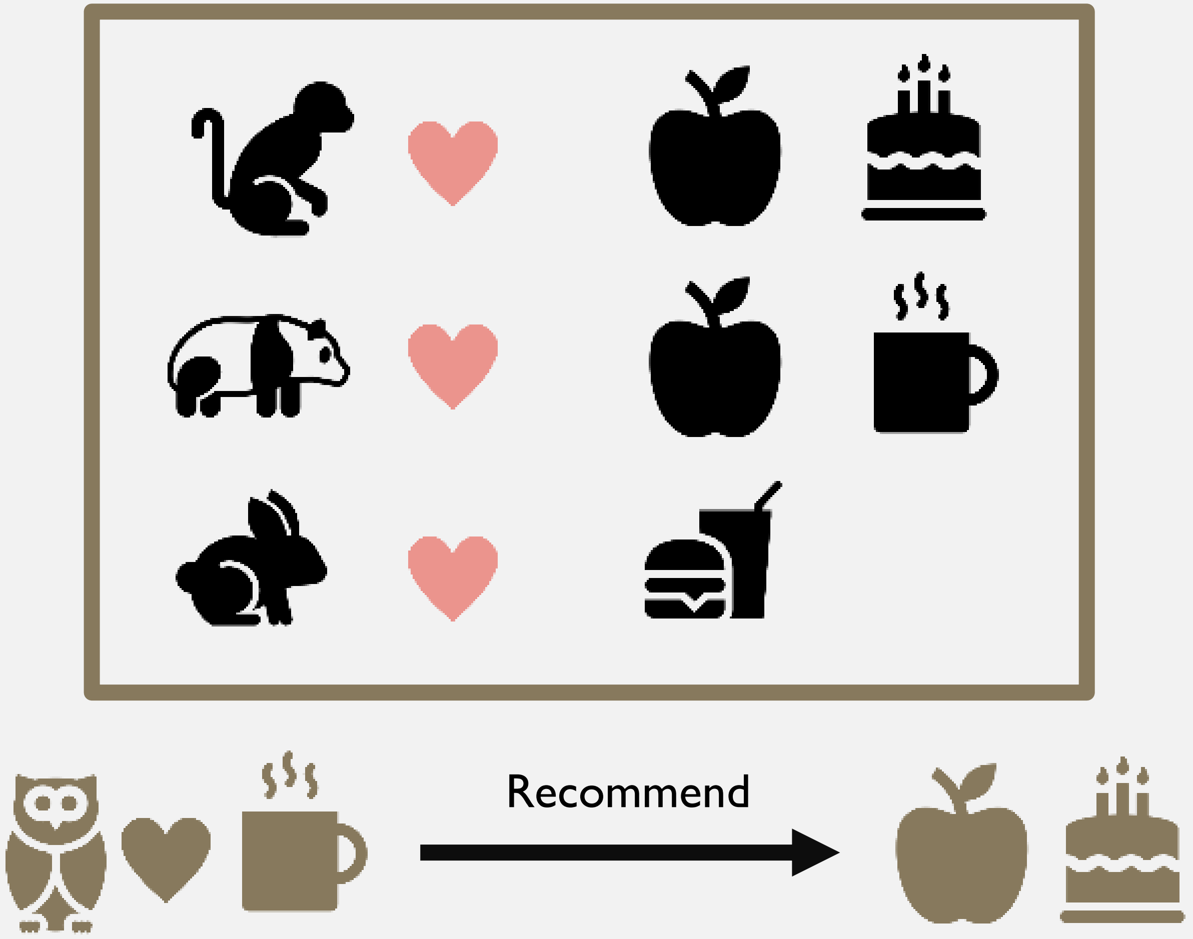
We want to show a visitor what they want, and give them an incentive that leads to a purchase.

WHAT CAN WE DO?

- Product Recommendations
  - For a customer, based on their past purchases, add-to-carts, products clicked
- Promotion/Coupon suggestions
  - For a customer, based on their store activity frequency and historical coupon usage
- Machine learning!
  - Personalized recommendations powered by machine learning

ML STRATEGIES

Collaborative Filtering



Regression

4	2		10
3	5	→	11
9	0		19
3	5		10

Training output:Y = 2A + B

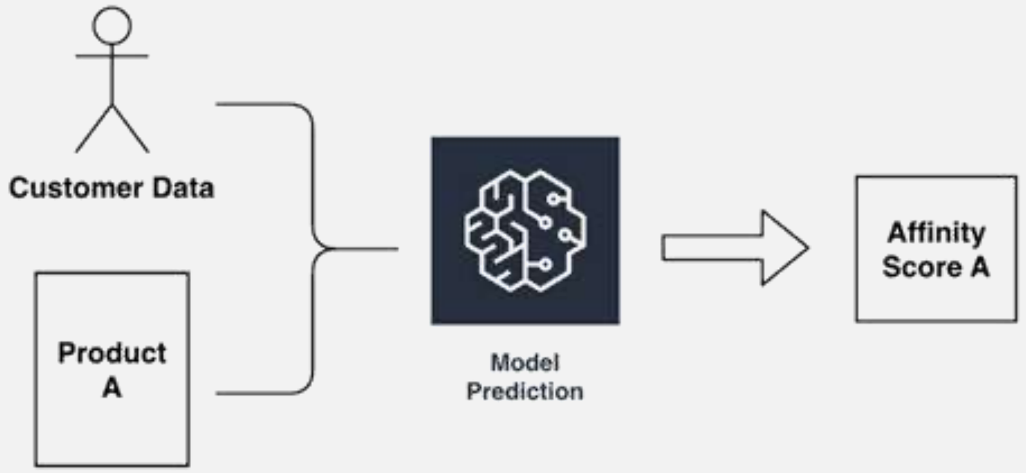
HOW ML WORKS

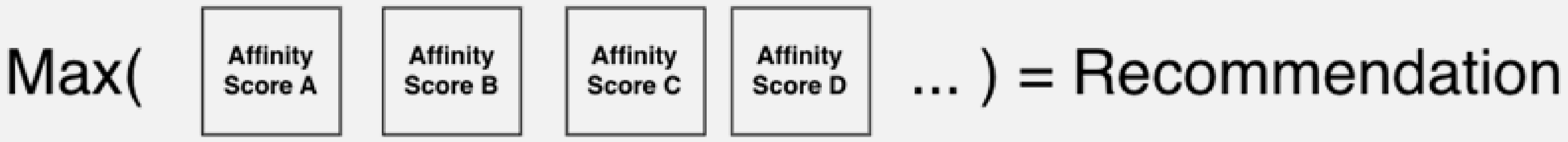
- What is ML? *The more data we give it, the better predictions it gives*
- We train ML models with preferences and behavior of customers
- In training, ML model learns and assigns weightings from various customer behaviors to various preferences/recommendations
- We want to name any customer, and it gives us the recommendation (products, or a coupon) for that customer, based on similar customers it has seen in training

PRODUCT RECOMMENDATIONS

Customer Data:
- List of purchased products
- List of products added to cart
- List of products clicked
- Average order value

Product:
- Sku





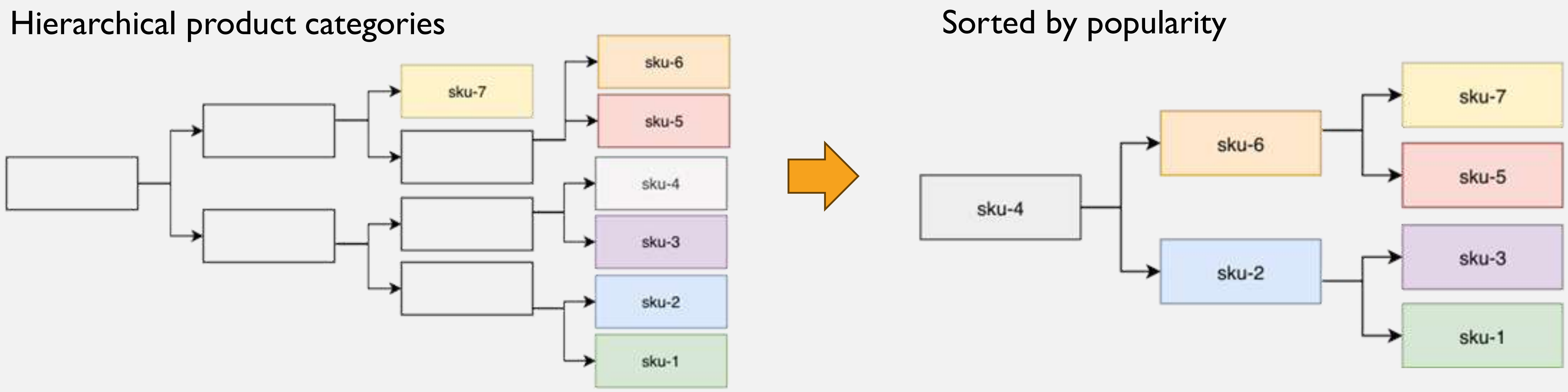
PRODUCT RECOMMENDATIONS CONT.

- We don’t want to try all products in a merchant’s catalog to create recommendations for 1 customer, in real time

- Let’s binary search for the best product
  - Need to sort our products
  - Sort products based on metrics that describe product
  - Let’s create a category tree that leads to sub categories or products

- We previously described customers by the products they purchased → We can describe products by which customers purchased the product, as a way to sort the products

PRODUCT PRE-PROCESSING



If Affinity Score > Affinity Score and Affinity Score > Affinity Score , Max(Affinity Score Affinity Score Affinity Score) = Recommendation

PRODUCT RECOMMENDATIONS CONT.

- Most popular product recommendations are structurally built into the algorithm
- We can suggest recommendations from other product categories
- We can suggest recommendations similar to other products
- Still only need one row in database per product, with added field of left and right product ID
- Preprocessed product tree lowers time complexity of real time prediction = smarter recommendation for the same compute, cost, and time
- More details documented here: formulating business problem into a numbers solution

BEST PROMOTION

- Automatically suggest a coupon for a customer
- Optimized for 10-week profit
- If a coupon leads to a purchase but the customer never returns without a big coupon, this is bad
- If a customer would have purchased with a 10% coupon, we do not want to offer a 20% coupon
- If a non-customer becomes converted to a customer, a temporary loss may lead to future profit
- We use ML to figure out the numbers

BEST PROMOTIONS ML DATA SCHEMA

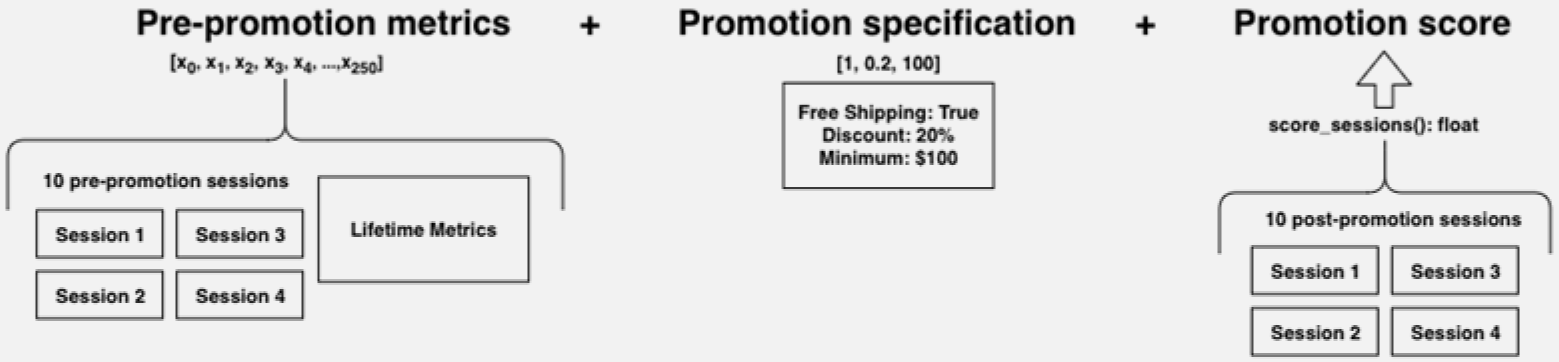
- Customer Data
  - Lifetime order value, count
  - Recent 10 store sessions
- Store Session
  - Order value, count
  - Add to cart value, count
  - Click count
  - Promotions available
  - Promotions used
  - Time since last session

- Promotions
  - Free shipping?
  - Discount amount
  - Discount minimum

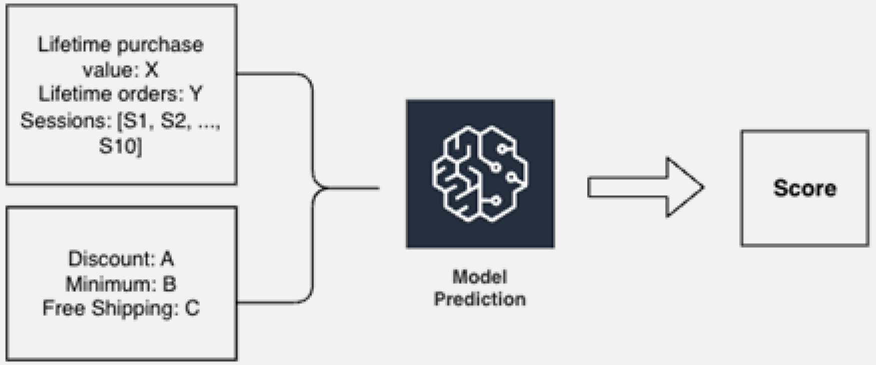
- Promotion Scoring: Number
  - Based on 10 post-promotion customer sessions

BEST PROMOTIONS CONT.

Training data row:
For a historical customer
+ promotion interaction



Getting a prediction:
For a customer +
arbitrary promotion



API ENDPOINTS

/product-recommendations

- Give merchant and customer ID, get product recommendations

/best-promotion

- Give merchant and customer ID, get recommended promotion

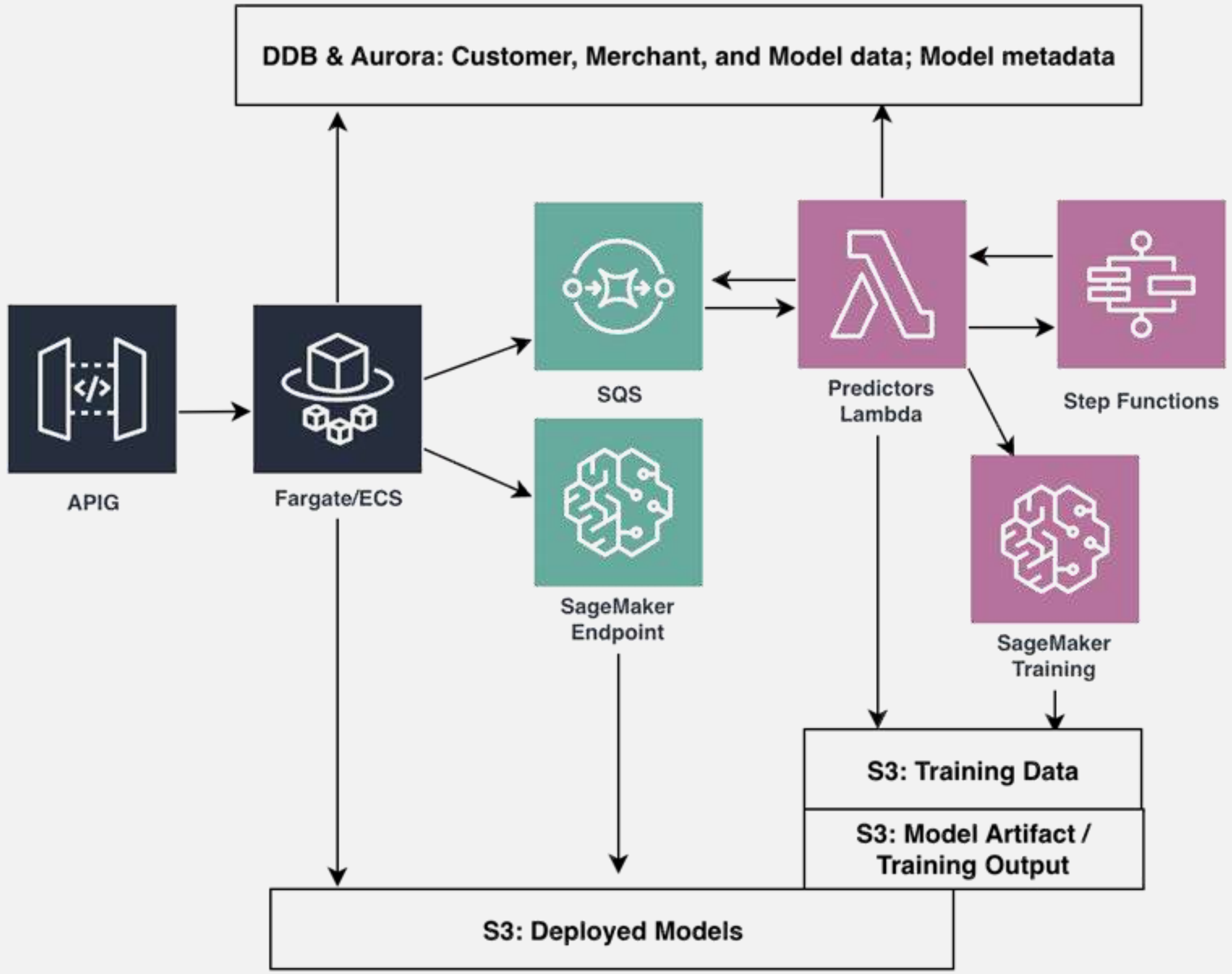
/predictors

- Give merchant ID, and ML models will be created/updated
  - This is the only prerequisite step to use the recommendation endpoints

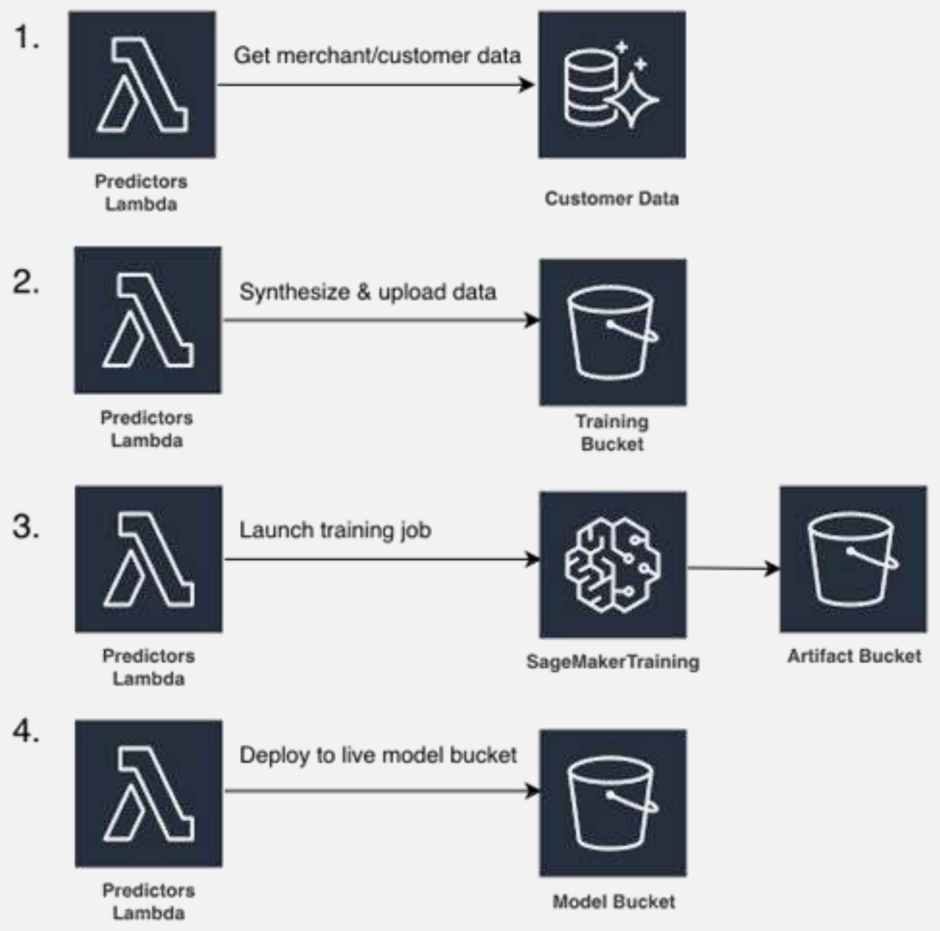
USE CASES FOR RECOMMENDATIONS API

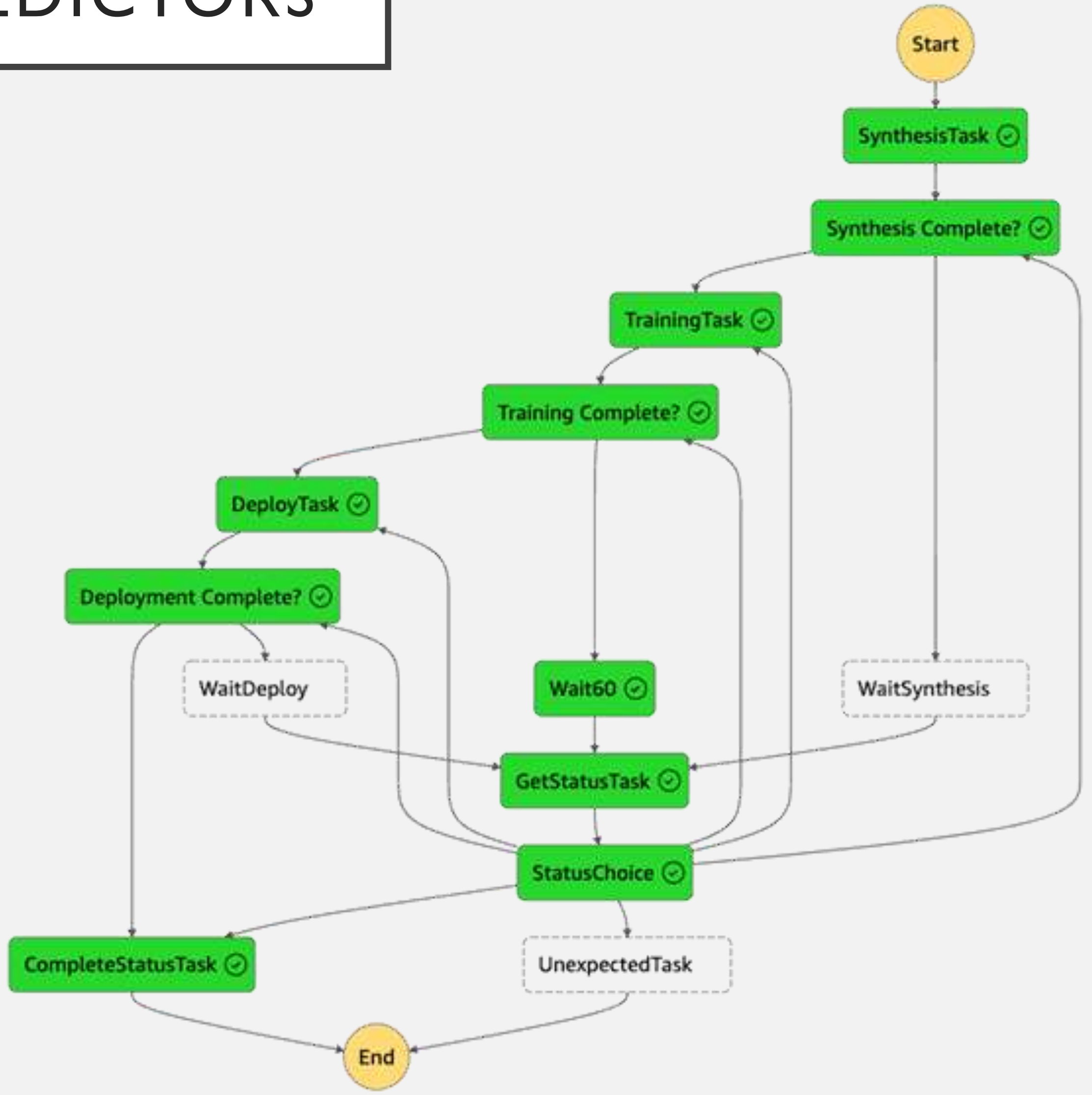
- Promotional emails
  - ”We think you’d like these” for BwP enabled products only
  - Better deal given in abandoned cart emails
- Additional recommendations at checkout
  - Perfect for products only enabled for BwP
- Shopify widget: stores can add product recommendation carousel
- Since this is an API, it can be integrated easily to anything

ARCHITECTURE



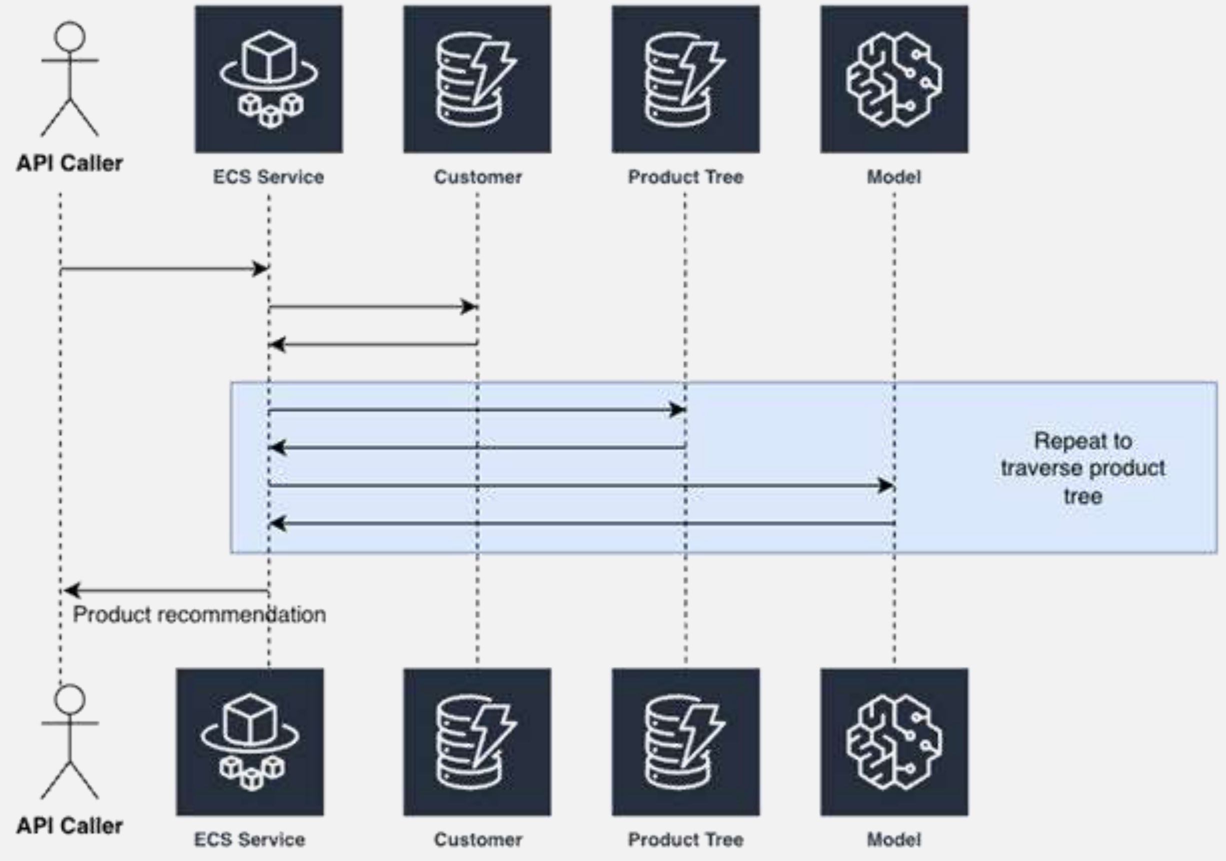
CREATING MODELS: POST /PREDICTORS





SERVING PREDICTIONS

1. Receive request
2. Get customer data
3. Get model identifier based on merchant
4. Get predictions from SageMaker endpoint
5. Return best prediction



DEMO

NEXT STEPS

- Caching real time customer browsing
- Pagination of results
- Balancing exploration of options vs returning only the best
- Visualization, metrics, and dashboards of suggestion impacts
- Optimizing recommendation feedback (ML)
- Infinite generation of results

KEY LEARNINGS

- Business problem → ML problem → Software solution
- First time using ML beyond a GPT wrapper
- First time using AWS beyond a S3 bucket
- Trying to design the systems involved
- Working in the Amazon environment

HINDSIGHT REFLECTIONS

- DLQ->SQS configuration
  - Avoid random missed messages from concurrent SQS messages when Lambda reserved concurrency is met. SQS retries failing → we want to set retry for DLQ to SQS instead of only SQS to Lambda
- Build tree traversal into ML docker image
  - Initially, using built in image, we had to alternate between SageMaker EC2 output and our own service logic
  - Ended up needing to customize our own docker image with predict function, we could directly put the business logic in the docker image, which gets rid of the latency between EC2 invokes.

THANKS!

Further links if you're interested

HLD: https://quip-amazon.com/aHgXAiW3obJX/Personalized-Promotions-HLD-V2

LLD: https://quip-amazon.com/U6IBA3f9Aaxx/Personalized-Promotions-LLD-V2

ML & Data: https://quip-amazon.com/bk02AacUavBi/Personalized-Promotions-Data-ML