
#2 Aria
#18 Vdara
#11 Paris
#38 Encore
#13 Cosmopolitan
#19 Hollywood
#22 Westgate
#1 MGM
#24 Bally's
#6 New York New York
#21 Tropicana
#20 Excalibur
Learning to Rank
with Eric
#14 Luxor

Agenda

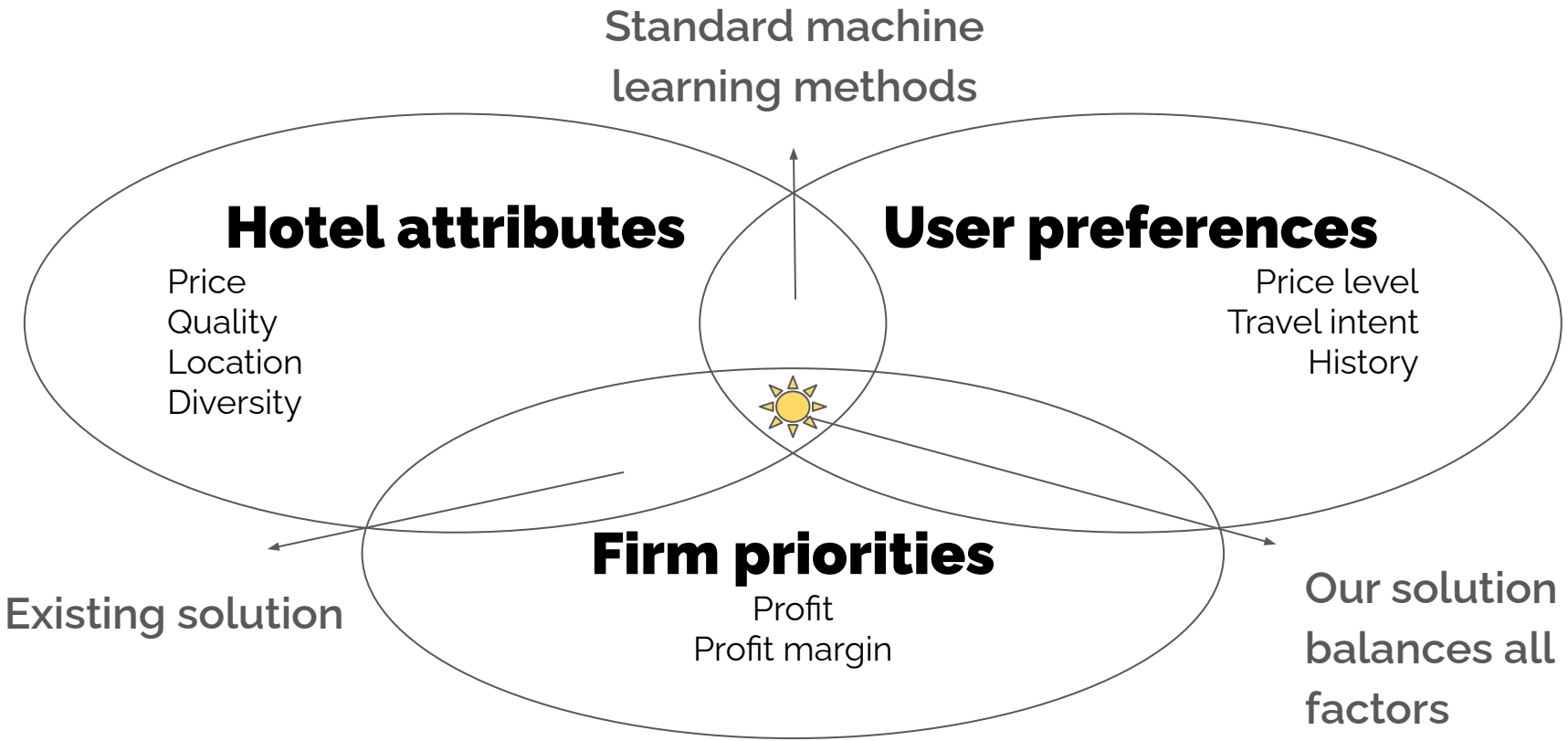
1. Introducing the problem
2. Evaluating performance
3. Interpreting the model
4. Designing the machine
5. Deploying to production

Historical attempts have not succeeded

$$\text{Points} = w_1 * \text{stars} + w_2 * \text{rating} + w_3 * \text{price} + w_4 * \text{spread} + \ldots$$

app_id	experiment_name	start_dt	ttm_lift
hotelstorm-www	Config:hs.exp.search.ranking_algo_points	Nov 2017	-0.06992
hotelstorm-www	Config:hs.exp.search.ranking_algo_points	Jan 2018	0.01064
hotelstorm-www	Config:hs.exp.search.ranking_algo_points	Feb 2018	0.0141
ALL-whitelabel-web	Config:testarossa.searchResults.rank.weights	Feb 2018	0.00466
hotelstorm-www	Config:hs.exp.search.ranking_algo_points_v2	Mar 2018	-0.03244
hotelstorm-www	Config:hs.exp.search.ranking_algo_points_v3	Apr 2018	0.01407

Ranking is a complex problem!



Our solution balances all factors

$$\sqrt{\text{profit} * \text{profit margin}}$$

TTM + Overrides (PO customizable)

$$\text{Ranking Score} = Relevance * \frac{Profit}{\sqrt{Revenue}}$$

Probability user picks this result, given the user picks a result

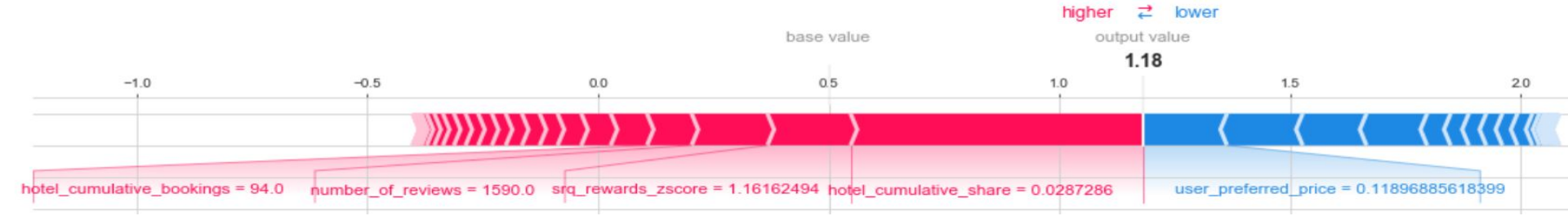
Published price

What are the drivers of hotel relevance?

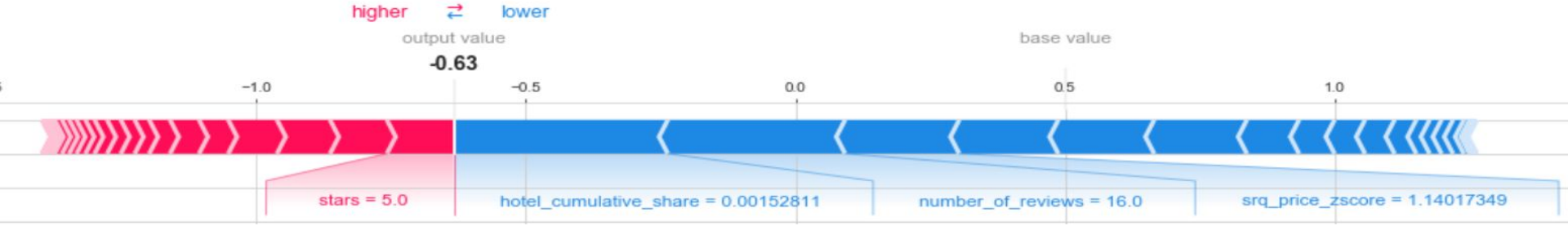
Credit: Matteo Colombo/Getty Images

Force plots reveal the effect of every feature

Grand Fiesta: High ranked because it's popular with decent rewards



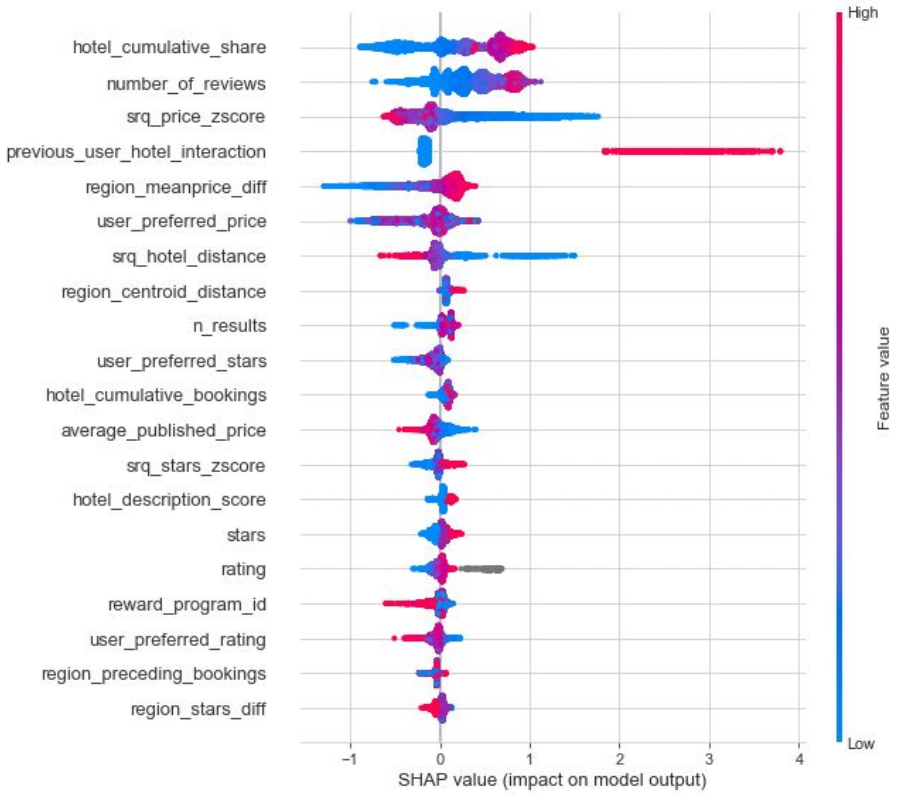
Sunset Royal: Low ranked because no history, reputation, and a high price



Every data field is credited with its contribution to the final prediction.

This credit is the data field's **SHAP value**.

Violin plots summarise aggregate effects



Previous performance

Relative price

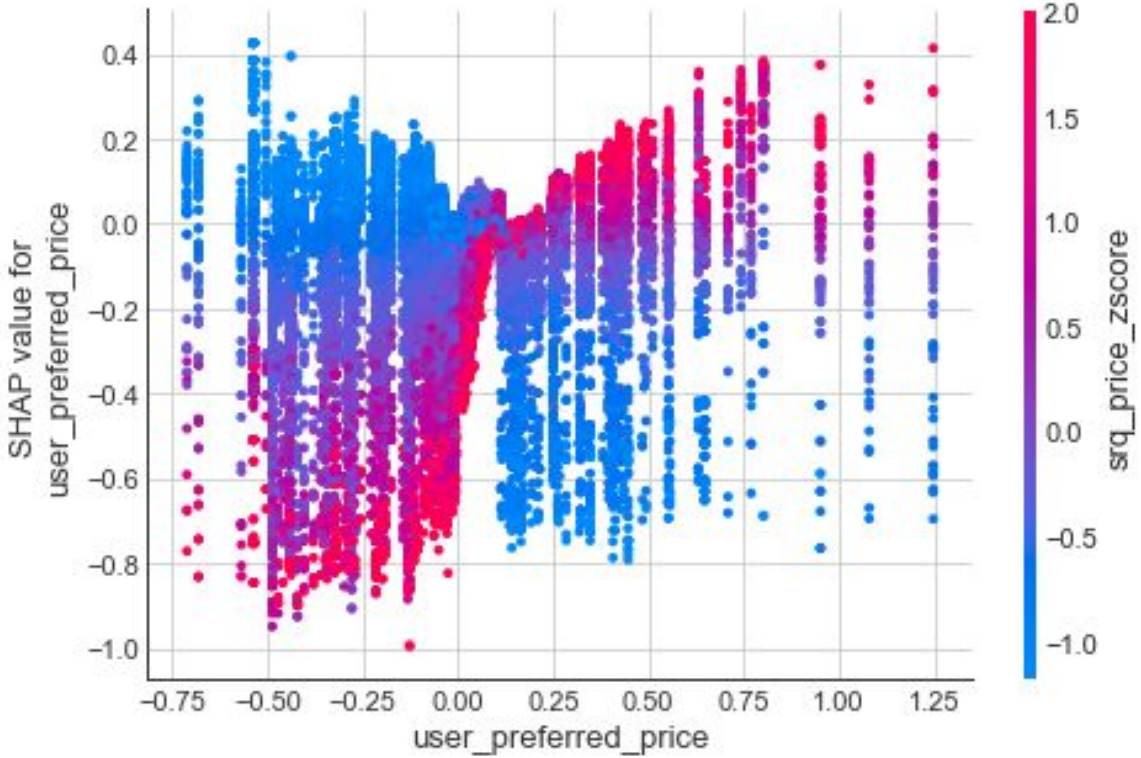
Review count

Distance

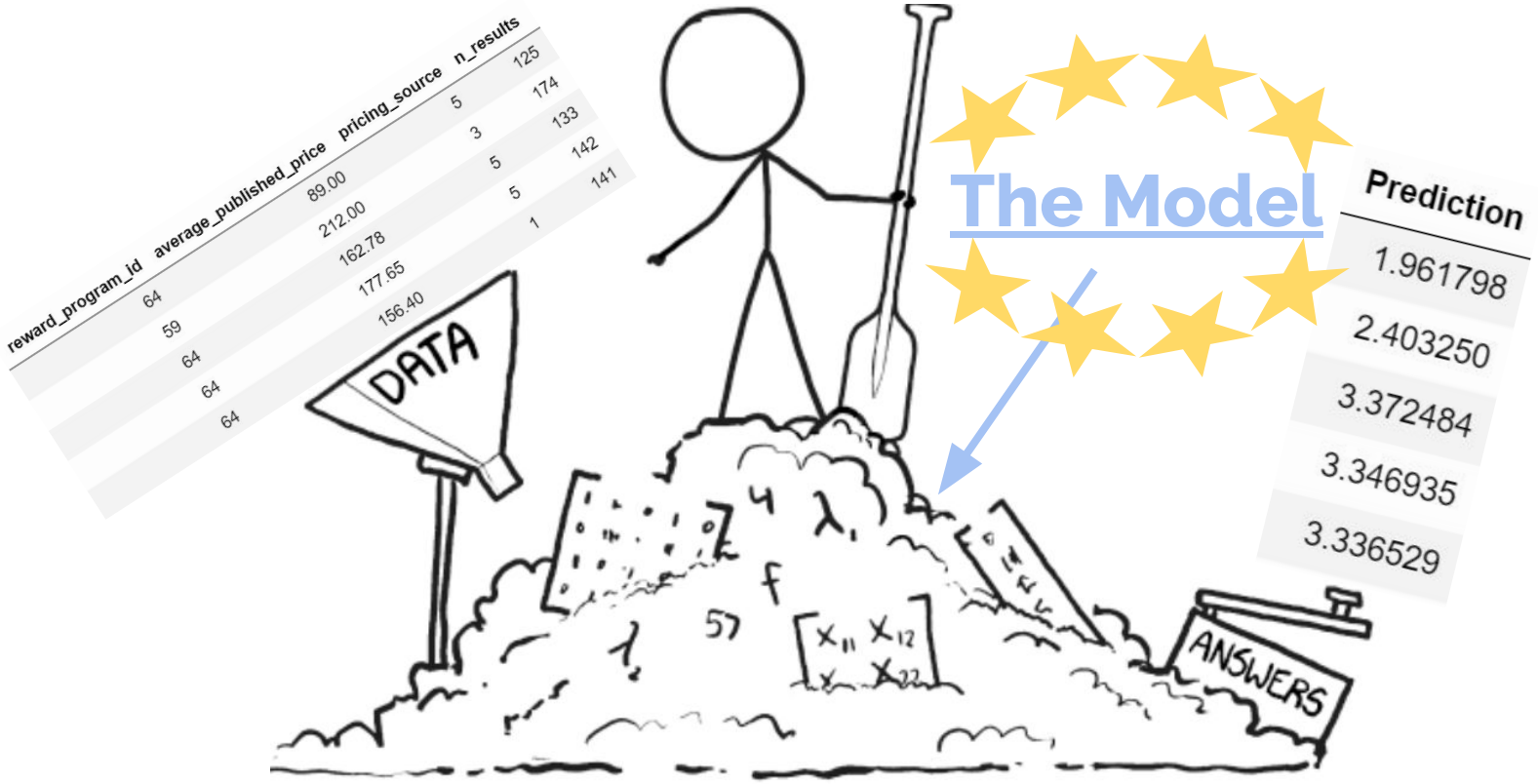
Stars

Partial dependence plots for feature interactions

Model is able to **adjust the score** based off of how closely the **hotel price** (srq_price_zscore) matches the **user's preferred price level**



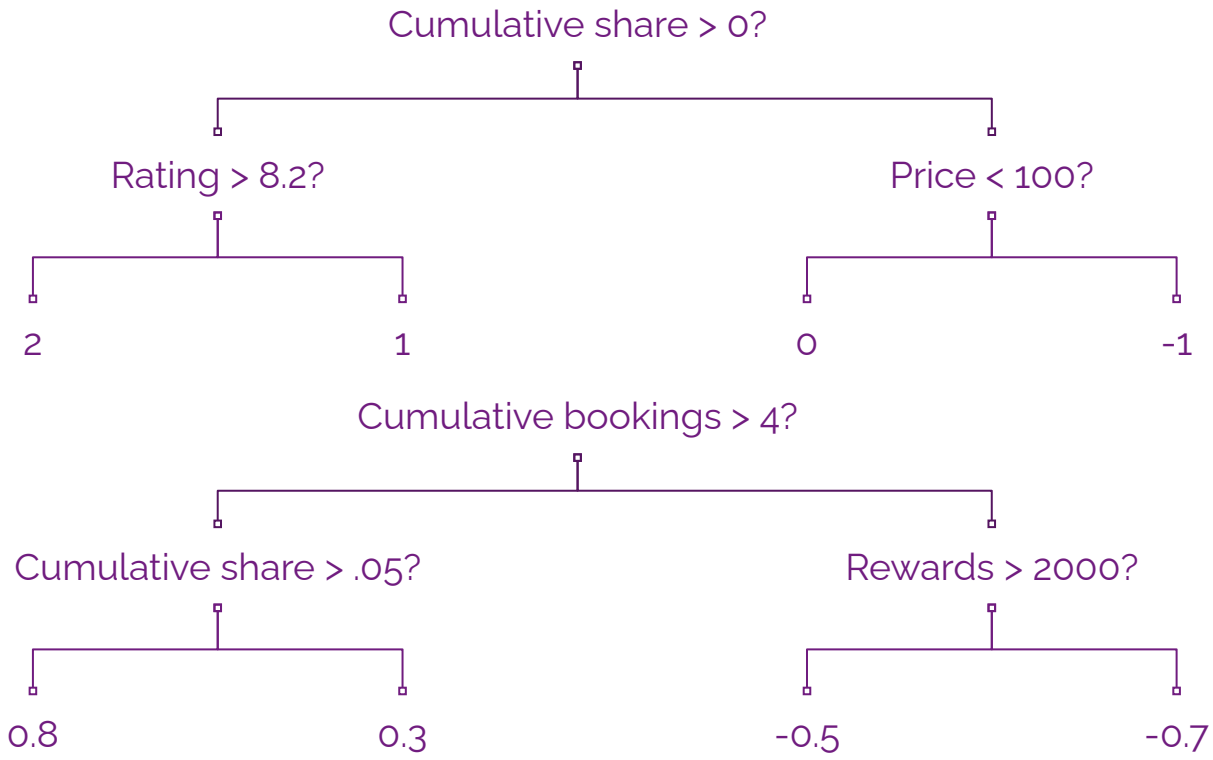
Modeling search result relevance



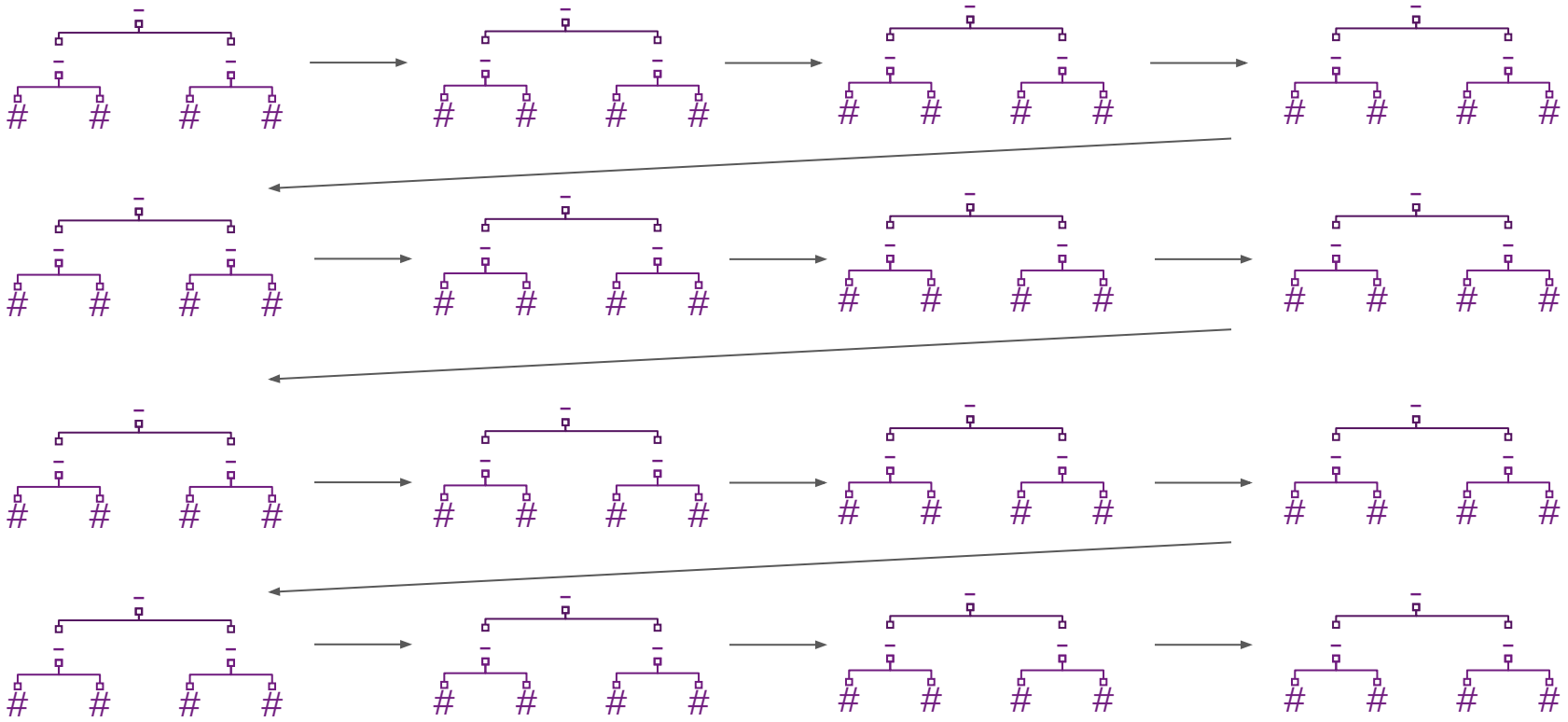
Credit: https://xkcd.com/1838/

Model of choice: gradient boosting tree

The second tree predicts the **error** of the first tree



We can continue gradient boosting ad infinitum



Predict which hotel is “preferred” by the customer

- Unconsidered
0 Irrelevant
1 Details
2 Payment
3 Purchased

1. Mgm Grand	2. Aria	3. Ti - Treasure Island	4. The Venetian®	5. Hard Rock
6. Paris	7. The Palazzo	8. The Cosmopolitan	9. Luxor	10. Sls
11. Tropicana	12. Westgate	13. The Linq	14. Bally's	15. Circus Circus
16. Golden Nugget	17. The Signature	18. El Cortez	19. Renaissance	20. Polo Towers
21. Waldorf Astoria	22. Palms Casino Resort	23. Red Rock	24. Nomad Las Vegas	25. Downtown Grand

A result’s label is determined based on its **funnel stage**

Predict which hotel is “preferred” by the customer

- Unconsidered
0 Irrelevant
1 Details
2 Payment
3 Purchased

1. Mgm Grand	2. Aria	3. Ti - Treasure Island	4. The Venetian®	5. Hard Rock
6. Paris	7. The Palazzo	8. The Cosmopolitan	9. Luxor	10. Sls
11. Tropicana	12. Westgate	13. The Linq	14. Bally's	15. Circus Circus
16. Golden Nugget	17. The Signature	18. El Cortez	19. Renaissance	20. Polo Towers
21. Waldorf Astoria	22. Palms Casino Resort	23. Red Rock	24. Nomad Las Vegas	25. Downtown Grand

Neighbors of positively labeled results were likely **considered**

Predict which hotel is “preferred” by the customer

- Unconsidered
0 Irrelevant
1 Details
2 Payment
3 Purchased

1. Mgm Grand	2. Aria	3. Ti - Treasure Island	4. The Venetian®	5. Hard Rock
6. Paris	7. The Palazzo	8. The Cosmopolitan	9. Luxor	10. Sls
11. Tropicana	12. Westgate	13. The Linq	14. Bally's	15. Circus Circus
16. Golden Nugget	17. The Signature	18. El Cortez	19. Renaissance	20. Polo Towers
21. Waldorf Astoria	22. Palms Casino Resort	23. Red Rock	24. Nomad Las Vegas	25. Downtown Grand

Results **unconsidered** by the user are left out of the dataset

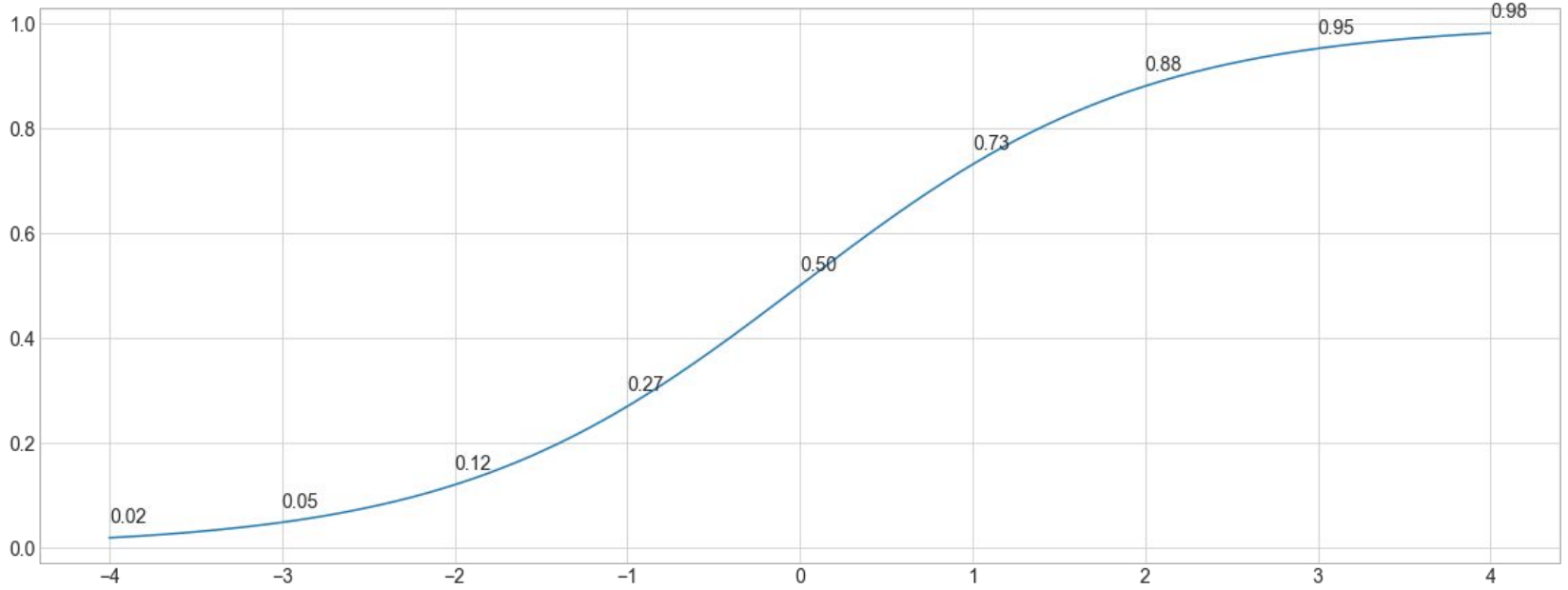
Error function compares scores against each other

$$\lambda_i = \sum_j [P_{\text{true}}(x_i >> x_j) - P_{\text{predicted}}(x_i >> x_j)] |\Delta_{NDCG}|$$

- The model is graded based on how far the predicted preference probability differs from the true preference probability
- **NDCG** is a metric which penalizes errors made high in the ranking more than mistakes made further down
- When we gradient boost with this error function, we are using a **Learning to Rank** model called **LambdaMART**

Map score differences to preference probabilities

$$P_{\text{predicted}}(x_i >> x_j) = \text{sigmoid}(x_i - x_j) = \frac{e^{x_i}}{e^{x_i} + e^{x_j}}$$



Converting scores to probabilities for profit reranking

$$\text{sigmoid}(x) \rightarrow \text{softmax}(\{x_1, ..., x_i, ..., x_n\})$$

$$\frac{e^x}{e^x + e^0} \rightarrow \frac{e^{x_i}}{e^{x_1} + ... + e^{x_i} + ... + e^{x_n}}$$

This **softmax probability** represents the chance that result x_i from the set $\{x_j\}$ is picked, given the user picks one result

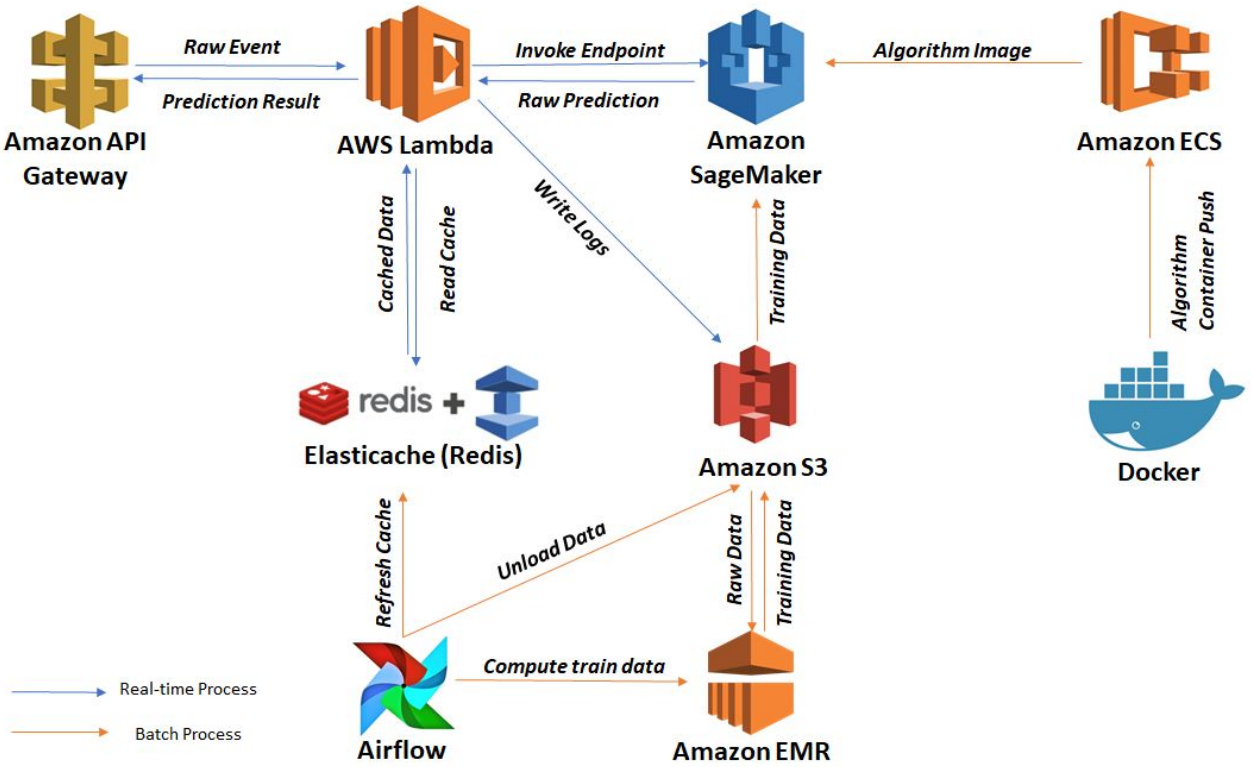
With these probabilities, we can rerank results and maximize expected profit!

(Actually, since the softmax normalization is constant across all results in the request, simply multiplying profit by the exponentiated scores achieves the profit-maximizing sort.)

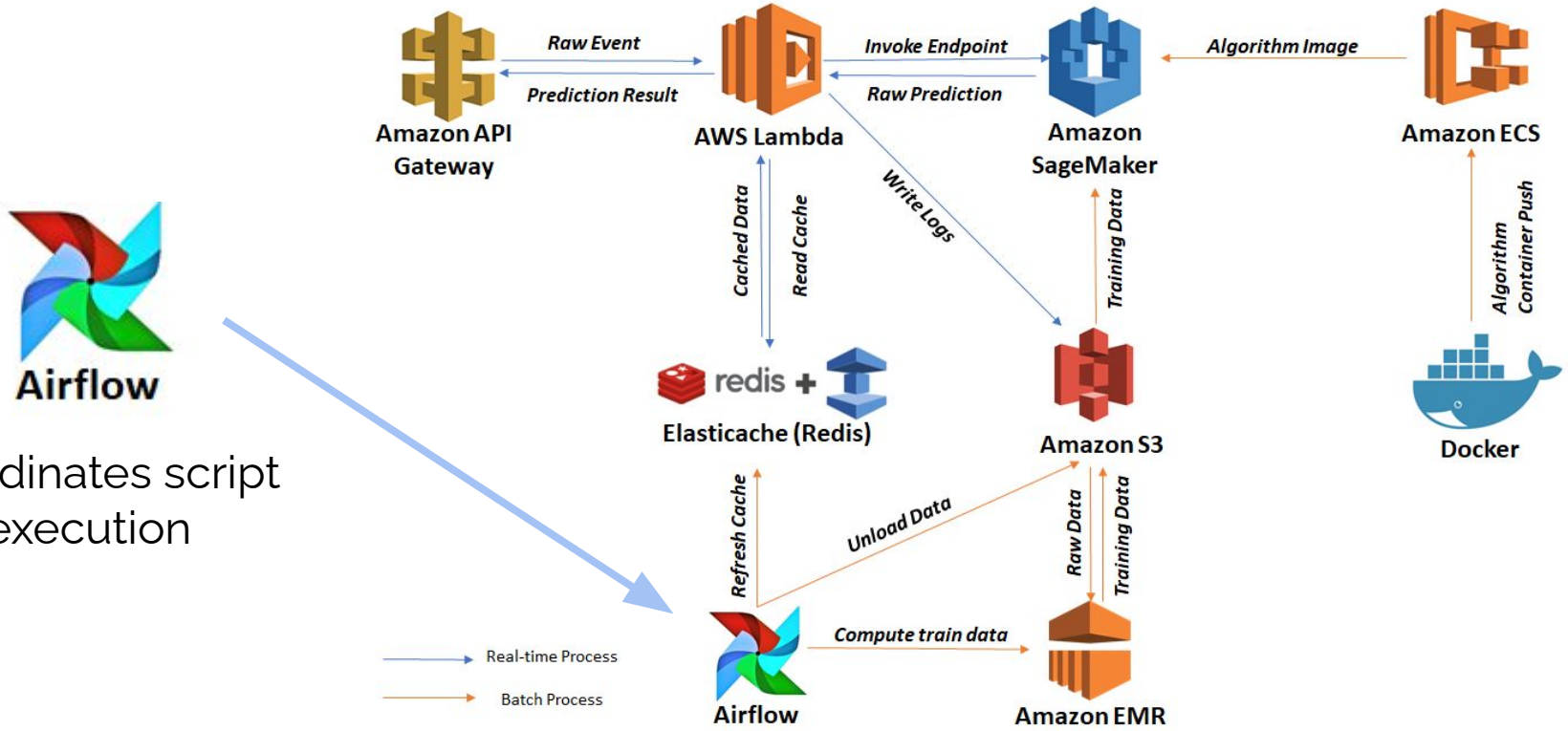
Deployment

Credit: Sergly Serdyuk/IStockPhoto/Thinkstock

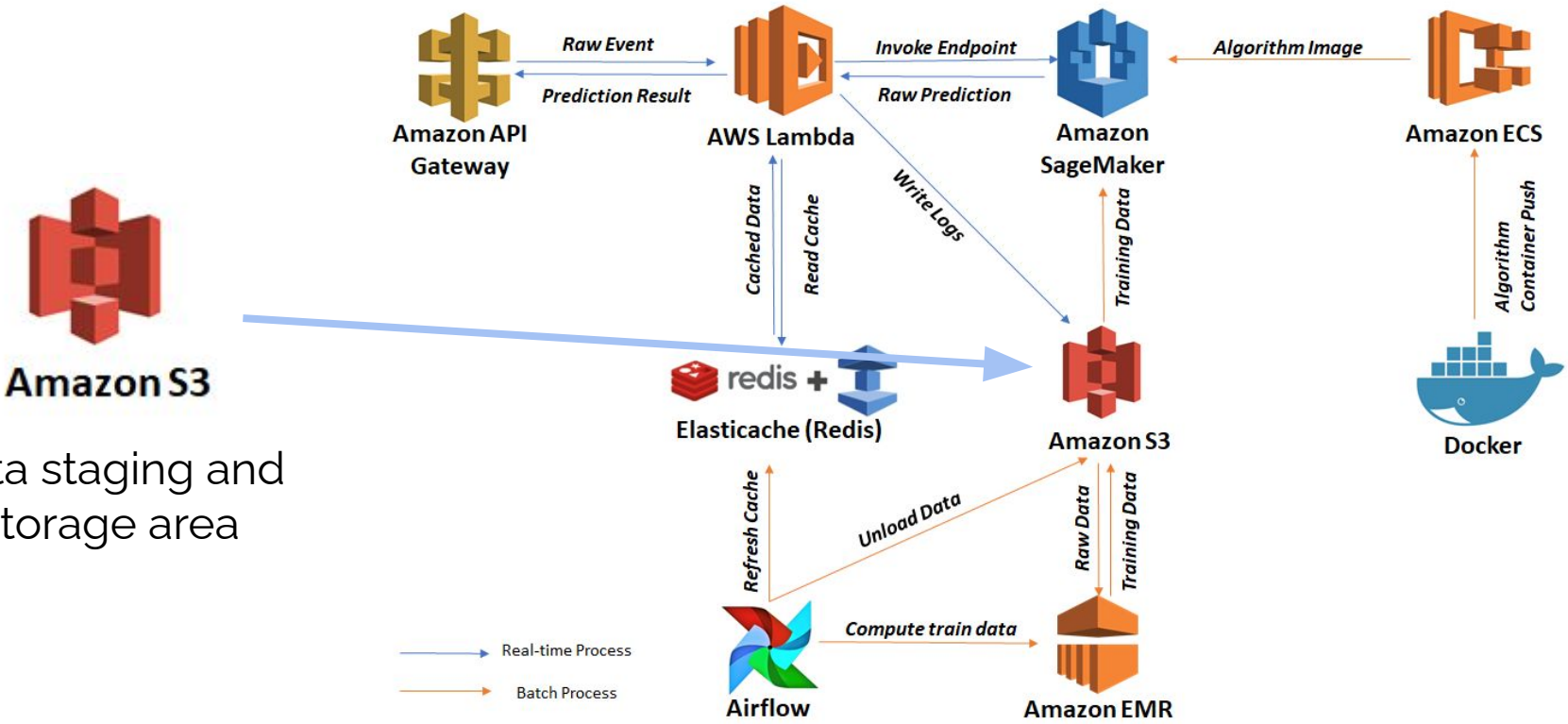
Model architecture



Model architecture

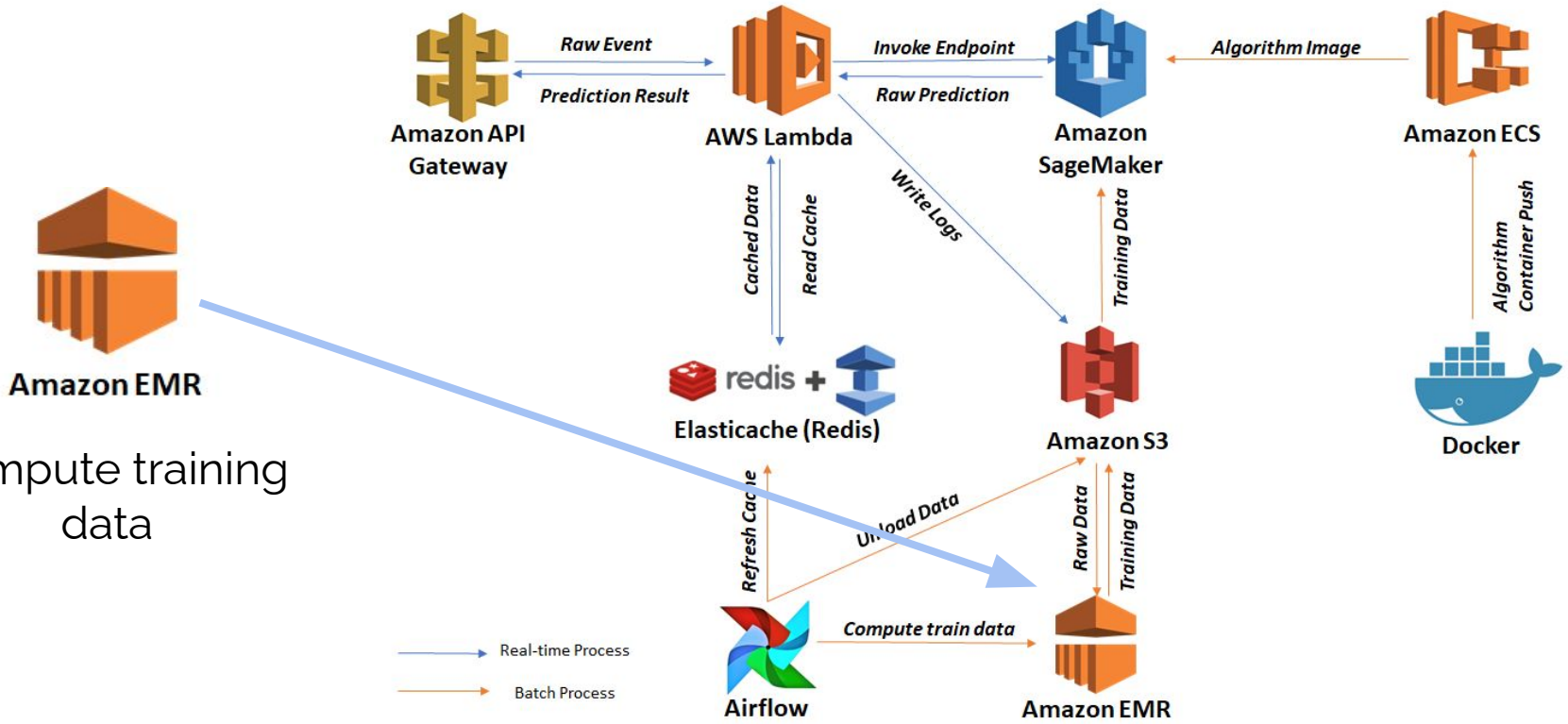


Model architecture



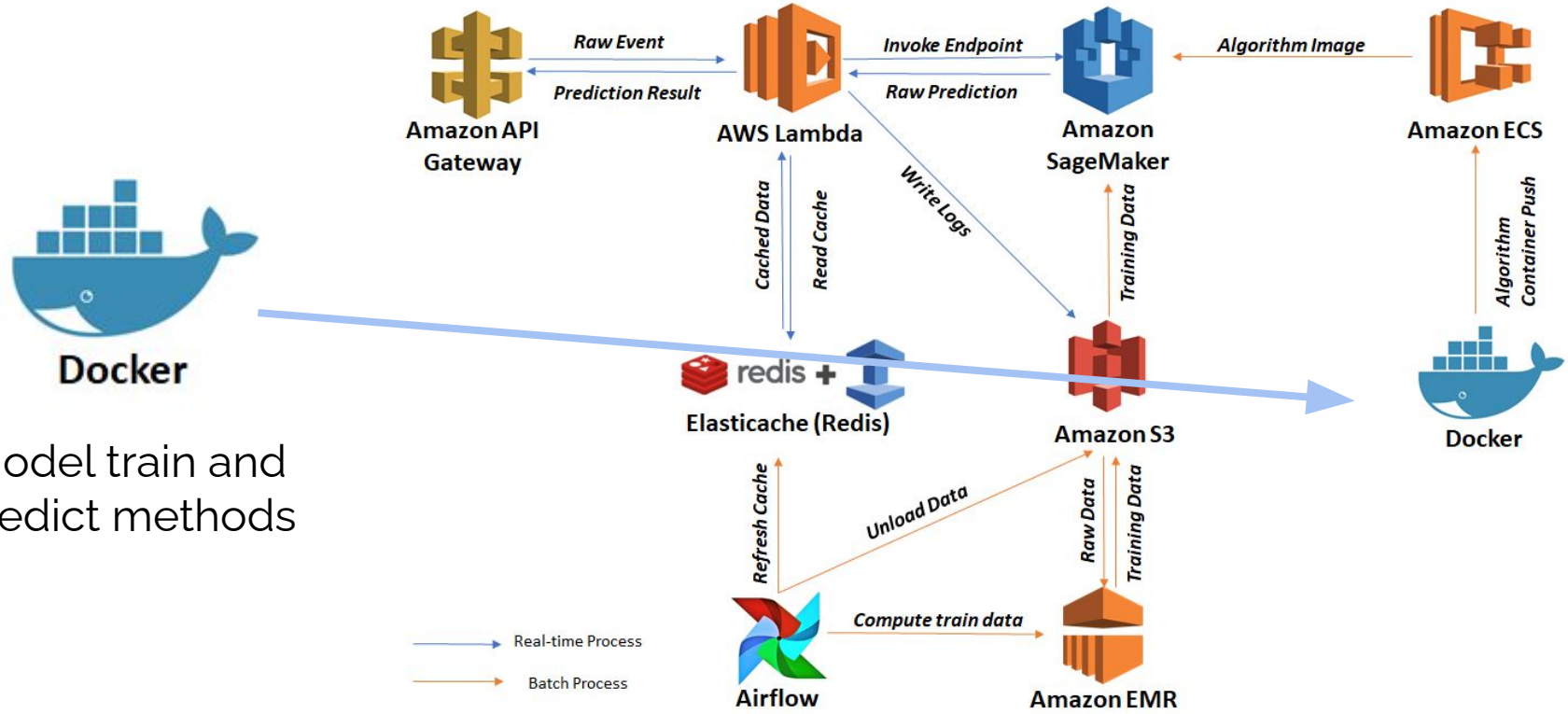
Data staging and storage area

Model architecture



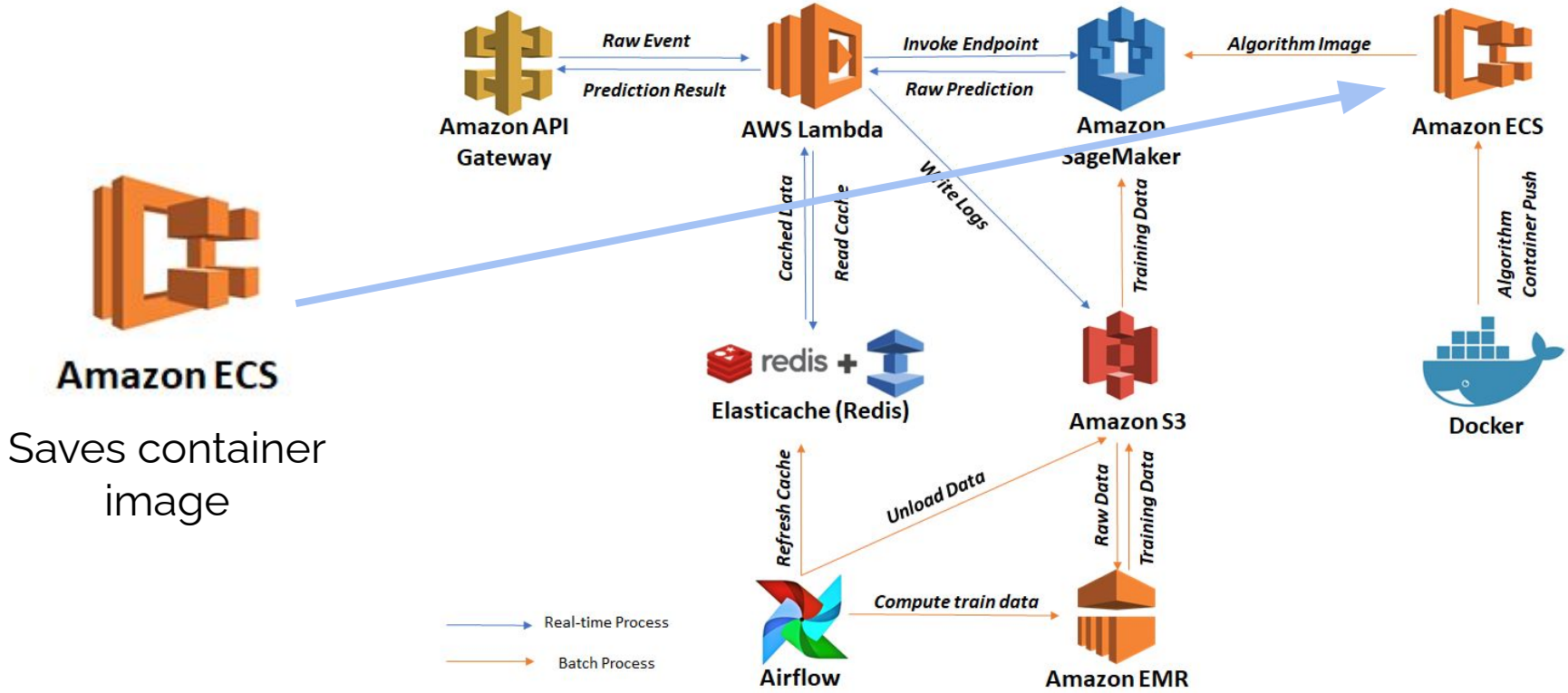
Compute training data

Model architecture

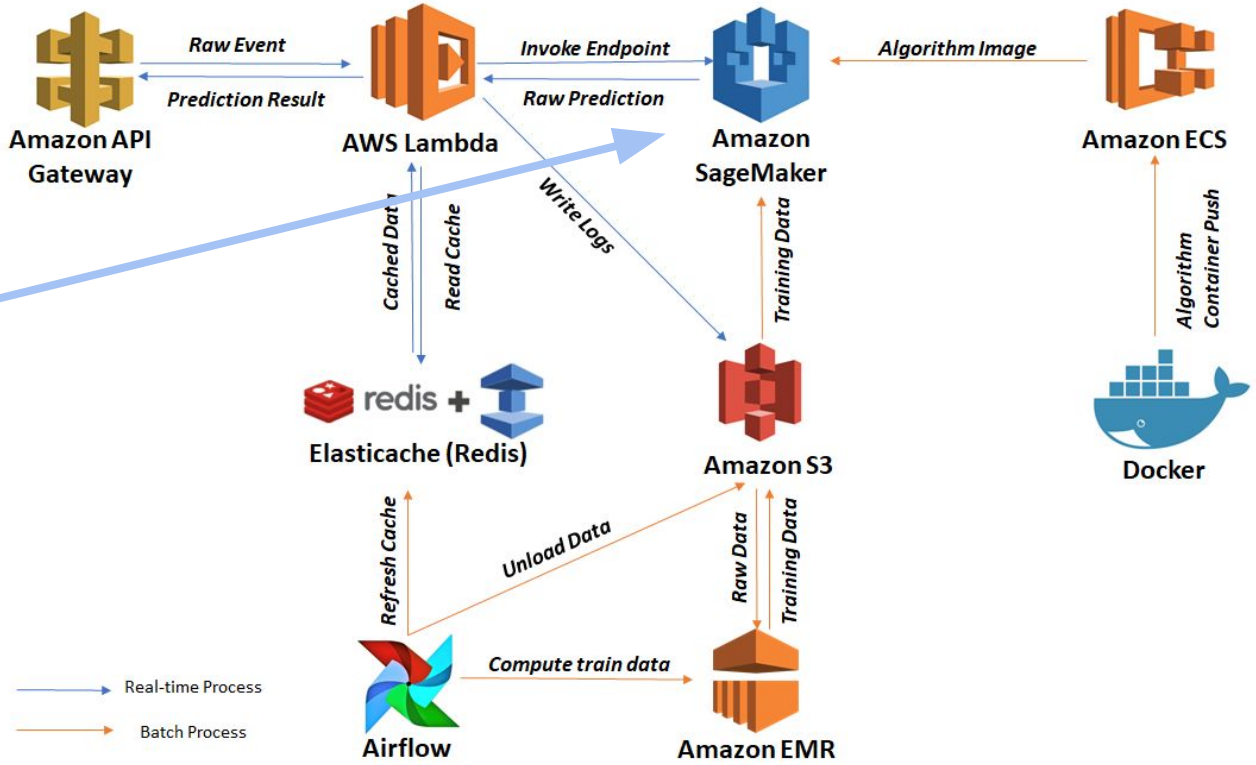


Model train and predict methods

Model architecture



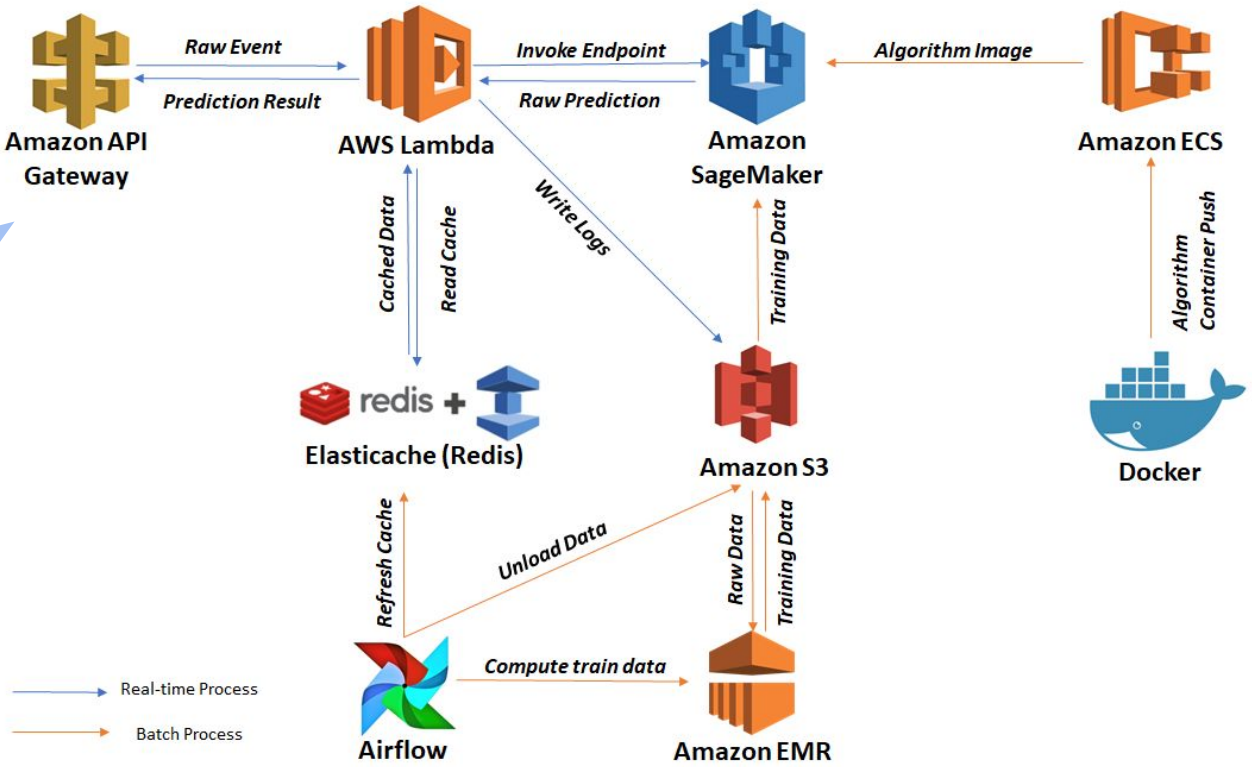
Model architecture



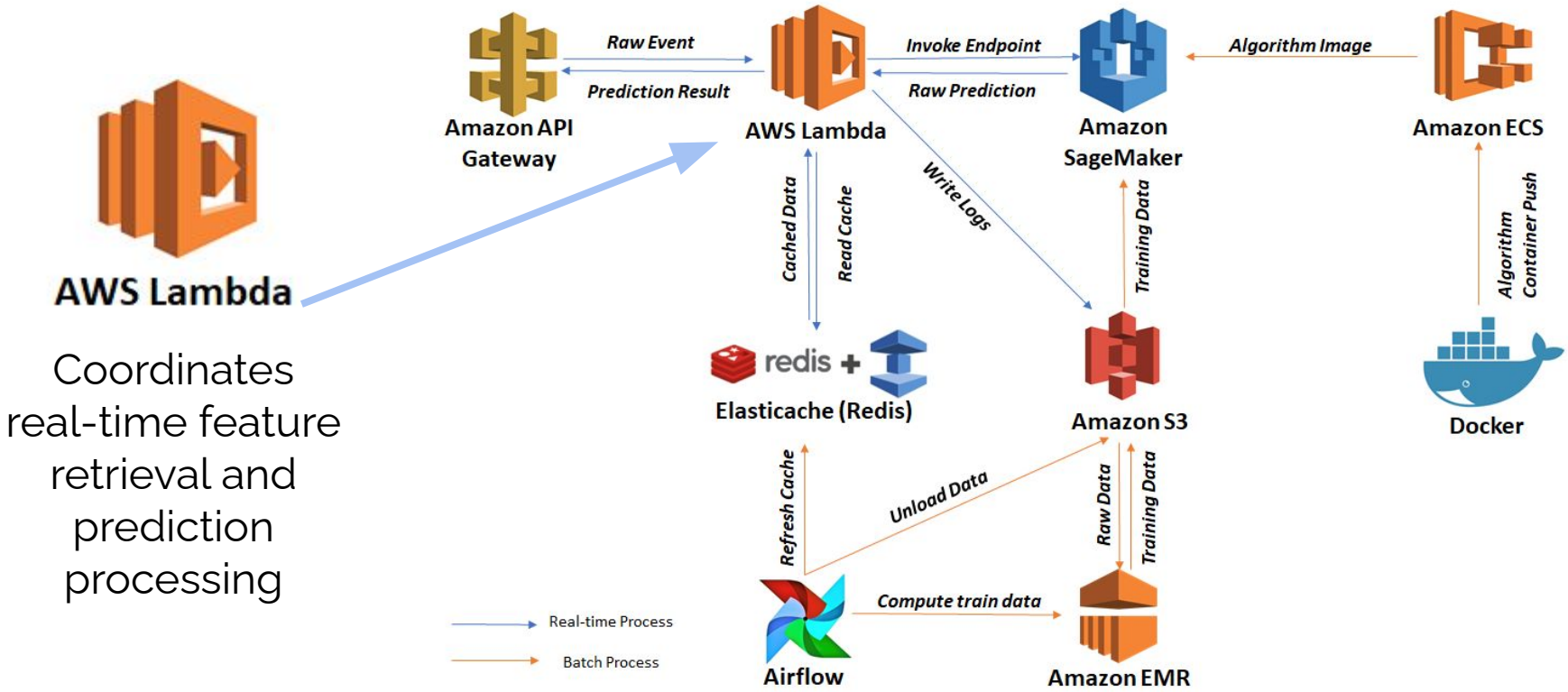
Trains model and hosts endpoint

Model architecture

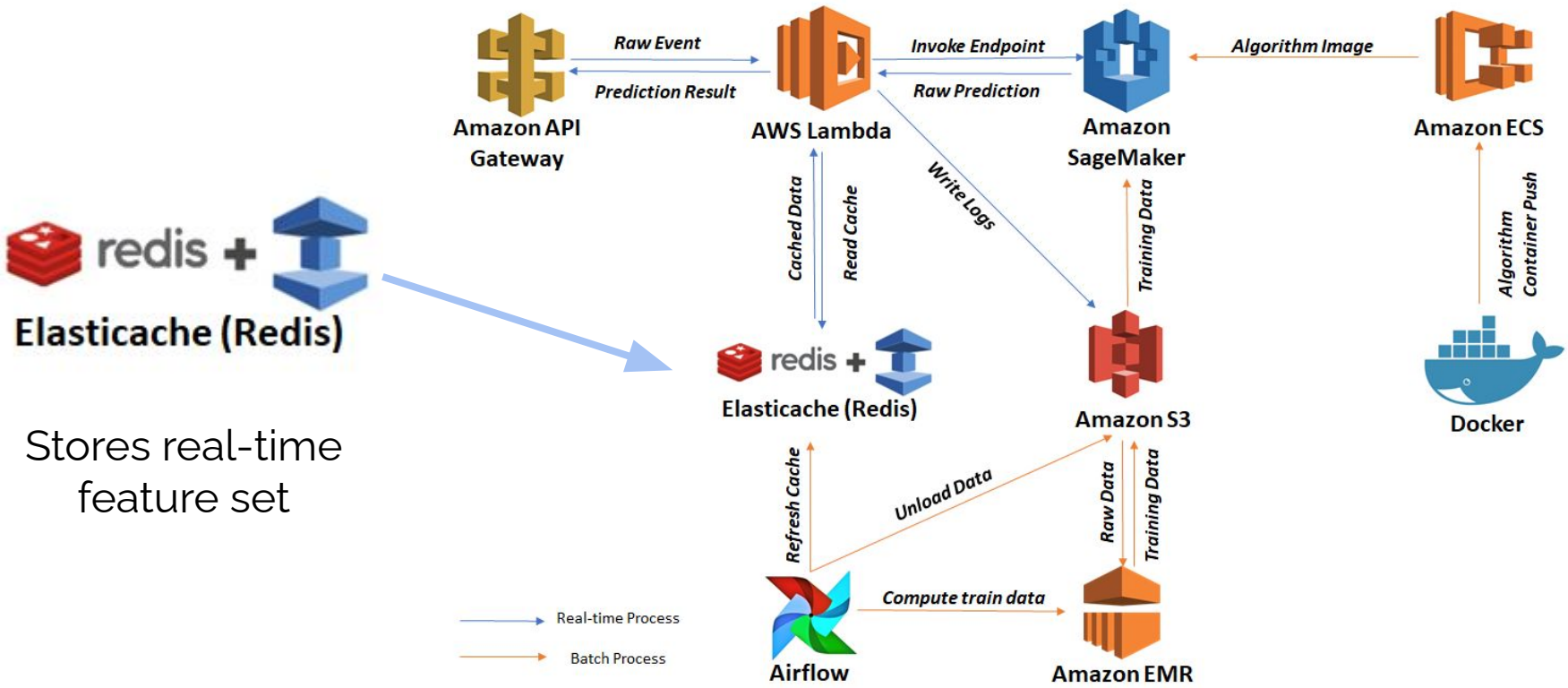
HTTP access to the API



Model architecture



Model architecture



Stores real-time feature set

Quality assurance at every step

Spark pull	Model training	Pre-deployment	Post-deployment
			
Data integrity - Nulls - Duplications - Ranges - Similarity to prod logs	**Model quality** - NDCG scores - SHAP signs - Correlations with previous models	**Infrastructure** - Unit tests - Batch load simulations	**Performance** - MBPR - Prediction histograms - Data integrity

Looking ahead

Engineering

Reducing Latency

Encapsulability

QA and monitoring

Broader model ecosystem

Theory

Embeddings as features

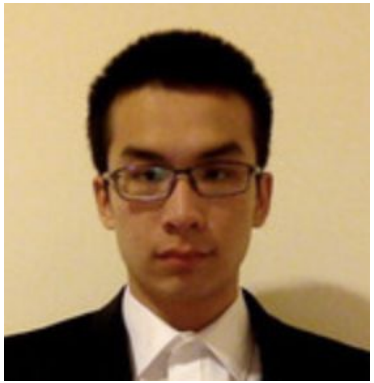
Alternate loss function

Debiasing by rank

Supporting dynamic reranking

Thanks to everyone who contributed!

Eric He
Research and Analysis



Renyu Zhang
Architecture and QA

Gary Ng
Planning and Review


We would love to take your questions!
Credit: Intercontinental Hotel Group
#2 Intercontinental NYC

The full list of mathematical equations

1. Map score differences to preference probabilities

$$P(x_i \triangleleft x_j) = \sigma(I - J) = \frac{e^I}{e^I + e^J}$$

2. Loss function adds NDCG-reweighting and rank debiasing

$$\lambda_{x_i} = \sum_{x_j} [P_{\text{true}}(x_i \triangleleft x_j) - P_{\text{predicted}}(x_i \triangleleft x_j)] * \frac{|\Delta \text{NDCG}|}{p^+(\text{rank}(x_i)) * p^-(\text{rank}(x_j))}$$

$$\text{DCG} = \sum_i \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)}$$

$$\text{NDCG} = \frac{\text{DCG}}{\text{IDCG}}$$

$$p^+_{raw}(1) = \sum_q \sum_j \frac{L(1,j)}{p^{-*}(j)}$$

$$p^-_{raw}(1) = \sum_q \sum_j \frac{L(1,j)}{p^{+*}(j)}$$

$$p^+(i) = \frac{1}{p^+_{raw}(1)} \sum_q \sum_j \frac{L(i,j)}{p^{-*}(j)}$$

$$p^-(i) = \frac{1}{p^-_{raw}(1)} \sum_q \sum_j \frac{L(i,j)}{p^{+*}(j)}$$

3. Multiply exponentiated relevance score by profitability metric

$$\text{Ranking Score} = Relevance * \frac{Profit}{\sqrt{Revenue}}$$

Updated architecture