Response Prediction and Ranking Models for Large-Scale Ecommerce Search

Seinjuti Chatterjee, Ravi Shankar Mishra, Sagar Raichandani, and Prasad Joshi

1 Problem Statement

User response prediction is the bread and butter of an ecommerce site. Every ecommerce site which is popular is running a response prediction engine behind the scenes to improve user engagement and to minimize the number of hops or queries that a user must fire in order to reach the destination item page which best matches the user's query. With the dawn of artificial intelligence (AI) and machine learning (ML), the whole merchandising process, web-commerce carousel product arrangement, personalized search results and user interactions can be driven by the click of a button. Modern-day ML platforms enable a dynamic cascade of models with different optimization functions which can be tuned towards a user's preference, taste and query trajectory.

In this paper, we talk about how Unbxd search services powers its user engagement and response prediction behind the scene using a plethora of optimized features across multiple channels and multiple domains. Elaborate feature engineering is deployed to understand the user's propensity to click. The search funnel lifecycle starts with a personalized search impression, captures a user click, progresses towards a cart and finally materializes into an order or sale. In this scenario, click through rate (CTR) modelling is the binary classification task of predicting whether a user would click given a ranked ordered set of products and conversion rate (CVR) modelling entails

S. Chatterjee (✉) · R. S. Mishra · S. Raichandani · P. Joshi
Unbxd, Bengaluru, India
e-mail: seinjuti@gmail.com

R. S. Mishra
e-mail: ravi@unbxd.com

S. Raichandani
e-mail: raichandanisagar@gmail.com

P. Joshi
e-mail: prasad.joshi@unbxd.com

A. K. Laha (ed.), *Applied Advanced Analytics*, Springer Proceedings in Business and Economics, https://doi.org/10.1007/978-981-33-6656-5_17

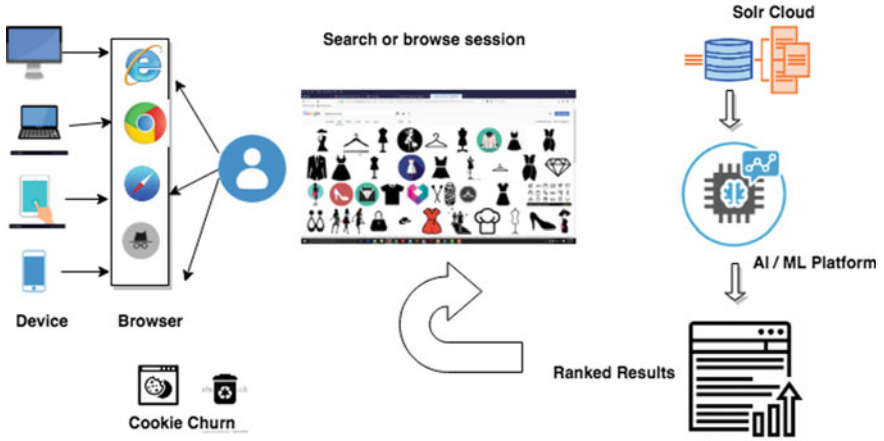


Fig. 1 View of user from search engine

the binary task of predicting whether user would purchase an item given he has shown click interest.

Figure 1 demonstrates the view of the user from a search engine's point and the view of the merchandizing website from the user's point. A user might be present in any device linked through the browser cookie or the device ID and might choose to initiate a search or browse session for the day; however, the click through and the conversion might potentially happen at a different device and at a different place (work or home) during a different time of day. Hence, an intelligent search engine must be able to stitch the user's trajectory seamlessly and understand the feature combinations leading up to an event for better engagement.

2 Literature Survey

As we started thinking about the bipartite query-product matching problem, we drew lot of similarity with the query-ads domain where personalizing the click prediction benefits both the users and the advertisers. In (Cheng & Cantu-Paz, 2010) the authors mention that the users will be presented ads in the manner that is most relevant to them, and the advertisers will receive clicks from users who are more engaged with the ads. In the search domain, however, most of the times search engines are burdened with the task of retrieval of the most relevant documents to improve precision and recall but in the process falling short of optimizing the business metrics like average order value, price per session, engagement rates such as click through, etc. In feature-based query performance as mentioned in (Kumar et al., 2018), the authors analyse user's behavioural patterns and build models to classify queries as high engagement queries, high sale through rate queries, thereby providing the search engine means to

drive its business metrics other than just optimizing precision and recall per search. In (Zhou et al., 2018), researchers at Alibaba attempted to understand the deep interest graph of a user and the context of an ad, thereby using deep learning to model higher order feature interactions which drive a click. They have closely modelled the user's historical data as a sequence model and built a network which given the current sequence can closely predict the future interactions of the user in terms of product affinity and personalized ads. In (Guo et al., 2017), authors mention that during their study in a mainstream apps market, they found that people often download apps for food delivery at meal time, suggesting a second-order interaction between app category and time stamp.

In this paper, we talk about the search business insights that Unbxd has gathered being one of the largest ecommerce search service providers across domains like electronics, furniture, fashion, grocery. These insights indicate strong correlations between user, context, category, time of day features and the performance metrics of a query. Starting with the business problem, we have implemented distributed models at scale which now define our AI or ML framework. Together with our inhouse A/B testing framework, we have demonstrated the capability of our ML models to our clients, and the overall journey has been summarized in this paper.

3 Algorithm

The naive model we started with at Unbxd is a composition of clickability and buyability of a product learnt over historical clicks, carts, and orders. Here, we rank all products which are deemed relevant for a particular query in descending order of following score.

$$\text{Score} = \sum_{m=0}^{m=60} \left[\left(\text{clicks} / \left(\text{a } e^{b/r} \right) \right) (1 + \text{ carts/clicks}) \left(1 + \frac{\text{orders}}{\text{carts}} \right)^2 \right] e^{-m/9} \quad (1)$$

where r = rank of the product and a and b are constants and m is lookback days which is number of days in past we want to consider. This approach is a relatively static approach but captures the recency of clicks, carts, and orders and can be considered a ranking by popularity score. The composite score acts like an overall boost factor to be overlaid on indigenous search ranking implemented in Apache Solr Search Platform (Solr) in order to bubble up the trending products. However, this score is not nimble enough to adapt to dynamic ranking depending on the device or browser or query context or location or time of day.

Each impression consists of various attributes extracted from the request side parameters such as site, query, device, user, time of day, day of week, query category, location. This impression must be now matched with the Solr retrieved document attributes like product category, price, keywords, reviews, related products, tokens, etc. Hence, now the problem morphs into a bipartite graph matching algorithm with

certain constraints and measured by the ranking loss function. Such features are called unigram features; since they only depend on one attribute; we can also use advanced features like query-dwell-time, time-to-first-click, time-to-first-cart, time-to-first-order, was-autosuggest-used, Wi-Fi-connection-type-of-user as mentioned in (Cheng & Cantu-Paz, 2010) depending on the data collection exposed through the search API.

Once we have collected such unigram features, we can fit the impressions data complete with the outcome to a **logistic regression** model as a binomial classification task which then estimates the probability of click given a new impression based on its features.

Logistic regression model: Given any input event, we assume that its outcome is a binary variable; i.e., it is either positive or negative. The logistic regression model calculates the probability of a positive outcome with the following function:

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) \text{ where } p = 1/\left(1 + e^{-w^T x}\right) \tag{2}$$

Here, x denotes vector of the features extracted from the input event and w is the vector of corresponding feature weight that we need to learn. LR model is trained with gradient descent. Assume that we have observed a set of events $X = \{x_i\}$ and their outcomes $Y = \{y_i\}$. Each event x_i can be represented with a set of features $\{x_{ij}\}$. We want to find the set of parameters w by maximizing the data likelihood $P(Y|X, w)$. This is equivalent to minimizing the following loss function:

$$L = -\log P(Y|X, w) = \sum_{i=0}^{n} \log P(y_i|x_i, w) \tag{3}$$

The beta coefficients of the model and ROC curve helps us understand the discriminating ability of the model between the positive and negative samples and ability to explain CTR through features.

4 Feature Selection

In Fig. 2, we show the factor map of the search session that is available to a third-party search engine. Some interesting features have been described below:

Data fields of a search session

- Outcome—click: 0/1 for non-click/click (can be cart or order depending on the model)
- Time Series Features
  - hour_of_day: int from 0 to 23, (parse format is YY-MM-DD-HH from session_time, so 14091123 means 23:00 on Sept. 11, 2014 UTC.)

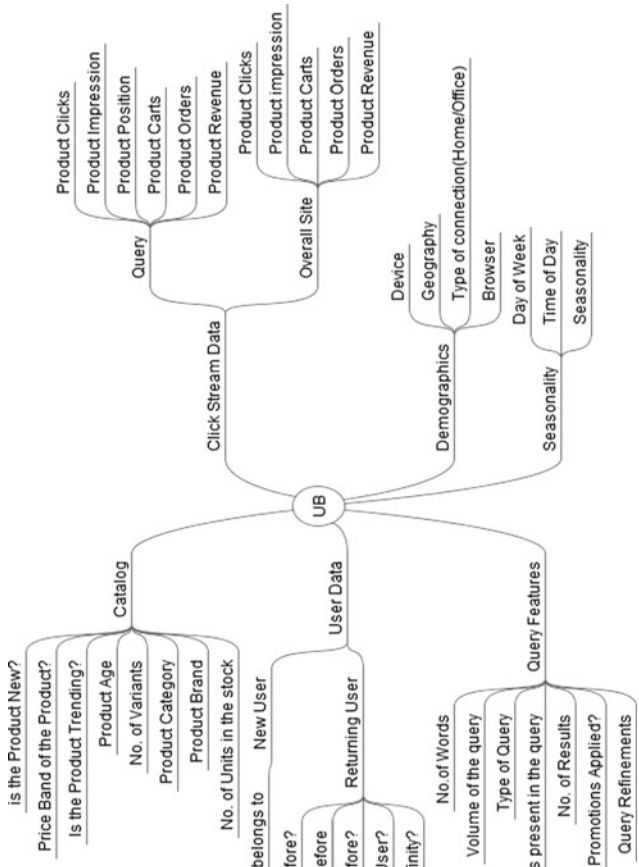


Fig. 2 Feature map of search session

  - day_of_week: 0–6 (parse format is YY-MM-DD-HH from session_time)
  - is_weekday: Boolean feature
  - is_weekend: Boolean feature

- Site Features
  - site_id: int
  - site_domain: string
  - site_catalog_size: int
  - site_daily_aggregate_clicks: int
  - site_daily_aggregate_impressions: int
  - site_daily_aggregate_carts: int
  - site_daily_aggregate_orders:int

- Product Features
  - product_category_in_serp: product category of search result page
  - product_id_in_serp: product id of search result page
  - product_pos_in_serp: product position in search result page
  - product_age (in terms of freshness: no of days old from the time of impression)
  - product_dynamic_popularity_score
  - product_num_stock_units

- Device Features
  - device_id
  - device_ip
  - device_model
  - device_type

- Location Features
  - country
  - pincode
  - region
  - latitude
  - longitude

- Query Features
  - query_tokens
  - query_length
  - query_refinements_in_same_session
  - query_entities (must have tokens, synonyms, entity recognition output—brand, product type, model no etc.)
  - query_daily_aggregate_clicks
  - query_daily_aggregate_orders
  - query_daily_aggregate_carts

- To account for position bias, we use a position-normalized statistic known as clicks over expected clicks (COEC) as defined in Cheng & Cantu-Paz, 2010

$$\text{COEC} = \frac{\sum_{r=0}^{R} c_r}{\sum_{r=0}^{R} ir_r * \text{CTR}_r}$$

where the numerator c_r is the total number of clicks received by a query-product pair; the denominator can be understood as the expected clicks (ECs) that an average product would receive after being impressed ir_r times at rank r, and CTR_r is the average CTR for each position in the result page (up to R), computed over all queries and products. We can obtain COEC statistic for specific query-product pairs, and this statistic is a good predictor of click probabilities as mentioned in (Cheng & Cantu-Paz, 2010). However, many data points are needed for this statistic to be significant but data for specific query-product pairs can be sparse and noisy.

In consideration of efficiency and robustness, we need to filter out certain types of features and this process is called **feature pruning**.

- Features with too few impressions. The simplest approach is to set a threshold on the number of impressions and filter out features with less impressions than the threshold. This step reduces the model size. However, one issue with this approach is that new features may get filtered out too easily. An alternative method is to put a threshold on the average number of impressions (averageImps) per feature defined as:

$$\text{average Imps} = \frac{\text{total Imps} + c}{\text{current Time} - \text{first Occurrence Time}} \tag{4}$$

where totalImps is total number of impressions received by the feature, currentTime is time of measurement in secs, firstOccurenceTime is time when feature received first impression and the constant c insures that new features will not be filtered out immediately.

- Features that are too old. If a feature is no longer active for a certain period, we may want to filter it out. In a fast-changing search space, features can become deprecated daily. Filtering such features out improves system efficiency.
- Features that have close to 0 weights. If a feature weight is close to 0, it means that this feature does not significantly affect the prediction, and if we filter this feature out, there should be little performance impact.

However, without expert feature engineering, exploring higher order features become daunting and learning sophisticated feature combinations behind user behavior is critical in maximizing CTR for search systems. The wide and deep model (Tze et al., 2016) from Google provides insights such as considering low- and high-order feature combinations simultaneously brings additional improvement over the cases of considering either alone. To this effect, we are in the process of implementing a

factorization-machine-based neural network for cross product transformations of the original feature space, but it is work in progress.

5 Business Insights

We present in this section some of the business insights our analysts have come up with which provides the intuition behind feature-based response prediction.

Figure 3 provides the intuition that country, region, and zip code are differentiator signals for deciding the propensity of the user to purchase.
In Figs. 4 and 5, we show that by channel (mobile, desktop) and by day of week (weekday vs. weekend) our search sessions volumes and conversions vary. We see that the weekday traffic post 9 am comes mostly from desktop which indicates a user browsing or searching from workplace leading up to a lower average order value (AOV) compared to a user logging in the weekend over mobile when the AOV and engagement both peak, hence opening up an opportunity window for response prediction models to promote bigger ticket items for a query during this time and thereby maximizing conversions and AOV.

In the fourth and fifth graph Figs. 6 and 7, we compare new versus existing users and their search volumes and conversions over various channels—social media, email, display ads, private apps, organic search, etc. By tracking the user type and channel, response prediction models can effectively maximize CTR and CVR.

In the sixth and seventh graphs Figs. 8 and 9, we show how location signals and query category can be correlated. This opens up the opportunity to response predictor models to utilize the user's location (work or home) and the region to optimize the search results for certain query categories. However for staple products like laundry the business metrics remain fairly uniform as shown in Fig. 10.

In the last graph Fig. 11 we note that behaviour in cities is markedly different than behaviour in non-urban areas.

6 ML Architecture

Here, in Fig. 12 we present the details of the ML relevancy platform we have built at Unbxd and how we use the platform to power our scalable distributed logistic regression-based modelling workflow for response prediction.

Distributed LR Training Details in Spark

- The algorithm takes the following inputs:

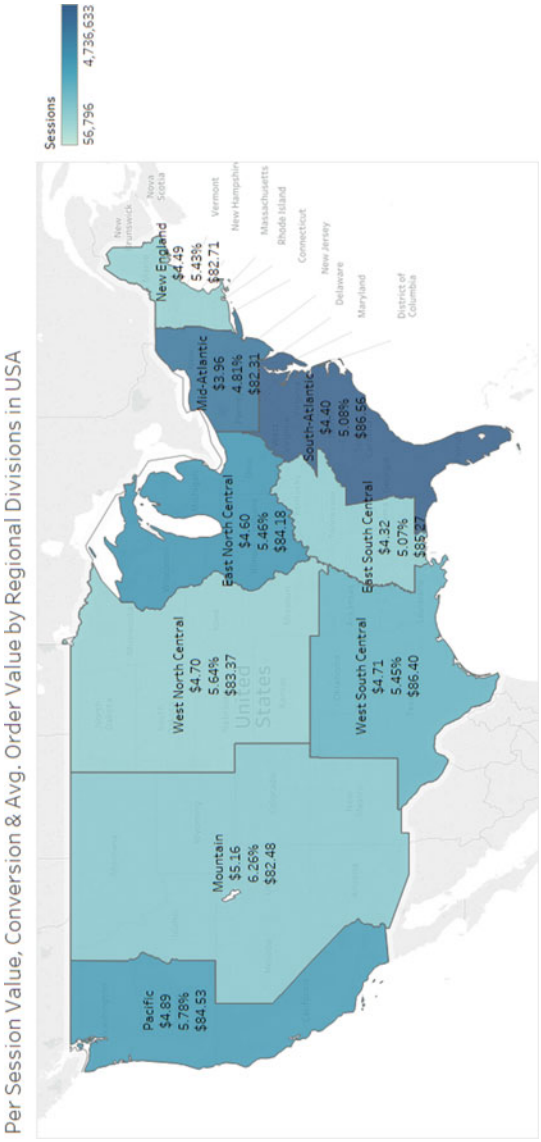


Fig. 3 Differences in per session value (i.e. revenue/num sessions), conversion, Avg. order value (i.e. revenue/transactions) across Regional Divisions in USA. Max differences observed are: Per Session Value– $5.16 versus $3.96, Conversion–6.26% versus 4.81%, Avg. order value–$86.56 versus $82.48

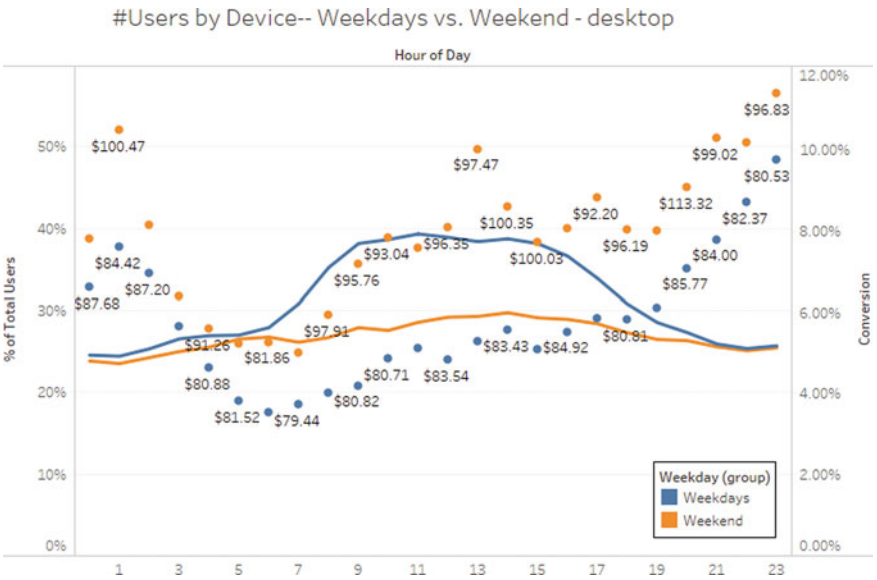


Fig. 4 Percentage of desktop users on weekdays versus weekend. The dots indicate conversion value (check *y*-axis to the right), whereas the text is the Avg. order value. What is evident is the decrease in desktop usage from morning to evening (work hours) on weekends compared to weekdays. Further, the jump in conversion from weekdays to weekends is accompanied by a significant jump in Avg. order value as well

- an existing model path or it can be empty
- a new set of training data on a periodical basis, where each data point represents an impression that consists of a set of attributes, the total number of impressions and the number of impressions with positive outcomes.
- model output path to save the new model
- configuration file to control the parameters.

The training algorithm proceeds as follows:

1. Read the configuration file to get all the training parameters.
2. Decay the number of data points/impressions in the existing model.
3. For $i = 1, 2, \ldots, n$, where n is the number of batches to split the data.

For $j = 1, 2, \ldots, m$, where m is the max number of iterations to run for each batch.

1. Calculate the feature weight updating factor with a map-reduce job using only the impressions which are part of the ith batch.
2. Apply the feature weight updates to the current model.
3. Check for model convergence and continue to next batch if converged.
4. Feature pruning based on the criterion mentioned above.

Some explanation about the algorithm:

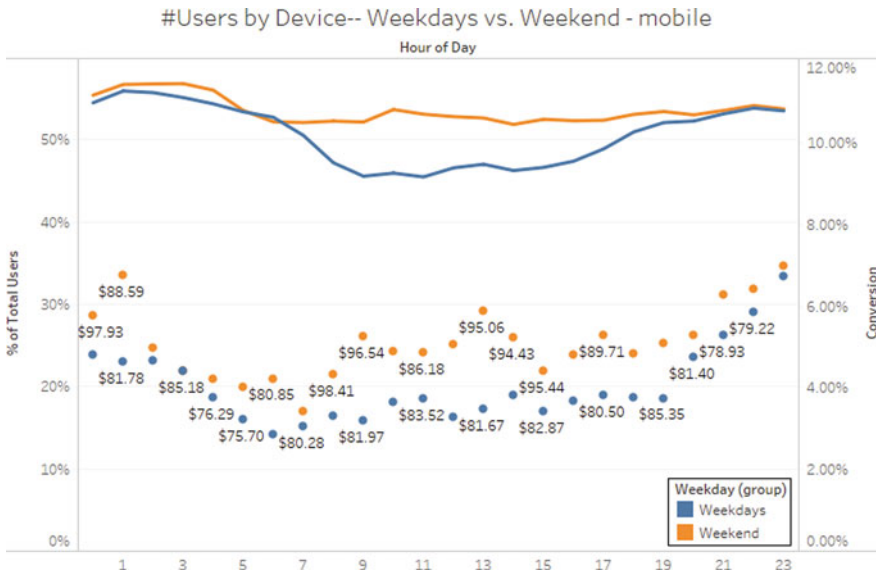


Fig. 5 Percentage of mobile users on weekdays versus weekend. The dots indicate conversion value (check *y*-axis to the right), whereas the text is the Avg. order value. What is evident is the increase in mobile usage from morning to evening (work hours) on weekends compared to weekdays. Further, the jump in conversion (or Avg. order value) from weekdays to weekends is not as strong as in desktop

- To determine a dynamic step size, we consider the number of impressions, while calculating the feature weight updating factor. Instead of using the raw counts, we want to weigh the recent events higher and therefore apply an exponentially decaying weights to the number of events.
- From Step 3, we know that the total number of map-reduce jobs is equal to the product of the number of batches and the number of iterations. If a batch converges before reaching the max number of iterations, we may have fewer jobs. If all the feature weight updating factors are close to 0, we consider the model to have converged with respect to the current batch of training data.
- The mapper class is designed to extract features from training data, make a prediction based on the current model, and output a weight updating factor for each feature.
- The reducer class is responsible for aggregating all the features stats. For each feature, the reducer outputs the sum of weight updating factors, the total number of events and the earliest timestamp.

The training parameters typically include the following:

- num_batches: number of batches we split the data into.
- max_num_iterations: maximum number of iterations we train for each batch.
- decay_factor: a time decay factor we apply to the existing model.

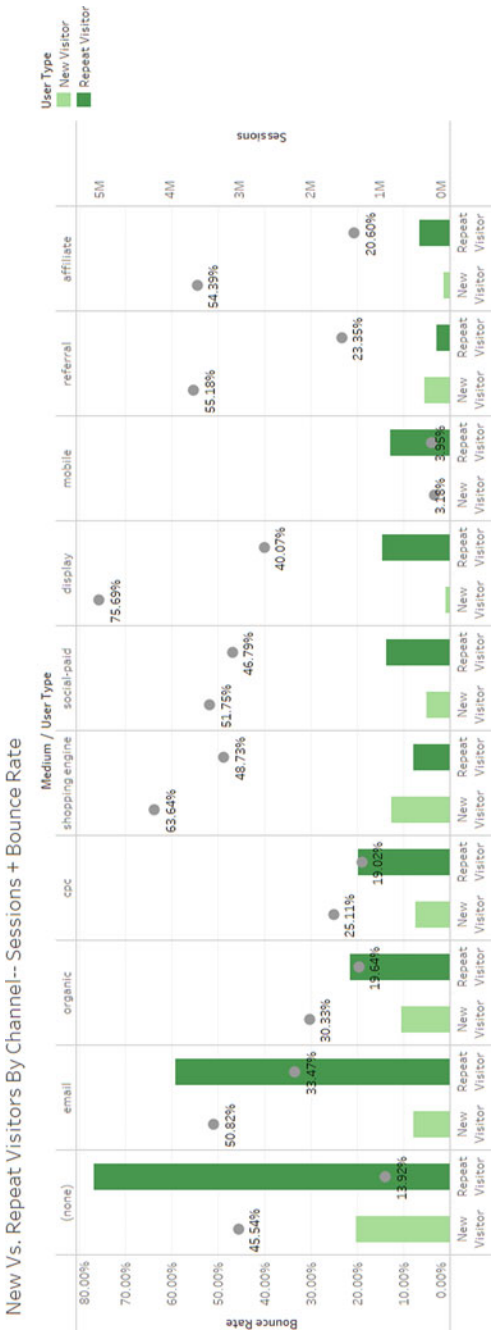


Fig. 6 Sessions and bounce rate by channel (filtered for top 10 across site) categorized by visitor type. Do note that channels referral (e.g., Facebook, YouTube) and shopping engine (e.g., Google, Bing) bring in more sessions from new visitors than repeat

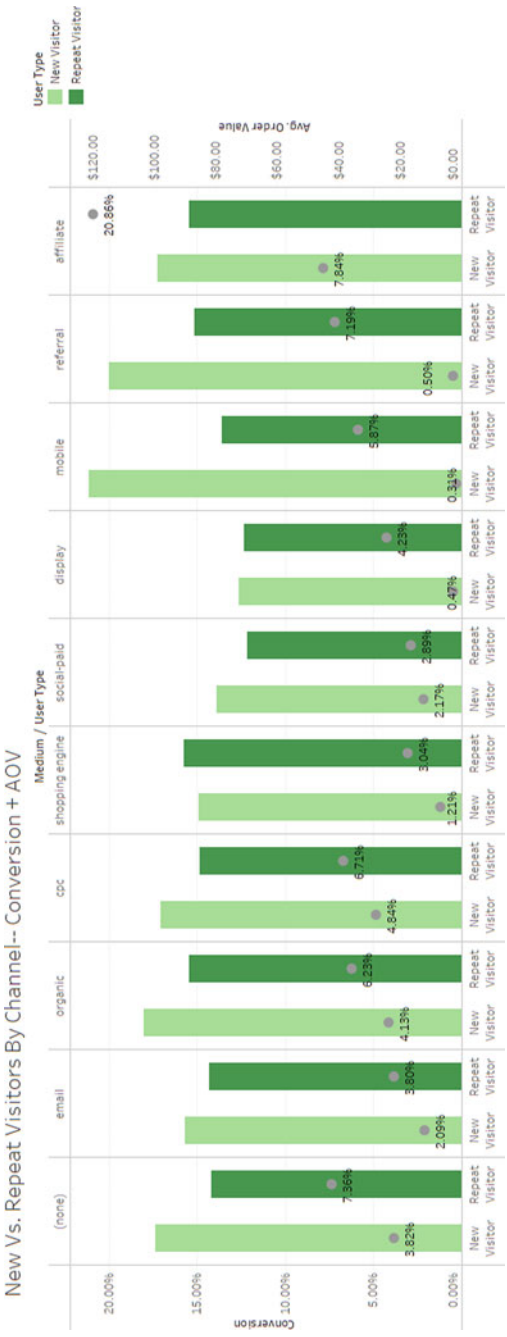


Fig. 7 Conversion and Avg. order value by channel (filtered for top 10 across site) categorized by visitor type. Do note that new visitors across all channels have higher Avg. order value

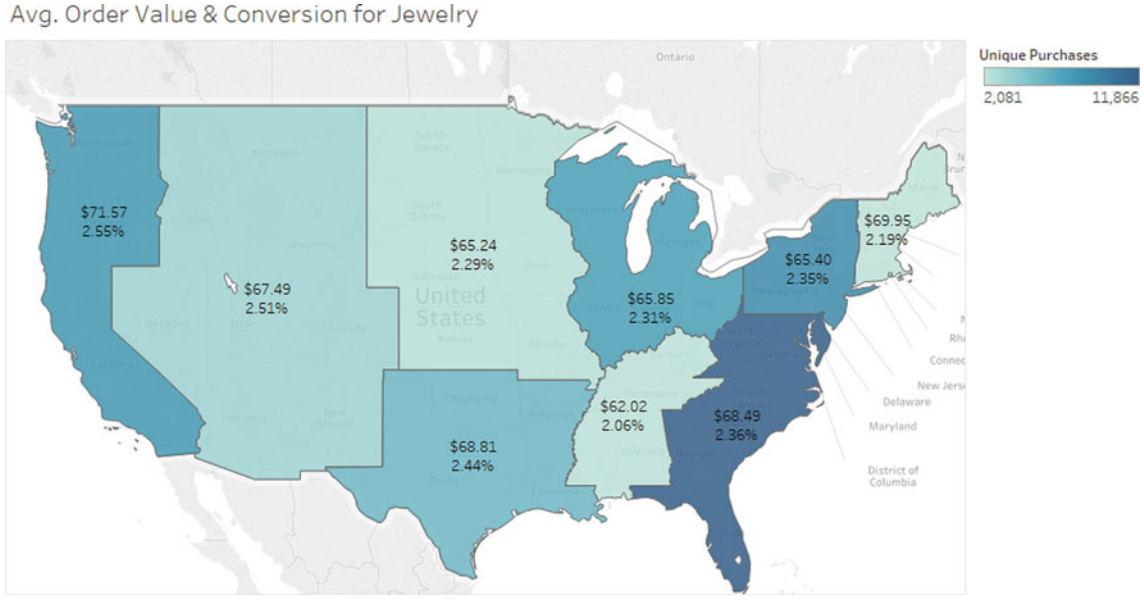


Fig. 8 AOV and conversion for jewellery (specifically necklaces, rings, earrings, beads and jewellery, etc.) products. Max differences observed: AOV– $71.57 versus $62.02, Conversion–2.55% versus 2.06%

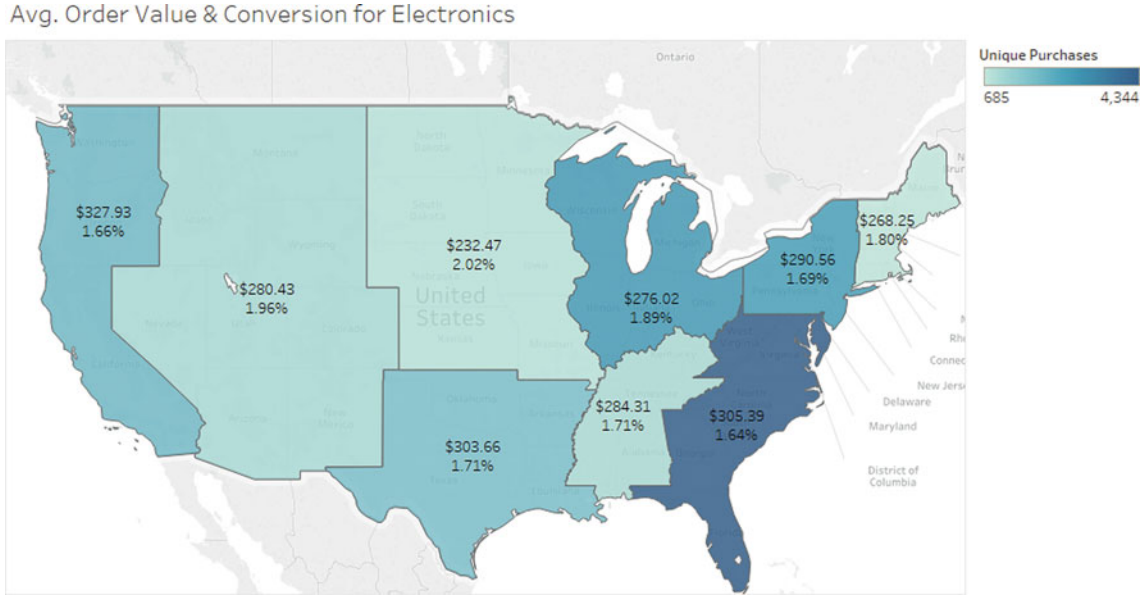


Fig. 9 AOV and conversion for electronics (specifically laptops, computers, computer accessories, TV accessories, cameras, etc.) products. Max differences observed: AOV–$327.93 versus $232.47, Conversion–2.02% versus 1.66%

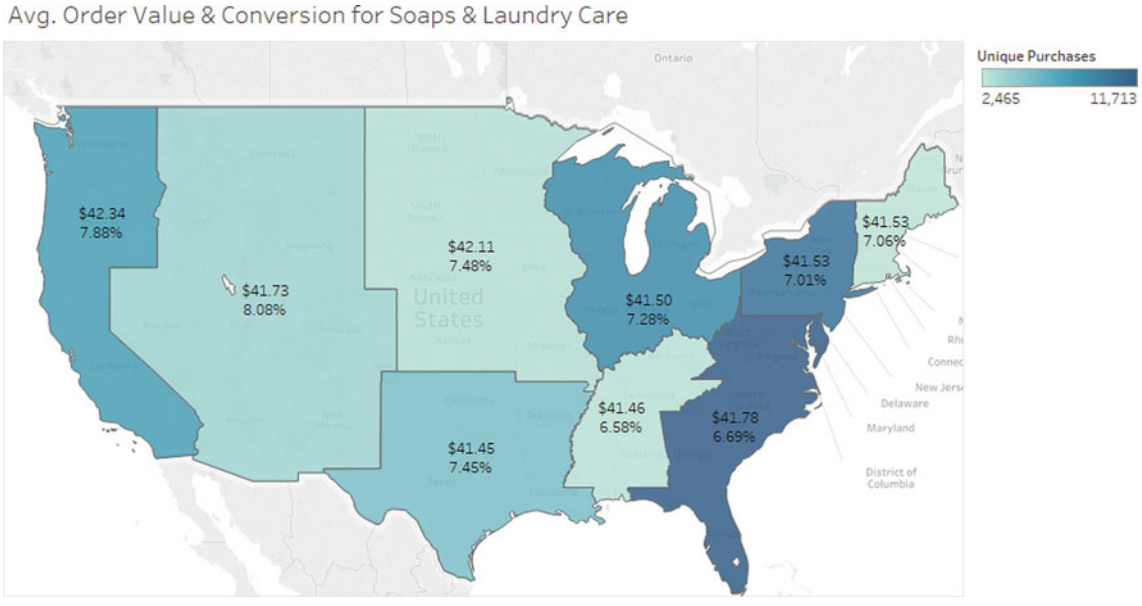
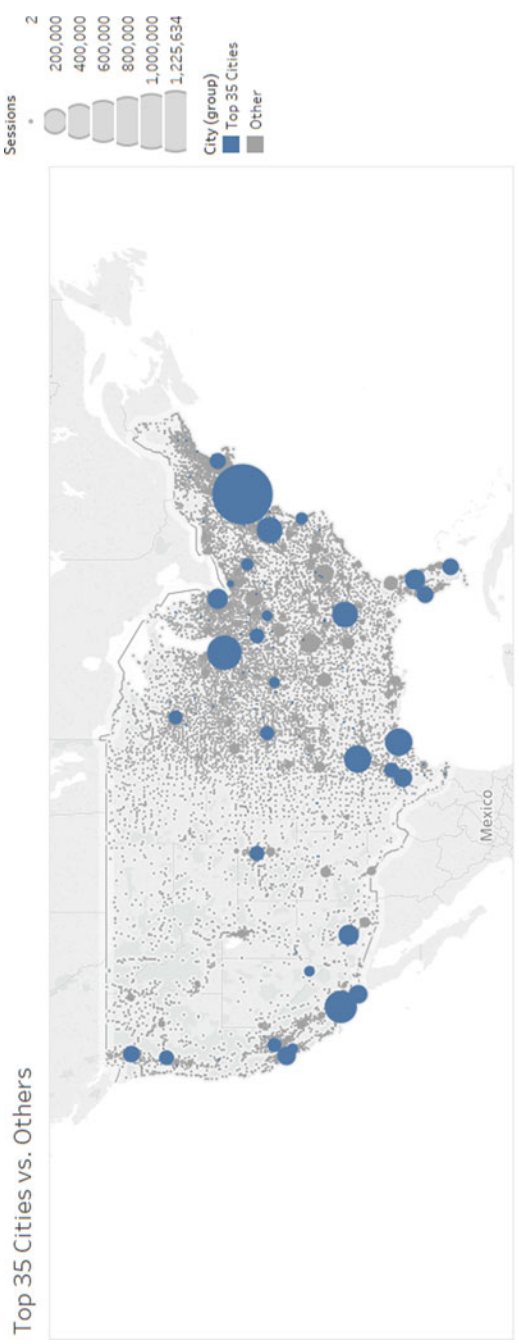


Fig. 10 AOV and conversion for soaps and laundry care products. Max differences observed: AOV–$71.57 versus $62.02, Conversion– 2.55% versus 2.06%



City (group)	Conversion	Per Session ..	Avg. Order V..	% Mobile Se..	Avg. Session..	Bounce Rate
Top 35 Cities	3.93%	$3.50	$88.95	65.04%	5.81	33.44%
Other	5.79%	$4.83	$83.45	47.84%	6.94	28.18%

Fig. 11 Key performance indicators of top 35 cities versus rest

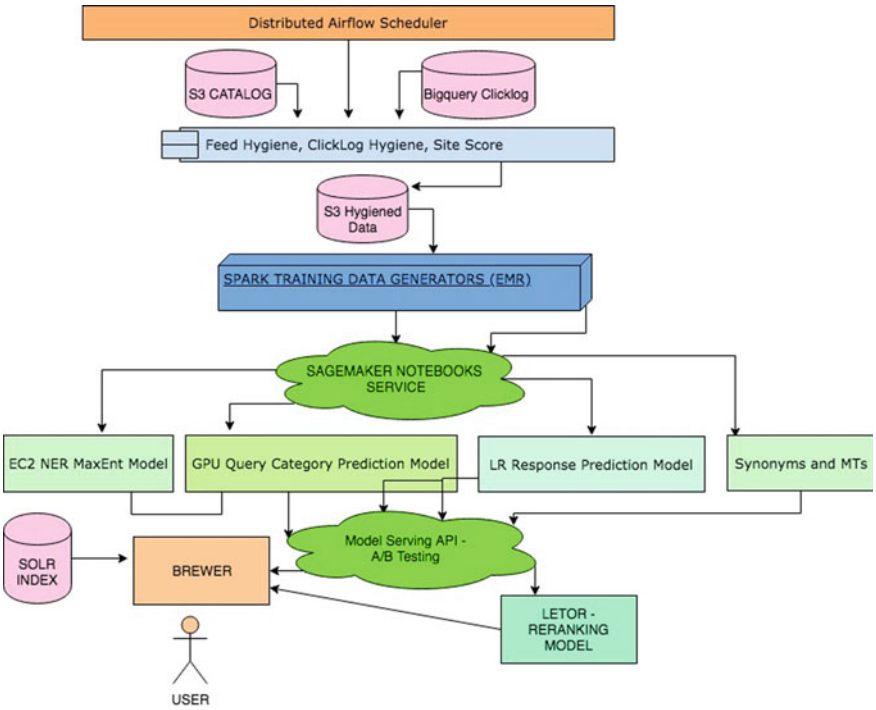


Fig. 12 ML architecture

- converge_threshold: a threshold to decide model training convergence.
- filtering_threshold: a threshold to filter out less frequent features.
- new_feature_bonus: allows new features to pass the filtering more easily.
- filter_by_total_events: whether to filter by total or average events.
- param_alpha, param_beta, param_gamma: learning step size parameters.
- prior: average response value.
- minimum_update_threshold: filter out updating factors below this threshold.
- minimum_update_events: filter out updating factors with too few events.

7 A/B Framework

When any response prediction or ranking model tries to change some of the system parameters, we need to measure its impact. How do we measure impacts of the model? To this effect, we have designed an inhouse A/B experimentation framework which works on these principles. Working within the space of incoming traffic and the system parameters, we have three key concepts:

- A domain is a segmentation of traffic.

- A layer corresponds to a subset of the system parameters or competing algorithms which have the same optimization criterion.
- An experiment is a segmentation of traffic where zero or more system parameters can be given alternate values that change the path how the incoming request is processed.

Domains contains layers and layers contains experiments. Domains let us have different partitioning of system parameters. For example, we can have a non-overlapping domain where we can change lots of parameters that might not normally be used together. Since a layer corresponds to subset of parameters that cannot be changed independently, experiments within a layer cannot run together for a request. Experiments from different layers however are free to run together on a request. Experiment allocation within a layer is based on some diversion type available in the request. This is to ensure stickiness with respect to that type. Diversion type can be device ID or browser cookie or query string**.** Experiments that target user behaviour would usually want that the users do not pop in and out of the experiment and therefore need user stickiness. Some experiments do not need user level stickiness but would operate at query level and therefore would want stickiness at query level. We partition the traffic space into say 1000 buckets and assign experiments within a layer to bucket ranges. The request is mapped to a bucket using a function of the bucket ID and layer modulus the number of buckets, e.g., f(id, layer) % 1000. Note that this function takes layer as an argument to ensure that the experiments in different layers are independently diverted.

The graph below, Fig. 13 shows the result of A/B experimentation between a control group and a response prediction model group. Unbxd clearly shows an uptick of 10% in CTR for torso and tail queries in terms of search volumes (where torso and tail refers to terciles of search volume distribution) and AOV for an American site where the users where shown search results ranked by the response predictor algorithm by the probability of click.

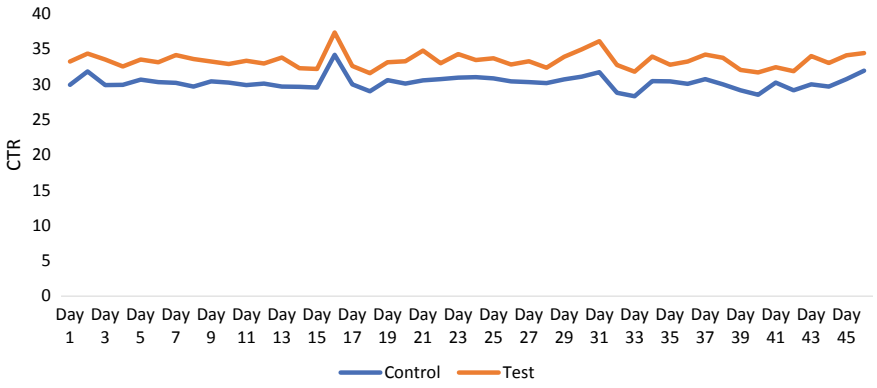


Fig. 13 Control versus test in A/B setting

8 Future Work

From the above feature-based response prediction model, we have been able to both personalize the search results for a user given a query and improve the CTR and CVR which is the revenue tracking metrics of the search business. Through feature-based ranking models, we have mostly captured the general trends of clickability of a product and improved the business performance metrics by driving a higher average order value. However, we have not explored the option of serendipity or cross learning when it comes to surprising the user or providing related product recommendation in the same search session. Window shopping and serendipity shopping is another paradigm which is also known to improve engagement of a shopper and a site. In literature, cross-selling products, "bought also bought", "viewed also viewed" are common basis for recommendations. In search however since user's context is set through a query, we cannot drift afar, but using an epsilon greedy approach or multi-armed bandits, we can exploit our feature-based predictions and explore with a subtle mix of random predictions. This would be the next set of ranking algorithms that we look forward to working on in the future.

References

Cheng, H., & Cantu-Paz, E. (2010). Personalized click prediction in sponsored search. https://www.wsdm-conference.org/2010/proceedings/docs/p351.pdf.

Guo, H., Tang, R., Ye, Y., Li, Z., & He, X. (2017). DeepFM: A factorization-machine based neural network for CTR prediction. https://arxiv.org/pdf/1703.04247.pdf.

Kumar, R., Kumar, M., Shah, N., & Faloutsos, C. (2018). Did we get it right? Predicting query performance in e-commerce search. https://sigir-ecom.github.io/ecom18Papers/paper23.pdf.

Tze Cheng, H., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhya, H., et al. (2016). Wide and deep learning for recommender systems. https://arxiv.org/pdf/1606.07792.pdf.

Zhou, G., Song, C., Fan, Y., Zhu, X., Zhu, H., Ma, X., et al. (2018). Deep interest network for click-through rate prediction. https://arxiv.org/pdf/1706.06978.pdf.