Analysis of E-commerce Ranking Signals via Signal Temporal Logic

Tommaso Dreossi
Amazon Search
Palo Alto, California, USA
dreossit@amazon.com

Giorgio Ballardin
Amazon Search
Palo Alto, California, USA
giobal@amazon.com

Parth Gupta
Amazon Search
Palo Alto, California, USA
guptpart@amazon.com

Jan Bakus
Amazon Search
Palo Alto, California, USA
jbakus@amazon.com

Yu-Hsiang Lin
Amazon Search
Palo Alto, California, USA
yuhsianl@amazon.com

Vamsi Salaka
Amazon Search
Palo Alto, California, USA
vsalaka@amazon.com

The timed position of documents retrieved by learning to rank models can be seen as signals. Signals carry useful information such as drop or rise of documents over time or user behaviors. In this work, we propose to use the logic formalism called Signal Temporal Logic (STL) to characterize document behaviors in ranking accordingly to the specified formulas. Our analysis shows that interesting document behaviors can be easily formalized and detected thanks to STL formulas. We validate our idea on a dataset of 100K product signals. Through the presented framework, we uncover interesting patterns, such as cold start, warm start, spikes, and inspect how they affect our learning to ranks models.

1 Introduction

Learning to rank (LTR) [13] is family of machine learning techniques to solve ranking problems. Given a training set of queries and documents sorted by some relevance degree, the goal of an LTR model is to learn a model that, given a query, sorts documents while maximizing the relevance score. The relevance score can be either manually prepared from human labelling or automatically derived from users interactions logs. Automatic labelling is best suited for large amount of data since it both relieves humans from the labelling task and objectively measures the user preferences. In the e-commerce context, clickthrough rate, that is the rate of the clicks received by a retrieved product for a given query, is a common example of automatic relevance score. In general, we call behavioral signals the signals generated by user-ranker interactions that can be used as relevance scores. A drawback of using behavioral signals for training ranking models is that products with low user interaction, such as new or rare products, lack of behavioral signals and hence are ranked as irrelevant. It takes time to gather enough information so that the ranker can place the products in their right position. This also leads to the causality dilemma: No behavioral signals causes poor ranking which in turn results in new products having a reduced likelihood of accruing behavioral data. This particular phenomenon is referred to as cold start which leads to a poor customer experience.

Cold start is just one particular problem rising from learning from behavioral signals. Other examples are warm start (product ranks too high too early), instability (sudden spikes or ditches in ranking position), or uncertainty (the ranker does not know how to rank due to lack of user interaction). These are examples of well know unwanted phenomena that an LTR model should avoid. Being able to isolate and measure these phenomena plays an important role in designing LTR rankers and preventing unwanted signal patterns.

T. Dang and S. Ratschan: 6th International Workshop on
Symbolic-Numeric Methods for Reasoning about CPS and IoT (SNR 2020)
EPTCS 331, 2021, pp. 33–42, doi:10.4204/EPTCS.331.3

Unluckily, there are few efficient tools for isolating known signal patterns. Some examples are: probabilistic anomaly detection methods [9], where a signal is considered to be an outlier accordingly to its probability of being observed; k-means-based approaches [12], where the distance of a signal from cluster centroids is a measure of diversity; ad-hoc classification or regression models [18], where a model is trained to detect a specific behavior. Note how these techniques either do not provide the flexibility for identifying a particular signal pattern or require to build rigid ad-hoc solutions.

In this work we present use of Signal Temporal Logic (STL) [14], as a tool for isolating and analyzing product and behavioral signals in the LTR context. In particular, we show how STL can be used to formally characterize well known signal behaviors, such as the undesirable cold start, warm start, product instability, etc., isolate them from a collection of signals, and afterwards analyze them so that we can take countermeasures in our LTR model. Intuitively, STL is temporal logic [17] (i.e., mathematical formalism for representing and monitoring properties involving time) particularly suitable for characterizing real-valued signals defined over real-time intervals. Examples of signals patterns in natural language that can be easily described by STL formulas are "some products always rank at position 1" or "every product will eventually rank at position 1".

The main contributions of this work are:

- Propose STL as a flexible tool for formally describing known signal patterns as formal logic formulas;
- Define a library of STL formulas encoding common unwanted signal behaviors in the LTR context (such as cold start, warm start, sudden ranking ditches or spikes, instability, etc.);
- Cluster a large set of product signals collected from a popular e-commerce website using the defined STL properies and analyze how performance metrics, such as clicks, impressions, and purchases are distributed across different clusters and product categories;
- Compare the expressivity and succinctness of STL with common signal clustering/filtering tools (such as k-means, Pandas queries, propositional logics, etc.)

Researchers have widely used temporal logics for formal verification purposes, where hardware and software systems are tested against properties that characterize the system's correctness [8, 10]. Over the years, researchers defined several types of temporal logics that usually vary in expressive power. Some examples are Linear Temporal Logic [17], Computation Tree Logic [3], or Metric Temporal Logic [11]. STL has been successfully applied to the cyber-physical system domain [8, 10], specifically for monitoring and testing devices that involve physical and computational components, such as drones [4, 16], self-driving vehicles [7, 19], and even medical devices [2]. The success of STL in these domains is mainly due to its expressiveness and the efficiency of tools, such as S-TaLiRo [1] and Breach [5], for reasoning with STL formulas. To the best of our knowledge, this is the first time that STL is used in the context of Information Retrieval and LTR rankers.

The paper is organized as follows. In Sec. 2 we explain the theoretical promise of STL. In Sec. 3 we define a library of STL properties useful for analyzing ranking signals. In Sec. 3.4 we also compare the expressiveness and succinctness of STL to other common formalisms. In Sec. 4, we present experiments on evaluating and analyzing the defined STL ranking properties on product signals collected from a popular e-commerce website. Finally, we draw concluding remarks in Sec. 5.

2 Signal Temporal Logic

In this section we define the Signal Temporal Logic [14], a formalism particularly suitable for properties of real-valued signals.

A signal is a function $s : D \to S$ with $D \subseteq \mathbb{R}_{\geq 0}$ an interval and $S \subseteq \mathbb{R}$. A trace $w = (s_1, \ldots, s_n)$ is a tuple of real-values signals defined over D.

Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be a set of predicates $\sigma_i : \mathbb{R}^n \to \mathbb{B}$, with $\sigma_i := p_i(x_1, \ldots, x_n) \lhd 0$, $\lhd \in \{<, \leq\}$, and $p_i : \mathbb{R}^n \to \mathbb{R}$.

Definition 1 (STL syntax)**.** *An STL formula is defined by the grammar:*

$$\varphi := \sigma | \neg\varphi | \varphi \wedge \varphi | \varphi U_I \varphi \tag{1}$$

where $\sigma \in \Sigma$ and $I \subset \mathbb{R}_{\geq 0}$ is a closed non-singular interval.

A shifted interval I is defined as $t + I = \{t + t' | t' \in I\}$.

Definition 2 (STL qualitative semantics)**.** *Let w be a trace, $t \in \mathbb{R}_{\geq 0}$, and φ be an STL formula. The qualitative semantics of φ is defined as follows:*

$$\begin{aligned}
&w,t \models \top \\
&w,t \models p(x_1, \ldots, x_n) \lhd 0 \textit{ iff } p(w(t)) \lhd 0 \textit{ with } \lhd \in \{<, \leq\} \\
&w,t \models \neg\varphi \textit{ iff } w,t \not\models \varphi \\
&w,t \models \varphi_1 \wedge \varphi_2 \textit{ iff } w,t \models \varphi_1 \textit{ and } w,t \models \varphi_2 \\
&w,t \models \varphi_1 U_I \varphi_2 \textit{ iff } \exists t' \in t + I \textit{ s.t. } w,t' \models \varphi_2 \\
&\qquad\qquad \textit{and } \forall t'' \in [t,t'] w,t'' \models \varphi_1
\end{aligned} \tag{2}$$

The peculiarity of STL is the interval-decorated until operator. Intuitively, $\varphi_1 U_I \varphi_2$ holds if φ_1 is true until φ_2 becomes true at a time instant in I.

We can define other common operators as syntactic abbreviations: $\bot := \neg\top, p(x) > 0 := \neg(p(x) \leq 0), p(x) \geq 0 := \neg(p(x) < 0), \varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2), \varphi_1 \implies \varphi_2 := \neg\varphi_1 \vee \varphi_2, F_I\varphi := \top U_I \varphi, G_I\varphi := \neg F_I \neg\varphi$.

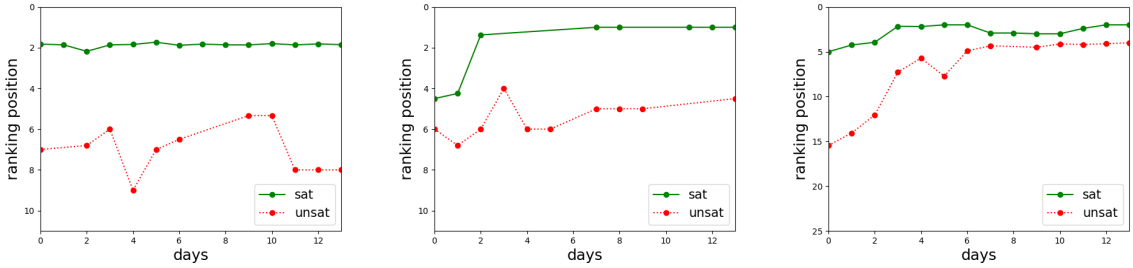
The "eventually" operator $F_I\varphi$ (also called "future") forces φ to be true at least once in I. The "always" operator $G_I\varphi$ (also called "globally") requires φ to be always true in I. We will omit the interval decoration I from temporal operators when the property predicates over the entire life of trace (e.g., we write $G\varphi$ as a shorthand for $G_{[0,+\infty]}\varphi$).

We say that a trace w satisfies φ if $w, 0 \models \varphi$. The satisfaction of a formula can be determined by recursively computing the satisfaction of its subformulas [14]. The evaluation of an STL formula is linear in the signal length [6].

STL has also several alternative semantics. The most popular is the qualitative semantics that, instead of a boolean satisfaction value, returns a real value encoding how robustly a trace satisfies a formula (i.e., the distance from satisfaction or violation). In this work, we consider only the qualitative semantics since we are interested in the binary classification of our signals. For more details on qualitative semantics see, e.g., [6].

3 Ranking Signal Properties

We now define three groups of specifications: 1) local properties, that predicate over parts of signals, 2) global properties, that involve whole signals, and 3) correctness properties, that monitor faulty behaviors such as missing data.



(a) $flat\_start$ (Eq. 3) (b) $cold\_start$ (Eq. 4) (c) $steady\_state$ (Eq. 6)

Figure 1: Ranking signals that satisfy (green) and do not satisfy (red) the STL specifications $flat\_start$ (Eq. 3), $cold\_start$ (Eq. 4), and $steady\_state$ (Eq. 6).

In the following experiments, we assume that the average daily ranking position of a products is given by a function $x : \mathbb{N} \to \mathbb{R}_{\geq 1}$, i.e., given a day $t_i \in \mathbb{N}$, $x(t_i)$ is the daily average position of a document. Let $x'(t_i)$ be the discrete-time derivative of x, i.e., $x'(t_i) = (x(t_{i+1}) - x(t_i))/(t_{i+1} - t_i)$.

3.1 Local Properties

We begin with properties that predicate over the initial days of products. We are interested in determining if products are ranked at a stable position (flat start) or if they gain (cold start) or loose (warm start) positions in the days after launch:

$$flat\_start := G_{[0,w]}(|x'| < \varepsilon) \tag{3}$$

$$cold\_start := G_{[0,w]}(x' \leq 0) \wedge F_{[0,w]}(x' < 0) \tag{4}$$

$$warm\_start := G_{[0,w]}(x' \geq 0) \wedge F_{[0,w]}(x' > 0) \tag{5}$$

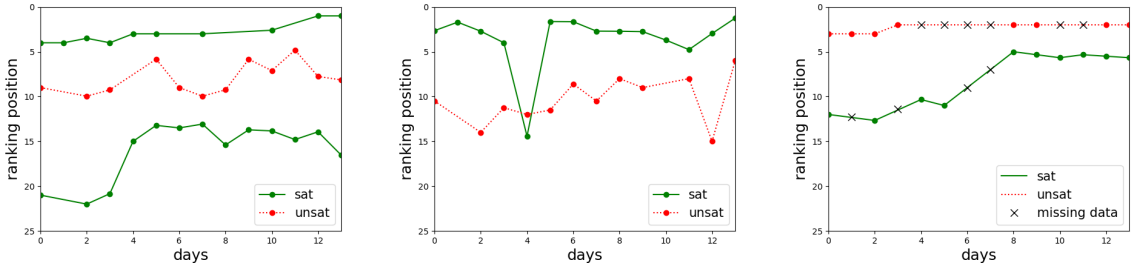
where $w \in \mathbb{N}$ and $\varepsilon \in \mathbb{R}_{\geq 0}$ are tunable parameters that define the length of the initial time window and noise tolerance.

The $flat\_start$ specification (Eq. 3) forces the first derivative x' to be always ε-close to zero (we include the ε tolerance to account for noise). The ranking signals that satisfy this specification are those that found their steady ranking position on launch day, i.e., those that experienced a flat start. Similarly, we define $cold\_start$ (Eq. 4) and $warm\_start$ (Eq. 5) specifications by requiring the ranking signals to always decrease/increase. The right-most conjunct ($F_{[0,w]}$) forces the signals to strictly grow/decrease at least once.

Fig. 1 shows some examples of trajectories that do and do not satisfy (green and red, respectively) the flat and cold start STL specifications with parameters $w = 3$ and $\varepsilon = 1$ for $flat\_start$ and $w = 3$ and $\varepsilon = 0$ for $cold\_start$.

3.2 Global Properties

We now increase the scope of our specifications by predicating over entire ranking signals.



(a) *reach* (Eq. 7) (b) *ditch* (Eq. 8) (c) *no_long_miss* (Eq. 11)

Figure 2: Ranking signals that satisfy (green) and do not satisfy (red) the STL specifications *reach* (Eq. 7), *ditch* (Eq. 8), and *no_long_miss* (Eq. 11).

In learning to rank, we often assume that a document reaches a steady state ranking position after an initial period during which behavioral data is collected. The following STL formula characterizes the existence of a steady state:

$$steady\_state := F_{[0,w]} G(|x'| < \varepsilon) \tag{6}$$

where $w \in \mathbb{R}_{\geq 0}$ defines the maximum stabilization time and $\varepsilon \in \mathbb{R}_{\geq 0}$ the tolerance to noise. The *steady_state* formula holds if there is a day in $[0, w]$ after which $|x'|$ is always smaller than ε, i.e., the trajectory maintains a steady state with ε tolerance. Fig. 1(c) shows two trajectories evaluated on *steady_state* with parameters $w = 3$ and $\varepsilon = 1$. The green signal satisfies the specification because it stabilizes after day 3 at position 2. On the other hand, the red trajectory does not satisfy the specification because on day 4 it experiences a drop in positions and thus it is not stable after day 3.

Next, we define a liveness property that checks if a product that reaches a certain ranking interval eventually hits a critical position:

$$reach := G((x < s) \implies F(x = r)) \tag{7}$$

with $s, r \in \mathbb{R}$. For instance, with parameters $s = 10$ and $r = 1$ we check whether products that reached the top 10 positions eventually rank in first position too. Fig. 2(a) depicts two signals that satisfy the *reach* specification and one that does not with parameters $s = 10$ and $r = 1$. The upper green signal satisfies the specification because it enters the $[1, 10]$ ranking interval and eventually reaches position 1 on day 12. The lower green signals also satisfies the requirement since it never enters the $[1, 10]$ range. However, the red signal does not satisfy the specification because its values become smaller than 10 but never reach the first position.

Finally, we analyze the stability of products. We define two specifications that capture signals with large bouncing drops or rises in a short time frame:

$$ditch := F((x' > d) \wedge F_{[0,w]}(x' < d)) \tag{8}$$

$$spike := F((x' < d) \wedge F_{[0,w]}(x' > d)) \tag{9}$$

The parameters $d, w \in \mathbb{R}_{>0}$ determine the signal drop/rise width and amplitude, respectively. These two formulas check if at any point in time there is a drop/rise of at least d positions followed by a rise/drop of at least d position within w days. Fig. 2(b) shows some signals evaluated on *ditch* with parameters $d = 10$ and $w = 2$. The green signal satisfies the specification since it experiences a ditch of at least 10 positions on day 3. The red signal does not satisfy the property since its ditch is not deep enough.

3.3 Correctness Properties

Finally, we hypothesize a scenario where some data might miss, i.e., the ranking position might be unknown. Let -1 denote the ranking position of a product on a day for which data is missing. The atomic predicate that holds if the ranking position is unknown is $miss := x = -1$.

In learning to rank systems, the first days after a product launch are crucial for the collection of behavioral data and the subsequent correct ranking. Hence, we do not want too much missing data in the days after a product launch:

$$no\_init\_miss := \neg(G_{[0,w]} miss) \tag{10}$$

where $w \in \mathbb{N}$ defines the initial time window. *no_init_miss* ensures that there are no w consecutive initial days of missing data. We can also extend this requirement to any part of a ranking signal and not just to its prefix. We define a specification that ensures that are no windows with too many consecutive days with missing data:

$$no\_long\_miss := G(miss \implies F_{[0,w]} \neg miss) \tag{11}$$

where $w \in \mathbb{N}$. *no_long_miss* ensures that if there is a missing data day, then eventually within w days there will be non missing data day. In Fig. 2(c), the red signal does not satisfy *no_long_miss* for $w = 3$, since after day 4 there are 4 consecutive days of missing data. On the other hand, the green signal satisfies the specification since it never has 3 consecutive days of missing data.

3.4 On STL's Succinctness and Efficiency

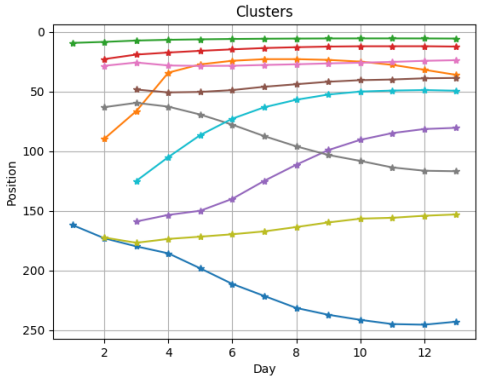


Figure 3: k-means (with k= 10) centroid signals for our product ranking signals dataset. From this analysis it seems most products have smooth trajectories. However, our STL-based analysis reveals that more that 50% of the signals experiences a spike or ditch of at least 10 ranking positions.

Before proceeding with the evaluation of our STL specifications, it is worth noticing how STL is both succinct and efficient when compared with standard logics.

For instance, we ran the clustering algorithm k-means (with k=10) on our dataset of product signals with the goal of mining the most representative signals. Fig. 3 shows the centroid signals of the clusters obtained by running k-means (with k=10) on product ranking signals dataset. From this analysis it seems most products have smooth trajectories. However, as we will later discover in Sec. 4 , our STL-based analysis reveals that 50% of all the analyzed products in this test experience a shift in ranking position over two consecutive days. This shows how clustering techniques might fail in isolating important signal patterns.

We also compared STL with classic logics and query formalisms. For instance, the translation of the *ditch* property (Eq. 8) in propositional logics is $\bigvee_{i=0}^{T}(xgd_i \wedge \bigvee_{j=0}^{w} xld_j)$ where $xgd_i := x'(i) > d$ and $xld_i := x'(i) < d$. For this formulation, we must know in advance the length T of the signal. In addition, it involves $T(1+w)$ operators against the 3 operators of Eq. 8.

The encoding of Eq. 8 in a first-order logic, e.g., semi-algebraic logics, would result in a formula with less operators $\exists t(x'(t) > d \wedge \exists t'(t \leq t' \leq t + w \wedge x'(t') < d))$ which is still less compact than Eq. 8 and

less efficient to evaluate. The evaluation of semi-algebraic formulas is generally doubly exponential in the number of quantifier alternations while the evaluation of an STL formula is linear in the signal length [6].

For completeness, we also include the translation of Eq. 8 into a `pandas` [15] query, a Python library for data manipulation and analysis. Let df be the dataframe with our signals where pos_i is the column with the ranking position on day i. The `pandas` query that isolates the signals that satisfy the $ditch$ property is $df[((df.pos_0 > d)\ \&\ (df.pos_1 < d \mid \ldots \mid df.pos_w < d)) \mid \ldots \mid ((df.pos_{T-w} > d)\ \&\ (df.pos_{T-w+1} < d \mid \ldots \mid df.pos_T < d))]$. The structure of this query is similar to the propositional encoding. Also in this case, we are dealing with a long query, we need to know in advance the length of the signal, and any parametric change of our requirement affects the query's structure. Finally, note that the encoding formula length explosion occurs for any specification that involves temporal operators and not just this particular case.

4 Experimental Evaluation

We now evaluate the STL specifications defined in Sec. 3 against a dataset of product signals. We conduct two analyses:

1. *Product categories*: Explore the correlation between query-product signals and product categories;
2. *Performance metrics*: Analyze how metrics such as impressions, clicks, and purchases distribute across different clusters of query-product signals.

Our analyses highlight how STL can be used to easily isolate unwanted signals behaviors. We will also discover that not all product categories are equally affected by LTR anomalies (e.g., cold start, instability, etc.) and that clicks, impressions, and purchases are not evenly distributed across signal patterns. Our dataset contains 100K examples from ten different product categories. Each data point contains the product's position for 14 days, number of searches, clicks, and purchases.

For the evaluation of the specifications, we relied on the `py-metric-temporal-logic library` [20]. Specifically, we implemented a pandas [15] user-defined function that, for each entry of our dataset, invokes `py-metric-temporal-logic` and evaluates our STL specifications.

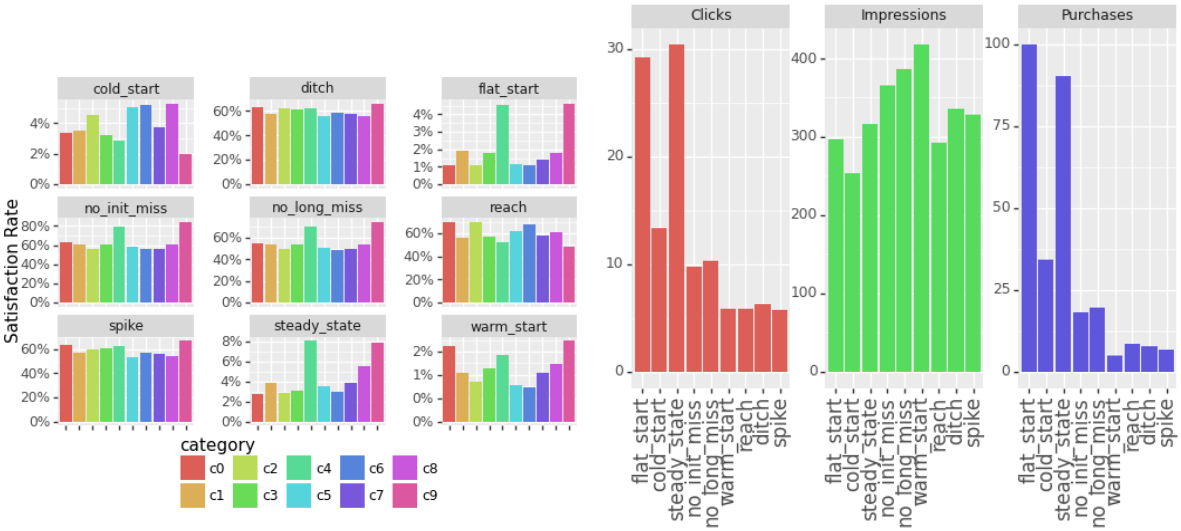
4.1 Product Categories

Do satisfaction rates of our STL specifications vary across categories? To address this question we compute the percentage of product signals that satisfies a given STL specification for each product category and STL specification combination. Results, reported in Fig. 4(a), highlight how satisfaction rates are not equally distributed across different categories.

c9 and c4 are the categories less affected by cold start with a 2% satisfaction rate opposed to c8, c7, and c5 with satisfaction rates 5%. c4 and c9 have also the highest flat start and steady state rates (5% and 8%, respectively) in contrast to the other categories. This means that c4 and c9 are the categories whose products most frequently find their natural ranking position from launch day and keep it constant over time.

Interestingly, c9, c4, together with c0, are the most affected by warm start (1.50%) suggesting that products from these categories might rank too high after launch or might include a high number of low engagements products (e.g., spam).

Finally, spike and ditch satisfaction rates are almost equally distributed. Remarkably, 60% of all the analyzed products experience a drop or ditch of at least 10 ranking position within two consecutive days.



(a) (b)

Figure 4: Specification satisfaction rates across product categories (left) and average total impressions, clicks, and purchases garnered by query-products that satisfy STL specifications (right).

4.2 Performance Metrics

We now analyze how impressions, clicks, and purchases are distributed across different specifications. We compute the average number of impressions, clicks, and purchases for query-documents tuples that satisfy a given STL specification. Fig. 4(b) shows the obtained distributions.

The signals within this test that garner the highest and lowest average number of impressions (400 and 250, respectively) are those that satisfy the warm start and cold start specifications, respectively. The most clicked products satisfy flat start and steady start (30 clicks) while the least clicked ones satisfy warm start, reach, ditch, and spike (5 clicks). The products that collect highest purchases satisfy flat start and steady state (90) followed by cold start (35), while the products with lowest purchases are those affected by warm start (5).

This analysis suggests that products affected by warm start, despite receiving the highest number of impressions, tend not to be clicked and consequently gather low purchases. We could speculate that warm start products are either not relevant to customer's searches or are low quality products that do not attract customer's attention. cold start products show the symmetric phenomenon. They receive a low number of impressions, they are clicked twice as much as warm start and gather 5x purchases compared to warm start. This might mean that cold start products are eventually discovered and purchased by customer despite being initially poorly ranked.

Outliers are flat start and steady state specifications. They receive 3x clicks (30 vs 10) and 4x purchases (100 vs 25) compared to the second mostly clicked and highest purchases cold start, no init miss, and no long miss. Products that have early flat and steady positions tend to rank very high and consequently collect the highest amount of clicks and purchases.

Note how both the product categories and performance metrics STL analyses revealed interesting behaviors of our learning to rank model. The obtained insights can be used, for instance, to rebalance our training sets by focusing on particular product segments or redesign relevance scores. Note also that these

are just demonstrative examples of how STL can be used to reason over ranking signals. Nothing prevents STL from being applied to more complex temporal properties or more sophisticated analyses.

5 Remarks

In this work, we proposed for the first time STL in the learning to rank context to cluster and analyze product ranking signals. We defined a library of properties that characterize unwanted product behaviors and analyzed the distribution of the satisfaction of properties over a dataset of 100K product signals. Our analysis showed how STL can be used to reason over ranking traces and reveal insights on the model under study. In the future, we plan to explore STL for online monitoring where the real-time detection of faulty behaviors can be used to trigger alarms and actuate repairing procedures.

References

[1] Yashwanth Annpureddy, Che Liu, Georgios Fainekos & Sriram Sankaranarayanan (2011): *S-taliro: A tool for temporal logic falsification for hybrid systems*. In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, pp. 254–257, doi:10.1007/978-3-642-19835-9_21.

[2] Ezio Bartocci & Pietro Lió (2016): *Computational modeling, formal analysis, and tools for systems biology*. *PLoS computational biology* 12(1), p. e1004591, doi:10.1371/journal.pcbi.1004591.t001.

[3] Edmund M. Clarke & E. Allen Emerson (1982): *Design and synthesis of synchronization skeletons using branching time temporal logic*. In Dexter Kozen, editor: *Logics of Programs*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 52–71, doi:10.1137/0201010.

[4] Ankush Desai, Tommaso Dreossi & Sanjit A Seshia (2017): *Combining model checking and runtime verification for safe robotics*. In: *International Conference on Runtime Verification*, Springer, pp. 172–189, doi:10.1007/978-3-642-35632-2_18.

[5] Alexandre Donzé (2010): *Breach, a toolbox for verification and parameter synthesis of hybrid systems*. In: *International Conference on Computer Aided Verification*, Springer, pp. 167–170, doi:10.1007/3-540-36580-X_22.

[6] Alexandre Donzé, Thomas Ferrere & Oded Maler (2013): *Efficient robust monitoring for STL*. In: *Computer Aided Verification*, Springer, pp. 264–279, doi:10.1016/S0019-9958(65)90241-X.

[7] Tommaso Dreossi, Alexandre Donzé & Sanjit A Seshia (2019): *Compositional falsification of cyber-physical systems with machine learning components*. *Journal of Automated Reasoning* 63(4), pp. 1031–1053, doi:10.1007/BF01475864.

[8] Georgios E Fainekos & George J Pappas (2009): *Robustness of temporal logic specifications for continuous-time signals*. *Theoretical Computer Science* 410(42), pp. 4262–4291, doi:10.1016/j.tcs.2009.06.021.

[9] Markus Goldstein & Seiichi Uchida (2016): *A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data*. *PloS one* 11(4), p. e0152173, doi:10.1371/journal.pone.0152173.t006.

[10] Xiaoqing Jin, Jyotirmoy V Deshmukh, James Kapinski, Koichi Ueda & Ken Butts (2014): *Powertrain control verification benchmark*. In: *Hybrid systems: Computation and Control*, ACM, pp. 253–262, doi:10.1145/2562059.2562140.

[11] Ron Koymans (1990): *Specifying real-time properties with metric temporal logic*. *Real-Time Systems* 2(4), pp. 255–299, doi:10.1007/BF01995674.

[12] Moisés F Lima, Bruno B Zarpelao, Lucas DH Sampaio, Joel JPC Rodrigues, Taufik Abrao & Mario Lemes Proença (2010): *Anomaly detection using baseline and k-means clustering*. In: *SoftCOM 2010, 18th International Conference on Software, Telecommunications and Computer Networks*, IEEE, pp. 305–309.

[13] Tie-Yan Liu (2009): *Learning to Rank for Information Retrieval*. Found. Trends Inf. Retr. 3(3), p. 225–331, doi:10.1561/1500000016.

[14] Oded Maler & Dejan Nickovic (2004): *Monitoring Temporal Properties of Continuous Signals*. In Yassine Lakhnech & Sergio Yovine, editors: Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 152–166, doi:10.1007/3-540-45739-9_14.

[15] Wes McKinney (2010): *Data Structures for Statistical Computing in Python*. In Stéfan van der Walt & Jarrod Millman, editors: Python in Science Conference, pp. 51 – 56, doi:10.25080/Majora-92bf1922-00a.

[16] Yash Vardhan Pant, Rhudii A Quaye, Houssam Abbas, Akarsh Varre & Rahul Mangharam (2019): *Fly-by-Logic: A Tool for Unmanned Aircraft System Fleet Planning Using Temporal Logic*. In: NASA Formal Methods Symposium, Springer, pp. 355–362, doi:10.1007/978-3-642-15297-9_9.

[17] A. Pnueli (1977): *The temporal logic of programs*. In: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), pp. 46–57, doi:10.1109/SFCS.1977.32.

[18] Ingo Steinwart, Don Hush & Clint Scovel (2005): *A classification framework for anomaly detection*. Journal of Machine Learning Research 6(Feb), pp. 211–232, doi:10.5555/1046920.1058109.

[19] Cumhur Erkan Tuncali, Georgios Fainekos, Hisahiro Ito & James Kapinski (2018): *Simulation-based adversarial test generation for autonomous vehicles with machine learning components*. In: 2018 IEEE Intelligent Vehicles Symposium (IV), IEEE, pp. 1555–1562, doi:10.1109/IVS.2018.8500421.

[20] Marcell Vazquez-Chanlatte (2019): *mvcisback/py-metric-temporal-logic: v0.1.1*, doi:10.5281/zenodo.2548862.