How to Grow a (Product) Tree
Personalized Category Suggestions for eCommerce Type-Ahead

Jacopo Tagliabue
Coveo Labs, New York, USA
jtagliabue@coveo.com

Bingqing Yu
Coveo, Montreal, CA
cyu2@coveo.com

Marie Beaulieu
Coveo, Quebec, CA
mabeaulieu@coveo.com

Abstract

In an attempt to balance precision and recall in the search page, leading digital shops have been effectively nudging users into select category facets as early as in the type-ahead suggestions. In *this* work, we present **SessionPath**, a novel neural network model that improves facet suggestions on two counts: first, the model is able to leverage session embeddings to provide scalable personalization; second, **SessionPath** predicts facets by *explicitly* producing a probability distribution at each node in the taxonomy path. We benchmark **SessionPath** on two partnering shops against count-based and neural models, and show how business requirements and model behavior can be combined in a principled way.

1 Introduction

Modern eCommerce search engines need to work on millions of products; in an effort to fight "zero result" pages, digital shops often sacrifice *precision* to increase *recall*[1], relying on *Learn2Rank* (Liu, 2009) to show the most relevant results in the top positions (Matveeva et al., 2006). While this strategy is effective in web search, when users rarely go after page one (Granka et al., 2004; Guan and Cutrell, 2007), it is only partially successful in product search: shoppers may spend time browsing several pages in the result set and *re-order* products based on custom criteria (Figure 1); analyzing industry data, up to 20% of clicked products occur *not* on the first page, with re-ranking in approximately 10% of search sessions.

Leading eCommerce websites leverage machine learning to suggest *facets* - i.e. product categories, such as *Video Games* for "nintento switch" - *during* type-ahead (Figure 2): narrowing down candidate products explicitly by matching the selected categories, shops are able to present less noisy result pages and increase the perceived relevance of their search engine. In *this* work we present **SessionPath**, a scalable and personalized model to solve facet prediction for type-ahead suggestions: given a shopping session and candidate queries in the suggestion dropdown menu, the model is asked to predict the best category facet to help users narrow down search intent. A big advantage of **SessionPath** is that it can complement any existing stack by adding facet prediction to items as retrieved by the type-ahead API.

Figure 1: Price re-ordering on *Amazon.com*, showing degrading relevance in the result set when querying for a console - "nintendo switch" - and then re-ranking based on price.

We summarize the main contributions of *this* work as follows:

- we devise, implement and benchmark several models of incremental complexity (as measured by features and engineering requirements); starting from a non-personalized count-based baseline, we arrive at **SessionPath**, an encoder-decoder architecture that explicitly models the real-time generation of paths in the catalog taxonomy;

[1]The "nintendo switch" query for a gaming console returns 50k results on *Amazon.com* at the time of drafting this footnote; 50k results are more products than the entire catalog of a mid-size shop such as **Shop 1** below.

Proceedings of the 3rd Workshop on e-Commerce and NLP (ECNLP 3), pages 7–18

Figure 2: Facet suggestions during type-ahead: shoppers can be nudged to pick a facet *before* querying, to help the search engine present more relevant results.

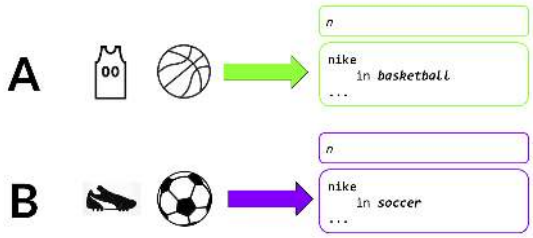

Figure 3: Shoppers in **Session A** and **Session B** have different sport intent, as shown by the products visited. By combining linguistic and behavioral in-session data, **SessionPath** provides in real-time personalized facet suggestions to the same "nike" query in the type-ahead.

- we discuss the importance of false positives and false negatives in the relevant business context, and provide decision criteria to adjust the precision/recall boundary after training. By combining the predictions of the neural network with a *decision module*, we show how model behavior can be tuned in a principled way by human decision makers, without interfering with the underlying inference process or introducing *ad hoc* manual rules.

To the best of our knowledge, **SessionPath** is the first type-ahead model that allows *dynamic* facet predictions: linguistic input and in-session intent are combined to adjust the target taxonomy depth (*sport / basketball* vs *sport / basketball / lebron*) based on real-time shopper behavior and model confidence. For this reason, we believe the methods and results here presented will be of great interest to any digital shop struggling to strike the right balance between precision and recall in a catalog with tens-of-thousands-to-millions of items.

2 Less (Choice) is More: Considerations From Industry Use Cases

The problem of narrowing down the result set before re-ranking is a known concern for mid-to-big-size shops: as shown in Figure 1-A, a common solution is to invite shoppers to select a category facet when still typing. Aside from UX considerations, restricting the result set may be beneficial for other reasons. On one hand, decision science proved that providing shoppers with *more* alternatives is actually less efficient (the so-called "paradox of choice" (Scheibehenne et al., 2010; Iyengar and Lepper, 2001)) - insofar as **SessionPath** helps avoiding unnecessary "cognitive load", it may be a welcomed ally in fighting irrational decision making; on the other, by restricting result set through facet selection, the model may reduce the long-tail effect of many queries on product visibility: when results are too many and items frequently changed, standard *Learn2Rank* approaches tend to penalize less popular items (Abdollahpouri et al., 2017; Anderson, 2006), which end up buried among noisy results far from the first few pages and never collect enough relevance feedback to rise through the top.

In *this* work, we extend industry best practices of facet suggestion in type-ahead by providing a solution that is dynamic in two ways: i) given the same query, session context may be used to provide a contextualized suggestion (Figure 3); ii) given two queries, the model will decide in real-time how deep in the taxonomy path the proposed suggestion needs to be (Figure 4): for some queries, a generic facet may be optimal (as we do not want to narrow the result set *too much*), for others a more specific suggestion may be more suitable. Given the natural trade-off between *precision* and *recall* at different depths, Section 7.2 is devoted to provide a principled solution.

3 Related Work

Facet selection. Facet selection is linked to *query classification* on the research side (Lin et al., 2018; Skinner and Kallumadi, 2019) and *query scoping* on the product side, i.e. pre-selecting, say, the facet *color* with value *black* for a query such as "black basketball shoes" (Liberman and Lempel, 2014; Vandic et al., 2013). Scoping may result in an aggressive restriction of the result set, lowering *recall* excessively: in most cases, an acceptable shopping experience would need to combine scoping with *query expansion* (Diaz et al., 2016). **SessionPath** is more flexible than query classification, by supporting explicit path prediction and incorporating in-session information; it is more gentle than scop-

ing (by nudging transparently the final user instead of forcing a selection behind the scene); it is more principled than expansion in balancing precision and recall.

Deep Learning in IR. The development of deep learning models for IR has been mostly restricted to the retrieve-and-rerank paradigm (Mitra and Craswell, 2017; Guo et al., 2016). Some recent works have been focused specifically on ranking suggestions for type-ahead: neural language models are proposed by Park and Chiba (2017); Wang et al. (2018b); specifically in eCommerce, Kannadasan and Aslanyan (2019) employs *fastText* to represent queries in the ranking phase and Yu et al. (2020) leverages deep image features for in-session personalization. While *this* work employs deep neural networks both for feature encoding and the inference itself, the proposed methods are agnostic on the underlying retrieval algorithm, as long as platforms can enrich type-ahead response with the predicted category. By providing a gentle entry point into existing workflows, a great product strength of **SessionPath** is the possibility of deploying the new functionalities with minimal changes to any architecture, neural or traditional (see also Appendix A).

4 Problem Statement

Suggesting a category facet can be modelled with the help of few formal definitions. A target shop E has products $p_1, p_2, ...p_n \in P$ (e.g. *nike air max 97*) and categories $c_{1,1}, c_{1,2}, ...c_{n,m} \in C$, where $c_{n,m}$ is the category n at *depth* m (e.g. at $m = 1$, [*soccer, volley, football, basketball*], at $m = 2$ [*shoes, pants, t-shirts*], etc.); a taxonomy tree T_m is an *indexed* mapping $P \mapsto C_m$, assigning a category to products for each depth m (e.g. *air max 97* $\mapsto_0$ *root*,$\mapsto_1$ *soccer*, $\mapsto_2$ *shoes*, $\mapsto_3$ *messi* etc.); *root* is the base category in the taxonomy, and it is common to all products (we will omit it for brevity in all our examples). In what follows, we use *path* to denote a sequence of categories (hierarchically structured) in our target shop (e.g. *root / soccer / shoes / messi*), and *nodes* to denote the categories in a path (e.g. *soccer* is a node of *soccer / shoes / messi*).

Given a browsing session s containing products $p_x, p_y, ...p_z$, and a candidate type-ahead query q, the model's goal is to learn both the optimal depth value m and, for each $k \leq m$, a contextual function $f(q, s) \mapsto C_k$. As we shall see in the ensuing section, **SessionPath** solution to this challenge is two-fold: a model generating a path first, and a decision module to pick the appropriate depth m (Figure 4).

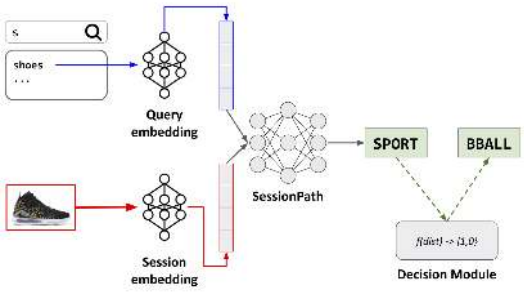

Figure 4: Functional flow for **SessionPath**: the current session and the candidate query "shoes" are embedded and fed to the model; the distribution over possible categories at each step of the taxonomy is passed to a *decision module*, that either cuts the generation at that step or includes the step in the final prediction. The decision process is repeated until either the module cuts or a max-length path is generated.

5 Baseline and Personalized Models

We approach the challenge *incrementally*, by first developing a count-based model (**CM**) that learns a mapping from queries to all paths (i.e. *sport* and *sport / soccer* are treated purely as "labels", so they are two completely unrelated target classes for the model); **CM** will both serve as a baseline for more sophisticated methods and as a fast reference implementation not requiring any deep learning infrastructure. We improve on **CM** with **SessionPath**, a model based on deep neural networks. From a product perspective, it is important to remember (Figure 4) that a *decision module* is called *after* a path prediction is made: we discuss how to tune this crucial part after establishing the general performance of the proposed models.

5.1 A Baseline Model

The intuition behind the count-based model is that we may gain insights on relevant paths linked to a query from the clicks on search results. Therefore, we can build a simple count-based model by creating a map from each query in the search logs to their frequently associated paths. To build this map, we first retrieve all products clicked after each query, along with their path; for a given query, we can then calculate the percentage of occurrence of each path in the clicked products. Since the

model is not hierarchical, it is important to note that *sport* and *sport / basketball* will be treated as completely disjoint target classes for the prediction. To avoid noisy results, we empirically determined a frequency threshold for paths to be counted as relevant to a certain query (80%); at prediction time, given a query in the training set, we retrieve all the paths associated with it and return the one with longest depth; for unseen queries, no prediction is made.

5.2 Modelling Session Context and Taxonomy Paths

The main conceptual improvements over **CM** are three:

- **SessionPath** produces predictions also for queries not in the training set;
- **SessionPath** introduces personalization, by combining the linguistic information contained in the query with in-session shopping intent;
- **SessionPath** is trained to produce the most accurate path by explicitly making a new prediction *at each node*, not predicting paths in a *one-out-of-many* scenario; in other words, **SessionPath** knows that *sport* and *sport / basketball* are related, and that the second path is generated from the first when a given distribution over sport activities is present.

To represent the current session in a dense architecture, we first train a skip-gram *prod2vec* model over user data for the entire website, mapping product to 50-dimensional vectors (Mikolov et al., 2013a; Grbovic et al., 2015). At training and serving time **SessionPath** retrieves the embeddings of the products in the target session, and use average pooling to calculate the context vector from the sequence of embeddings, as shown by Covington et al. (2016); Yu et al. (2020). To represent the candidate query, an encoding of linguistic behavior that generalizes to unseen queries is needed. We tested different strategies:

- *word2vec*: we train a skip-gram model from Mikolov et al. (2013b) over product short descriptions from the catalog. Since most search queries are less than three words long, we opted for a simple and fast average pooling of the embeddings in the tokenized query;
- *character-based language model*: inspired by Skinner (2018), we train a char-based language model (single LSTM layer with hidden dimension 50) on search logs and product descriptions from the target shop; a standard LSTM approach was found ineffective in preliminary tests, so we opted instead for using the "Balanced pooling" strategy from Skinner and Kallumadi (2019), where the dense representation for the query is obtained by taking the last network state and then concatenating it together with average-pooling (Wang et al., 2018a), max-pooling, and min-pooling;
- *pre-trained language model*: we map the query to a 768-size vector using BERT (Devlin et al., 2019) (as pre-trained for the target language by Magnini et al. (2006));
- *Search2Prod2Vec + unigrams*: we propose a "small-data" variation to *Search2Vec* by Grbovic et al. (2016), where queries (on a web search engine) are embedded through events happening before and after the search event. Adapting the intuition to product search, we propose to represent queries through the embeddings of products clicked in the search result page; in particular, each query q is the weighted average of the corresponding *prod2vec* embeddings; it can be argued that the *clicking* event is analogous to a "pointing" signal (Tagliabue and Cohn-Gordon, 2019), when the *meaning* of a word ("shoes") is understood as a function from the string to a set of objects falling under that concept (e.g. Chierchia and McConnell-Ginet (2000)). In the spirit of compositional semantics (Baroni et al., 2014), we generalize this representation to unseen queries by building a unigram-based language model, so that "nike shoes" gets its meaning from the composition (average pooling) of the meaning of *nike* and *shoes*.

To generate a path explicitly, we opted for an encoder-decoder architecture. The encoder employs the wide-and-deep approach popularized by Cheng et al. (2016), and concatenates textual and non-textual feature to obtain a wide representation of the current context, which is passed through a dense layer to represent the final encoded state. The decoder is a word-based language model (Zoph and Le, 2016) which produces a sequence of nodes (e.g.

Shop	Queries (with context)	Products
Shop 1	270K (185K)	29.699
Shop 2	270K (227K)	93.967

Table 1: Descriptive statistics for the dataset.

sport, *basketball*, etc.) conditioned on the representation created by the encoder; more specifically, the architecture of the decoder consists of a single LSTM with 128 cells, a fully-connected layer and a final layer with softmax output activation. The output dimension corresponds to the total number of distinct nodes found in all the paths of the training data, including two additional tokens to encode the start-of-sequence and end-of-sequence. For training, the decoder uses the encoded information to fill its initial cell states; at each timestep, we use teacher forcing to pass the target character, offset by one position, as the next input character to the decoder (Williams and Zipser, 1989). Empirically, we found that robust parameters for the deep learning methods are a learning rate of 0.001, time decay of 0.00001, early stopping with $patience = 20$, and mini-batch of size 128; furthermore, the Adam optimizer with cross-entropy loss is used for all networks, with training up to 300 epochs. Once trained, the model can generate a path given an encoded session representation and a start-of-sequence token: after the first step, the decoder uses autoregression sequence generation (Bahdanau et al., 2015) to predict the next output token.

6 Dataset

We leverage behavioral and search data from two partnering shops in *Company*'s network: **Shop 1** and **Shop 2** have uniform data ingestion, making it easy to compare how well models generalize; they are mid-size shops, with annual revenues between 20 and 100 million dollars. **Shop 1** and **Shop 2** differ however in many respects: they are in different verticals (*apparel* vs *home improvement*), they have a different catalog structure (603 paths organized in 2-to-4 levels for each product vs 985 paths in 3 levels for all products), and different traffic (top 200k vs top 15k in the Alexa Ranking). Descriptive statistics for the training dataset can be found in Table 1: data is sampled for both shops from June-August in 2019; for testing purposes, a completely disjoint dataset is created using events from the month of September.

Model	D=1	D=2	D=last
CM	0.63	0.53	0.22
MLP+BERT	0.72	0.59	0.33
SP+BERT	0.77	0.64	0.40
SP+LSTM	0.79	0.68	0.43
SP+W2V	0.82	0.71	0.46
SP+SV	**0.87**	**0.79(0.01)**	**0.55**
CM	0.41	0.34	0.24
MLP+BERT	0.61	0.50	0.39
SP+BERT	0.66	0.55	0.45
SP+LSTM	0.67	0.57	0.46
SP+W2V	0.69	0.59	0.47
SP+SV	**0.80**	**0.71**	**0.59**

Table 2: Accuracy scores for $depth = 1$, $depth = 2$, $depth = last$, divided by **Shop 1** (*top*) and **Shop 2** (*bottom*). We report the mean over 5 runs, with SD if $SD \geq 0.01$.

7 Experiments

We perform offline experiments using search logs for **Shop 1** and **Shop 2**: for each search event in the dataset, we use products seen before the query (if any) to build a session vector as explained in Section 5.2; the path of the products *clicked after* the query is used as the target variable for the model under examination.

7.1 Making predictions

We benchmark **CM** and **SessionPath** from Section 5, plus a multi-layer perceptron (**MLP**) to investigate the performance of an intermediate model: while not as straightforward as **CM**, **MLP** is considerably easier to train and serve than **SessionPath** and it may therefore be a compelling architectural choice for many shops (see Appendix A for practical engineering details); **MLP** concatenates the session vector with the *BERT* encoding of the candidate query, and produces a distribution over all possible *full-length* paths (*one-out-of-many* classification, where the target class comprises all the paths at the maximum depth for the catalog at hand). Table 2 shows accuracy scores for three different depth levels in the predicted path: **SP+BERT** is **SessionPath** using *BERT* to encode linguistic behavior, **SP+W2V** is using *word2vec*, **SP+SV** is using *Search2Prod2Vec* and **SP+LSTM** is using the language model. Every **SessionPath** variant outperforms the count-based and neural baselines, with *Search2Prod2Vec* providing up to 150% increase over **CM** and 67% over **MLP**. **CM** score is

penalized not only by the inability to generalize to unseen queries: even when considering previously seen queries in the test set, **SP+SV**'s accuracy is significantly higher (0.58 vs 0.27 at $D = last$), showing that neural methods are more effective in capturing the underlying dynamics. Linguistic representations learned *directly* over the target shop outperform bigger models pre-trained on generic text sources, highlighting some differences between general-purpose embeddings and shop-specific ones, and suggesting that off-the-shelf NLP models may not be readily applied to short, keyword-based queries. While fairly accurate, **SP+W2V** is much slower to train compared to **SP+SV** and harder to scale across clients, as it relies on having enough content in the catalog to train models that successfully deal with shop lingo. On a final language-related note, it is worth stressing that click-based embeddings built for **SP+SV** show not just better performance over *seen* queries (which is expected), but better generalization ability in the *unseen* part as well compared to *BERT* embeddings (0.82 vs 0.70 at $D = 1$ for **Shop 1**, 0.76 vs 0.63 for **Shop 2**).

In the spirit of ablation studies, we rerun **SP+SV** and **SP+BERT** *without* session vector. Interestingly enough, context seems to play a slightly different role in the two shops and the two models: **SP+BERT** is greatly helped by contextual information, especially for *unseen* queries (0.28 vs 0.21 at $D = last$ for **Shop 1**, 0.40 vs 0.15 for **Shop 2**), but the effect for **SP+SV** is smaller (0.50 vs 0.42 for **Shop 2**); while models on **Shop 2** show a bigger drop in performance when removing session information, generally (and unsurprisingly) session-aware models provide better generalization on *unseen* queries across the board. By comparing **SessionPath** with a simpler neural model (such as **MLP**), it is clear that session plays a bigger role in *MLP*, suggesting that **SessionPath** architecture is able to better leverage linguistic information across cases.

Finally, we investigate sample efficiency of chosen methods by training on smaller fractions of the original training dataset: Table 3 reports accuracy of four methods when downsampling the training set for **Shop 1** to $1/10^{th}$ and $1/4^{th}$ of the dataset size. **CM**'s inability to generalize cripples its total score; **MLP** is confirmed to be simple yet effective, performing significantly better than the count-based baseline; **SP+SV** is confirmed to be the best performing model, and even with only $1/10^{th}$ of samples outperforms all other models from Table 2: by leveraging the bias encoded in the hierarchical structure of the products, **SP+SV** allows paths that share nodes (*sport*, *sport / basketball*) to also share statistical evidence, resulting in a very efficient learning.

Model (D=last)	**1/10**	**1/4**
CM	0.18	0.20
MLP+BERT	0.28	0.30
SP+BERT	0.31	0.34
SP+SV	**0.51**	**0.53**

Table 3: Accuracy scores (**D=last**) when training on portions of the original dataset for **Shop 1**.

Accuracy provides a strong argument on the efficacy of the proposed models in industry, and it is in fact widely employed in the relevant literature: Vandic et al. (2013) employs click-based accuracy for label prediction, while Molino et al. (2018) (in a customer service use cases) uses accuracy at different depths for sequential predictions that are somewhat similar to **SessionPath**. However, *accuracy* by itself falls short to tell the whole story on product decisions: working with *Coveo*'s clients, it is clear that not all shops are born equal - some (e.g. mono-brand fashion shops) strongly favor a smaller and cleaner result page; others (e.g. marketplaces) favor bigger, even if noisier, result sets. Section 7.2 presents our contribution in analyzing the business context and proposes viable solutions.

7.2 Tuning the decision module

Consider the two possible decisions in the scenario depicted in Figure 5: given "nike shoes" as query and basketball shoes as session context, **SessionPath** prediction is *shoes / nike / basketball*. According to scenario **1**, a decision is made to cut the path at *shoes / nike*: the resulting set of products contain a mixed set of shoes from the target brand, with no specific sport affinity; in scenario **2**, the decision module allows the prediction of a longer path, *shoes / nike / basketball*: the result page is smaller and only contains basketball shoes of the target brand. Intuitively, a perfect model would choose **2** only when it is "confident" of the underlying intention, as expressed through the combination of language and behavioral clues; when the model is less confident, it should stick to **1** to avoid hiding from the shopper's possible interesting products.

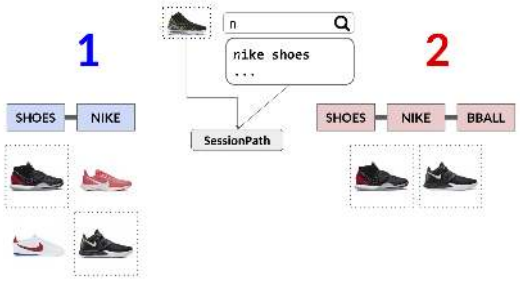

Figure 5: Two scenarios for the decision module after **SessionPath** generates the *shoes / nike / basketball* path, with input query "nike shoes" and Lebron James basketball shoes in the session. In *Scenario 1 (blue)*, we cut the result set after the second node - *shoes / nike* - resulting in a mix set of shoes; in *Scenario 2 (red)*, we use the full path - *shoes / nike / basketball* - resulting in only basketball shoes (dotted line products). How can we define what is the optimal choice?

To quantify how much confident the model is at any given node in the predicted path, at each node s_n we output the multinomial distribution d over the next node s_{n+1}[2] and calculate the Gini coefficient of d, $g(d)$:

$$g(d) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} |x_i - x_j|}{2n^2 \bar{x}} \qquad \text{(GI)}$$

where n is the total number of classes in the distribution d, x_i is the probability of the class i and $\bar{x}$ is the mean probability of the distribution.

Once $GI = g(d)$ is computed, a *decision rule* $DR(GI)$ for the decision module in Figure 4 is given by:

$$DR(x) = \begin{cases} 1 & \text{if } x \geq ct \\ 0 & \text{otherwise} \end{cases}$$

where 1 means that the module is confident enough to add the node to the final path that will be shown to the user, while 0 means the path generation is stopped at the current node. ct is our *confidence threshold*: since different values of ct imply more or less aggressive behavior from the model, it is important to tune ct by taking into account the relevant business constraints.

[2]Non-existent paths account for less then 0.005% of all the paths in the test set, proving that **SessionPath** is able to accurately learn transitions between nodes and suggesting that an explicit check at decision time is unnecessary. Of course, if needed, a "safety check" may be performed at query time by the search engine, to verify that filtering by the suggested path will result in a non-empty set.

Gini Threshold	Precision	Recall
0.996	0.65	0.99
0.993	0.82	0.91
0.990	0.93	0.77
0.980	0.99	0.74

Table 4: Precision and recall at different decision thresholds for **Shop 1**.

Due to the contextual and interactive nature of **SessionPath**, we turn search logs into a "simulation" of the interactions between hypothetical shoppers and our model (Kuzi et al., 2019). In particular, for any given search event in the test dataset - comprising products seen in the session, the actual query issued, all the products returned by the search engine, the products clicked from the shopper in the result page -, and a model prediction (e.g. *sport / basketball*), we construct two items:

- **golden truth set**: which is the set of the paths corresponding to the items the shopper deemed relevant in that context (relevance is therefore assessed as *pseudo-feedback* from clicks);
- **filtered result set**: which is the set of products returned by the engine, *filtering* out those not matching the prediction by the model (i.e. simulating the engine is actually working with the categories suggested by **SessionPath**).

With the *golden truth set*, the *filtered result set* and the original *result page*, we can calculate *precision* and *recall* at different values of ct (please refer to Appendix B for a full worked out example).

Table 4 reports the chosen metrics calculated for **Shop 1** at different values of ct; the trade-off between the two dimensions makes all the point Pareto-optimal: there is no way to increase performance in one dimension without hurting the other. Going from the first configuration ($ct = 0.996$) to the second ($ct = 0.993$) causes a big jump in the metric space, with the model losing some recall to gain *considerably* in precision. To get a sense of how the model is performing in practice, Figure 6 shows three sessions for the query "nike shoes": when session context is empty (session 1), the model defaults to the broadest category (*sneakers*); when session is *running*-based or *basketball*-based, the model adjusts its aggressiveness depending on the threshold we set. It is interesting to note that while the prediction for 2 at $ct = 0.97$ is

wrong at the last node (product is *a7*, not *a3*), the model is still making a *reasonable* guess (e.g. by guessing sport and brand correctly).

In our experience, the adoption of data-driven models in traditional digital shops is often received with some skepticism over the "supervision" by business experts (Baer and Kamalnath, 2017): a common solution is to avoid the use of neural networks, in favor of model interpretability. **SessionPath**'s decision-based approach dares to dream a different dream, as the proposed architecture shows that we can retain the accuracy of deep learning and still provide a meaningful interface to business users – here, in the form of a precision/recall space to be explored with an easy-to-understand parameter.

Figure 6: Sample **SessionPath** predictions for the candidate query "nike shoes", with two thresholds (gray, green) and three sessions, $1, 2, 3$ (no product for session 1, a pair of running shoes for 2, a pair of basketball shoes for 3). The model reacts quickly both across sessions (switching to relevant parts of the underlying product catalog) and across threshold values, making more aggressive decisions at a lower value (green).

8 Conclusions and Future Work

This research paper introduced **SessionPath**, a personalized and scalable model that dynamically suggests product paths in type-ahead systems; **SessionPath** was benchmarked on data from two shops and tested against count-based and neural models, with explicit complexity-accuracy trade-offs. Finally, we proposed a confidence-based decision rule inspired by customer discussions: by abstracting away model behavior in one parameter, we wish to solve the often hard interplay between business requirements and machine behavior; furthermore, by leveraging a hierarchical structure of product concepts, the model produces predictions that are suitable to a *prima facie* human inspection (e.g. Figure 6).

While our evaluation shows very encouraging results, the next step will be to A/B test the proposed models on a variety of target clients: **Shop 1** and **Shop 2** data comes from search logs of a last-generation search engine, which possibly skewed model behavior in subtle ways. With more data, it will be possible to extend the current work in some important directions:

1. while *this* work showed that **SessionPath** is effective, the underlying deep architecture can be improved further: on one hand, by doing more extensive optimization; on the other, by focusing on how to best perform linguistic generalization: *transfer learning* (between tasks as proposed by Skinner and Kallumadi (2019), or across clients, as described in Yu et al. (2020)) is a powerful tool that could be used to improve performances further;

2. the same model can be applied with almost no changes to the search workflow, to provide a principled way to do personalized query scoping. A preliminary A/B test on **Shop 1** using the *MLP* model on a minor catalog facet yielded a small (2%) but statistically significant improvement ($p < 0.05$) on click-through rate and we look forward to extending our testing;

3. we could model path depth *within the decoder itself*, by teaching the model when to stop; as an alternative to learning a decision rule in a supervised setting, we could leverage reinforcement learning and let the system improve through iterations - in particular, the choice of cutting the path for a given query and session vector could be cast in terms of contextual bandits;

4. finally, *precision* and *recall* at different depths are just a first start; preliminary tests with *balanced accuracy* on selected examples show promising results, but we look forward to performing user studies to deepen our understanding of the ideal decision mechanism.

Personalization engines for digital shops are expected to drive an increase in profits by 15% by the end of 2020 (Gillespie et al., 2018); facet suggestions help personalizing the search experience as early as in the type-ahead drop-down window: considering that search users account globally for almost 14% of the total revenues (Charlton, 2013),

and that category suggestions may improve click-through-rate and reduce cognitive load, **SessionPath** (and similar models) may play an important role in next-generation online experiences.

Acknowledgments

Thanks to (in order of appearance) Andrea Polonioli, Federico Bianchi, Ciro Greco, Piero Molino for helpful comments to previous versions of this article. We also wish to thank our anonymous reviewers, who greatly helped in improving the clarity of our exposition.

References

Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. 2017. Controlling popularity bias in learning-to-rank recommendation.

Chris Anderson. 2006. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion.

Tobias Baer and Vishnu Kamalnath. 2017. Controlling machine-learning algorithms and their biases.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Marco Baroni, Raffaela Bernardi, and Roberto Zamparelli. 2014. Frege in space: A program of compositional distributional semantics.

Graham Charlton. 2013. Is site search less important for niche retailers?

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Gregory S. Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & deep learning for recommender systems. In *DLRS 2016*.

Gennaro Chierchia and Sally McConnell-Ginet. 2000. *Meaning and Grammar (2nd Ed.): An Introduction to Semantics*. MIT Press, Cambridge, MA, USA.

Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 191–198, New York, NY, USA. Association for Computing Machinery.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Fernando Diaz, Bhaskar Mitra, and Nick Craswell. 2016. Query expansion with locally-trained word embeddings. *ArXiv*, abs/1605.07891.

Penny Gillespie, Jason Daigler, Mike Lowndes, Christina Klock, Yanna Dharmasthira, and Sandy Shen. 2018. Magic quadrant for digital commerce. Technical report, Gartner.

Laura A. Granka, Thorsten Joachims, and Geri Gay. 2004. Eye-tracking analysis of user behavior in www search. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, page 478–479, New York, NY, USA. Association for Computing Machinery.

Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, Fabrizio Silvestri, Ricardo Baeza-Yates, Andrew Feng, Erik Ordentlich, Lee Yang, and Gavin Owens. 2016. Scalable semantic matching of queries to ads in sponsored search advertising. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, page 375–384, New York, NY, USA. Association for Computing Machinery.

Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. 2015. E-commerce in your inbox: Product recommendations at scale. In *Proceedings of KDD '15*.

Zhiwei Guan and Edward Cutrell. 2007. An eye tracking study of the effect of target rank on web search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, page 417–420, New York, NY, USA. Association for Computing Machinery.

Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A deep relevance matching model for ad-hoc retrieval. In *CIKM '16*.

Sheena Iyengar and Mark Lepper. 2001. When choice is demotivating: Can one desire too much of a good thing? *Journal of personality and social psychology*, 79:995–1006.

Manojkumar Rangasamy Kannadasan and Grigor Aslanyan. 2019. Personalized query auto-completion through a lightweight representation of the user context. *CoRR*, abs/1905.01386.

Saar Kuzi, Abhishek Narwekar, Anusri Pampari, and ChengXiang Zhai. 2019. Help me search: Leveraging user-system collaboration for query construction to improve accuracy for difficult queries. In *Proceedings of the 42nd International ACM SIGIR*

Conference on Research and Development in Information Retrieval, SIGIR'19, page 1221–1224, New York, NY, USA. Association for Computing Machinery.

Sonya Liberman and Ronny Lempel. 2014. Approximately optimal facet value selection. *Sci. Comput. Program.*, 94(P1):18–31.

Y. Lin, A. Datta, and G. D. Fabbrizio. 2018. E-commerce product query classification using implicit user's feedback from clicks. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1955–1959.

Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331.

Bernardo Magnini, Amedeo Cappelli, Emanuele Pianta, Manuela Speranza, V Bartalesi Lenzi, Rachele Sprugnoli, Lorenza Romano, Christian Girardi, and Matteo Negri. 2006. Annotazione di contenuti concettuali in un corpus italiano: I - cab. In *Proc.of SILFI 2006*.

Irina Matveeva, Chris Burges, Timo Burkard, Andy Laucius, and Leon Wong. 2006. High accuracy retrieval with multiple nested ranker. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, page 437–444, New York, NY, USA. Association for Computing Machinery.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA. Curran Associates Inc.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Bhaskar Mitra and Nick Craswell. 2017. Neural models for information retrieval. *ArXiv*, abs/1705.01509.

Piero Molino, Huaixiu Zheng, and Yi-Chia Wang. 2018. Cota: Improving the speed and accuracy of customer support through ranking and deep networks. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

Dae Hoon Park and Rikio Chiba. 2017. A neural language model for query auto-completion.

Benjamin Scheibehenne, Rainer Greifeneder, and Peter M. Todd. 2010. Can There Ever Be Too Many Options? A Meta-Analytic Review of Choice Overload. *Journal of Consumer Research*, 37(3):409–425.

Michael Skinner. 2018. Product categorization with lstms and balanced pooling views. In *eCOM@SIGIR*.

Michael Skinner and Surya Kallumadi. 2019. E-commerce query classification using product taxonomy mapping: A transfer learning approach. In *eCOM@SIGIR*.

Jacopo Tagliabue and Reuben Cohn-Gordon. 2019. Lexical learning as an online optimal experiment: Building efficient search engines through human-machine collaboration. *ArXiv*, abs/1910.14164.

Damir Vandic, Flavius Frasincar, and Uzay Kaymak. 2013. Facet selection algorithms for web product search. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, CIKM '13, page 2327–2332, New York, NY, USA. Association for Computing Machinery.

Cheng Wang, Mathias Niepert, and Hui Li. 2018a. LRMM: Learning to recommend with missing modalities. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3360–3370, Brussels, Belgium. Association for Computational Linguistics.

Po-Wei Wang et al. 2018b. Realtime query completion via deep language models. In *eCOM@SIGIR*, volume 2319 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks.

Bingqing Yu, Jacopo Tagliabue, Federico Bianchi, and Ciro Greco. 2020. An image is worth a thousand features: Scalable product representations for in-session type-ahead personalization. In *Companion Proceedings of the Web Conference*, New York, NY, USA. Association for Computing Machinery.

Barret Zoph and Quoc V. Le. 2016. Neural architecture search with reinforcement learning. *ArXiv*, abs/1611.01578.

A Architectural Considerations

Figure 7 represents a functional overview of a type-ahead service: when *User X* on a shop starts typing a query after browsing some products, the query seed and the session context are sent to the server. An existing engine - traditional or neural - will then take the query and the context and produce a list of top-*k* query candidates, ranked by relevance, which are then sent back to the client to populate the dropdown window of the search bar.

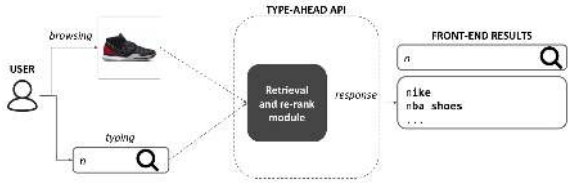

Figure 7: High-level functional overview of an industry standard API for type-ahead suggestions: query seed and possibly session information about the user are sent by the client to the server, where some retrieval and re-ranking module produces the final top-k suggestions and prepares the response for front-end consumption.

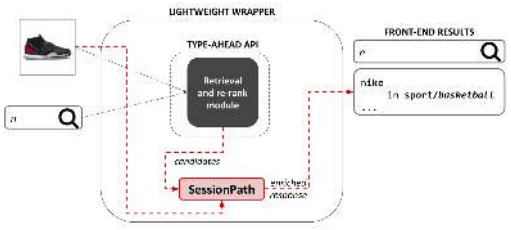

Figure 8: A lightweight **SessionPath** functional integration: starting from a standard flow (Figure 7, a simple wrapper around the existing module sends the same session information and the top-n suggestions to **SessionPath**, for dynamic path prediction. The final response is then obtained by simply augmenting the existing response containing query candidates with category predictions.

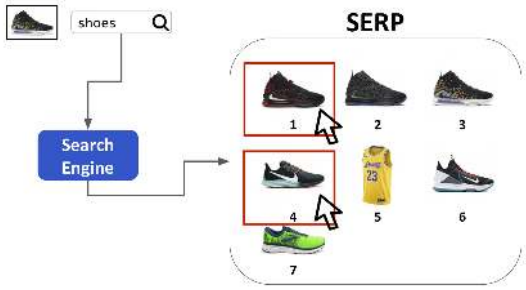

Figure 9: A sample row in the test set, displaying search results (7 products in 4 paths) for the query "shoes" and a session containing a pair of *LeBron James* basketball shoes. In this example, the shopper clicked on products P_1 and P_4.

As depicted in Figure 8, category suggestions can be quickly added to any existing infrastructure by treating the current engine as a "black-box" and adding path predictions at run-time for the first (or the first k, since requests to the model at that point can be batched with little overhead) query candidate(s). In this scenario, the decoupling between retrieval and suggestions is absolute, which may be a good idea when the stacks are very different (say, traditional retrieval *and* neural suggestions), but less extreme solutions are obviously possible. The crucial engineering point is that path prediction (using any of the methods from Section 7) can be added and tested quickly, with few conceptual and engineering dependencies: the more traditional the existing stack, the more an incremental approach is recommended: count-based first - since predictions can be served simply from an in-memory map -, **MLP** second - since predictions require a small neural network, but they are fast enough to only require CPU at query time -, and finally the full **SessionPath** - which requires dedicated hardware considerations to be effective in the time constraints of the type-ahead use case. As a practical suggestion, we also found quite effective when using simpler models (e.g. **MLP**) to first test it at a *given depth*: for example, you start by only classifying the most likely nodes in template *sport / ?*, and then incrementally increase the target classes by adding more diverse paths.

Adding a lightweight wrapper around the original bare-bone endpoint allows for other improvements as well: for example, considering typical power-law of query logs, a caching layer can be used to avoid a full retrieving-and-rerank iteration for frequent queries; obviously, this and similar features are independent from **SessionPath** itself.

B Metrics Calculation: a Worked-Out Example

For the sake of reproducibility, we present a worked out example of metrics calculations for offline testing of the decision module (Section 7.2). Figure 9 depicts an historical interaction from the search logs: a session containing a product, a query issued by the user and the search result page ("serp"), containing seven items belonging to the following paths:

$$
\begin{aligned}
P_1 &= \textit{sport / basketball / lebron} \\
P_2 &= \textit{sport / basketball / lebron} \\
P_3 &= \textit{sport / basketball / lebron} \\
P_4 &= \textit{sport / running / sneakers} \\
P_5 &= \textit{sport / basketball / jerseys} \\
P_6 &= \textit{sport / basketball / curry} \\
P_7 &= \textit{sport / running / sneakers}.
\end{aligned}
$$

Click-through data (i.e. products in the serp clicked by the user) indicates that P_1 and P_4 are relevant, and so the associated paths are ground truths (*sport/basketball/lebron* and *sport/running/sneakers*). We now present the full calculations in three scenarios, corresponding to three level of depths in the predicted path.

Scenario 1 (general): prediction is *sport*. In this case, result set would be intact, so: *True Positives* (**TP**) are P_1, P_2, P_3, P_4, P_7, *False Positives* (**FP**) are P_5, P_6, *False Negatives* (**FN**) are $\emptyset$. *Precision* is: **TP / (TP + FP)** = $5/(5+2) = 0.71$, *Recall* is: **TP / (TP + FN)** = $5/(5+0) = 1.0$ (with no cut, all truths are retrieved so 1.0 is the expected result).

Scenario 2 (intermediate): prediction is *sport/basketball*. In this case, filtering the result set according to the decision made by the model would give P_1, P_2, P_3, P_5, P_6 as the final set. So: **TP** = P_1, P_2, P_3, **FP** = P_5, P_6, **FN** = P_4, P_7; *Precision* = $3/(3+2) = 0.6$, *Recall* = $3/(3+2) = 0.6$.

Scenario 3 (specific): prediction is *sport / basketball / lebron*. In this case, filtering the result set according to the decision made by the model would give P_1, P_2, P_3 as the final set. So: **TP** = P_1, P_2, P_3, **FP** = $\emptyset$, **FN** = P_4, P_7; *Precision* = $3/(3+0) = 1.0$, *Recall* = $3/(3+2) = 0.6$.

The full calculations show very clearly the natural trade-off discussed at length in Section 7.2: the deeper the path, the more precise are the results but also the higher the chance of hiding valuable products from the shopper.