Learning to Rank Instant Search Results with Multiple Indices: A Case Study in Search Aggregation for Entertainment

Scott Rome
Scott_Rome@comcast.com
Applied AI Research, Comcast
Philadelphia, PA, USA

Sardar Hamidian
Sardar_Hamidian@comcast.com
Applied AI Research, Comcast
Washington, DC, USA

Richard Walsh
Richard_Walsh@comcast.com
Applied AI Research, Comcast
Sunnyvale, CA, USA

Kevin Foley
Kevin_Foley@comcast.com
Applied AI Research, Comcast
Washington, DC, USA

Ferhan Ture
Ferhan_Ture@comcast.com
Applied AI Research, Comcast
Washington, DC, USA

ABSTRACT

At Xfinity, an instant search system provides a variety of results for a given query from different sources. For each keystroke, new results are rendered on screen to the user, which could contain movies, television series, sporting events, music videos, news clips, person pages, and other result types. Users are also able to use the Xfinity Voice Remote to submit longer queries, some of which are more open-ended. Examples of queries include incomplete words which match multiple results through lexical matching (i.e., "ali"), topical searches ("vampire movies"), and more specific longer searches ("Movies with Adam Sandler"). Since results can be based on lexical matches, semantic matches, item-to-item similarity matches, or a variety of business logic driven sources, a key challenge is how to combine results into a single list. To accomplish this, we propose merging the lists via a Learning to Rank (LTR) neural model which takes into account the search query. This combined list can be personalized via a second LTR neural model with knowledge of the user's search history and metadata of the programs. Because instant search is under-represented in the literature, we present our learnings from research to aid other practitioners.

CCS CONCEPTS

• **Information systems** → **Retrieval models and ranking**; *Users and interactive retrieval*; Multimedia and multimodal retrieval.

KEYWORDS

learning to rank, neural networks, NLP retrieval, off-policy evaluation, information retrieval

ACM Reference Format:
Scott Rome, Sardar Hamidian, Richard Walsh, Kevin Foley, and Ferhan Ture. 2022. Learning to Rank Instant Search Results with Multiple Indices: A Case Study in Search Aggregation for Entertainment. In *Proceedings of the 45th Int'l ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '22), July 11–15, 2022, Madrid, Spain.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3477495.3536334

SIGIR '22, July 11–15, 2022, Madrid, Spain.

ACM ISBN 978-1-4503-8732-3/22/07.
https://doi.org/10.1145/3477495.3536334

1 INTRODUCTION

Instant search is an under-studied use case in information retrieval (IR), which has been noted by industry practitioners [9, 13]. Instant search is characterized as a search where a new set of results is returned for every keystroke and is applicable across a variety of different industries and use cases [21]. To meet strict latency requirements, inverted indices are typically used in this setting. Inverted indices are a map from string queries to sets of results. Most IR systems employ at least two steps: a candidate selection step from which a small subset is selected from a large pool of options followed by an ordering step which reranks the retrieved items. This approach has been implemented by several companies for both search and recommendation problems [3, 7].

When aiming to provide a wide range of results in a single list beyond lexical matches, numerous challenges are apparent when implementing a robust instant search service. First, how should one combine results for a given search query when there are multiple matches present (lexical, semantic, etc.)? When using several candidate sources, how does one prevent ancillary matches from being ranked higher than more logical matches? Should one consider item type (in our case, movies, television shows, sporting events, etc.) in ranking?

To that end, our goal is to present proposed solutions to these challenges by describing an approach to instant search that can serve millions of users on a platform. We will discuss many practical points around making a production instant search system work at a large scale. These points will focus on the design of the various necessary indices, model architectures, business logic, and metrics and sessionization criteria which in combination create a cohesive experience. We finally will present online experiments of the proposed system and discuss its performance against a global popularity sorting algorithm.

2 APPROACH

The following section discusses different components of the proposed search framework shown in Figure 1. Candidate generation, availability filtering, and reranking constitute the key components of the process. In the candidate generation step, asynchronous calls are made to multiple indices. For example, a query for "*THE OFFICE*" may return the lexical match *The Office* and an

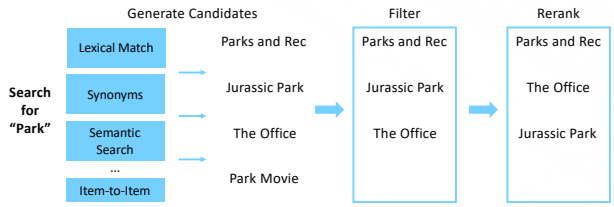

Figure 1: An example diagram of the search flow for the query "Park".

item-to-item similarity match like *Parks and Recreation*. The candidate results may include movies, television shows, sporting events, and music videos among other types, which can be lexical matches, semantic matches, or item-to-item similarity. Then, the candidates need to be filtered based on if the item is available to the user. (Availability will be discussed in Section 2.3.2.) The results are next combined into a single list through a heuristic before being sent to the reranking step, which consists of two deep learning models that combine the different candidate lists and fine-tune the results to the user. In the final stage of the pipeline, business logic would be applied to generate the final rankings while also respecting product requirements.

2.1 Candidate Generation

Generating candidates can be accomplished via lexical matching, semantic matching, item-to-item similarity matching, and trending items. In the following sections, all specifications relating to these possible indices are described. We also note that there can also be a fallback index that is used when other indices have no matches, but that index is outside the scope of this work. As a general framework, these approaches are applicable across a wide range of disciplines with reasonable results while being easy to implement in a performant, production-ready way to handle millions of users.

2.1.1 Lexical Matching. Lexical matching is based on a relevance score derived from the combination of text-based prefix matching and, if desired, popularity. Items whose title contain the query as a prefix are included in a candidate list and the results are reranked by a global popularity score. This component can be implemented using (for instance) Apache SOLR, a Lucene-based [8] technology.

2.1.2 Semantic Search Model. For semantic matching, an unsupervised zero-shot retrieval system using a twin neural network (sometimes called a siamese neural network) [2] and pre-trained natural language processing (NLP) model with an additional match boosting component can be implemented. A similar use of pre-trained NLP models and Siamese Networks for this type of application has been demonstrated by other practitioners [20].

In particular, a pre-trained sentence embedding model (Universal Sentence Encoder [4]) would be applied to encode a vector representation of brief text synopses of indexed content. Better performance may be achieved by augmenting these brief synopses with additional words from the item's *metadata*, such as the genre of the program (i.e. "Mystery," "Romantic Comedy," etc.) and the names of cast members with lead roles. An example of the resulting text blob for a fabricated horror-comedy movie is shown in Figure 2.

> "A Yellow Search Paper starring Scott Rome, Sardar Hamidian, Richard Walsh, Kevin Foley and Ferhan Ture. A machine learning research team stumbles upon a research paper written by a mysterious practitioner. Little did they know that reading the paper induces madness. Movie. Horror Comedy. Rated R. 2022"

Figure 2: A toy example of metadata representing the input data to one of the towers of a twin neural network for semantic retrieval.

At runtime, a query would be encoded to a vector representation via the same sentence embedding model, and an approximate K-nearest neighbor search can retrieve items with metadata that are semantically most similar to the query text.

To improve the quality of the matching, one can add a second component to boost items whose metadata approximately contain n-grams from the query, which can improve performance on more diverse queries. For example, queries for "Native American movies" were returning "Wild West movies" as well. By adding this second component, which can be referred to as "approximate n-gram boosting", the metadata must include the term "Native American" somewhere in the description for the item to be a viable match. In approximate n-gram boosting, word-level n-grams of the item metadata are computed and hashed into N buckets, resulting in a multi-hot vector representation. The semantic vector representation of the items discussed previously and the n-gram boosting vectors are concatenated together and stored to be searched against. As far as the authors know, this "approximate n-gram boosting" is novel in the case of multimedia semantic search.

2.1.3 Item-to-Item Similarity Candidates. For item-to-item similarity candidates in search, one can use a collaborative filtering-based approach such as described in Jojic et al. Lists of similar results to a query may be pre-computed and stored in a key-value mapping to improve runtime performance. For example, the key "*THE OFFICE*" will have a list of similar item identifiers stored as its value. Such key-value pairs are identified via an offline processing job with a heuristic that identifies search queries with few lexical matches. This process creates similar candidate lists for incomplete queries like "*THE OFF*" as well.

2.1.4 Trending Candidates. Trending candidates can be identified via an offline processing job which tags items as trending and stores them in an in-memory cache. At runtime, trending items are identified and can be boosted to the top of the list based on business logic. To identify trending items, a simple anomaly detection algorithm can be used. Data for each item over the last N days are used to compute a distribution of the log of the daily search result click totals. A t-test can then be employed to determine if the current day's click volume for that item is anomalous. For new items, we utilize a threshold for the total clicks for the first M days to indicate if the item should be boosted.

2.2 Reranking

A two-step approach with two deep learning models to perform reranking can be implemented. The end result of the system is a single list of content items which combines and ranks the search results from various inverted indices. This approach can solve the problems created by having multiple sources of candidates using a machine learning model to blend the various candidate lists into a ranked list, which is then ready to be personalized.

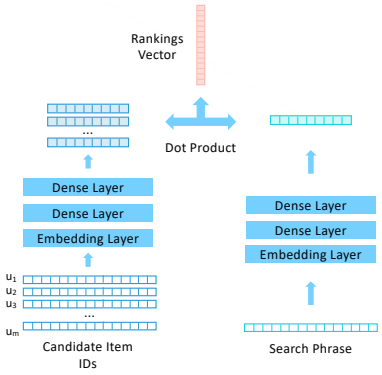

Figure 3: A diagram of the model used as the first step in the proposed reranking pipeline.

2.2.1 Combining Candidate Lists. For reranking, a two-tower model such as shown in Figure 3 can be utilized. The first tower breaks down the query into word-level n-grams, which are embedded and averaged. Similarly, the second tower embeds the item identifier being ranked. This model is trained via a standard pairwise Learning to Rank approach to capture item popularity for a given query. This approach allows the model to combine the various candidates into a cohesive list. For the experiments found in Section 3, training data consisted of aggregated click data, a "group by" sum of clicks aggregated at the (query, item ID) level, and the final pairwise training set numbered approximately 10 million pairs. The model was retrained nightly using two weeks of recent data. This model was also stored with a set where the elements were the concatenation of the query and the item identifier. If a query/item pairing was not present in the set at runtime, the model would not rerank the results and instead a business logic ordering was preserved.

2.2.2 Personalization. The second model in the pipeline personalizes the top N results. The second model's architecture (Figure 4) is inspired by published works where recurrent neural networks (RNNs) are trained end-to-end in a Learning to Rank paradigm [9, 15, 22]. RNNs used in this fashion allow one to identify the similarity between the search query and the item title. The input for the character-level LSTM layer is of the form "<s> *query* <sep> *title* <e>" concatenated with a segment embedding described by Hashemi et al. to differentiate between query characters and title characters. The LSTM output is combined with embeddings of the item IDs and metadata to form a search context vector. A dot product of the search context vector and the user search click history is computed for the final ranking.

For the experiment in Section 3, the model was trained end-to-end from data using a point-wise learning to rank approach. We only included query sessions (defined in Section 2.3.1) for training that had at least one search result click event. A typical training consisted of 150 million examples taken over a recent month of production data. The training regimen used a multi-GPU scheme and was programmed using Keras and TensorFlow [1, 6].

2.2.3 Additional Business Logic. After the reranking has been completed, it is often still desirable to apply a final step of business logic to meet product requirements. An example product requirement could be as follows: item-to-item similarity matches which do not also contain a lexical match must appear after lexical matches in the final list. To accomplish this and other similar requirements, one can classify items into primary, secondary, and tertiary groups based on the candidate generation source and metadata. For instance, item-to-item similarity matches could be secondary results, whereas lexical matches could be defined as primary results. One could then enforce that primary results are always presented before secondary results, and likewise for tertiary results. The results would remain in the order assigned by the models inside each tier. This business logic would ensure that results are presented in a reasonable and expected order.

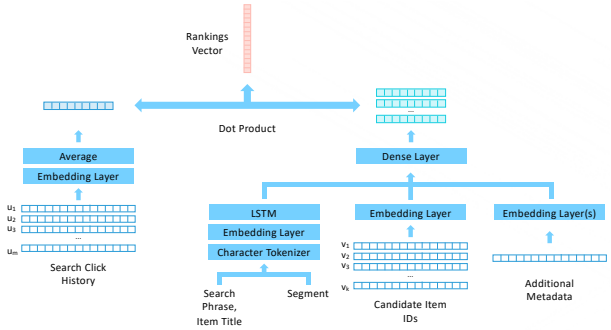

Figure 4: A diagram of the architecture of the proposed personalization model for instant search.

2.3 Metrics and Offline Evaluation

In similar settings, it has been shown that users judge the quality of a search experience by whether the search was successful and the effort it took to complete the search [10, 13]. With that in mind, one option is to implement hero metrics of Successful Search Rate (SSR), defined as the percentage of sessions ending in a search result click with no follow-up search within T minutes, and the Average Number of Keystrokes (ANK) until the correct search result is clicked. A myriad of auxiliary metrics that quantify the customer experience including Time to Success, Dead End Search Rate, and Percent of Enriched Queries (i.e., queries with non-lexical matches) can also be tracked. Metrics should be calculated at the session level and could be used for A/B testing and monitoring of the system.

2.3.1 Sessionization Logic. A key element of the approach is the definition of a search session. A search session in the instant search setting can be comprised of numerous prefix queries as a user types

in the desired query. For example, when a user is searching for "The Office", their queries would form a list L = ["*T*", "*TH*", "*THE*", "*THE* ", "*THE O*", ...] which need to be grouped together to form a "query session". Query sessions are formed via an edit distance heuristic using Levenshtein Distance which takes into account situations where letters may not be sent to the backend due to the speed of data entry. For instance, the previous query session L may be sent to an API as such: L' = ["*T*", "*THE*", "*THE* ", ...]. Query sessions can then be grouped again by time to form a "search session", where the overall experience metrics can be calculated to validate the performance of the system and characterize customer success and effort.

2.3.2 Offline Evaluation. In the off-policy evaluation literature, randomization and propensity weighting are commonly used to create accurate offline estimates of online results [5, 14, 16–19]. Many typical randomization schemes used like uniform shuffling as in Li et al. could lead to a poor customer experience. However, in many use cases, there is a concept of item availability for a user which has been discussed by other industry practitioners [13]. A user may not be eligible to view a given item due to their subscriptions, and in which case, the item would be removed from the results. This leads to different users having different result sets. This pseudo-randomization may not be sufficient for accurate estimates, but we have found in practice that offline estimates of Normalized Discount Cumulative Gain (NDCG) and Mean Reciprocal Rank (MRR) are directionally accurate when using such pseudo-randomized data as described above.

3 RESULTS

The results discussed below are from online A/B tests with metrics calculated using session data. The A/B tests were typically run for 2 weeks with treatment and control receiving an equal amount of traffic. The significance level required was $\alpha = .01$. Sample Ratio Mismatch (SRM) tests were performed to ensure the splits observed were valid. Because metrics were calculated at a session level while treatment was assigned at the account level, a delta method correction to the variance was applied. For details and best practices, we recommend Kohavi et al.

Figure 5 documents the improvements observed from introducing a similar reranking pipeline as described above. In fact, there was a strong improvement on our key metrics when only introducing the first step of the reranking pipeline and a second strong improvement when the personalization model was added. The personalized model made a larger impact on shorter queries (e.g. "*MA*"), particularly in "effort" metrics like Average Number of Keystrokes and Time to Success and relevancy metrics like NDCG and MRR. These cases will have many possible outcomes for the search phrase and the personalization model was able to improve the rankings significantly. In the cases where the user is typing a query, a search if unsuccessful is typically due to the user abandoning the session (e.g., the search took too much effort to find what the user wanted) or the item the user was searching for was not available to them. Thus, we did not expect nor did we see a large increase in Search Success Rate by deploying any machine learning based ranking; however, there was a small improvement. In the case of dictated queries matching the title of an item, like "*THE OFFICE*", the pool of search results is much smaller, and there are fewer options to reorder the results. So, in these cases we only saw a modest improvement in relevancy metrics like NDCG. However, for longer queries that are not exact title matches, we saw adequate improvements in effort and success metrics for both dictated and typed queries. Such queries frequently appear in search fallback scenarios, where a system returns "best effort" results (e.g., using edit distance based algorithms and n-gram matching) when no first order candidates are found.

Percent Improvement

Search Success Rate	Time to Click	Number of Keystrokes
.5-5%	10-20%	10-20%

Figure 5: Metric improvements of a reranking step as proposed in Section 2.2 over a global popularity sort, calculated through online experiments on typed queries. Values are given in ranges for business privacy purposes.

On the other hand, we observed the largest gains in Search Success Rate from the introduction of new indices. By handling more search use cases, the system produces fewer dead ends for search sessions, resulting in users being able to find content for more open ended queries and when the original item they searched for was unavailable.

4 DISCUSSION

Due to the dearth of papers on instant search, we have presented a potential implementation of an instant search system. The challenges faced with returning a single list containing multiple different types of results (i.e., movies, television series, music videos, sporting events, etc.) and match types (lexical, semantic, etc.) extend beyond the media domain and will be relevant for many practitioners working in product search use cases. We hope that the lessons outlined above can help others identify a path forward to implement their own large-scale instant search systems.

COMPANY PORTRAIT

Comcast Corporation (Nasdaq: CMCSA) is a global media and technology company that connects people to moments that matter. We are principally focused on connectivity, aggregation, and streaming with 57 million customer relationships across the United States and Europe. Visit www.comcastcorporation.com for more information.

PRESENTER BIO

As a senior principal researcher at Comcast Cable, Scott currently works on the Search team building deep learning models to rank results for customers on Xfinity's X1, XClass, and Flex. In the past, he had trained contextual bandit models for recommendations for the Xfinity Assistant while utilizing off-policy evaluation techniques. His previous roles include data science stints in advertising, healthcare, and finance. He holds a doctoral degree in mathematics from Drexel University and enjoys cooking, reading, and spending time with his family.

DISCLAIMER

The foregoing is a discussion of research and development and not a representation of implementations.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature Verification Using a "Siamese" Time Delay Neural Network. In *Proceedings of the 6th International Conference on Neural Information Processing Systems* (Denver, Colorado) *(NIPS'93)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 737–744.

[3] Yuri M. Brovman, Marie Jacob, Natraj Srinivasan, Stephen Neola, Daniel Galron, Ryan Snyder, and Paul Wang. 2016. Optimizing Similar Item Recommendations in a Semi-Structured Marketplace to Maximize Conversion. In *Proceedings of the 10th ACM Conference on Recommender Systems* (Boston, Massachusetts, USA) *(RecSys '16)*. Association for Computing Machinery, New York, NY, USA, 199–202. https://doi.org/10.1145/2959100.2959166

[4] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Brian Strope, and Ray Kurzweil. 2018. Universal Sentence Encoder for English. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Brussels, Belgium, 169–174. https://doi.org/10.18653/v1/D18-2029

[5] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. 2019. Top-k off-policy correction for a REINFORCE recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM, 456–464.

[6] François Chollet et al. 2015. Keras. https://keras.io.

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems* (Boston, Massachusetts, USA) *(RecSys '16)*. Association for Computing Machinery, New York, NY, USA, 191–198. https://doi.org/10.1145/2959100.2959190

[8] Apache Software Foundation. 2011. Apache Lucene - Scoring. http://lucene.apache.org/java/3_4_0/scoring.html letzter Zugriff: 20. Oktober 2011.

[9] Helia Hashemi, Aasish Pappu, Mi Tian, Praveen Chandar, Mounia Lalmas, and Benjamin Carterette. 2021. Neural Instant Search for Music and Podcast. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery Data Mining* (Virtual Event, Singapore) *(KDD '21)*. Association for Computing Machinery, New York, NY, USA, 2984–2992. https://doi.org/10.1145/3447548.3467188

[10] Christine Hosey, Lara Vujović, Brian St. Thomas, Jean Garcia-Gathright, and Jennifer Thom. 2019. Just Give Me What I Want: How People Use and Evaluate Music Search. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300529

[11] Oliver Jojic, Manu Shukla, and Niranjan Bhosarekar. 2011. A Probabilistic Definition of Item Similarity. In *Proceedings of the Fifth ACM Conference on Recommender Systems* (Chicago, Illinois, USA) *(RecSys '11)*. Association for Computing Machinery, New York, NY, USA, 229–236. https://doi.org/10.1145/2043932.2043973

[12] R. Kohavi, D. Tang, and Y. Xu. 2020. *Trustworthy Online Controlled Experiments: A Practical Guide to A/B Testing*. Cambridge University Press. https://experimentguide.com/

[13] Sudarshan Lamkhede and Sudeep Das. 2019. Challenges in Search on Streaming Services: Netflix Case Study. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Paris, France) *(SIGIR'19)*. Association for Computing Machinery, New York, NY, USA, 1371–1374. https://doi.org/10.1145/3331184.3331440

[14] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. 2011. Unbiased Offline Evaluation of Contextual-bandit-based News Article Recommendation Algorithms. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining* (Hong Kong, China) *(WSDM '11)*. ACM, New York, NY, USA, 297–306. https://doi.org/10.1145/1935826.1935878

[15] Dae Hoon Park and Rikio Chiba. 2017. A Neural Language Model for Query Auto-Completion. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Shinjuku, Tokyo, Japan) *(SIGIR '17)*. Association for Computing Machinery, New York, NY, USA, 1189–1192. https://doi.org/10.1145/3077136.3080758

[16] Alex Strehl, John Langford, Lihong Li, and Sham M Kakade. 2010. Learning from Logged Implicit Exploration Data. In *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (Eds.). Curran Associates, Inc., 2217–2225. http://papers.nips.cc/paper/3977-learning-from-logged-implicit-exploration-data.pdf

[17] Adith Swaminathan and Thorsten Joachims. 2015. Batch Learning from Logged Bandit Feedback through Counterfactual Risk Minimization. *Journal of Machine Learning Research* 16, 52 (2015), 1731–1755. http://jmlr.org/papers/v16/swaminathan15a.html

[18] Adith Swaminathan and Thorsten Joachims. 2015. The Self-Normalized Estimator for Counterfactual Learning. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 3231–3239. http://papers.nips.cc/paper/5748-the-self-normalized-estimator-for-counterfactual-learning.pdf

[19] Adith Swaminathan, Akshay Krishnamurthy, Alekh Agarwal, Miroslav Dudík, John Langford, Damien Jose, and Imed Zitouni. 2016. Off-policy evaluation for slate recommendation. *CoRR* abs/1605.04812 (2016). arXiv:1605.04812 http://arxiv.org/abs/1605.04812

[20] Alexandre Tamborrino. 2022. *Introducing Natural Language Search for Podcast Episodes*. https://engineering.atspotify.com/2022/03/introducing-natural-language-search-for-podcast-episodes/

[21] Ganesh Venkataraman, Abhimanyu Lad, Viet Ha-Thuc, and Dhruv Arya. 2016. Instant Search: A Hands-on Tutorial. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Pisa, Italy) *(SIGIR '16)*. Association for Computing Machinery, New York, NY, USA, 1211–1214. https://doi.org/10.1145/2911451.2914806

[22] Tian Wang, Yuri M. Brovman, and Sriganesh Madhvanath. 2021. Personalized Embedding-based e-Commerce Recommendations at eBay. *CoRR* abs/2102.06156 (2021). arXiv:2102.06156 https://arxiv.org/abs/2102.06156