Bootstrapping Interactive Recommender Systems

Sally Goldman, Dima Kuzmin, Steffen Rendle, Li Zhang
Moustafa Alzantot, Ajit Apte, Ajay Joshi, Anand Kesari
Santiago Ontanon, Anushya Subbiah, Martin Zinkevich
John Anderson, Monica Lenart, Sarvjeet Singh, Cathy Yip
{sgoldman,dimakuzmin,srendle,liqzhang}@google.com
Google Research
Mountain View, CA, USA

ABSTRACT

Modifying recommender systems for new kinds of user interactions is costly and exploration is slow since machine learning models can be trained and evaluated on live data only after a product supporting these new interactions is deployed. Our data bootstrapping approach moves the task of developing models for new interactions into the input representation allowing a standard machine learning model to be used to train a model capturing the new interactions. More specifically, we use data obtained from a launched system to generate simulated data that includes the new interactions options. This data-centric approach helps accelerate model and algorithm development, and reduces the time to launch new interaction experiences. We also present machine learning methods designed specifically to work well with limited and noisy data produced via data bootstrapping.

CCS CONCEPTS

• **Computer Methodologies-Artificial Intelligence-Learning**;

KEYWORDS

Recommender Systems, Machine Learning, Data Bootstrapping, Interactive Recommender Systems, Information Retrieval

ACM Reference Format:
Sally Goldman, Dima Kuzmin, Steffen Rendle, Li Zhang, Moustafa Alzantot, Ajit Apte, Ajay Joshi, Anand Kesari, Santiago Ontanon, Anushya Subbiah, Martin Zinkevich, and John Anderson, Monica Lenart, Sarvjeet Singh, Cathy Yip. 2022. Bootstrapping Interactive Recommender Systems. In *SIGKDD '22: ACM Special Interest Group on Knowledge Discovery and Data Mining, August 14–18, 2022, Washington, DC*. ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Modern recommender systems are powered by, or heavily rely on, machine learning techniques that learn from the logs of user-interactions on a launched system. In this process, the recommender system tries to learn the "best" system response for a user action. Since these models mainly learn from user interactions with launched systems, they are effective when the logged data faithfully captures all interactions (user actions and system responses) supported by the product user interface. We refer to the data obtained from such logs as *native data*. Introducing new UIs (e.g. critiquing, suggestion chips, or explicit user preferences) to an existing product leads to the following novel problem that is the focus of our work. *How can we apply machine learning to interactive recommenders for which there is no native data?*

Building a recommender system for which there is no native data is very costly making exploration of new UI elements time consuming since only after a product supporting these new interaction options is built and deployed for a sufficient time can a ML model can be trained and evaluated on live traffic. One approach is to use rule-based system or some other ad-hoc approach to build a system from which one can then obtain native data. A novel aspect of our work is a data-centric approach to this problem that moves the task of developing models for new interactions into the data representation allowing a standard machine learning model to be used to train a model capturing the new interactions. Once native data is obtained then more traditional, well-studied machine learning approaches can be applied.

Our approach is based upon **data bootstrapping**, in which data obtained from a launched system is used to generate simulated data for a system with new user interactions to help accelerate model and algorithm development, and reduce the time to launch new interaction experiences. Data bootstrapping is different from bootstrapping recommender systems with respect to new users or items [7] or statistical bootstrapping methods such as boosting or bagging. In this paper we illustrate our framework on three examples of new user experiences: critiquing, suggestion chips, and explicit user preference. The solutions that we describe focus on simplicity for demonstrating the application of our framework.

Observe that for each new user interaction introduced, we need to design a method specific to that problem to generate simulated data that captures the new interactions using the existing data. The key is that once this is done the same ML approach is used for all the problems versus having to create rule-based or ad-hoc approach to train the initial models from which native data can then be obtained.

We also present ML modeling approaches geared towards this setting, and evaluation methods that enable a faster development cycle than traditional training approaches and evaluations based on A/B testing. Our machine learning architecture consists of (1) an

SIGKDD 2022, August 14–18, 2022, Washington, DC
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

input representation that supports different entity types including annotations (e.g., modifier, position), (2) a projection that maps the symbolic input to an intermediate representation, and (3) a prediction task that utilizes the intermediate representations to generate useful outputs, for example, relevance scores. Our machine learning models are designed to work well with limited and noisy data that has been produced by our data bootstrapping methods. It uses pretrained embeddings for general domain understanding and utilizes this both for content and user representation. The model is tailored for combining these signals with a low number of free parameters, that can be learned from small and noisy bootstrapped data. After an initial launch, the bootstrapped components can be slowly transitioned to the traditional ML approach using real logs from the bootstrapped system, and the machine learning model can be made less dependent on external data like pretrained embeddings.

2 DATA BOOTSTRAPPING FRAMEWORK

In this section we formalize our data bootstrapping framework. We assume that we have access to data/logs D from a launched recommender system $\mathcal{S}$ with user action space $\mathcal{A}$ and system response space $\mathcal{R}$ to bootstrap recommendation system $\mathcal{S}'$ with richer interactions as captured by user action space $\mathcal{A}'$ and system response space $\mathcal{R}'$. While this framework is broader than its use for recommender systems, here we focus on this use case. Data bootstrapping can be achieved through a combination of launched product logs, user simulations, and rater interactions with a prototyped system that include the new product UIs.

Let $D = \langle A_i, R_i \rangle^*$ be logged data from the launched recommender system $\mathcal{S}$ where $A_i \in \mathcal{A}$ is the i^{th} user action and $R_i \in \mathcal{R}$ is the system response to A_i. We illustrate our framework on a standard query-based recommender system $\mathcal{S}$ in which $\mathcal{A}$ includes sending a query or clicking to engage with one of the recommendations (e.g., listen to music, play a video, open a URL), and $\mathcal{R}$ includes sending a slate (ordered list) of recommendations back to the user or responding to the user click. For demonstration of our framework, we extend $\mathcal{S}$ with the following new interaction options (which can be individually considered or combined):

Critiquing $\mathcal{A}'$ adds to $\mathcal{A}$ critiques of the form $\langle$ modifier, term $\rangle$ from which the expectation is that the next slate changes as specified (e.g. ⟨more, chicken⟩ or ⟨not, spicy⟩ for a recipe recommender). The zero-state slate can be modeled by an empty query. For a survey on the critiquing problem see [4].

Suggestion Chips $\mathcal{R}'$ adds to $\mathcal{R}$ suggestion chips (proposed critiques) to steer their recommendations. $\mathcal{A}'$ adds to $\mathcal{A}$ the action of clicking on one of the provided suggestion chips. See [11] for an overview of work on preference elicitation within recommender systems.

Explicit User Preferences Allowing the system to ask for the user's preferences before or during the recommender experience. Let $\mathcal{S}'$ include a set of terms and actions (e.g. genre, artist for a music recommender) each with a thumbs-up button, and $\mathcal{A}'$ includes clicking on any number of thumbs up. There is no direct system response but rather the expectation that a user's recommendations are influenced by their stated preferences.

Note that our bootstrapping framework is not limited to these three use cases above. We use them as examples to illustrate the framework.

This paper categorizes different approaches to transform logs D of an existing product to a bootstrapped data set D' to train a ML model M' for a recommender system $\mathcal{S}'$ with richer interactions. In particular, we look at **offline bootstrapping** in which D' is generated only using D, and **online bootstrapping** in which a user or user simulation interacts with a prototype for S' in order to generate D' that captures the way in which users might naturally interact with $\mathcal{S}'$, which might differ dramatically from the way that user interact with $\mathcal{S}$. A combination of offline and online bootstrapping can accelerate model and algorithm development and reduce the time to launch new interaction models

3 RELATED WORK

Machine learning approaches for recommender systems have been extensively studied (e.g. [5, 9]), and much previous work discusses the difficulties and time-consuming development cycle in designing the first iterations of machine learned recommender systems and the difficulties in debugging ML pipelines, and the lack of appropriate training data. There is work (e.g. [10]) that discusses the challenges in the ML pipeline related to deployment in a launched system including data management but don't include any discussion of data bootstrapping.

There is research focused on ML deployment that include some form of data bootstrapping. For example, [12] focuses on bootstrapping large-scale recommender systems as a way to significantly reduce development cycles and bypasses various real-world difficulties including issues related to data sparsity and data skew issues related to user interface options. A longitudinal study of three years of development of related pins in Pintrest, exploring the evolution of the system and its components rom prototypes to present state is presented in [9]. The focus on how the initial product must be built without training data until the recommender s bootstrapped but their solution focuses on engineering solutions to move from purely heuristic to machine learned systems. The use of coview data logged from the same browsing sessions is used by eBay [2]. There is also work focused on debiasing user feedback, assuming the existence of interaction logs and an extra annotated dataset[13].

There is also work on using pretrained models to help address data limitations. For example [6] using pretrained NLP models. A technique based on a pointwise ranking approach to reduce the ranking problem to a binary classification problem optimized on past user purchase behavior is used by [2].

As far as we are aware, our work is the first that focuses on data bootstrapping, and presents a machine learning architecture and learning techniques that are geared toward this setting. We illustrate our approach using examples in which new user interaction interfaces are added to already launched products.

4 OFFLINE BOOTSTRAPPING

One approach to bootstrapping is *offline bootstrapping* in which $D = \langle A_i, R_i \rangle^*$ obtained from a launched recommender system $\mathcal{S}$, is used as input to a data transformation function F to obtain $D' = F(D)$. A single interaction in D could be mapped to any number (possibly

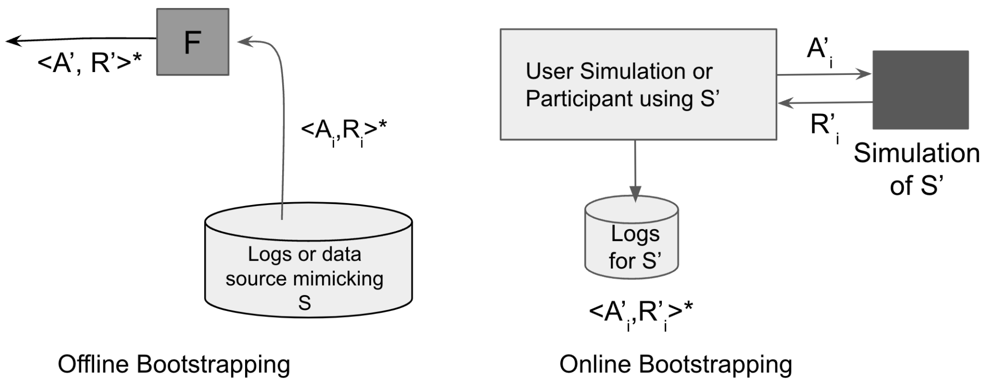

Figure 1: Offline and Online Data Bootstrapping

0) interactions in D'. The left diagram of Figure 1 illustrates this approach. The transformation F could be performed in a variety of ways including: transforming $\langle A_i, R_i \rangle$ to $\langle A'_{i'}, R'_{i'} \rangle^*$, transforming or using aspects of D to generate D', or generating D' independent of D. For example for the Critiquing problem, we could find sequence of queries for which terms are added at each step, and apply this as training data for training a critiquing model. See Section 6 for details.

The key drawback of this approach is that user interactions in the launched system might not faithfully capture the natural interactions of the user when provided with the expanded interactions available in $\mathcal{S}'$.

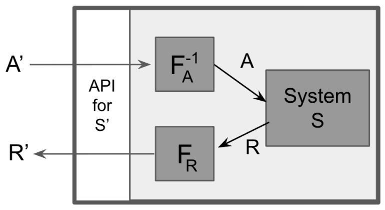

Figure 2: Creating a prototype for S' by wrapping S along with functions to transform A' to A and R to R'.

5 ONLINE BOOTSTRAPPING

Another approach is illustrated in the right diagram of Figure 1 in which a rater directly interacts with the expanded interactions options available in $\mathcal{S}'$. This approach allows training data to be obtained that captures natural user journeys with the UI of $\mathcal{S}'$. The drawback of online bootstrapping is that it adds a reliance on the ability to simulate S'. That is, a way to map user action A'_i to system response R'_i. We refer to this as a prototype for S'.

Within online bootstrapping, we need to generate both the User actions and System responses. The two broad classes of **User actions modeling** are

Raters selected to match end users as best possible interact with the system potentially guided to follow some specified user case in order to capture the way in which Users will want to interact with the system.

User Simulations in which a user model simulates the interactions with the system. This area of work is very well studied (e.g. [16]) so we don't further address it here. In general, some sort of Rater-based approach is needed to develop and validate the User Simulation model.

This data-centric approach helps accelerate model and algorithm development and reduce the time to launch new interaction experiences. These two models can be used together with raters to generate some data for fine-tuning and to validate the user model with the user simulations used for generating data for training models.

Another key aspect for online bootstrapping prototypes is the way in which the system response r is computed for user action a. The broad category of options for addressing this problem are: as follows.

Human is when a person is fully responsible of generating R'. The drawback of this approach is that it is not grounded in any backend and can't leverage the knowledge and corpus breadth of a machine learned system.

Algorithmic in which heuristics or a learned model is used to build a prototype for $\mathcal{S}'$ by wrapping the trained machine learning model M for $\mathcal{S}$) through functions F_A^{-1} that transforms A'_i to A_i and F_R that transform R_i to R'_i as illustrated in Figure 2. The drawback of this approach is that the user interface of $\mathcal{S}$ often lacks some non-trivial functionality needed in computing both F_A^{-1} and F_R for which creating a heuristic or machine model is on-trivial.

Wizard-of-Oz (Human-in-the-loop) in which a human-in-the-loop (HIL) helps ensure that the prototype performs well so that the user experience matches that of the aspirational system so that the data captured will represent natural user journeys within S'. having a HIL that provides an intermediary between the other two approaches in which the human can correct F_A^{-1} and F_R to help fill gaps to both gather training data to train ML model M' for S' and for evaluation including headroom analysis in which the HIL can provide the function of a system component that might be hard to build in order to measure how much the system performance and user experience are affected by having this component.

6 BOOTSTRAPPING EXAMPLES

In this section, we describe and discuss advantages and drawbacks of several data bootstrapping approaches for the problems described in Section 1. For online bootstrapping, we focus on the algorithmic approach. Having a human directly respond or introducing a wizard-of-oz can supplement the algorithmic approaches described below.

6.1 Critiquing Example

The intuition behind our offline bootstrapping is to use the portions of D that can be viewed as a sequence of critiques from a slate. We first partition D into a sequence of action and responses from a single session that ends in an engagement with an item in the previous slate. For ease of exposition, denote one such a sub-sequence as $D_P = A_0, A_1, \ldots, A_{k-1}, A_k$ and since all actions other than A_k are queries, we write this as $Q_1, Q_1, \ldots, Q_{k-1}, E$ where E indicates the item the user selected. We define **Critique** (Q_i, Q_j) to denote a procedure that takes a bag of words representation of Q_i and Q_j and uses NLU to generate a critique $C = \langle$ modifier, token $\rangle$ if Q_j can be modeled as Q_i followed by C and otherwise null is returned. For example, **Critique**("chicken recipe", "Thai recipe") would return null whereas **Critique**("Chinese recipe", "Chinese chicken recipe") would return the critique ⟨more, chicken⟩. One could decide upon the desired semantics, here we view "chicken recipe" to "Thai recipe" as abandoning the request for a chicken recipe and instead asking for a Thai recipe.

Algorithm 1 Critique Offline Bootstrapping Example

```
1:  procedure CritiqueBootstrap($<A, R>$)
2:      $A = \langle Q_0, \ldots, Q_{k-1}, E\rangle$                ▷ $Q$ query, $E$ item click
3:      $R = \langle R_0, \ldots, R_k\rangle$                       ▷ $R$ system response
4:      $A'_0 \leftarrow Q_0, R'_0 \leftarrow R_0$
5:      for $i \leftarrow 1$ to $k-1$ do
6:          $C = \text{Critique}(Q_{i-1}, Q_i)$   ▷ model as critique if possible
7:          if $C$ is not null then
8:              $A' \leftarrow A + C, R' \leftarrow R' + R_i$
9:          else        ▷ can't model $Q_i$ as critique applied to $Q_{i-1}$
10:             Clear $A'$ and $R'$
11:             $A'_0 \leftarrow Q_i, R'_0 \leftarrow R_i$
12:     $A' \leftarrow A' + E, R' \leftarrow R' + R_k$
13:     return $\langle A', R'\rangle$
```

To make this concrete, let's suppose we have the sequence of queries "chicken recipe", "Thai recipe", "Chinese recipe", "Chinese chicken recipe", "Chinese chicken recipe not spicy" followed by engagement (the user clicks on a result returned from the query "Chinese chicken recipe not spicy"). Our goal is to convert this into a training example for $\mathcal{S}'$. To achieve this goal, we find a suffix of the sequence of queries for which each query can be modeled as a critique of the one that preceded it. For this example, we would drop "chicken recipe" and "Chinese recipe" leaving us with an example starting with the query "Chinese recipe" followed by the critique ⟨more, chicken⟩ and then ⟨not, spicy⟩. We apply Algorithm 1 to all such sub-sequences in D. While this only represents a subset of the traffic, it provides a way to obtain data set that can be used as an initial step in training a model for $\mathcal{S}'$.

An alternative is to use an online bootstrapping approach. In order to do this some technique is needed to generate a system response R' for user action A'. One simple solution is to maintain a query q that is initialized when the user issues a query. Then for ⟨modifier, term⟩ simply append the string "modifier token" to q up until the user issues a new query, at which point q is reset.

6.2 Suggestion Chips Example

Our bootstrapping approach replaces two-turn conversations within D where the user issues a query, makes a refinement, and then clicks on an item in the returned slate by a simulated experience in which the first part of this interaction is replaced by the selection of a suggestion chip included with the initial slate. As part of our solution, we also use of a pretrained joint embeddings between the items in the slate (e.g. videos) and preference terms (e.g. from metadata or other sources) for the items in the slate.

Algorithm 2 describes our bootstrapping in detail. We start with the portions of the logs D of the form $\langle R_{i-1}, A_i, R_i, \text{click } r_i\rangle$ in which R_{i-1} is the context (slate) for query A_i, R_i is the slate returned, and A_{i+1} is the user selecting slate item $r_i \in R_i$. We assume $|R_i|$ is $O(100)$. We use our pretrained embeddings to cluster the items in R_i and name each cluster using a representative term (e.g. a cuisine, ingredient, genre, artist,...). Observe that a term is different than a token which is a word, bi-gram, sequence of words within a query. Next, we use this portion of D to simulate an interaction for $\mathcal{S}'$ in which suggestion chips are added to R_{i-1} and the user selects one in lieu of issuing query A_i.

We randomly select whether to use "MORE" or "NOT" modifiers in the simulated suggestion chips. For ease of exposition, in our simulated data all suggestion chips in a response use the same modifier. We could instead randomly selecting a modifier for each simulated chip. Also, we select the term to place on the chips independent of A_i but could incorporate that. More complex approaches, such as that of [15] which presents a gradient-based EVOI optimization could be applied to the bootstrapped data.

For this problem, using an on-line bootstrapping approach is more challenging since a heuristic would be needed to generate suggestion chips to show to the user. One way to do directly use metadata associated with the results in the previous slate. Having a Wizard-of-Oz would be a good online bootstrapping approach for this problem, since the Wizard could generate some suggestion chips which would provide good training data.

6.3 Explicit User Preferences Example

Similar to our offline bootstrapping algorithm for the Suggestion Chip problem, we will use a clustering-based approach for the Explicit User Preference problem. The key difference is that here we simulate user preferences via selecting portions of D from some sample users. For User i, let R_i be the set of all items in the slates that the user saw and let $C_i \subset R_i$ be those that the user clicked on. We then model the user clicking thumbs up button for a terms in clusters with items in C_i.

For online bootstrapping, one good option here is to use a curated set of preference options.

Algorithm 2 Suggestion Chip Offline Bootstrapping Example

```
1: procedure SuggestionChipBootstrap($<R_{i-1}, A_i, R_i, r_i>$)
2:     $D' \leftarrow \{\}$
3:     for each tuple $R_{i-1}, A_i, R_i, r$ do                ▷ $r_i \in R_i$ selected
4:         cluster $R_i$ using a pretrained joint embedding
5:         for each cluster $c$ do
6:             $t_c \leftarrow$ term that maximizes mutual info for $c$
7:         $x \leftarrow$ cluster that includes item $r_i$
8:         randomly select modifier of "MORE" or "NOT"
9:         if modifier is "MORE" then
10:            $A'_i$ = click "MORE $t_x$"
11:            Chips $C_i \leftarrow$ "MORE $t_x$"      ▷ simulate selected chips
12:            for some random $x^-$ not in cluster $x$ do
13:                $C_i \leftarrow C_i$+ "MORE $t_{x^-}$"          ▷ add more chips
14:            $R'_{i-1} = R_{i-1}$ with chips $C_i$         ▷ show $C_i$ with $R_{i-1}$
15:        if modifier is "NOT" then          ▷ simulate "not" chips
16:            pick random $y \in R_i$ not in cluster $x$
17:            $A'_i$ = click "NOT $t_y$"
18:            Chips $C_i \leftarrow$ "NOT $t_y$"
19:            for random $x^-$ in cluster $x$ do
20:                $C_i \leftarrow C_i$+ "NOT $t_{x^-}$"
21:            $R'_{i-1} = R_{i-1}$ with chips $C_i$
22:        $R'_i \leftarrow R_i, A'_{i+1} \leftarrow$ click $r_i$             ▷ So $A'_{i+1} = A_{i+1}$
23:        add to $D'$: $\langle R'_{i-1}, A'_i, R'_i, A'_{i+1}\rangle$
24:    return $D'$
```

Algorithm 3 Explicit Preference Offline Bootstrapping Example

```
1: procedure ExplicitPreferenceBootstrap($<C_i, R_i>$)
2:     Input: $\langle C_i, R_i\rangle$                 ▷ $C_i \subset R_i$ given thumbs-up
3:     $D' \leftarrow \{\}$
4:     for all $i$ do
5:         cluster $R_i$ using a pretrained joint embedding
6:         for each cluster $c$ do
7:             $t_c \leftarrow$ term that maximizes mutual info for $c$
8:         Initialize Preference options $R'_i \leftarrow \{\}$
9:         Initialize thumbs up actions $A'_i \leftarrow \{\}$
10:        Clicks $\leftarrow$ clusters containing some item in $C_i$
11:        for cluster $x$ in Clicks do
12:            $R'_i \leftarrow R'_i + x$
13:            $A'_i \leftarrow A'_i$+ thumbs up for $x$
14:        for random $y$ not in Clicks do
15:            $R'_i \leftarrow R'_i + y$
16:        add to $D'$: $\langle R'_i, A'_i\rangle$
17:    return $D'$
```

7 MACHINE LEARNING TECHNIQUES

Our ultimate goal is to train a machine learning model, using the bootstrapped data, to predict the best response to a (potentially hypothetical) user action. In principle, this can be solved as a generic machine learning problem, orthogonal to the data bootstrapping procedure. However, the tension between the rich interaction we wish to support and the relative lower quality and/or quantity of the training data we create through bootstrapping poses as a major challenge to the modeling task. In this section we describe a model architecture and demonstrate its effectiveness for solving this challenge.

7.1 Supporting Rich Interaction with Limited Data

Since we would like to support novel user interactions, they tend to be rich, often sophisticated, for example, involving better understanding of the context or the preferences of different forms. On the other hand, the bootstrapped data may not match the product scenario well. Or it may have too low quantity or coverage, if generated with human in the loop. Hence, it would be very difficult, to train such a model from scratch using only the bootstrapped data.

We take a two pronged approach to deal with this challenge. On one hand, we rely on pre-trained features to capture the "innate" semantics of the entities/user preferences. On the other hand, we create granular models to fuse the semantics from multiple sources and to interpret the user's interactions in this "fused" semantic space.

Pre-trained features. It is common to utilize pretrained features, such as embeddings, when there is a lack of training data. For our purpose, it is also important to utilize features from difference sources. For example, in order to understand the user's implicit preference, it would be useful to utilize the embeddings learned from implicit feedback, e.g. by the collaborative filtering method on the co-visit pattern. On the other hand, to understand the user's explicit preference, it would be useful to utilize the "content" embeddings, e.g. derived from the metadata of the entities.

Granular fusing. Since we utilize pre-trained features from diverse sources, it requires our model to be able to "fuse" them into a common semantic space. In addition, to support rich interaction, our model needs to be able to model the context, for example the sequence of interactions, at granular levels. Our model hence need to "inter-weave" along both the feature axis and the action axis.

To summarize, our model needs to take the history of the interaction, represented as features or embeddings from different sources, and summarize it into an intermediate representation, or embeddings, for the final prediction tasks. The model will need to be able to model diverse entities/actions as well as features from diverse sources. In addition, our model needs to be convenient in handling different scenarios so to relieve the user from complicated modeling tasks. In the following, we present a data-centric modeling technique which generalizes the position embeddings in the Transformer model to tree structured data for satisfying these multiple needs.

7.2 Machine Learning Method

The key motivating observation behind our machine learning method is that the diverse entities, interactions, and features can be very well represented by tree structured data. Hence most of our needs would be satisfied if we could have a model architecture which can represent any tree structured data in its general form while taking the structure into consideration.

In addition, the Transformer model architecture, by keeping the attention between all the pairs of the input data, is well suited to model granular correlations among the input. Instead of using relative position encodings to capture structure (as done in previous work in Transformers to capture tree structure [1]), we propose a different modeling approach that emphasizes the tree structure by first "projecting" all the leaves in the tree to a common space, guided by the tree structure, and then combining them using the Transformer model. Specifically, the our machine learning approach creates the final representation of the input data through the following steps.

Representation. Represent the input as a tree, where each leaf node is associated with an input feature, and each internal node represents the data "type" or "modifier".

Parameterization. Parameterize each leaf node as an embedding (usally a pre-trained embeddings) and each internal node as a transformation, for example, a two-layer neural network.

Projection. For each leaf node, apply the internal node transformation on the path leading to the root to the leaf embeddings. Intuitively, this is to "project" all the leaf embeddings in the same semantic space for further combination. It is especially important when the leaf embeddings are pre-trained.

Combination. Apply the Transformer model to the projected embeddings, and output the top layer embeddings as the representation of the data.

We have described above how we map the input data into an internal representation, or embeddings. This representation can be further fed into any final prediction task e.g. with regression loss or pair-wise rank loss, for training the model. Figure 3 illustrates how they are used for learning a ranking score of a "Target" given a "Slate" and a "Query" using the pair-wise loss – first apply multiple neural network layers to map the internal representation to a score, and then create pair-wise rank loss between positive example and negative example.

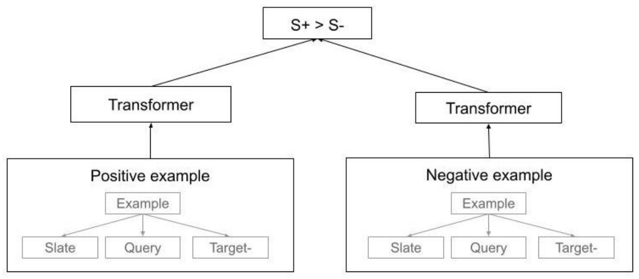

Figure 3: Reranker trained using pairwise loss.

7.3 Examples

Below we give a detailed example of the above steps for the offline bootstrapping algorithms given for the critiquing example from Section 6. The same basic approach also applies for the suggestion chip and explicit preference examples.

Representation. We use the Critique training data from Algorithm 1 to illustrate our input representation which is shown in Figure 4. Recall that A_0' is a user query for which the system response is the slate R_0' which we'll say consists of n items. We capture the sequence of critiques $A_1', \ldots, A_k'$ where each critique is represented by its index (position), modifier (e.g. "more"), and token (e.g. "chicken"). For the suggestion chip and explicit preference examples, the critique portion of the graph is replaced by suggestion chips, and for the explicit preference problem there might not be any context.

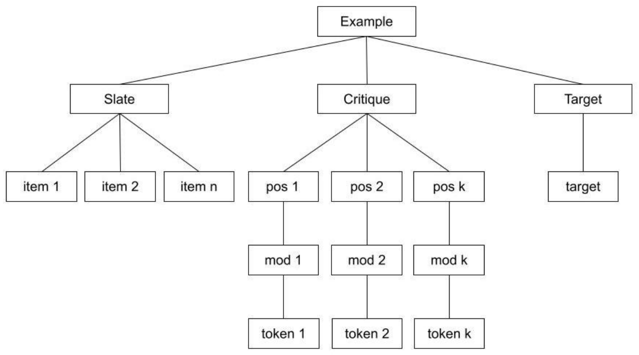

Figure 4: Training example graph for the Critique problem that captures the position within the critique of each modifier that itself is represent by the modifier (e.g. "more", "not") and the token being modified.

One slight complexity of the representation is that the critique (or query in some other applications) might be tokenized into overlapping terms: they can be unigrams, bigrams, and named entities. Hence the position node in Figure 4 is best modelled as an interval node which indicates the span of each term. One strategy is to directly replace each position node with an interval node, indexed by both the start and end positions. The other strategy is to split each interval node into two nodes, as shown in Figure 5.

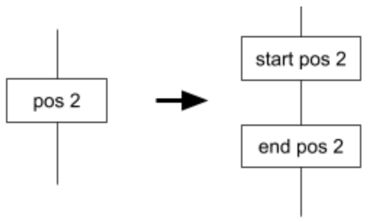

Figure 5: Replacing a position node by interval nodes.

Note that if we have features from different sources, we can further split the leaf node by adding an internal node to indicate the source of the input features.

Parameterization. Given a data item represented as a tree structure, we obtain its representation throughout parameterization, projection, and combination steps. In parameterization, the leaf nodes are parameterized as dense vectors, a.k.a. embeddings. They can be numerical values, viewed as one dimensional embeddings, or higher dimensional embeddings from categorical values. And these embeddings can be pre-trained, i.e. as numerical values which are not modified during the training, or train-able during the training.

The internal nodes are parameterized as transformations (possibly with parameters), e.g. a two layer neural network. For example, the node "pos-k" in Figure 4 can be parameterized as the addition of the input embeddings to a trainable vector dependent on k (or position embeddings).

Projection. For each leaf node, we project the leaf embeddings by following the path from the leaf to the root by composing the "internal-node" transformation to the leaf node. For example, the "pos-k" node would add the position embeddings to the input, equivalent to the position embeddings in the Transformer model. For the transformation composition to work, we need to make sure the embedding dimensions can be "chained" through each path, i.e. the output dimensions of a node should match the input dimensions of the node above it. In addition, we require the output embedding size to be equal for the nodes below the root, i.e. "Slate", "Critique", "Target" nodes in Figure 4.

Combination. All the leaf nodes, after the projection, are now mapped to the space with the same dimensions. These projected embeddings become the input embeddings to the Transformer model. The output is then treated as the final representation of the data.

8 EVALUATION

We briefly explore alternatives to traditional A/B testing for evaluation. Figure 6 illustrates an approach in which a rater receives the entire set of user actions and system responses and then both rates the overall performance as well as individual response quality and may also add annotations (e.g. marks erroneous responses). See [8] as just one example. Observe that this can be applied with offline bootstrapping approaches as well as with online approaches. In the case of online approaches the rater could be a different or the same as the person serving as the User. When using online bootstrapping, another option is to also include evaluation steps as the interaction is happening as illustrated in Figure 7. Along with more standard evaluations measuring quality, when including a human-in-the-loop to help provide human-level performance in the prototype for $\mathcal{S}'$, one can measure the headroom (improved performance) possible in a fully functioning S'. When a User simulation model is used instead of the User, by having this model also simulate user satisfaction, this User simulation model can also be used for the evaluation (e.g. [16]. Another research direction is to have a simulated model for user satisfaction and compare the performance on simulated and human evaluations as a way to validate the user satisfaction model (e.g. [14]).

There is work such as MultiWOZ [3] introduces a data set of a fully-labeled collection of human-to-human dialogues. We have also applied a novel approach in which there is a human-in-the-loop (who we will refer to as the Wizard). In this, scenario a crowd compute worker serving as the User will connect to a plugin that has been instrumented to pass requests between the user and backend through a Wizard who can help facilitate the conversation both making it more natural and reducing dead-ends. By doing so, you can both capture data of an improved experience and the

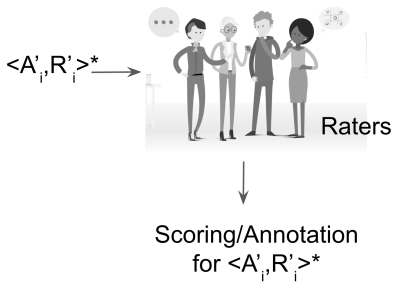

Figure 6: Post session evaluation.

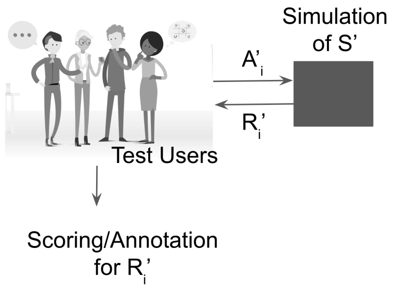

Figure 7: In session evaluation.

interactions of the Wizard to achieve these aspirational experiences. Secondly, you can measure headroom for a new feature by having and experimental set-up in which a user does not know whether or not there is a Wizard. Half of the time, there will be a Wizard who will follow directions in a way to meet the goal of the new feature (e.g. deciding when to ask the User a question in a conversational system). Then user evaluations (at the conversational turn and overall) can be used to measure how much the new feature being considered will improve the user experience with respect to the particular evaluation set-up created.

9 CONCLUSION

We presented a novel data bootstrapping framework that moves the task of developing models for new interactions into the input representation allowing a standard machine learning model to be used to train a model capturing the new interactions. We illustrated our framework on three examples of new user experiences: critiquing, suggestion chips, and explicit user preference. We also presented ML modeling approaches geared towards this setting to enable a faster development cycle than traditional training approaches and evaluations based on A/B testing. Our machine learning architecture consists of (1) an input representation that supports different entity types including annotations (e.g., modifier, position), (2) a transformer that maps the symbolic input to an intermediate representation, and (3) a prediction task that utilizes the intermediate representations to generate useful outputs, for example, relevance scores, and is designed to work well with limited and noisy data that has been produced by our data bootstrapping methods.

REFERENCES

[1] Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. ETC: Encoding Long and Structured Inputs in Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 268–284.

[2] 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*. Association for Computing Machinery, New York, NY, USA, 199–202.

[3] Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. MultiWOZ - A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Brussels, Belgium, 5016–5026.

[4] Li Chen and Pearl Pu. 2012. Critiquing-based recommenders: survey and emerging trends. *User Modeling and User-Adapted Interaction* 22, 1 (2012), 125–150.

[5] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. Association for Computing Machinery, New York, NY, USA, 191–198.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. (2019), 4171–4186.

[7] Nadav Golbandi, Yehuda Koren, and Ronny Lempel. 2010. On Bootstrapping Recommender Systems. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*. Association for Computing Machinery, New York, NY, USA, 1805–1808.

[8] Aldo Lipani, Ben Carterette, and Emine Yilmaz. 2021. How Am I Doing?: Evaluating Conversational Search Systems Offline. *ACM Trans. Inf. Syst.* 39, 4 (2021), 1046–8188.

[9] David C. Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C. Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. 2017. Related Pins at Pinterest: The Evolution of a Real-World Recommender System. In *Proceedings of the 26th International Conference on World Wide Web Companion*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 583–592.

[10] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D. Lawrence. 2022. Challenges in Deploying Machine Learning: A Survey of Case Studies. *ACM Comput. Surv.* (2022).

[11] Pearl Pu and Li Chen. 2008. User-involved preference elicitation for product search and recommender systems. *AI magazine* 29, 4 (2008), 93–93.

[12] Zhen Qin, Honglei Zhuang, Rolf Jagerman, Xinyu Qian, Po Hu, Dan Chary Chen, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2021. Bootstrapping Recommendations at Chrome Web Store. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, New York, NY, USA, 3483–3491.

[13] Tobias Schnabel and Paul N. Bennett. 2020. *Debiasing Item-to-Item Recommendations With Small Annotated Datasets*. Association for Computing Machinery, New York, NY, USA, 73–81.

[14] Ryuichi Takanobu, Qi Zhu, Jinchao Li, Baolin Peng, Jianfeng Gao, and Minlie Huang. 2020. Is Your Goal-Oriented Dialog Model Performing Really Well? Empirical Analysis of System-wise Evaluation. In *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Association for Computational Linguistics, 1st virtual meeting, 297–310.

[15] Ivan Vendrov, Tyler Lu, Qingqing Huang, and Craig Boutilier. 2020. Gradient-Based Optimization for Bayesian Preference Elicitation. *Proceedings of the AAAI Conference on Artificial Intelligence* 34 (2020), 10292–10301.

[16] Shuo Zhang and Krisztian Balog. 2020. Evaluating Conversational Recommender Systems via User Simulation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, New York, NY, USA, 1512–1520.