Building Higher-Order Abstractions from the Components of Recommender Systems

Serdar Kadıoğlu[1, 2], **Bernard Kleynhans**[1]

[1] AI Center of Excellence, Fidelity Investments, Boston, USA
[2] Department of Computer Science, Brown University, Providence, USA
serdar.kadioglu@fmr.com, bernard.kleynhans@fmr.com

Abstract

We present a modular recommender system framework that tightly integrates yet maintains the independence of individual components, thus satisfying two of the most critical aspects of industrial applications, *generality* and *specificity*. On the one hand, we ensure that each component remains self-contained and is ready to serve in other applications beyond recommender systems. On the other hand, when these components are combined, a unified theme emerges for recommender systems. We present the details of each component in the context of recommender systems and other applications. We release each component as an open-source library, and most importantly, we release their integration under MAB2REC, an industry-strength open-source software for building bandit-based recommender systems. By bringing standalone components together, MAB2REC realizes a powerful and scalable toolchain to build and deploy business-relevant personalization applications. Finally, we share our experience and best practices for user training, adoption, performance evaluation, deployment, and model governance within the enterprise and the broader community.

Introduction

The applications of personalization and recommenders are widespread in the industry, see, e.g., (Amatriain and Basilico 2016) for a comprehensive overview. Despite being so pervasive, building recommender *systems* still require complex machinery of data preparation, transformation, modeling, and production deployment. Each of these components requires significant domain expertise and serious engineering effort to deploy a robust system that can operate at scale.

Thanks to the decade-long efforts in the field, several specialized frameworks exist today for building recommender systems. Most prominent players of the technology industry contributed dedicated software such as Torch Recommenders from Meta (Naumov et al. 2019), TensorFlow Recommenders from Google (Pasumarthi et al. 2019), Recommenders from Microsoft (Argyriou, González-Fierro, and Zhang 2020), and Merlin Recommenders from NVIDIA (Oldridge et al. 2020). These frameworks can be used to create models for large-scale recommendation systems with a strong focus on scalability, performance, and deployment.

Despite these efforts, two issues remain; one challenge for users of these systems and another challenge for companies who want to deploy them.

First, these are monolithic frameworks that are specialized in one and only one task, i.e., building recommenders. The effective use of these systems still requires expert skills. This poses a significant learning curve for data scientists and practitioners, and even with the best effort, the skill is not immediately transferable to another task, e.g., propensity modeling, natural language processing, text featurization, feature selection, pattern mining, and so on. Such in-depth knowledge might be desirable for recommendation experts, and at the same time, is a blocker for the broader community interested in personalization applications.

Second, these frameworks require serious engineering capabilities to meld the training and testing pipeline together in an effort for real-time performance at scale, plus hardware requirements for extremely data-hungry neural networks that expect millions of user and item interactions. This engineering requirement is a barrier for industry players that are not necessarily software-driven, unlike the high-tech companies that built these systems for their specific needs. Personalization is not a privileged application that should remain only accessible to technology companies, and most other businesses have opportunities to personalize touchpoints for their end-users. In fact, from the end-user's perspective, today, we would like any interaction to be personalized seamlessly for our needs regardless of the provider.

On the one hand, we have practitioners with a general background in machine learning but cannot commit to becoming recommenders system experts and prefer more transferable skills. On the other hand, we have companies that must meet their business demand for personalization but cannot commit to building and maintaining the large-scale infrastructure only to satisfy requirements of a single task.

To address these challenges, we present a *modular framework*, MAB2REC[1], to build recommender systems from higher-order abstractions. We treat recommender systems with plug-and-play components that remain useful in isolation and present their unification under MAB2REC. MAB2REC leverages several industry-strength libraries that we also contribute to the open-source community.

[1] http://github.com/fidelity/mab2rec

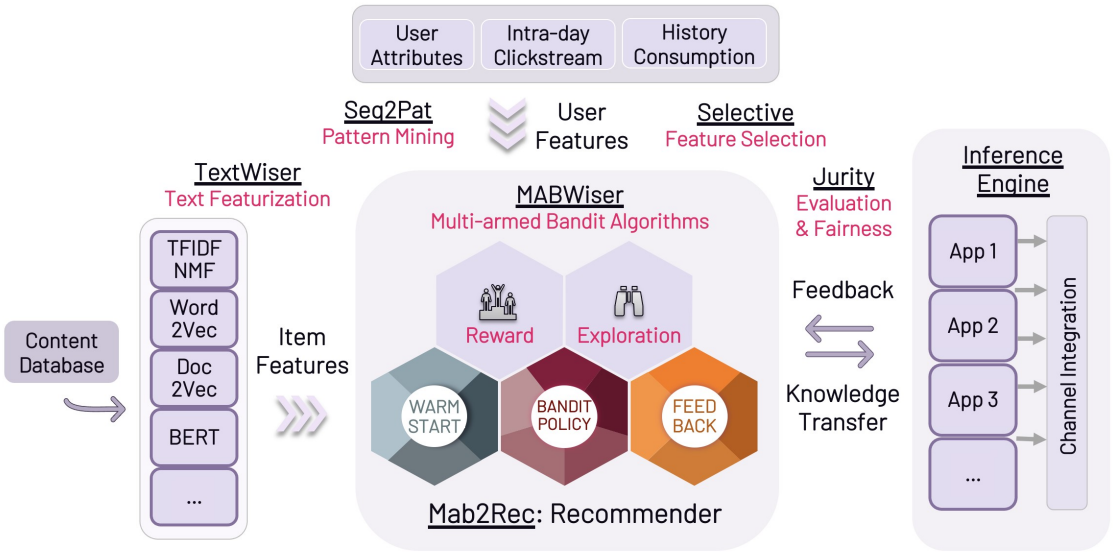

Figure 1: Building higher-order abstractions from the components of Recommender Systems.

Higher-Order Abstractions

Recommender systems are based on three main inputs; information about users, items, and historical user-item interactions. We realize this abstraction in our modular design and dedicate standalone components to each building block of the system.

As shown in Figure 1, our components revolve around item features using natural language processing via TEXTWISER, user features using sequential pattern mining via SEQ2PAT and SELECTIVE, learning from their interaction using multi-armed bandit algorithms via MABWISER, and finally, performance and fairness evaluation via JURITY.

When these higher-order components are integrated together through MAB2REC, we realize a powerful tool chain that serves to build and deploy industrial applications of recommender systems. At the same time, each component is standalone and ready to reuse in other applications.

At a high-level, our components include:

1. Feature selection and generation: SELECTIVE (Kadıoğlu, Kleynhans, and Wang 2021; Kleynhans, Wang, and Kadıoğlu 2021), SEQ2PAT (Wang et al. 2022; Kadıoğlu et al. 2023; Wang and Kadıoğlu 2022; Ghosh et al. 2022), and TEXTWISER (Kilitcioglu and Kadıoğlu 2021)
2. Recommendation models: MABWISER (Strong, Kleynhans, and Kadıoğlu 2019, 2021; Kilitçioglu and Kadıoğlu 2022)
3. Performance and fairness evaluation:JURITY (Michalský and Kadıoğlu 2021; Cheng, Kilitçioglu, and Kadıoğlu 2022)
4. Integration framework: MAB2REC (this paper)

We note that each component offers novel algorithmic improvements as published in a series of articles and practical solutions as released in open-source libraries that are readily available to the community.

Contextual multi-armed bandit (MAB) algorithms to train recommendation models is at the heart of our system. Our method of choice is motivated by the online learning nature of bandit algorithms to address the notorious cold-start problem and their principled approach to balance the exploitation-and-exploration trade-off (Strong, Kleynhans, and Kadıoğlu 2021).

Usage Example

Let us showcase a simple usage example to make the idea behind MAB2REC more concrete. Three input elements of recommenders; users, items, and interactions, form the basis of our public interface trained via a bandit learning policy.

```
# LinGreedy bandit recommender to select
# top-4 items with 10% random exploration
rec = BanditRecommender(top_k=4,
      LearningPolicy.LinGreedy(epsilon=0.1))

# Train on historic interactions
# together with user and item features
train(rec, data='train_data.csv',
      user_features='user_features.csv',
      item_features='item_features.csv')

# Score trained model for new users in test data
df = score(rec, data='test_data.csv',
           user_features='user_features.csv'
           item_features='item_features.csv')
```

Given historical user-item interactions, which are tuples of $\langle user_{id}, item_{id}, reward \rangle$, e.g., click event as binary reward, together with user and item features, which are numerical vectors for each user and item, MAB2REC provides a high-level interface to create and train a multi-armed bandit model based on the specified learning policy. This trained recommender is then evaluated on test data to generate top-k recommendations for new users.

In the following sections, we present the details of each component; features, models, evaluation, overview the benefits of our approach, and share best practices on how to scale across the enterprise.

Feature Selection & Generation

Contextual information is a key component of modern recommender systems. However, not all available contextual information is relevant or immediately usable within a recommender model. Depending on the data type, we need different approaches to extract useful features or representations. As part of MAB2REC, we highlight three distinct contributions for creating features from Structured Tabular Data, Unstructured Text Data, and Semi-Structured Sequential Data.

Structured Tabular Data

In the industry, it is common to have large amounts of raw tabular data. This is especially true for user contexts, as most companies collect a wide range of user attributes. However, determining the most relevant features for a given application is non-trivial. For this purpose, we open-source SELECTIVE[2]. This white-box feature selection library supports unsupervised and supervised selection methods with automatic task detection for classification and regression.

SELECTIVE (Kadıoğlu, Kleynhans, and Wang 2021; Kleynhans, Wang, and Kadıoğlu 2021) provides a variety of filtering and embedded selection methods with varying degrees of complexity, from simple variance, statistics- and correlation-based methods to embedded penalized linear regression models and non-linear tree-based methods. The library allows benchmarking with multiple selection methods using cross-validation for robustness and offers built-in parallelization for scalability.

In the recommenders setting and in other learning tasks, SELECTIVE helps determine the subset of tabular user features that are most relevant for predicting a user's item preference. Note that this is not a one-off exercise independent of the modeling approach. Instead, it needs to be regularly re-evaluated as more interaction data and new features become available. By offering a standardized and efficient toolkit for feature selection, our goal is to enable a wide range of practitioners to identify robust candidate features consistently with minimal effort.

Unstructured Text Data

To complement structured tabular data, we next turn to unstructured text data as a common source of input for recommenders. Various text embeddings have become prevalent for consuming raw text. However, there is no silver bullet as to which featurization technique or combination would provide the best performance for downstream applications. The choice of embedding techniques ranges widely, from simple counting-based TF-IDF to word embeddings like Word2Vec (Mikolov et al. 2013), Doc2Vec (Le and Mikolov 2014), and more sophisticated transformer language models like BERT (Devlin et al. 2018) and GPT-3 (Brown et al. 2020).

Benchmarking becomes imperative when faced with several options to find the best-performing technique. Beyond performance, other factors include inference time, simplicity, maintainability, hardware requirements, deployment constraints, and reproducibility. The task becomes harder when we consider combinations of featurization methods or their transformation using dimensionality reduction or decomposition techniques such as non-negative matrix factorization and singular value decomposition.

Thankfully there exists a rich set of tools that we can utilize, including SPACY, FLAIR, and HUGGINGFACE TRANSFORMERS. Still, the availability of various techniques in standalone, isolated libraries does not provide the needed uniformity. This is especially problematic in the industry to experiment with many different tools, build a custom layer on top to find the best approach and re-do the work when the scenario changes in a new use case. Even worse, this is an effort that *every* data scientist in the organization has to repeat *for each* application.

To address this challenge, we introduced a novel context-free grammar of embeddings (Kilitcioglu and Kadıoğlu 2021). This grammar allows us to systematically represent the language of all valid featurization techniques. The main idea is to treat each instantiation of embedding and transformation combinations as a sentence subject to language membership against the grammar of embeddings. At its core, the grammar utilizes two production rules, *concatenate* and *transform*, that define arbitrarily complex yet valid text featurization pipelines. We refer to (Kilitcioglu and Kadıoğlu 2021) for the complete set of rules.

This grammar is implemented in TEXTWISER[3] that lends itself to a high-level user interface enabling rapid experimentation with various featurization methods as shown in the example below. TEXTWISER hosts more than 25 different text embeddings with over 100 pre-trained models and serves as a building block within recommender applications consuming unstructured text as part of item features.

```
# Unstructured Text Data
item_text = ["Some document", "More text"]

# Example Embedding - I: Counting based
emb = TextWiser(Embedding.TfIdf())

# Example Embedding - II: Document vector
emb = TextWiser(Embedding.Doc2Vec())

# Text Featurization
item_features = emb.fit_transform(item_text)
```

Semi-Structured Sequential Data

The other data type we consider is semi-structured sequential data, particularly sequential clickstream, to better understand users' digital behavior. Clickstream has unique properties, such as its real-time and streaming nature. On the one hand, it provides *unstructured text* such as web pages. On the other hand, it yields sequential information where visits can be viewed as *structured events* representing user journeys. Each sequence is an ordered set

[2]http://github.com/fidelity/selective

[3]http://github.com/fidelity/textwiser

of *items* associated with a set of *attributes* to capture item properties, e.g., price, timestamp. Recent efforts focused on designing models and architectures to deal with sequential data. These include CNN-based Caser (Tang and Wang 2018), RNN-based GRU4Rec (Hidasi et al. 2015), and Transformer-based BERT4Rec (Sun et al. 2019), and Transformer4Rec (de Souza Pereira Moreira et al. 2021). These approaches utilize deep neural networks and advanced architectures to exploit semi-structured input and capture session behavior.

We consider an alternative approach based on sequential pattern mining (SPM) (Gan et al. 2019). A *pattern* is a subsequence that occurs in at least one sequence maintaining the original item ordering. The idea of SPM is to search for patterns with high *frequency* of occurrence. However, finding the entirety of frequent patterns is highly costly as the set is typically too large and may not provide significant insights. It is thus important to search for patterns that are not only frequent but also capture specific properties. This motivated Constraint-based SPM (CSPM) (Pei, Han, and Wang 2007) to incorporate constraint reasoning into data mining to find smaller subsets of interesting patterns.

In (Wang et al. 2022; Kadıoğlu et al. 2023), we applied SPM using declarative constraint models based on multi-valued decision diagrams (Hosseininasab, van Hoeve, and Ciré 2019). The idea is embodied in our open-source SEQ2PAT[4] library to support pattern mining applications. This is joint industry/academia collaboration with CMU[5]. Here is a quick usage example to highlight the declarative nature of our sequence models.

```
# Seq2Pat over 3 sequences of items
s2p = Seq2Pat(sequences=[["C", "B", "A"],
                         ["A", "B"],
                         ["C", "A", "C", "D"]])

# Price attribute corresponding to each item
s2p = Attribute(values=[[1, 3, 3],
                        [5, 3],
                        [4, 5, 2, 1]])

# Average price constraint in declarative form
s2p.add_constraint(3 <= price.average() <= 4)

# Patterns that occur at least twice (A-D)
patterns = s2p.get_patterns(min_frequency=2)
```

SEQ2PAT allows knowledge discovery in large sequence databases subject to desired properties, e.g., find frequent patterns from sessions where users spend at least a minimum amount of time on particular items with a specific price range. Such patterns serve as important signals.

Beyond recommenders, SEQ2PAT serves as an integrator technology for automated feature generation from sequential data. To that end, we designed an algorithm for embedding CSPM in Dichotomic Pattern Mining (DPM) (Wang and Kadıoğlu 2022) that exploits the dichotomy of positive versus negative outcomes in user cohorts.

With DPM, we identify frequent patterns that distinguish between positive and negative outcomes uniquely, e.g., buy vs. no buy. Our experimental results on real-world e-commerce shopping intent prediction and intrusion detection show that models built on top of SEQ2PAT patterns are competitive with the state-of-the-art LSTM models (Requena et al. 2020), and best results are achieved when frequent and deep patterns are combined (Ghosh et al. 2022).

Contextual Recommendation Models

In the previous section, we studied how to create user and item features from structured tabular data, unstructured text data, and semi-structured sequential data. Given the user and item features, we can now turn our attention to training recommendation models.

Traditional recommender systems, such as content-based and collaborative filtering methods, use simple user models. For example, user-based collaborative filtering generally models the user as a vector of item ratings. This approach ignores the fact that users interact with the system within a particular context and that item preferences within a context may differ in another context. Furthermore, it is well-known that focusing purely on exploitation fails to capture the dynamic nature of user preferences and available items and needs a principled approach to explore different policies for continuous learning.

To balance the *exploration-exploitation trade-off*, we target content-and context-aware recommenders. Multi-Armed Bandits (MAB) is a well-known algorithm family that focuses on sequential decision-making. MAB algorithms define each "arm" as a decision that an agent can make, generating either a deterministic or stochastic "reward". At each time step, the agent faces a decision on whether to utilize an arm that has a high expected reward ("exploit") or to try out new arms to learn something new ("explore"). The agent's goal is to maximize the cumulative reward in the long run, for which it must balance exploration with exploitation.

In recommender systems, the arms correspond to the different items in the system that the agent can recommend, and the reward is based on user interactions with the suggested item(s), e.g., click or no-click. We note that this reward can be further tuned to accommodate other information, such as multi-objective models that combine engagement (e.g., click) with action (e.g., purchase).

Contextual Multi-Armed Bandits (CMABs) utilize a state ("context") that captures side information that might affect the reward for a given arm. Formally, the reward for an arm becomes a function of the selected arm and the state.

Recommender systems commonly utilize CMABs to decide on the item to recommend at a given time step for a given context. While context application specific, it often contains user representations, which can be generated using our libraries described in the previous section.

Our contribution in this line of research is the introduction of MABWISER[6]. This open-source Python-native library offers context-free, non-parametric, and parametric MAB algorithms.

[4]http://github.com/fidelity/seq2pat

[5]https://www.cmu.edu/tepper/news/stories/2023/may/fidelity-ai.html

[6]http://github.com/fidelity/mabwiser

As shown below, MABWISER utilizes a familiar scikit-learn style interface, allows high-level access to various learning policies, and is highly parallelized for efficiency when training on large datasets.

```
# Data
arms = ['Item1', 'Item2']
decisions = ['Item1', 'Item1', 'Item2']
rewards = [0, 1, 0]
contexts = [[3, 1], [1, 1], [2, 1]]

# Model
mab = MAB(arms, LearningPolicy.UCB1(alpha=1.25),
          NeighborhoodPolicy.KNearest(k=2))

# Train
mab.fit(decisions, rewards, contexts)

# Test
mab.predict(test_contexts)
```

Let us also acknowledge powerful bandit libraries such as VOWPAL WABBIT (Langford, Li, and Strehl 2007), YELP MOE (Liu, Vesdapunt, and Wang 2014) and CONTEXTUAL R (van Emden and Kaptein 2018) for their excellent contributions. MABWISER differs from the existing work in several dimensions. First, it is a pure Python library without external dependencies. It offers easy access to several linear and non-linear contextual policies, including tree-bandits. A unique differentiator of MABWISER, which is not possible in other libraries, is the novel hybridization of non-parametric neighborhood policy with its parametric learning policy counterpart and providing a high-level interface to experiment with different policies.

In our Usage Example, notice how the learning policy parameter of MABWISER is exposed in MAB2REC. In the context of recommenders, this bandit policy is now paired with a top-k parameter to reflect the recommendation task. MAB2REC uptakes all available learning policies from MABWISER and the ones to become available in the future. This again shows how we improve individual components in isolation without dependency while their combination benefits immediately from each such improvement.

System Evaluation

Performance Metrics

Recommender systems have been evaluated in many, often incomparable, ways (Herlocker et al. 2004). While the literature on recommender system evaluation offers a large variety of evaluation metrics, little guidance is provided on how to choose among them (Schröder, Thiele, and Lehner 2011). Furthermore, we often find inconsistent implementations for the same metric in open-source software.

To this end, we release JURITY[7] our open-source Python-native contribution that supports both standard recommendation metrics as well as fairness metrics and bias mitigation techniques.

There is no "silver-bullet" to evaluate the quality of a recommender system algorithm, and typically several performance evaluation methods are required (Chen and Liu 2017). For offline evaluation, JURITY includes *accuracy*, *ranking*, and *diversity* related metrics. In particular, accuracy metrics JURITY includes the Click-Through Rate (CTR) metric with Inverse Propensity Score (IPS) and Doubly Robust (DR) options to address offline policy bias. For ranking metrics, JURITY offers to Mean Average Precision (MAP) and Normalized Discounted Cumulative Gains (NDCG). For diversity metrics, JURITY measures the similarity of recommended items across different users (Inter-list Diversity) and the similarity of items within lists of top-k recommendations (Intra-list Diversity). These metrics are related to coverage, which measures the universe of items the system covers in recommendations, and serendipity, which measures the extent the recommended items are both attractive and surprising to users (Herlocker et al. 2004).

In practice, several metrics at different top-k values have to be evaluated together. As stated in (Good et al. 1999), coverage (or diversity) decreases as a function of accuracy. Therefore, it is critical to use a standardized evaluation to ensure consistency and explain trade-offs such as accuracy vs. diversity when communicating with stakeholders.

Beyond recommenders, we utilized JURITY in the fairness evaluation of propensity models, a well-known industry application for lead generation and marketing campaign optimization. Specifically, we studied two practical scenarios that are often neglected in the academic literature; when the protected membership attribute of individuals is not available,and when the ground truth label is not available (Thielbar et al. 2023).

Fairness Metrics

In addition to performance analysis, we also have to consider fairness evaluation of recommenders. Bias in machine learning has been increasingly documented due to historical bias present in training data concerning protected attributes (Angwin et al. 2022; Buolamwini and Gebru 2018). In the context of recommender systems, items, and users can be abstracted as objects and subjects. Individuals that may be discriminated against can take either role. For example, when recommending offers, the subjects are people. Similarly, when looking for a ride, the recommended objects are taxi drivers. Therefore, fairness in recommender systems has multiple viewpoints referred to as *consumer fairness* (C-fairness) and fairness for objects referred to *producer fairness* (P-fairness). Similar to performance metrics, there is no consensus on fairness metrics.

In JURITY, our contribution is to extend the well-known Disparate Impact (Gómez, Boratto, and Salamó 2021) and Statistical Parity (Gao and Shah 2020) to multi-class and multi-label recommendation settings for fairness and diversity (Cheng, Kilitçioglu, and Kadıoğlu 2022). These metrics measure the *ratio* or *difference*, respectively, of probabilities for predicted positive outcomes among the two membership groups for each item. In addition, JURITY provides bias mitigation based on the equalized odds (Hardt, Price, and Srebro 2016) to adjust model predictions accordingly.

[7] http://github.com/fidelity/jurity

Applications & Benefits of Mab2Rec

In the previous sections, we presented the details of each component. In this section, we first highlight the benefits of our unification framework, cover applications powered by MAB2REC, and finally present examples where our components are successfully deployed beyond recommenders.

The novelty of MAB2REC stems from a formal treatment of recommender systems with a modularity lens that spans multiple subject areas studied in-depth within each component. Our modular approach yields benefits that address the two common challenges practitioners and companies face when deploying personalization applications.

From a data scientist perspective, utilizing individual, self-contained components, each released as a standalone open-source library, help practitioners extend their toolbox in several important directions beyond recommenders.

From a company perspective, modularity is especially desirable for industry players who cannot afford to build a specialized technology stack for every business problem but prefer leverage and reuse across solutions. This applies to capability view and also upskilling talent. As MAB2REC is a Python-native stack without low-level dependencies, it is ready-to-use with minimal overhead. This is realized in the following successfully deployed applications.

Article Recommendation: This personalization application (Verma et al. 2023) generates article recommendations for customers of a large financial services company where MAB2REC is used to train and deploy parametric contextual multi-armed bandit models and balance the exploration-exploitation trade-off, which is particularly important for news article setting. We also showed subtle non-deterministic behavior of the well-known LinTS bandit policy that went unnoticed in the community for 10+ years, provided a simple fix to ensure reproducibility, and proved its correctness (Kilitçioglu and Kadıoğlu 2022).

EaSe – Embeddings-as-a-Service: In this application, presented in detail in (Kilitcioglu and Kadıoğlu 2021), TEXTWISER is used to create pre-computed embeddings of large volumes of text that serves users with embedding vectors on-demand via a REST API. By abstracting away the complications of pre-processing pipelines, embedding models, and pooling methods, the embedding service save compute resources through pre-computed embeddings, promote sharing of language models across teams, and improve privacy by limiting access to raw text.

Open-Source: MAB2REC paved the road to several open-source libraries, which, in aggregation, surpassed 300K+ downloads in the community. We also received new contributions from external users. The self-contained nature of each component promoted collaboration with researchers in natural language processing, pattern mining, and optimization who might not necessarily be interested in recommender systems. Beyond MAB2REC, our components found applications in other domains, including ALNS (Wouda and Lan 2023) for adaptive large neighborhood search for solving optimization problems and GOLEM (Nikitin et al. 2021) for designing automated machine learning pipelines.

Scaling Recommenders in the Enterprise

Finally, let us share our best practices for effectively scaling recommenders using MAB2REC in industrial settings, from user training and adoption to performance evaluation and deployment.

User Training: For upskilling talent within the enterprise, we share our know-how in feature engineering, model selection, and model evaluation through demos, end-to-end tutorials[8] and Coursera guided courses. The MAB2REC library includes several notebook tutorials that shows how to go from raw interaction data to deploying a trained recommendation model.

Adoption: To scale adoption it is essential to minimize any barriers to entry (e.g., usage examples, good documentation, etc.). Equally important is the need to create a community of collaboration, communication, learning, and feedback. To cultivate a community within our organization, we organized an internal recommender systems competition where 100+ participants submitted competing algorithms evaluated through an anonymized, Kaggle-like real-time leaderboard.

Evaluation: To easily benchmark multiple algorithms using several evaluation metrics, MAB2REC offers benchmarking functionality to speed-up evaluations across the organization and JURITY standarizes it.

Deployment: MAB2REC is not coupled with any specific inference engine; hence, various deployment paths are viable from on-prem to cloud platforms, e.g., Amazon AWS, Microsoft Azure, or Google Cloud. Different business units might prefer operating on different technology stacks, which remains agnostic to MAB2REC. To facilitate development and deployment, we provide easy `pip install`[9] and a docker image available on GitHub Cloud[10] that is ready to run MAB2REC and all our libraries presented in this paper.

Conclusion

Recommender systems are of great practical interest in the industry while offering challenging algorithmic questions that propel research in the field. In this paper, we presented a modular recommender system framework, MAB2REC, to build recommender systems from higher-order abstractions.

Our design tightly integrates and yet clearly marks off independent abstractions, thus satisfying two of the most critical aspects of industrial applications, generality, and specificity. Our components from feature selection (SELECTIVE) and generation (SEQ2PAT and TEXTWISER), to recommendation models (MABWISER), and evaluation (JURITY) are highly specialized in their verticals while being generally applicable beyond recommenders as we showcased in this paper. We also showed that when MAB2REC brings these components together, it realizes a sophisticated toolchain for recommender systems to build and deploy business-relevant applications. Finally, we discussed how to disseminate knowledge and user adoption and shared our best practices within the enterprise and the broader community.

[8] http://github.com/fidelity/mab2rec/notebooks

[9] pip install mab2rec

[10] docker pull ghcr.io/skadio/atlas_docker:0.4.0

References

Amatriain, X.; and Basilico, J. 2016. Past, present, and future of recommender systems: An industry perspective. In *ACM RecSys*.

Angwin, J.; Larson, J.; Mattu, S.; and Kirchner, L. 2022. Machine bias. In *Ethics of data and analytics*, 254–264. Auerbach Publications.

Argyriou, A.; González-Fierro, M.; and Zhang, L. 2020. Microsoft Recommenders: Best Practices for Production-Ready Recommendation Systems. In *Proceedings of the Web Conference*, 50–51.

Brown, T.; Mann, B.; Ryder, N.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. In *NeurIPS*, volume 33, 1877–1901.

Buolamwini, J.; and Gebru, T. 2018. Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. In *FAccT*, Machine Learning Research.

Chen, M.; and Liu, P. 2017. Performance evaluation of recommender systems. *Int. J. of Performability Engineering*, 13(8): 1246.

Cheng, D.; Kilitçioglu, D.; and Kadıoğlu, S. 2022. Bias mitigation in recommender systems to improve diversity. In *CIKM*, CEUR.

de Souza Pereira Moreira, G.; Rabhi, S.; Ak, R.; and Schifferer, B. 2021. *End-to-End Session-Based Recommendation on GPU*, 831–833. New York, NY, USA: ACM.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*.

Gan, W.; Lin, J. C.-W.; Fournier-Viger, P.; Chao, H.-C.; and Yu, P. S. 2019. A Survey of Parallel Sequential Pattern Mining. *KDD*.

Gao, R.; and Shah, C. 2020. Counteracting bias and increasing fairness in search and recommender systems. In *ACM RecSys*.

Ghosh, S.; Yadav, S.; Wang, X.; Chakrabarty, B.; and Kadıoğlu, S. 2022. Dichotomic Pattern Mining Integrated with Constraint Reasoning for Digital Behaviour Analyses. *Frontiers in AI*.

Gómez, E.; Boratto, L.; and Salamó, M. 2021. Disparate impact in item recommendation: A case of geographic imbalance. In *ECAI*.

Good, N.; Schafer, J. B.; Konstan, J. A.; Borchers, A.; Sarwar, B.; Herlocker, J.; Riedl, J.; et al. 1999. Combining collaborative filtering with personal agents for better recommendations. *AAAI-IAAI*.

Hardt, M.; Price, E.; and Srebro, N. 2016. Equality of opportunity in supervised learning. *NeurIPS*, 29.

Herlocker, J. L.; Konstan, J. A.; Terveen, L. G.; and Riedl, J. T. 2004. Evaluating collaborative filtering RecSys. *ACM TOIS*.

Hidasi, B.; Karatzoglou, A.; Baltrunas, L.; and Tikk, D. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*.

Hosseininasab, A.; van Hoeve, W.; and Ciré, A. A. 2019. Constraint-Based SPM with Decision Diagrams. In *AAAI*.

Kadıoğlu, S.; Kleynhans, B.; and Wang, X. 2021. Optimized Item Selection to Boost Exploration for RecSys. In *CPAIOR*.

Kadıoğlu, S.; Wang, X.; Hosseininasab, A.; and van Hoeve, W.-J. 2023. Seq2Pat: Sequence-to-pattern generation to bridge mining with machine learning. *AI Magazine*.

Kilitcioglu, D.; and Kadıoğlu, S. 2021. Representing the Unification of Text Featurization using a Context-Free Grammar. *AAAI*.

Kilitçioglu, D.; and Kadıoğlu, S. 2022. Non-Deterministic Behavior of TS with Linear Payoffs and How to Avoid It. *TMLR*, 2022.

Kleynhans, B.; Wang, X.; and Kadıoğlu, S. 2021. Active Learning Meets Optimized Item Selection. In *DSO Workshop, IJCAI*.

Langford, J.; Li, L.; and Strehl, A. 2007. Vowpal Wabbit. https://github.com/VowpalWabbit/. Accessed: 2023-11-28.

Le, Q.; and Mikolov, T. 2014. Distributed representations of sentences and documents. In *ICML*, 1188–1196.

Liu, E.; Vesdapunt, N.; and Wang, J. 2014. Metric Optimization Engine. http://github.com/Yelp/MOE. Accessed: 2023-11-28.

Michalský, F.; and Kadıoğlu, S. 2021. Surrogate Ground Truth Generation to Enhance Binary Fairness Evaluation in Uplift Modeling. In *IEEE ICMLA*, 1654–1659.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*, 3111–3119.

Naumov, M.; Mudigere, D.; Shi, H.-J. M.; Huang, J.; Sundaraman, N.; Park, J.; Wang, X.; Gupta, U.; Wu, C.-J.; Azzolini, A. G.; et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*.

Nikitin, N. O.; Vychuzhanin, P.; Sarafanov, M.; and Polonskaia, I. S. 2021. Automated evolutionary approach for the design of composite machine learning pipelines. *Future Generation CS*.

Oldridge, E.; Perez, J.; Frederickson, B.; Koumchatzky, N.; Lee, M.; Wang, Z.; Wu, L.; Yu, F.; Zamora, R.; Yılmaz, O.; et al. 2020. Merlin: a gpu accelerated recommendation framework. *IRS*.

Pasumarthi, R. K.; Bruch, S.; Wang, X.; Li, C.; Bendersky, M.; Najork, M.; Pfeifer, J.; Golbandi, N.; Anil, R.; and Wolf, S. 2019. Tf-ranking: Scalable tensorflow library for learning-to-rank. In *KDD*.

Pei, J.; Han, J.; and Wang, W. 2007. Constraint-Based Sequential Pattern Mining: The Pattern-Growth Methods. *JIIS*, 28(2).

Requena, B.; Cassani, G.; Tagliabue, J.; Greco, C.; and Lacasa, L. 2020. Shopper intent prediction from clickstream e-commerce data with minimal browsing information. *Scientific Reports*, 10.

Schröder, G.; Thiele, M.; and Lehner, W. 2011. Setting goals and choosing metrics for recommender system evaluations. In *ACM RecSys UCERSTI2 workshop*, volume 23.

Strong, E.; Kleynhans, B.; and Kadıoğlu, S. 2019. MABWiser: A Parallelizable Contextual MAB Library for Python. In *IEEE ICTAI*.

Strong, E.; Kleynhans, B.; and Kadıoğlu, S. 2021. MABWiser: Parallelizable Contextual Multi-armed Bandits. *IJAIT*, 30(4).

Sun, F.; Liu, J.; Wu, J.; Pei, C.; Lin, X.; Ou, W.; and Jiang, P. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. In *ACM CIKM*.

Tang, J.; and Wang, K. 2018. Personalized Top-N Sequential Recommendation via Convolutional Sequence Embedding. *arXiv:1809.07426*.

Thielbar, M.; Kadıoğlu, S.; Zhang, C.; Pack, R.; and Dannull, L. 2023. Surrogate Membership for Inferred Metrics in Fairness Evaluation. In *Learning and Intelligent Optimization*, 424–442. ISBN 978-3-031-44505-7.

van Emden, R.; and Kaptein, M. 2018. contextual: Evaluating contextual multi-armed bandit problems in R. *arXiv:1811.01926*.

Verma, G.; Sengupta, S.; Simanta, S.; Chen, H.; Perge, J. A.; Pillai, D.; McCrae, J. P.; and Buitelaar, P. 2023. Empowering RecSys using automatically generated KG and RL. *arXiv:2307.04996*.

Wang, X.; Hosseininasab, A.; Colunga, P.; Kadıoğlu, S.; and van Hoeve, W.-J. 2022. Seq2Pat: Sequence-to-Pattern Generation for Constraint-based Sequential Pattern Mining. In *AAAI-IAAI*.

Wang, X.; and Kadıoğlu, S. 2022. Dichotomic Pattern Mining with Applications to Intent Prediction from Semi-Structured Clickstream Datasets. In *KDF-AAAI-22*.

Wouda, N. A.; and Lan, L. 2023. ALNS: a Python implementation of the adaptive large neighbourhood search metaheuristic. *JOSS*.