`Seq2Pat`: Sequence-to-Pattern Generation for Constraint-Based Sequential Pattern Mining

Xin Wang[1], Amin Hosseininasab[2], Pablo Colunga[1], Serdar Kadıoğlu[1], Willem-Jan van Hoeve[3]
[1]AI Center of Excellence, Fidelity Investments, Boston, USA
[2]Warrington College of Business, University of Florida, Gainesville, USA
[3]Tepper School of Business, Carnegie Mellon University, Pittsburgh, USA
xin.wang@fmr.com, amin.hosseininasab@warrington.ufl.edu, pablo.colunga@fmr.com,
serdar.kadioglu@fmr.com, vanhoeve@andrew.cmu.edu

Abstract

Pattern mining is an essential part of knowledge discovery and data analytics. It is a powerful paradigm, especially when combined with constraint reasoning. In this paper, we present `Seq2Pat`, a constraint-based sequential pattern mining tool with a high-level declarative user interface. The library finds patterns that occur frequently in large sequence databases subject to constraints. We highlight key benefits that are desirable, especially in industrial settings where scalability, explainability, rapid experimentation, reusability, and reproducibility are of great interest. We then showcase an automated feature extraction process powered by `Seq2Pat` to discover high-level insights and boost downstream machine learning models for customer intent prediction.

Introduction

Sequential Pattern Mining (SPM) is highly relevant in various practical applications including the analysis of medical treatment history (Bou Rjeily et al. 2019), customer purchases (Requena et al. 2020), call patterns and digital clickstream (Agrawal and Srikant 1995; Srikant and Agrawal 1996). A recent survey has more details (Gan et al. 2019).

In SPM, we are given a set of sequences that is referred to as *sequence database*. As shown in the example in Table 1, each sequence is an ordered set of *items*. Each item might be associated with a set of *attributes* to capture item properties, e.g., price, timestamp. A *pattern* is a subsequence that occurs in at least one sequence in the database maintaining the original ordering of items. The number of sequences that contain a pattern defines the *frequency*. Given a sequence database, SPM is aimed at finding patterns that occur more than a certain frequency threshold.

In practice, finding the entire set of frequent patterns in a sequence database is not the ultimate goal. The number of patterns 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 of the application at hand. This has motivated research in Constraint-based SPM (CSPM) (Pei, Han, and Wang 2007; Chen et al. 2008). The goal of CSPM is to incorporate constraint reasoning into sequential pattern mining to find smaller subsets of interesting patterns.

SEQUENCE DATABASE ⟨(item, price, timestamp)⟩
⟨ (A, 5, 1), (A, 5, 1), (B, 3, 2), (A, 8, 3), (D, 2, 3)⟩
⟨(C, 1, 3), (B, 3, 8), (A, 3, 9)⟩
⟨(C, 4, 2), (A, 5, 5), (C, 2, 5), (D, 1, 7)⟩

Table 1: Example sequence database with three sequences.

As an example, let us consider online retail clickstream analysis. We might not be interested in all frequent browsing patterns. For instance, the pattern $\langle login, logout\rangle$ is likely to be frequent but offers little value. Instead, we seek recurring clickstream patterns with unique properties, e.g., frequent patterns from sessions where users spend at least a minimum amount of time on a particular set of items with a specific price range. Such constraints help reduce the search space for the mining task and help discover patterns that are more effective in knowledge discovery than arbitrarily frequent clickstreams.

The main goal of this paper is to introduce an innovative tool to support CSPM applications and their deployment in real-world scenarios. Despite the applicability of pattern mining and its potential to generate insights when combined with constraint reasoning, library support remains limited for off-the-shelf tools. In particular, the Python ecosystem, one of the most commonly used technology stacks in machine learning, lacks such support. We designed the `Seq2Pat`[1] library to fill this gap and provide a stable and efficient pattern mining tool with easy access for Python users. Furthermore, beyond frequent pattern mining, our library enables users to introduce complex constraints in a declarative manner to search for meaningful patterns. This is a unique contribution of `Seq2Pat` that is not fully supported in existing mining libraries.

`Seq2Pat` is a tool built in collaboration between academia and industry to serve researchers and practitioners for knowledge discovery in large sequence databases. The tool finds sequential patterns that occur frequently. Furthermore, it supports constraint-based reasoning to specify desired properties by leveraging the state-of-the-art multi-valued decision diagram representation of the sequence database (Hosseininasab, van Hoeve, and Ciré 2019).

[1]https://github.com/fidelity/seq2pat

```
# Sequence database over items {A, B, C, D}
database = [["A", "A", "B", "A", "D"],
            ["C", "B", "A"],
            ["C", "A", "C", "D"]]

# Seq2Pat over three sequences
seq2pat = Seq2Pat(database)

# Minimum frequency threshold
min_frequency = 2

# Find patterns that occur at least twice
patterns=seq2pat.get_patterns(min_frequency)

# Patterns
# >>> ["A", "D"], ["B", "A"], ["C", "A"]
```

Figure 1: `Seq2Pat` usage example on a toy database with three sequences over the item set $\{A, B, C, D\}$. We discover three patterns, $\{[A, D], [B, A], [C, A]\}$, that occur in at least two sequences.

In the following, we start with an illustrative usage example and then provide details of the mining algorithm based on multi-valued decision diagrams. We then highlight key features, especially desirable in industrial settings. Finally, we demonstrate an AI application where the library serves as an integration technology between raw data and machine learning models for downstream customer intent prediction.

Usage Example

Let us start with a simple usage example based on the sequence database in Table 1 to make the idea behind `Seq2Pat` more concrete. The first example demonstrates sequential pattern mining and the second example demonstrates constraint-based sequential pattern mining.

The example in Figure 1 shows how to find frequent sequential patterns in a given sequence database. The database is represented as a list of sequences each with a list of items. There are three sequences over the item set $\{A, B, C, D\}$. The $min\_frequency$ parameter configures the search for patterns that occur in at least two sequences. This example does not enforce any constraints. When this is the case, `Seq2Pat` performs sequential pattern mining. As a result, we discover three patterns $\{[A, D], [B, A], [C, A]\}$ subject to minimum frequency threshold. Notice that each pattern occurs in exactly two sequences satisfying the minimum frequency. More specifically;

- The pattern $[A, D]$ is a subsequence of the first and the third sequence.
- The pattern $[B, A]$ is a subsequence of the first and the second sequence.
- The pattern $[C, A]$ is a subsequence of the second and the third sequence.

Our main design goal is to facilitate the interaction between the sophisticated pattern mining algorithm and the raw sequence data. In Section , we provide a high-level overview of the underlying algorithm.

```
# Prices for each sequence
prices = [[5, 5, 3, 8, 2],
          [1, 3, 3],
          [4, 5, 2, 1]]

# Timestamps for each sequence
timestamps = [[1, 1, 2, 3, 3],
              [3, 8, 9],
              [2, 5, 5, 7]]

# Attributes
price = Attribute(prices)
time = Attribute(timestamps)

# Declarative constraints
seq2pat.add_constraint(3<=price.average()<=4)
seq2pat.add_constraint(3<=price.median()<=4)
seq2pat.add_constraint(0<=time.gap()<=2)
seq2pat.add_constraint(0<=time.span()<=2)

# Minimum frequency threshold
min_frequency = 2

# Find patterns that occur at least twice
# subject to price and time constraints
patterns=seq2pat.get_patterns(min_frequency)

# Patterns
# >> ["A", "D"]
```

Figure 2: `Seq2Pat` usage example extended with additional attributes and constraints. Only a single pattern, $[A, D]$, satisfies all constraints and the minimum frequency.

While this example uses strings (and more precisely, characters) to represent the items, integer-based representation is also supported. Similarly, while this example uses an integer value for the frequency threshold, it is possible to use a floating number $\in (0, 1]$ to specify an occurrence percentage in the size of the sequence database.

Declarative Constraints

Next, we extend the initial example with more data to introduce various constraints to enforce desired properties on the resulting patterns.

As shown in Figure 2, we incorporate two attributes; *price* and *timestamp*. Conceptually, the idea is to capture frequent patterns in the database from users who have spent at least a minimum amount of time on certain items within specific price ranges. We assume information is available to associate each event in each sequence with the corresponding price and timestamp value. We then encapsulate this information in $Attribute$ objects so that the user can interact with the raw data. The attribute object allows reasoning about the properties of the pattern. For instance, the first condition restricts the average price of items in a pattern to be between the range $[3, 4]$. This condition is added to the system as a *constraint*. As such, `Seq2Pat` is now tasked with performing constraint-based sequential pattern mining.

Notice how operator overloading for arithmetics and compound expressions enables a user-friendly interface and improves the modeling experience. It is possible to create nested conditions that restrict upper and lower bounds simultaneously. Our design allows declarative modeling where the user can add and drop various constraints on attributes. In Section , we provide formal details about these constraints.

The first example in Figure 1 found three patterns $\{[A, D], [B, A], [C, A]\}$. When we introduce the constraints in the second example as in Figure 2, $[A, D]$ is the only remaining pattern that meets all of the four constraints. The other patterns do not satisfy the conditions.

Let us examine the details of constraint satisfaction for one of the constraints; the average price. The important observation is that the first sequence in the database exhibits three different subsequences of $[A, D]$. Notice the subsequences exhibit different price averages. In the first sequence, the first and the second occurrence of $[A, D]$ has $price\_average([5, 2]) = 3.5$ while the third occurrence has $price\_average([8, 2]) = 5$. The first two subsequences are feasible with respect to the price constraint, while the third subsequence is infeasible. One satisfying subsequence suffices for constraint feasibility. Hence the first sequence supports the pattern $[A, D]$. Similarly, the last sequence in the database satisfies the constraint with $price\_average([5, 1]) = 3$ for the $[A, D]$ pattern. Two sequences supporting the pattern for the price average meets the minimum required frequency condition. The reasoning for other constraints follows similarly. No other patterns satisfy the constraints and frequency condition. As a result, the library returns the only solution with $[A, D]$.

CSPM using Decision Diagrams

We formalize our problem of solving SPM and discuss briefly how decision diagrams help constraint reasoning.

In SPM, a *sequence database* $\mathcal{SD}$ is defined as a collection of N item sequences $\{S_1, S_2, \ldots, S_N\}$. Each *sequence* is modeled as an ordered list of items, where items $i \in I$. The items are associated with a set of *attributes* $\mathbb{A} = \{\mathcal{A}, \ldots, \mathcal{A}_{|\mathbb{A}|}\}$. As in our example, the attributes can be price and timestamp. A pattern $P = \langle i_1, i_2, \ldots, i_{|P|}\rangle$ is a subsequence of some $S \in \mathcal{SD}$. A subsequence relation $P \preceq S$ holds if and only if there exists an embedding $e : e_1 \le e_2 \le \ldots \le e_{|P|}$ such that $\mathcal{S}[e_j] = i_j$, where $\mathcal{S}[e_j]$ denotes the ${e_j}^{th}$ position of $\mathcal{S}$, $i_j \in P$ and $j \in \{1, 2, \ldots, |P|\}$. In our usage example, $P = \langle A, D\rangle$ is a subsequence of $S = \langle A, A, B, A, D\rangle$ with three possible embeddings (1, 5), (2, 5) and (4, 5). A pattern is identified to be frequent if it is a subsequence of at least θ number of sequences in $\mathcal{SD}$, where θ is a given frequency threshold. Then SPM aims to find a set of all frequent *patterns* in $\mathcal{SD}$.

The technology behind our approach is based on Multi-valued Decision Diagrams (MDDs) (Bergman et al. 2016). MDDs are widely used as efficient data structures (Wegener 2000) and for discrete optimization (Bergman et al. 2016). More recently, MDDs were utilized for CSPM (Hosseininasab, van Hoeve, and Ciré 2019) to encode the sequences and associated attributes of sequence databases.

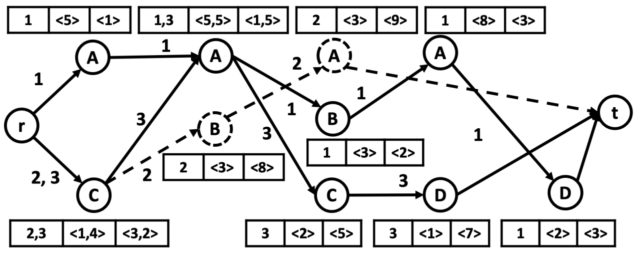

Figure 3: The MDD encoding of our usage example from Table 1 and Figure 2. Each node is associated with the sequence ID and attributes, $[ID, \langle\text{price}\rangle, \langle\text{timestamp}\rangle]$.

An MDD $M = (U, A)$ is a layered directed acyclic graph where U is the set of nodes and A is the set of arcs. Set U is partitioned into layers $(l_0, l_1, \ldots, l_{m+1})$ such that layers $l_i : 1 \le i \le m$ correspond to position i of a sequence $S \in \mathcal{SD}$. The first and the last layer consists of the root r and terminal node t modeling the start and end of all sequences. An arc $a = (u, v) \in A$ is directed from a node $u \in l_j$ to a node $v \in l_{j'} : j' > j$ and represents the next possible item after node u for all sequences in $\mathcal{SD}$. MDD nodes store constraint-specific information to support propagation.

Figure 3 shows the MDD encoding of our usage example. Each sequence is represented by a path from r to t. The MDD structure is used to enforce constraints by removing infeasible patterns and store constraint-specific information for tackling non-monotone constraint efficiently. For instance, imposing the constraints removes the nodes and arcs shown in dashed lines. We refer to (Hosseininasab, van Hoeve, and Ciré 2019) for more details.

The MDD approach accommodates multiple attributes and constraint types. It is shown to be competitive with or superior to existing CSPM algorithms in terms of scalability and efficiency (Hosseininasab, van Hoeve, and Ciré 2019). The `Seq2Pat` library makes this efficient algorithm accessible to a broad audience with a user-friendly interface.

Available Constraints

`Seq2Pat` supports several constraint types. Let c denote a threshold and $C_{type}(\cdot)$ is a function imposed on attributes with a certain type of operation. Each constraint is presented for both a minimum and maximum threshold c.

- **Average**: This constraint specifies the average value of an attribute across all events in a pattern.

$$\begin{array}{c} C_{avg}(\mathcal{A}) \le c \\ C_{avg}(\mathcal{A}) \ge c \end{array} \tag{1}$$

- **Gap**: This constraint specifies the difference between attribute values of every two consecutive events in a pattern.

$$\begin{array}{l} C_{gap}(\mathcal{A}) \le c := \alpha_j - \alpha_{j-1} \le c \\ \qquad\qquad\qquad\qquad \alpha_j \in \mathcal{A}, 2 \le j \le |P| \\ C_{gap}(\mathcal{A}) \ge c \end{array} \tag{2}$$

- **Median**: This constraint specifies the median value of an attribute across all events in a pattern.

$$\begin{array}{c} C_{med}(\mathcal{A}) \le c \\ C_{med}(\mathcal{A}) \ge c \end{array} \tag{3}$$

- **Span**: This constraint specifies the difference between the maximum and the minimum value of an attribute across all events in a pattern.

$$\begin{array}{l} C_{spn}(\mathcal{A}) \leq c := \max\{\mathcal{A}\} - \min\{\mathcal{A}\} \leq c \\ C_{spn}(\mathcal{A}) \geq c \end{array} \quad (4)$$

The difficulty of imposing a constraint in SPM depends on its monotone, anti-monotone, or non-monotone property. A constraint is *anti-monotone* if its violation by a sequence implies violation by all super-sequences of that sequence. A constraint is *monotone* if its violation by a sequence implies that all subsequences violates the constraint. Constraints that are neither monotone nor anti-monotone are called *non-monotone*. In SPM, imposing anti-monotone constraints is relatively easy, whereas imposing monotone constraints and non-monotone constraints are harder for mining algorithms.

The maximum gap constraint is prefix anti-monotone, the minimum gap and maximum span constraints are anti-monotone, and the minimum span constraint is monotone. The average and median constraints $C_{avg}(\mathcal{A})$ and $C_{med}(\mathcal{A})$ are non-monotone, and challenging to implement, despite their need in real-world applications. Previous works cannot handle monotone and non-monotone constraints and have to address them at the post-processing step. Contrarily, `Seq2Pat` enforces non-monotone constraints *during* the mining process, thanks to its underlying MDD structure, thereby improving the efficiency of the mining process.

Library Highlights

The main design goal behind `Seq2Pat` is twofold: to support the state-of-the-art MDD approach for CSPM in a user-friendly fashion in Python, and to accommodate several constraint types in a declarative fashion, including complex constraints such as average and median that are not supported in existing tools.

There exist other key considerations to ease application development, deployment, and maintenance costs. These include extensive documentation, a stable and well-tested library, graceful error handling to guide users, and scalability. We highlight some of these library features next.

Expressiveness As demonstrated in our usage examples, the `Seq2Pat` API is designed to provide a user-friendly, class-based interface. The user interacts with the system by adding and removing constraints. The Python ecosystem further supports pattern mining with other powerful libraries such as `pandas` for data manipulation, `numpy` and `scipy` for scientific computation, and `sklearn` for machine learning algorithms, all of which can interoperate with `Seq2Pat`. This facilitates benchmarking with `Seq2Pat` as part of a larger machine learning pipeline.

Efficiency When working with large sequence databases, scalability is crucial. With this requirement in mind, the library is written in `Cython` to bring together the efficiency of a low-level `C++` backend and the expressiveness of a high-level Python public interface. `Cython` (Behnel et al. 2011) is a superset of Python that enables type declarations. It translates `C++` code into optimized code compiled as Python extension modules. `Cython` is commonly used to optimize heavy computations in Python to improve performance and programming productivity, e.g. in `sklearn` (Pedregosa et al. 2011) and `surprise` (Hug 2020). `Seq2Pat` is implemented with special attention to ensure reproducibility, a highly desirable feature in industrial applications, while minimizing any runtime and memory overhead when interfacing `Python` and `C++` modules.

Installation & Dependencies The library is indexed in `PyPI`. This allows a simple installation with a single `pip install seq2pat` command hiding the complex backend from the user. The setup automatically downloads and installs necessary packages, such as `Cython`. The only requirement is to have a `C++` compiler, e.g., gcc or clang. For the power user, we provide installation instructions to build the library from scratch using the source code on different operating systems.

Coding Standards & Testing The library adheres to the PEP-8 style guide for Python coding standards and is compliant with numpydoc documentation standards. All available functionality is tested via standalone unit tests to verify the correctness of the algorithms, including invalid cases. The test suite provides more than 95% code coverage. The source code is peer-reviewed for both architecture and implementation. There is special attention on immutable data containers for reproducible results and strict error checking of input parameters to help users avoid simple mistakes.

Documentation & Examples The library is well documented[2]. Publicly available methods are complete with source code documentation, including their arguments, default parameter settings, return values, and exception cases. The library supports typing to provide the user with argument hints and annotations. We provide installation instructions, detailed usage examples[3] and an API reference guide.

Seq2Pat as an Integration Technology

Our tool is readily available for any mining application that deals with data encoded as sequences of symbols. For continuous sequences, such as time series, discretization can be performed (Fournier-Viger et al. 2017; Lin et al. 2007). The original work presents successful applications using streaming data from MSNBC, an e-commerce website, and online news. These are considerably large benchmarks with 900K sequences of length more than 29K, containing up to 40K items (Hosseininasab, van Hoeve, and Ciré 2019).

Beyond pattern mining, we envision `Seq2Pat` as an integration technology to enable other AI applications. It can be used to capture data characteristics for downstream AI models. `Seq2Pat` generates succinct representations from large volumes of digital clickstream activity. The patterns found then become consumable for subsequent machine learning models. This generic process alleviates manual feature engineering and automates feature generation. In the next section, we present a demonstration of this integration on a public dataset.

[2]https://fidelity.github.io/seq2pat

[3]https://github.com/fidelity/seq2pat/blob/master/notebooks/

Customer Intent Prediction

We demonstrate how to automate the feature extraction process leveraging Seq2Pat. For this purpose, we use a publicly available dataset on online shopper behavior. The patterns generated by our tool serve as input features for downstream machine learning models to predict shopper intent. We show that the auto-generated features by Seq2Pat match the hand-crafted features explicitly designed for this dataset and improves the performance of predictive models.

In the following, we describe the dataset, the pattern mining and feature generation process, the machine learning models, and the training setup. We then present numeric results and study feature importance to drive insights and explanations from auto-generated Seq2Pat features.

Clickstream Data The dataset contains rich clickstream behavior on online users browsing a popular fashion e-commerce website (Requena et al. 2020). It consists of 203,084 shoppers' click sequences. There are 8,329 sequences with at least one purchase, while 194,755 sequences lead to no purchase. The sequences are composed of symbolized events as shown in Table 2. Each sequence has length L within the range $5 \leq L \leq 155$. Sequences leading to purchase are labeled as positive (+1); otherwise, labeled as negative (0), resulting in a binary classification problem.

Sequential Pattern Mining We divide the customers into two groups as purchasers and non-purchasers. Then, we apply Seq2Pat to each group independently to mine patterns specific to each characteristic (Wang and Serdar 2022). For each event, we have two attributes: the sequential order in a sequence, $\mathcal{A}_{order}$, and the dwell time on a page, $\mathcal{A}_{time}$. We enforce two constraints to seek interesting patterns. First, we require the maximum length of a pattern to be 10. Additionally, we seek page views where shoppers spend at least 20 secs on average. More precisely, we set $C_{spn}(\mathcal{A}_{order}) \leq 10$ and $C_{avg}(\mathcal{A}_{time}) \geq 20_{(sec)}$. Note that the maximum span constraint is anti-monotone and is relatively easy to enforce. In contrast, the average constraint is non-monotone and is one of the challenging constraints that is supported uniquely by Seq2Pat. We set the $min\_frequency$ threshold as the 30% of the total number of sequences. Seq2Pat finds 457 frequent patterns in purchase sequences, in 2 minutes on a Linux RHEL7 OS, 16-core 2.2GHz CPU, and 64 GB of RAM, and 236 frequent patterns from the non-purchase sequences, in 35 minutes on the same machine, with some overlap between the two groups.

Feature Generation When the two different sets of patterns from purchaser and non-purchaser are compared, there are 244 unique purchaser patterns and 23 unique non-purchaser patterns. The groups share 213 patterns in common. In combination, we have 480 unique frequent patterns. We generate the feature space via one-hot encoding. For each sequence, we create 480-dimensional feature vector with a binary indicator to denote the existence of a pattern.

Predictive Modeling To study the behavior of auto-generated patterns, we develop four different models for purchase prediction: *LightGBM* (Ke et al. 2017), shallow neural network using one hidden layer (*Shallow_NN*), Long Short Term Memory (*LSTM*) network (Hochreiter and Schmidhuber 1997) from (Requena et al. 2020) that uses sequences as input, and *LSTM* boosted with Seq2Pat patterns (*LSTM_seq2pat*). *LSTM* applies one hidden layer on the output of the last layer as input followed by a fully connected layer to make predictions. *LSTM_seq2pat* uses the same architecture with the only difference that Seq2Pat based features are concatenated to the output of *LSTM* and are used together as input to the hidden layer.

SYMBOL	EVENT
1	Page view
2	Detail (see product page)
3	Add (add product to cart)
4	Remove (remove product from cart)
5	Purchase
6	Click (click on result after search)

Table 2: The symbols used to depict clickstream events.

Training Setup We use 80% of the data as the train set and 20% as the test set and repeat this split 10 times for robustness. We compare the average results for each model based on Precision, Recall, F1 score, and the area under the ROC curve, also known as AUC.

Hyper-parameter Tuning: We apply 3-fold cross-validation for hyper-parameter tuning in the first train-test split. We apply grid search on the number of iterations [400, 600, 800, 1000] for *LightGBM*, number of nodes in the hidden layer [32, 64, 128, 256, 512] for *Shallow_NN* and *LSTM* models, number of *LSTM* units [32, 64, 128]. We use 10% of train set as a validation set to determine if training meets early stop condition. When the loss on validation set stops decreasing steadily, training is terminated. The validation set is used to determine a decision boundary on the predictions for the highest F1 score. The final parameters are 400 iterations for *LightGBM*, 64 nodes for *shallow_NN*, 32 units for the two *LSTM* models with 64 and 128 nodes in hidden layers.

Numerical Results Table 3 presents the average results that compare model performance. For feature space, we either use the patterns found by Seq2Pat, the raw clickstream events, or their combination. Notice *LightGBM* and *Shallow_NN* cannot operate on clicstream events. Instead, using auto-generated Seq2Pat features, *LightGBM* and *Shallow_NN* achieve a performance that closely match the results given in the reference work (Requena et al. 2020). The difference is, models in (Requena et al. 2020) use hand-crafted features, while we automate the feature generation process. When a more sophisticated model such as *LSTM* is used purely on clickstreams, it outperforms *LightGBM* and *Shallow_NN*, where the latter two models lack the ability to capture the sequential nature of the data. When *LSTM* is combined with Seq2Pat, *LSTM_seq2pat* further improves the performances in terms of Recall, F1 score and AUC. We conclude that the features extracted automatically via Seq2Pat boost ML models in downstream task for shopper intent prediction.

MODEL	FEATURES SPACE	PRECISION(%)	RECALL(%)	F1(%)	AUC(%)
LightGBM	Seq2Pat Patterns	44.70 (± 1.92)	63.15 (± 4.65)	52.20 (± 0.65)	94.98 (± 0.15)
Shallow_NN	Seq2Pat Patterns	44.40 (± 2.18)	64.11 (± 4.57)	52.31 (± 0.54)	95.00 (± 0.17)
LSTM	Clickstream	**54.96** (± 1.77)	69.53 (± 4.31)	61.28 (± 0.95)	96.41 (± 0.15)
LSTM_seq2pat	Clickstream + Seq2Pat Patterns	54.35 (± 2.40)	**73.64** (± 4.70)	**62.39** (± 0.81)	**96.76** (± 0.12)

Table 3: Comparison of averaged classification performance by different methods over 10 random Train-Test splits.

Feature Importance Lastly, we study feature importance to drive high-level insights and explanations from auto-generated `Seq2Pat` features. We examine the SHapley Additive exPlanation (SHAP) (Lundberg et al. 2020) value of features from the *LightGBM* model. Figure 4 shows the top-20 features with the highest impact. Our observations match previous findings in (Requena et al. 2020). The pattern $\langle 3, 1, 1 \rangle$ provides the most predictive information, given that the symbol (3) stands for adding a product. Repeated page views as in $\langle 1, 1, 1, 1, 1, 1, 1 \rangle$, or specific product views, $\langle 2, 1, 1, 1 \rangle$ are indicative of purchase intent, whereas web exploration visiting many products, $\langle 1, 1, 2, 1, 2 \rangle$, are more negatively correlated to a purchase. Interestingly, searching actions $\langle 6 \rangle$ have minimum impact on buying, raising questions about the quality of the search and ranking systems. Most frequent patterns also yield new insights not covered in the existing hand-crafted analysis. Most notably, we discover that removing a product but then remaining in the session for more views, $\langle 4, 1, 1 \rangle$ is an important feature, positively correlated with a purchase. These might be scenarios where customers need specific products, hinting at business potential for further incentives such as prompting a virtual chat agent or recommending personalized promotions.

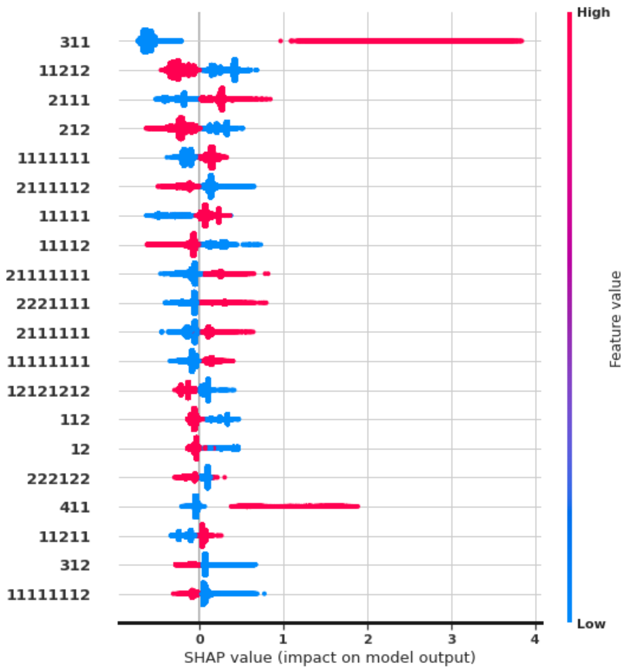

Figure 4: SHAP values of auto-generated features by `Seq2Pat`. Top-20 features are ranked by importance in descending order. Color indicates high (in red) or low (in blue) feature value. Horizontal location indicates the correlation of the feature value to a high or low model prediction.

Related Work

Historically, SPM was introduced in the context of market basket analysis (Agrawal and Srikant 1995) with several algorithm such as GSP (Srikant and Agrawal 1996), PrefixSpan (Pei et al. 2001), SPADE (Zaki 2001) and SPAM (Ayres et al. 2002). Mining the complete set of patterns imposes high computational costs and contains a large number of redundant patterns. Thus CSPM is proposed to alleviate this problem (Bonchi and Lucchese 2005; Nijssen and Zimmermann 2014; Aoga, Guns, and Schaus 2017). Constraint Programming and graphical representation of the sequence database have been shown to perform well for CSPM (Kemmar et al. 2017; Guns et al. 2017; Borah and Nath 2018; Hosseininasab, van Hoeve, and Ciré 2019).

Although a few Python libraries exist for SPM, see, e.g., (Gao 2019; Dagenais 2016), to the best of our knowledge, `Seq2Pat` is the first CSPM library in Python that supports several anti-monotone and non-monotone constraint types. Unfortunately, other CSPM implementations are either not available in Python, hence missing the opportunity to integrate with ML applications, or limited to a few constraint types, most commonly, gap, maximum span, and regular expressions (Yu and Hayato 2006; Bermingham 2018; Aoga, Guns, and Schaus 2016; Fournier-Viger et al. 2016).

Conclusion

Pattern mining is an essential part of data analytics and knowledge discovery from sequential databases. It is a powerful tool, especially when combined with constraint reasoning to specify desired properties. In a collaboration between academia and industry, we open-sourced `Seq2Pat` to improve applied AI innovation and deployment of pattern mining systems. `Seq2Pat` provides an easy-to-use high-level Python API for CSPM applications without sacrificing performance, thanks to its efficient low-level C++ backend. This enables researchers and practitioners to take advantage of the state-of-the-art MDD algorithm in a declarative fashion with modeling support for several constraint types. Finally, `Seq2Pat` can play an integrator role for automated feature generation for downstream machine learning tasks as demonstrated here on a customer intent prediction problem from fashion e-commerce.

References

Agrawal, R.; and Srikant, R. 1995. Mining Sequential Patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, 3–14.

Aoga, J. O.; Guns, T.; and Schaus, P. 2016. An efficient algorithm for mining frequent sequence with constraint programming. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 315–330.

Aoga, J. O.; Guns, T.; and Schaus, P. 2017. Mining Time-Constrained Sequential Patterns with Constraint Programming. *Constraints*, 22(4): 548–570.

Ayres, J.; Flannick, J.; Gehrke, J.; and Yiu, T. 2002. Sequential PAttern Mining Using a Bitmap Representation. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 429–435.

Behnel, S.; Bradshaw, R.; Citro, C.; Dalcin, L.; Seljebotn, D. S.; and Smith, K. 2011. Cython: The Best of Both Worlds. *Computing in Science and Engineering*, 13(2): 31–39.

Bergman, D.; Ciré, A. A.; van Hoeve, W.; and Hooker, J. N. 2016. *Decision Diagrams for Optimization*. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer.

Bermingham, L. 2018. Sequential pattern mining algorithm with DC-SPAN, CC-SPAN. https://github.com/lukehb/137-SPM. Accessed: 2021-09-16.

Bonchi, F.; and Lucchese, C. 2005. Pushing Tougher Constraints in Frequent Pattern Mining. In *Proceedings of the 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, 114–124.

Borah, A.; and Nath, B. 2018. FP-Tree and Its Variants: Towards Solving the Pattern Mining Challenges. In Somani, A. K.; Srivastava, S.; Mundra, A.; and Rawat, S., eds., *Proceedings of First International Conference on Smart System, Innovations and Computing*, 535–543.

Bou Rjeily, C.; Badr, G.; Hajjarm El Hassani, A.; and Andres, E. 2019. *Medical Data Mining for Heart Diseases and the Future of Sequential Mining in Medical Field*, 71–99. Springer.

Chen, E.; Cao, H.; Li, Q.; and Qian, T. 2008. Efficient Strategies for Tough Aggregate Constraint-Based Sequential Pattern Mining. *Information Sciences*, 178: 1498–1518.

Dagenais, B. 2016. Simple Algorithms for Frequent Item Set Mining. https://github.com/bartdag/pymining. Accessed: 2021-09-16.

Fournier-Viger, P.; Lin, C.; Gomariz, A.; Gueniche, T.; Soltani, A.; Deng, Z.; and Lam, H. T. 2016. The SPMF Open-Source Data Mining Library Version 2. In *Proceedings of the 19th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2016) Part III*, 36–40.

Fournier-Viger, P.; Lin, J. C.-W.; Kiran, R.-U.; Koh, Y.-S.; and Thomas, R. 2017. A Survey of Sequential Pattern Mining. *Data Science and Pattern Recognition*, 1(1): 54–77.

Gan, W.; Lin, J. C.-W.; Fournier-Viger, P.; Chao, H.-C.; and Yu, P. S. 2019. A Survey of Parallel Sequential Pattern Mining. *ACM Transactions on Knowledge Discovery from Data*, 13(3).

Gao, C. 2019. Sequential pattern mining algorithm with PrefixSpan, BIDE, and FEAT. https://github.com/chuanconggao/PrefixSpan-py. Accessed: 2021-09-16.

Guns, T.; Dries, A.; Nijssen, S.; Tack, G.; and De Raedt, L. 2017. MiningZinc: A declarative framework for constraint-based mining. *Artificial Intelligence*, 244: 6–29.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780.

Hosseininasab, A.; van Hoeve, W.; and Ciré, A. A. 2019. Constraint-Based Sequential Pattern Mining with Decision Diagrams. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, 1495–1502.

Hug, N. 2020. Surprise: A Python library for recommender systems. *Journal of Open Source Software*, 5(52): 2174.

Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; and Liu, T.-Y. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 3149–3157.

Kemmar, A.; Lebbah, Y.; Loudni, S.; Boizumault, P.; and Charnois, T. 2017. Prefix-projection global constraint and top-k approach for sequential pattern mining. *Constraints*, 22: 265–306.

Lin, J.; Keogh, E.; Wei, L.; and Lonardi, S. 2007. Experiencing SAX: A Novel Symbolic Representation of Time Series. *Data Mining and Knowledge Discovery*, 15(2): 107–144.

Lundberg, S. M.; Erion, G.; Chen, H.; DeGrave, A.; Prutkin, J. M.; Nair, B.; Katz, R.; Himmelfarb, J.; Bansal, N.; and Lee, S.-I. 2020. From Local Explanations to Global Understanding with Explainable AI for Trees. *Nature Machine Intelligence*, 2: 56—67.

Nijssen, S.; and Zimmermann, A. 2014. Constraint-Based Pattern Mining. *Frequent Pattern Mining*, 147–163.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830.

Pei, J.; Han, J.; Mortazavi-Asl, B.; Pinto, H.; Chen, Q.; Dayal, U.; and Hsu, M.-C. 2001. PrefixSpan,: mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings 17th International Conference on Data Engineering*, 215–224.

Pei, J.; Han, J.; and Wang, W. 2007. Constraint-Based Sequential Pattern Mining: The Pattern-Growth Methods. *Journal of Intelligent Information Systems*, 28(2): 133–160.

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*, 2020: 16983.

Srikant, R.; and Agrawal, R. 1996. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, 3–17.

Wang, X.; and Serdar, K. 2022. Dichotomic Pattern Mining with Applications to Intent Prediction from Semi-Structured Clickstream Datasets. In *Knowledge Discovery from Unstructured Data in Financial Services Workshop at AAAI*.

Wegener, I. 2000. *Branching programs and binary decision diagrams: theory and applications*. Society for Industrial and Applied Mathematics (SIAM).

Yu, H.; and Hayato, Y. 2006. Generalized sequential pattern mining with item intervals. *Journal of Computers*, 1(3): 51–60.

Zaki, M. J. 2001. SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning*, 42(1–2): 31–60.