jKarma: a Highly-Modular Framework for Pattern-Based Change Detection on Evolving Data (Discussion Paper)

Angelo Impedovo, Corrado Loglisci, Michelangelo Ceci, Donato Malerba

Dept. of Computer Science, University of Bari "Aldo Moro", Bari, Italy
{name.surname}@uniba.it

Abstract. Pattern-based change detection (PBCD) describes a class of change detection algorithms for evolving data. Contrary to conventional solutions, PBCD seeks changes exhibited by the patterns over time and therefore works on an abstract form of the data, which prevents the search for changes on the raw data. Moreover, PBCD provides arguments on the validity of the results because patterns mirror changes occurred with any form of evidence. However, the existing solutions differ on data representation, mining algorithm and change identification strategy, which we can deem as main modules of a general architecture, so that any PBCD task could be designed by accommodating custom implementations for those modules. This is what we propose in this paper through *jKarma*, a highly-modular framework for designing and performing PBCD.

Keywords: Change Detection, Evolving Data, Software Framework

1 Introduction

Pattern-based change detection (PBCD) refers to the class of change detection solutions able to find out data-points in which the data distribution changes by acting on the patterns rather than on raw data. Despite the attention it could raise, we ascertain lacking in comprehensive environments able to investigate the problem with alternative solutions or even with integrable implementations. Its main peculiarity is working in an unsupervised fashion, without relying on labeling, which often makes it preferable to the supervised approaches.

The blueprint relies on three main methodological decisions, that is, data description, pattern mining algorithm, and change identification strategy. Pattern mining algorithms are in charge of building an abstract representation of the evolving data (patterns). The change identification strategy is in charge of

searching for changes expressed by the patterns by the effect of possible distribution drifts in the underlying data. In PBCDs, the changes correspond to variations that occurred on the patterns discovered over time. While the decision on which technique to use for the pattern mining and change identification components determines the algorithmic aspects of a PBCD solution, the data representation strictly concerns the formalism of the evolving data, characteristics of the original data to consider and pattern language. For instance, the PBCDs implemented in [1,2] identify the changes through a generic notion of Jaccard dissimilarity defined for three different types of patterns, that is, *frequent subnetworks*, and *δ-closed itemsets*.

Our purpose is to provide the users with a software framework that supports the study of a predictive problem (change detection) through an unsupervised data mining task (pattern mining) while disseminating existing PBCDs and promoting the development of new ones. As our best knowledge, this is the first solution that combines change detection and pattern mining, while they have been explored as separate tasks in existing frameworks. For instance, MOA [3] has been designed to work on evolving data and offers algorithms that deal with concept drift in predictive tasks. SPMF [4] presents several classes of patterns (such as, sequential patterns and periodic patterns), but no one defined for change detection. In this discussion paper, we accomplish this with jKarma, a framework written in Java and proposed in [5] which offers loosely coupled modules, does not require programming efforts and enables the use of reusable, off-the-shelf and ad-hoc implementations for algorithmic components.

2 Background and PBCD architecture

In this section we provide preliminary notions and explain the conceptual architecture under which PBCD solutions can be collocated. Given the set of items I, a transactional database is the time-ordered sequence $D = \langle T_1, T_2, \ldots, T_n \rangle$. Each $T_i \subseteq I$ is a transaction observed in t_i and uniquely identified by id i. Thus, a pattern $P \subseteq I$ is a set of $|P|$ items, and, for PBCD purposes, they are discovered from transactions in time windows. A window $W = [t_i, t_j]$, with $t_i < t_j$, is the sequence of $|W| = j - i + 1$ transactions $\{T_i, \ldots, T_j\} \subseteq D$. P_W denotes the set of patterns discovered on the window W.

In the blueprint of PBCD, the *Mining step* and the *Identification step* search for change-points on evolving data by using *Time-windows models*. In particular, two time-windows W and W', $W = [t_b, t_e]$ and $W' = [t'_b, t'_e]$ ($t_b \leq t'_b \leq t_{e+1}$, $t_e < t'_e$) are built (Figure 1, Step 2) and input to a pattern mining algorithm, which discovers two pattern sets P_W and $P_{W'}$ (Figure 1, Step 3). In these terms, the changes are attributed to the patterns which make P_W different from $P_{W'}$. In particular, we can determine the *i)* amount of the change through a quantification of the difference between the two pattern sets, *ii)* temporal collocation of the changes (change-points) as the time in which the difference-patterns occur (Figure 1, Step 4). For this core procedure, jKarma offers a general architecture that supports software modularity (Figure 1). It makes the decisions on the specific

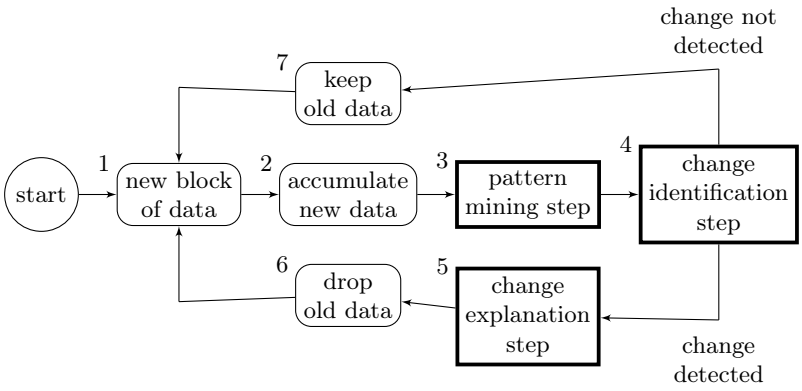

Fig. 1: Overview of the PBCD architecture

implementation for *Time-windows models*, *Mining step* and *Identification step* independent from each other. Indeed, the time-window models allow us to build sub-sequences of data regardless of their original structure (such as, itemsets, subgraphs, subtrees) and the choice of the specific model to use (such as, sliding, landmark, tilted) is not constrained neither by pattern mining nor change identification, since the time-windows are only in charge of to scan evolving data and account for new (recent) transactions and old (past) transactions (Figure 1, Steps 2, 6 and 7). The sole assumption of jKarma is the the availability of evolving data in the form of transactional databases.

The Identification step (Figure 1, Step 4) is in charge of spotting variations which the new pattern set $P_{W'}$ presents in comparison with the old pattern set P_W. To do that, jKarma makes available different implementations of dissimilarity measures (such as Jaccard dissimilarity, etc.) defined on several notions of evidence of the patterns (such as relative frequency, frequency ratio, periodicity, etc.). Not all the dissimilarity values are worthwhile of interest, but only those that exceed a desired degree of change, as well as, not all the patterns exhibit a variation in the evidence, but only that exceed a desired degree of evidence. This enables jKarma to provide "explanations" of the changes in the form of patterns that better express the underlying changes (Figure 1, Step 5).

3 Software Framework & Functionalities

jKarma is an open-source highly-modular framework written in Java 8 for defining and executing custom PBCDs. The framework, publicly available under the Apache License 2.0, exposes an API easing the rapid prototyping of custom PBCD strategies, tailored for data coming from transactional data sources, by implementing the general architecture seen in Section 2. Custom PBCDs are in-

`https://jkarma.bitbucket.io/`

stantiated by composition, meaning that existing modules for the pattern mining, change identification and change explanation steps can be combined to design PBCDs ready to be used. Every PBCD defined with jKarma is completely independent from other data mining and machine learning libraries, and third-parties data sources, thus offering two advantages, *i)* integrability with existing projects using their data sources (such as, relational databases, graph databases, xml documents), and *ii)* interoperability with existing analytics frameworks.

The framework has been developed as a multi-module Maven project in which two modules expose the APIs for defining custom pattern mining strategies (`jkarma-mining`) and custom PBCDs pipelines on top of previously defined pattern mining strategies (`jkarma-pbcd`). In particular, the main functionalities are served by two main factory classes: *i)* `org.jkarma.mining.structures.Strategies` constructs `MiningStrategy` objects implementing the pattern mining algorithm to be used in the *Mining step* of the PBCD architecture,
ii) `org.jkarma.pbcd.detectors.Detectors` constructs `PBCD` objects implementing the details of every step involved in the PBCD architecture.

The expressiveness of the programming interface enables the modular design of custom PBCD strategies. This is done by reusing existing software for the (*mining step* and *identification step*) in the PBCD architecture. In the current version, it is possible to devise PBCDs based on 5 pattern mining algorithms (Eclat, diffEclat, LCM and LCM-Max, and PFPM), 3 pattern languages (itemsets, subgraphs, subtrees), 4 time-window models (blockwise sliding/landmark, cumulative sliding/landmark) and 2 search algorithms (depth-first search and beam search). Furthermore, the API allows the user to implement his modules when necessary.

4 Illustrative Examples

In this section, we report some illustrative examples of how jKarma can be used for building different PBCDs by following a component-based architectural model. Specifically, in the following we will show how the user can specify the details about the *Mining step*, the *Detection step*, and the *Explanation step* in a two-step approach: the first step uses the `Strategies` class to define a `MiningStrategy` object, while the second step injects it into a `PBCD` object via the `Detectors` class. It is evident that the choices are domain-specific and affect the behavior of the PBCDs, thus resulting in different change detection results.

4.1 Definition of mining strategies

The mining strategy is defined by instantiating a generic `MiningStrategy<A,B>` object, hence by specifying the set of items (of type `A`), the pattern language, the pattern evidence criterion (implemented in a class of type `B`), the mining algorithm and the search strategy of patterns. Listing 1.1 shows the definition of a mining strategy, based on the Eclat algorithm, which searches for connected subgraphs. The pattern evidence criterion filters out frequent connected subgraphs

(FCSs) whose frequency is lower than the minimum threshold (`minSupp`). Eclat computes the frequency of a pattern by inspecting its *tidset*, a data structure collecting the identifiers of the transactions in which the pattern occurs.

```
public MiningStrategy<LabeledEdge, TidSet>
defineStrategy(double minSupp) {
    TidsetProvider<LabeledEdge> accessor = new TidsetProvider<>(Windows.blockwiseSliding());
    return Strategies.uponSubgraphs().eclat(minSupp)
        .limitDepth(3).dfs(accessor);
}
```

Listing 1.1: FCS mining strategy based on Eclat.

The strategy, which is an object of type `MiningStrategy <LabeledEdge, TidSet>`, is instantiated by the `uponSubgraphs` method that specifies the FCSs pattern language. The `eclat` method injects the mining algorithm into the mining strategy, while the `limitDepth` method limits the maximum number of edges in every FCS. Then, an instance of type `TidsetProvider <LabeledEdge>` (`accessor`) scans the transactions and build the tidsets. Finally, the `dfs` method finalizes the strategy and forces the Eclat algorithm to run in a DFS fashion.

An interesting aspect is that Eclat is used to mine FCSs, while natively it is a frequent itemset mining algorithm. In fact, in jKarma the pattern language is decoupled from the mining algorithm, so that equivalent strategies on different languages (e.g.: itemsets and subtrees) can be defined. For instance, the Eclat algorithm can be forced to discover frequent subtrees by replacing the `uponSubgraphs` method with the `uponSubtrees` one. However, since both the strategies are based on the Eclat algorithm, they will compute the frequencies of patterns by intersecting `TidSet` objects. Although this is a good choice on sparse datasets, it could be time-consuming for dense datasets [6], for which the diffEclat algorithm is more appropriate, since the frequency is computed using `DiffSet` data structures. In jKarma, mining strategies based on *diffEclat* are easily instantiated by i) invoking the `diffEclat` method instead of the `eclat` method, and ii) replacing the `TidsetProvider` data accessor with a `DiffsetProvider`. However, the main pitfall of the mining strategies discussed so far is their exhaustiveness, which leads to the discovery of complete sets of patterns. The exhaustive search is caused by the `dfs` method, which forces the mining algorithm to work in exhaustive mode. jKarma can be used to define non-exhaustive strategies based on beam-search and heuristics as done in [7].

4.2 Definition of PBCDs

As introduced in Section 2, PBCD relies on the sets of patterns P_W and $P_{W'}$ to i) compute the dissimilarity score $d(P_W, P_{W'})$ and quantify the degree of change and ii) arrange a change explanation. The dissimilarity score is computed on two equally-sized vector encodings F_W and $F_{W'}$, in which the i-th element corresponds to the weight associated with the i-th pattern in the enumeration of $P_W \cup P_{W'}$. This way, the change is quantified through vector measures, instead of set-based ones. Different weighting schemes and vector encodings could determine different change detection results.

In jKarma, a PBCD pipeline is defined by injecting a `MiningStrategy<A,B>` instance into a `PBCD<C,A,B,D>` object via the `Detectors` class. This ensures the type-checking consistency between the patterns discovered in the *mining step* and those used in the *identification step*. The generic type `C` specifies the type of transactions that will be consumed by the PBCD, while the generic type `D` denotes the pattern weighting scheme adopted. Finally, a PBCD is finalized by providing details on the *identification step* and *explanation step*.

In the following example, a `PBCD` is built by passing a `MiningStrategy` to the `upon` method. Then, a binary weighting scheme and the Jaccard dissimilarity measure are specified via the `unweighted` method. The PBCD will use the `isFrequent` predicate when constructing the binary vector encodings, while the `UnweightedJaccard` computes the dissimilarity score. This PBCD explains changes by discovering emerging patterns via the `Descriptors.eps` method. Finally, the PBCD is finalized with the `build` method which i) sets the minimum change threshold to 0.5, and ii) arranges data in blocks of 15 transactions.

```
public PBCD<LabeledGraph, LabeledEdge, TidSet, Boolean>
buildPBCD(MiningStrategy<LabeledEdge,TidSet> strategy) {
    UnweightedJaccard m = new UnweightedJaccard();
    return Detectors.upon(strategy)
        .unweighted((p,t)->Patterns.isFrequent(p, minFreq, t), m)
        .describe(Descriptors.eps(minGr)).build(0.5, 15);
}
```

Listing 1.2: PBCD based on the unweighted jaccard dissimilarity between binary-valued vector encodings of patterns.

4.3 A complete example: the KARMA algorithm

Detecting changes is particularly relevant for dynamic networked data, that is, networks which evolve over time, for which no common notion of change exists. In fact, different methods ascribe changes to variations in the observed nodes, while others focus on edges or subgraphs observed over time, which leads to clearly different results. Moreover, many proposed methods are not part of existing software frameworks, which limits their versatility. Listing 1.3 reports a complete example in which jKarma is used so as implementing the KARMA PBCD algorithm presented in [1], which detects changes in dynamic networks by observing variations in the FCSs discovered over time. The example also shows how users can react to changes, by following the event-listener paradigm: the `changeDetected` method will be be executed in case of detected changes, otherwise, the `changeNotDetected` method will be executed.

```
public PBCD<LabeledGraph, LabeledEdge, TidSet, Boolean>
getKARMA(double minSupp, double minChange, double minGr) {
    //auxiliary components
    TidSetProvider<LabeledEdge> dataAccessor = new
         TidSetProvider<>(Windows.cumulativeLandmark());
    UnweightedJaccard m = new UnweightedJaccard();
    Descriptor descriptor = Descriptors.partialEps(minSupp, minGr);

    //mining strategy definition
    MiningStrategy<LabeledEdge, TidSet> strategy = Strategies.uponSubgraphs().eclat(minSupp)
```

https://bitbucket.org/jkarma/demo-karma-pbcd/

```
        .limitDepth(3).dfs(dataAccessor);

    //PBCD definition
    return Detectors.upon(strategy)
        .unweighted((p,t)->Patterns.isFrequent(p, minSupp, t), m)
        .describe(descriptor).build(minChange, 15);
}

public void runKARMA(Stream<LabeledGraph> dataSource) {
    PBCD<LabeledGraph, LabeledEdge, TidSet, Boolean> detector = this.getKarma(0.15, 0.2, 1.2);
    //change detection event listening
    detector.registerListener(new PBCDEventListener<LabeledEdge, TidSet>(){
        public void changeDetected(
        ChangeDetectedEvent<LabeledEdge,TidSet> e){
            //reaction to change detected
        }
        public void changeNotDetected(
        ChangeNotDetectedEvent<LabeledEdge,TidSet> e){
            //reaction to change not detected
        }
    });
    //consume the data source
    dataSource.forEach(detector);
}
```

Listing 1.3: Example of jKarma implementing the KARMA PBCD [1].

Indeed, jKarma enables the users to detect changes in dynamic networks with alternative approaches. The example shows how to instantiate the KARMA algorithm, which is a good choice when the change has to be detected on subgraphs. However, the solution could not be the best one when changes affects only some attributes of nodes. To this end, jKarma can be used to rapid prototyping of new algorithms in Java.

4.4 Comparative evaluation

To show the effectiveness of jKarma in deploying actionable PBCDs, we compare the detection accuracy and running times of four PBCD algorithms (KARMA, PBCD-1, PBCD-2, and StreamKrimp) on three synthetic datasets with same minimum frequency and change thresholds (equal to 0.5). Specifically, KARMA, PBCD-1, and PBCD-2 have been implemented in jKarma. PBCD-1 and PBCD-2 are non-exhaustive variants of the exhaustive KARMA algorithm that make use of the landmark window model and sliding window model, respectively. While StreamKrimp [8] is a non-exhaustive PBCD based on frequent itemsets discovered according to the MDL principle. The results (Table 1) show that non-exhaustive PBCDs (PBCD-1, PBCD-2, and StreamKrimp) are more accurate than those exhaustive (KARMA). Although exhaustive, KARMA is more efficient than StreamKrimp, which is not implemented with jKarma. Finally, PBCD-1 offers the higher accuracy, while PBCD-2 has the lower running times.

5 Conclusions

We have introduced jKarma, an highly-modular framework for defining and executing customized pattern-based change detection approaches for evolving data,

`https://bitbucket.org/jkarma/datasets`

Table 1: Running times and accuracies on synthetic data.

dataset	Running times (seconds)				dataset	Accuracy			
	PBCD-1	PBCD-2	KARMA	S.Krimp		PBCD-1	PBCD-2	KARMA	S.Krimp
synth-drifts-1	12.913	6.194	60.763	86.130	synth-drifts-1	0.987	0.918	0.804	0.931
synth-drifts-2	12.284	6.522	55.982	77.138	synth-drifts-2	0.991	0.916	0.799	0.911
synth-drifts-3	12.603	6.463	58.137	76.750	synth-drifts-3	0.988	0.918	0.796	0.916

in Java. jKarma enables the modular definition of custom PBCDs, with reduced or none implementation efforts, by following a component-based architectural model. The framework comes as a Java software library which is completely independent of other data mining frameworks and existing data sources. As future work, we plan to investigate the periodicity of the changes [9].

Acknowledgments

We acknowledge the support of the MIUR - Ministero dell'Istruzione dell'Università e della Ricerca through the project "TALIsMan - Tecnologie di Assistenza personALizzata per il Miglioramento della quAlità della vitA" (Grant ID: ARS01_01116), funding scheme PON RI 2014-2020

References

1. C. Loglisci, M. Ceci, A. Impedovo, D. Malerba, Mining microscopic and macroscopic changes in network data streams, Knowl. Based Syst. 161 (2018) 294–312. doi:10.1016/j.knosys.2018.07.011.
2. D. Trabold, T. Horváth, Mining strongly closed itemsets from data streams, in: 20th International Conference, DS 2017, Kyoto, 2017, pp. 251–266.
3. A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: massive online analysis, J. Mach. Learn. Res. 11 (2010) 1601–1604.
4. P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu, V. S. Tseng, SPMF: a java open-source pattern mining library, J. Mach. Learn. Res. 15 (1) (2014) 3389–3393.
5. A. Impedovo, C. Loglisci, M. Ceci, D. Malerba, jkarma: A highly-modular framework for pattern-based change detection on evolving data, Knowl. Based Syst. 192 (2020) 105303. doi:10.1016/j.knosys.2019.105303.
6. M. J. Zaki, K. Gouda, Fast vertical mining using diffsets, in: Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, USA, 2003, pp. 326–335.
7. A. Impedovo, M. Ceci, T. Calders, Efficient and accurate non-exhaustive pattern-based change detection in dynamic networks, in: Discovery Science - 22nd International Conference, DS 2019, Split, Croatia, October 28-30, Proceedings, 2019.
8. M. van Leeuwen, A. Siebes, Streamkrimp: Detecting change in data streams, in: ECML/PKDD 2008, Belgium 2008, Proceedings, Part I, pp. 672–687.
9. C. Loglisci, M. Ceci, A. Impedovo, D. Malerba, Mining spatio-temporal patterns of periodic changes in climate data, in: 5th International Workshop, NFMCP 2016, Held in Conjunction with ECML-PKDD 2016, Italy, 2016, Revised Selected Papers, pp. 198–212. doi:10.1007/978-3-319-61461-8_13.