Multi-User Segmentation for Process-Based Habit Mining

Giacomo Lanciano
lanciano.1487019@studenti.uniroma1.it

December 13, 2017

Abstract

This report presents the activities done and the results achieved during the Excellence Programme of the M.Sc. in Engineering in Computer Science at Sapienza University of Rome (A.Y. 2016/2017). This subject of this project is the design and implementation of techniques for Process-Based Habit Mining, in the context of Ambient Intelligence. This work has been supervised by Prof. Massimo Mecella and his research group. In particular, we focused on facing the problem of tracking the actions of multiple users in a smart environment, analysing a raw sensor log coming from a network of PIR[1] sensors.

1 Introduction

The goal of this work is to use supervised learning techniques to identify the sub-sequences of a given sensor log, possibly coming from a smart environment populated by many users, that are related to the actions of individual users. These information can be fed as input to many other kinds of analysis. For instance, we could see these sub-sequences as traces and mine the habits of the various users in the environment by applying Process Mining (a discipline that sits between data mining and process modeling and analysis and, hence, can be considered one of the links between data science and process science [5]) as depicted in [1].

In these settings, the real challenge is to deal with unlabelled sensor logs to create an effective training set (made of positive samples only, in this case) for some supervised learning model. Indeed, due to the presence of many users in the environment, the traces corresponding to their actions could be interleaved and, since we rely on PIR measures only, we have no indications about which user an entry in the log is related to. Besides, even though we were able to tell which user a measure refers to, we would still have to deal with the lack of *case-ids*[2] , that are necessary to apply most of process discovery techniques. Many approaches have been proposed to deal with unlabelled event logs [2, 6] but they are suitable only for well-structured and established (and often simple) business process. Due to the highly-variable nature of human behaviour (i.e. the presence of a huge amount of noise in a sensor log), our task is way more challenging.

The core of our work consists in a (batch) algorithm for sensor logs segmentation, such that the smaller chunks (i.e. sub-sequences, traces) that are produced can be considered (up to a certain degree of confidence) as related to the actions of individual users. The resulting segmentation of the given log is intended to be used as the training set of a supervised learning model, such as a One-class SVM (due to the lack of negative

[1]Passive Infra-Red.

[2]the identifiers of individual real executions of a business process.

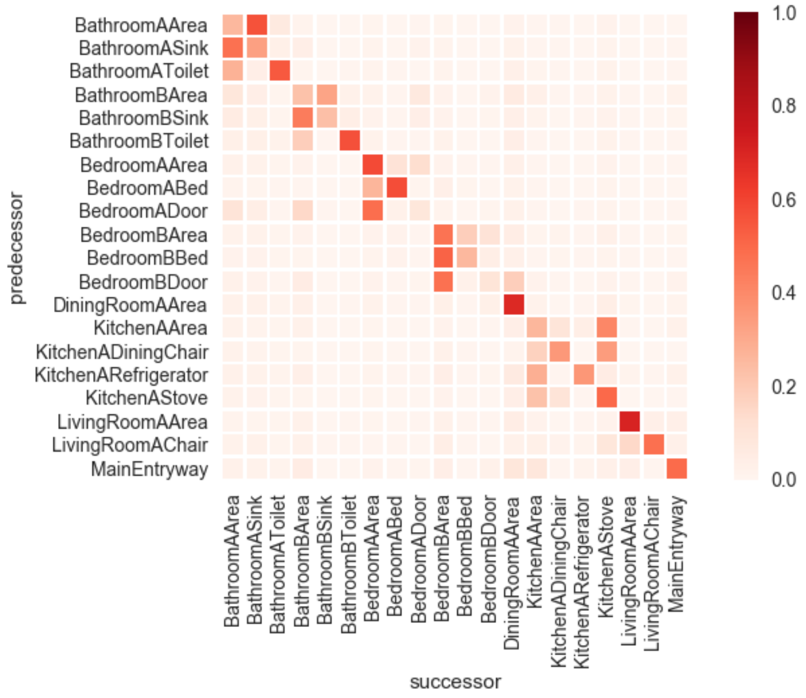

Figure 1: Topological compatibility matrix for the complete sensor log.

samples), in order to provide an *on-line* method for the detection of *single-user* segments. The validation of such a model has been conducted by taking into account only some combinations of those segments that were detected as corresponding to *crossings* between users trajectories.

The software produced during our work has been developed in *Python 3* and is accessible through a *Github* repository[3]. The sensor log considered in this work ($\sim$396.000 measurements) and additional material are also available[4].

The rest of the report is organized as follows. Section 2 elaborates on the algorithm designed to segment a sensor log in single-user chunks. Furthermore, we present the kernel function chosen for the application of supervised learning to this problem. Section 3 reports on the experiments conducted on a sensor log coming from a network of PIR sensors. Section 4 concludes the report.

2 Sensor Log Segmentation

In this section, we introduce the key concepts that are the foundations of our segmentation algorithm. The underlying assumption of our approach is that there exists a locality correlation among the sensors triggered by the actions of an individual user. In other words, we expect the single-user sub-sequences of the sensor log to be constituted by measurements coming from sensors that are placed close to each other.

Algorithm 1 Sensor Log Segmentation

```
$\mathcal{S} \leftarrow \emptyset$                                                      ▷ collection of segments.
**for all** $m \in \mathcal{L}$ **do**
    **if** there exists a single *open* $s \in \mathcal{S}$ *compatible* with $m$ **then**      ▷ A-step.
        $s \leftarrow s \| m$
    **if** there exist *open* $s_1, \dots, s_n \in \mathcal{S}$ *compatible* with $m$ **then**      ▷ B-step.
        mark $s_1, \dots, s_n \in \mathcal{S}$ as *closed*
        create a new *open* segment $s'$ initialized with $m$
        $\mathcal{S} \leftarrow \mathcal{S} \cup s'$
    **if** there exists no *open* $s \in \mathcal{S}$ *compatible* with $m$ **then**      ▷ C-step.
        create a new *open* segment $s'$ initialized with $m$
        $\mathcal{S} \leftarrow \mathcal{S} \cup s'$
mark all *open* $s \in \mathcal{S}$ as *closed*
remove all $s \in \mathcal{S}$ such that $|s| < N$                                  ▷ noise filtering.
**return** $\mathcal{S}$
```

2.1 Segmentation Algorithm

To support the locality correlation assumption, we devised a method to decide whether two measurements in a sensor log are *compatible.* Indeed, we compute the extent at which the sensors they come from are likely to be in direct succession in the given sensor log. Then, given a fixed threshold, we use this result to decide whether measurements may belong to the same single-user sub-sequence. To do that, we need to compute the compatibility degree for each possible pair of sensors occurring in the sensor log.

Let $\mathcal{M}$ be a *topological compatibility matrix*, such that $\mathcal{M}_{ij}$ is the probability that a measurement m_i (from sensor i) is *directly followed* by a measurement m_j (from sensor j) in a single-user segment, computed as the number of occurrences of the pattern $m_i \rightarrow m_j$ divided by the number of measurements coming from i in the log. Given a threshold $0 \leq t \leq 1$, we say that m_j is *compatible* with m_i if $\mathcal{M}_{ij} \geq t$ (*not compatible*, otherwise). In fig. 1, the *topological compatibility matrix* for the sensor log considered during our work is shown. As one can clearly see, the highest compatibility values are distributed along the diagonal of the matrix. This result indeed supports the initial assumption (i.e. locality correlation) since, in general, a sensor is followed either by itself or by another one that is placed nearby (notice that sensors placed in the same area of the environment have similar identifiers and result close to each other when alphabetically ordered). Besides, we noticed that the highest *diagonal* values are related with objects that, in general, are involved il long interactions with users (e.g. the bed). Provided a way to quantify the compatibility between two sensor measurements, we can trivially extend this notion to segments by saying that a measurement is *compatible* with a segment if it is so with the last measurement in the segment.

Algorithm 1 describes the procedure we use to compute the single-user segments and that takes as inputs: a sensor log $\mathcal{L}$, the *topological compatibility matrix* $\mathcal{M}$ related to $\mathcal{L}$, a compatibility threshold T and a noise threshold[5] N. Notice that $\mathcal{M}$ and T are needed to decide about the *compatibility.* We interpret the execution of a *B-step* as the intersection of different users trajectories, as we can see in fig. 2. Each time a *B-step* occurs we save the pointers to the segments that are marked as *closed* in a separated

[3] `https://github.com/giacomolanciano/multi-user-segmentation`
[4] `https://goo.gl/RizbXT`
[5] minimum sequence length.

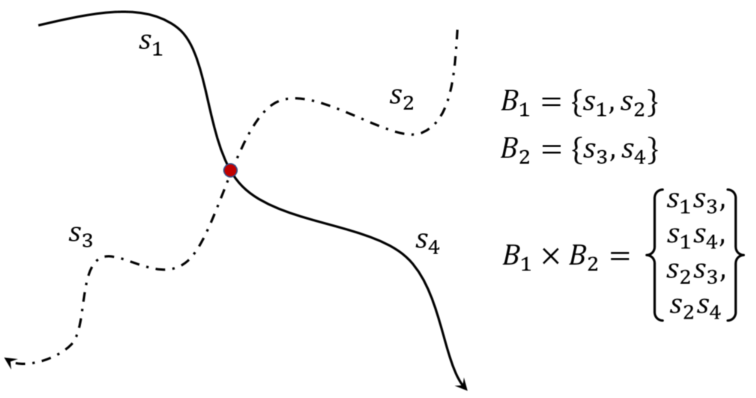
s_1
s_2
s_3
s_4
$B_1 = \{s_1, s_2\}$
$B_2 = \{s_3, s_4\}$
$B_1 \times B_2 = \begin{Bmatrix} s_1 s_3, \\ s_1 s_4, \\ s_2 s_3, \\ s_2 s_4 \end{Bmatrix}$

Figure 2: Two (intersecting) single-user trajectories and the corresponding B-step.

collection B_1. Basically, when multiple users *cross* the same point, we need to solve the problem of tracking the continuations of their individual trajectories. Indeed, the segments in B_1 are treated as *single-user* and we have to consider all their possible continuations after the *crossing*. Notice that in this way we implicitly prune the set of possible solutions by discarding the (incoherent) possibility that segments in B_1 follow each other, since it would mean to go back in time.

Then, for each new segment open between the current *B-step* and the next that is compatible with at least one segment in B_1, we add it to another collection B_2. This collection indeed contains all the possible continuations after the *crossing*. Hence, the Cartesian product $B_1 \times B_2$ contains all the possible trajectories that could be occurred, that contribute to the validation set for our One-class SVM. Conversely, the training set consists in the (overlapping) *n-gram*[6] representation of the resulting segments. Notice that we assumed the noise threshold N to be greater or equal to the length of a single n-gram.

2.2 Spectrum Kernel

To apply One-class SVM to our problem, we use a particular kind of *string kernel*, the so-called **Spectrum Kernel**. As described in [3], this kernel function is designed to be very simple and efficient to compute.

In this case, the input space $\mathcal{X}$ is constituted by all finite length sequences of characters from an alphabet $\mathcal{A}$, such that $|\mathcal{A}| = l$. Given $k \geq 1$, the k-spectrum of a sequence of characters is the set of all the k-length (contiguous) sub-sequences that it contains.

The feature map $\Phi_k : \mathcal{X} \to \mathbb{R}^{l^k}$ is indexed by all possible sub-sequences a of length k from alphabet $\mathcal{A}$ and it is defined as

$$\Phi_k(x) = (\phi_a(x))_{a \in \mathcal{A}^k}$$

where $\phi_a(x)$ are the *occurrences* of sub-sequence a in sequence x. Thus, $\Phi_k(x)$ is a weighted representation of the k-spectrum of input sequence x. Hence, the k-spectrum kernel is defined as

$$K_k(x, y) = \langle \Phi_k(x), \Phi_k(y) \rangle$$

Notice that feature vectors are *sparse*: the number of non-zero coordinates is bounded by $|x| - k + 1$. This fact leads to the possibility of applying very efficient approaches when computing the kernel function.

[6]a contiguous sequence of n items (syllables, letters, words, etc.) from a given sequence of text.

3 Experiments

Before applying algorithm 1 the complete sensor log has been pre-processed by removing entries related to sensors reset. Indeed, after being idle for a certain amount of time, PIR sensors automatically trigger to reset. Thus, such measures (labelled with *OFF*) are not informative.

In addition, we associated to each sensor an (alternative) identifier consisting of a single symbol and substituted every measurement with the one related to the sensor they came from. In this way, it is easier to build the training and validation sets for the sequence classifier. Then, the segmentation has been performed setting $T = 0.1$ and $N = 3$. In these settings, we obtained 17732 segments with maximum length equal to 7136 and average length equal to 21.

Since *scikit-learn*[4] One-class SVM implementation is not designed to handle strings vectors, the n-grams representations of the sequences have been translated using a simple binary encoding of symbols, allowing us to deal with integers vectors. Then, the model has been trained considering, for the sake of efficiency, only the sequences longer or equal than 15 (5742 segments). The whole process took $\sim$ 12 hours to terminate. Using the same length threshold, we then performed the validation of the model using the validation set (15432 segments) built according to the B-steps of the segmentation algorithm. The computation took $\sim$ 32 hours to terminate and the final results showed that the 50% of the segments in this set were compliant with the model. We repeated the experiment considering only the sequences longer or equal than 20 (4341 segments) and got the same result for compliance (training time: $\sim$ 7 hours, validation time: $\sim$ 19 hours, validation set: 12454 segments).

4 Conclusions

Notice that, because of the lack of a ground truth, it is impossible (for the time being) to conduct a proper evaluation of the approach. Thus, the 50% refers to the percentage of times that the classifier labelled a concatenation of two segments as a *single-user* sequence. Indeed, in most of the cases, the situation captured by a B-step is the one shown in fig. 2, i.e. a *crossroad* between two user trajectories that has 4 possible combinations, and only 2 are correct (that is the 50%). In conclusion, the reported results are not telling the accuracy of the classifier, but at least they show that it is behaving coherently with our definition of the validation set.

References

[1] Marcella Dimaggio, Francesco Leotta, Massimo Mecella, and Daniele Sora. Process-based habit mining: Experiments and techniques. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse, France, July 18-21, 2016*, pages 145–152, 2016.

[2] Diogo R. Ferreira and Daniel Gillblad. Discovering process models from unlabelled event logs. In *Proceedings of the 7th International Conference on Business Process Management*, BPM '09, pages 143–158, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] Christina S. Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the 7th Pacific*

Symposium on Biocomputing, PSB 2002, Lihue, Hawaii, USA, January 3-7, 2002, pages 566–575, 2002.

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

[5] Wil M. P. van der Aalst. *Process Mining: Data Science in Action.* Springer, Heidelberg, 2 edition, 2016.

[6] Michal Walicki and Diogo R. Ferreira. Sequence partitioning for process mining with unlabeled event logs. *Data Knowl. Eng.*, 70(10):821–841, October 2011.