Pattern Recognition in the Usage Sequences of Medical Apps

Chloé Adam

université PARIS-SACLAY

CentraleSupélec

NNT : 2019SACLC027

Thèse de doctorat

Analyse des Séquences d'Usage d'Applications Médicales

Thèse de doctorat de l'Université Paris-Saclay
préparée à CentraleSupélec

École doctorale n° 573 Interfaces : Approches interdisciplinaires : fondements, applications et innovation
Spécialité de doctorat : Mathématiques appliquées

Thèse présentée et soutenue à Gif sur Yvette, le 1er avril 2019, par

Chloé ADAM

Composition du jury :

Paul-Henry COURNÈDE *Professeur, CentraleSupélec, Université Paris-Saclay*	Directeur de thèse
Antoine ALIOTTI *Docteur, GE Healthcare*	Co-Encadrant
Thierry ARTIÈRES *Professeur, Ecole Centrale Marseille*	Rapporteur
Marianne CLAUSEL *Professeur, Université de Lorraine*	Rapporteuse
Céline HUDELOT *Professeur, CentraleSupélec, Université Paris-Saclay*	Présidente du jury
Fragkiskos D. MALLIAROS *Maître de Conférences, CentraleSupélec, Université Paris-Saclay*	Examinateur
Nicolas THOME *Professeur, CNAM Paris*	Examinateur

Acknowledgements

Cette thèse aura été une expérience extrêmement enrichissante. Il me tient à cœur de remercier toutes celles et ceux sans qui cet aboutissement n'aurait pas été possible.

Tout d'abord, je tiens à exprimer ma reconnaissance à Paul-Henry Cournède, mon directeur de thèse. Merci pour ta bienveillance à mon égard et la confiance que tu as pu m'accorder tout au long de cette thèse. Tu n'as cessé de m'encourager, tu m'as écoutée et tu m'as conseillée. J'ai énormément appris à tes côtés. Sans toi, cette aventure n'aurait pas été aussi instructive, scientifiquement et sur de nombreux autres aspects.

Je remercierai ensuite Antoine Aliotti, mon manager qui m'a accompagnée depuis mon arrivée en stage à GE Healthcare puis pendant ces années de thèse. J'aimerais également remercier Florence Moreau, à l'origine de ce projet. Merci de m'avoir donné l'immense opportunité de faire cette thèse.

Je tiens à exprimer ma gratitude à Marianne Clausel et Thierry Artières, pour avoir accepté de rapporter ma thèse. Merci également à Céline Hudelot, Fragkiskos D. Malliaros et Nicolas Thome, membres de mon jury, pour l'intérêt que vous avez porté à mon travail.

Merci à tous mes collègues des équipes AW et Cloud de GE Healthcare, je pense notamment à Lorraine qui s'est aventurée de nombreuses fois au fin fond du code du logiciel pour m'aider à comprendre les fichiers de logs, Guillaume qui a largement contribué à l'amélioration du contenu des logs à mon arrivée ou encore Guy pour la mise en place du processus de récupération de ces derniers. Merci aussi à Alexandre, Zineb, Mathieu, Laurent, Régis, Laura, Duc, Dreena, Florian, Véronique, Alyson et bien d'autres pour tous les bons moments passés ensemble, au bureau ou aux afterworks. Grâce à vous, je garderai un très bon souvenir de mon passage à GE Healthcare.

Merci aussi à tous les membres du laboratoire MICS, et en particulier de l'équipe Biomathematics. J'ai adoré travailler à vos côtés, même si je n'ai jamais vraiment réussi à percer au baby-foot, pratique culte au sein de l'équipe. Merci à Sarah, Véronique, Sylvie et à tous les doctorants : Antonin, Xiangtuo, Walid, Mahmoud, Mathilde, Andreas, Sylvain. Je tiens à remercier tout particulièrement Gautier et Benoit, pour votre aide tout au long de ma thèse, les séances de débogage (en Python et sous Windows en plus !), les dernières relectures... Votre soutien a été précieux ces derniers mois.

Il me tient également à cœur de remercier Jean-Pierre Lavigne et Judith Sausse respectivement directeur du CPP de Nancy et directrice des Études de l'École des Mines de Nancy lors de mon passage dans ces deux établissements. Merci de m'avoir aidée à concilier ma passion, le volley, et mes études lors de mes années de classes préparatoires et d'école d'ingénieurs. Votre soutien a été décisif dans les moments difficiles.

Un peu plus éloignées mais tout aussi importantes, je remercie mes amies d'enfance, rencontrées pour la plupart sur un terrain de volley : Aurélie, Laure, Pauline, Mélissa, Marie, pour tous les moments inoubliables passés ensemble. Un petit clin d'œil également aux volleyeuses en devenir : Eva et Zoé.

Enfin, merci à toi Julie, collègue lors de mes débuts à GE, tu as rapidement pris une place importante dans ma vie. Toujours présente pour les coups durs et surtout les coups de folie ! Merci d'être là pour moi.

Je terminerai en remerciant mes parents, mon petit frère, Quentin, et ma grand-mère pour leur soutien infaillible. Merci de m'avoir toujours donné le courage et la force d'aller au bout des choses.

Merci à tous.
Chloé

Résumé

Les radiologues utilisent au quotidien des solutions d'imagerie médicale pour le diagnostic, la préparation des interventions chirurgicales ou le traitement de suivi. Les progrès techniques actuels de l'imagerie médicale représentent un énorme défi pour leur flux de travail. Ils s'attendent à ce que les applications de visualisation avancées permettent de d'analyser plus rapidement les images des patients.

L'amélioration de l'expérience utilisateur est toujours un axe majeur de l'effort continu visant à améliorer la qualité globale et l'ergonomie des produits logiciels. L'amélioration de la qualité du logiciel implique également la réduction du nombre de dysfonctionnements qui provoquent des interruptions désagréables pour les utilisateurs. Évidemment, les conséquences de ces perturbations sont plus ou moins graves selon le domaine d'utilisation. En imagerie médicale, les pannes de logiciels sont un enjeu crucial puisque la vie des patients est en jeu, en particulier dans les cas de radiologie interventionnelle. Les applications de monitoring permettent d'enregistrer l'évolution des différents paramètres du logiciel et du système lors de leur utilisation et en particulier les actions successives effectuées par les utilisateurs dans l'interface du logiciel. Ces interactions peuvent être représentées sous forme de séquences d'actions.

Sur la base de ces données, ce travail traite de deux sujets industriels : les pannes logicielles et l'ergonomie des logiciels. Ces deux thèmes impliquent d'une part la compréhension des modes d'utilisation, et d'autre part le développement d'outils de prédiction permettant soit d'anticiper les pannes, soit d'adapter dynamiquement l'interface logicielle en fonction des besoins des utilisateurs.

Tout d'abord, nous visons à identifier les origines des crashes du logiciel qui sont essentielles afin de pouvoir les corriger. Bien que l'occurrence d'un crash soit explicitement enregistrée, les combinaisons d'actions qui l'ont déclenché ne sont pas évidentes. Pour ce faire, nous proposons d'utiliser un test binomial afin de déterminer quel type de pattern est le plus approprié pour représenter les signatures de crash. Ces types de patterns tiennent compte de l'ordre et/ou de la proximité des actions qui les composent. L'amélioration de l'expérience utilisateur par la personnalisation et l'adaptation des systèmes aux besoins spécifiques de l'utilisateur exige une très bonne connaissance de la façon dont les utilisateurs utilisent le logiciel. Afin de mettre en évidence les tendances d'utilisation, nous proposons de regrouper les sessions similaires. Pour ce faire, nous comparons trois types de représentation de session dans différents algorithmes de

clustering. Nous utilisons des indices de validité de cluster pour déterminer la configuration fournissant la meilleure partition. Les clusters obtenus et les sessions correspondantes sont ensuite visualisées et interprétées à l'aide d'un outil de process mining.

La deuxième contribution de cette thèse concerne le suivi dynamique de l'utilisation du logiciel. Nous proposons deux méthodes – basées sur des représentations différentes des actions d'entrée – pour répondre à deux problématiques industrielles distinctes : la prédiction de la prochaine action et la détection du risque de crash logiciel. Les deux méthodologies tirent parti de la structure récurrente des réseaux LSTM (Long Short Term Memory) pour capturer les dépendances entre nos données séquentielles ainsi que leur capacité à traiter potentiellement différents types de représentations d'entrée pour les mêmes données. Pour aborder la tâche de surveillance des crashes, nous proposons d'utiliser des vecteurs composés d'actions significatives contenues dans les crashes en tant qu'entrées du LSTM – avec l'idée de tirer parti de sa capacité d'apprendre des informations passées lesquelles sont pertinentes pour détecter les signatures de crash. La méthode surpasse les méthodes de classification de séquences de l'état de l'art. À des fins de comparaison, ces baselines sont également alimentées par les signatures de crash significatives. Compte tenu de l'historique des actions de l'utilisateur dans l'interface, notre deuxième méthode de suivi vise à prédire la prochaine action de l'utilisateur. Le réseau neuronal récurrent proposé surpasse les algorithmes de l'état de l'art avec des vecteurs de type one-hot en entrée. Par ailleurs, nous proposons de donner en entrée du LSTM des embeddings d'actions. Cette représentation continue est plus performante que la représentation de type one-hot et sa dimension inférieure réduit en même temps le coût de calcul. Des expériences sont menées afin de déterminer la meilleure stratégie à la fois sur des ensembles d'entraînement obtenus par des caractéristiques explicites et sur des ensembles d'entraînement obtenus par des techniques non supervisées.

Toutes nos approches sont démontrées sur des logs provenant de 50 hôpitaux différents dans le monde.

Mots-clés: exploration de motifs fréquents; données séquentielles; représentations pour l'apprentissage; représentations d'action; réseaux de neurones récurrents LSTM; représentations de session; clustering; logs de logiciel; détection des signatures de crash; suivi des risques; prédiction des actions utilisateurs; logiciel d'imagerie médicale.

Abstract

Radiologists use medical imaging solutions on a daily basis for diagnosis, surgery preparation or follow-up treatments. Today's technical advances in medical imaging represent a huge challenge for their workflows. They expect more advanced visualization applications in order to deliver effective therapies to patients faster.

Improving user experience is a major line of the continuous effort to enhance the global quality and usability of software products. The overall software quality amelioration also involves reducing the number of defects and malfunctions which cause unpleasant interruptions for users. Obviously, the consequences of these disruptions are more or less severe depending on the field of use. In radiology, software crashes are a crucial issue since patients' lives are at stake, especially in interventional radiology cases. Monitoring applications enable to record the evolution of various software and system parameters during their use and in particular the successive actions performed by the users in the software interface. These interactions may be represented as sequences of actions.

Based on this data, this work deals with two industrial topics: software crashes and software usability. Both topics imply on one hand understanding the patterns of use, and on the other developing prediction tools either to anticipate crashes or to dynamically adapt software interface according to users' needs.

First, we aim at identifying crash root causes. It is essential in order to fix the original defects. While the occurrence of a crash is explicitly recorded, the patterns of actions that triggered it are not obvious. For this purpose, we propose to use a binomial test to determine which type of patterns is the most appropriate to represent crash signatures. These types of patterns take into account the order and the proximity of the actions composing them. We conclude that subsquences are the most appropriate formalism to determine crash signatures and apply the method to a critical case.

The improvement of software usability through customization and adaptation of systems to each user's specific needs requires a very good knowledge of how users use the software. In order to highlight the trends of use, we propose to group similar sessions into clusters. For this purpose we compare 3 session representations as inputs of different clustering algorithms. We use cluster validity indices to determine the configuration providing the best partition. Sessions from the resulting clusters are then visualized and interpreted using a process mining

tool. The resulting clusters show significant differences in terms of workflows and confirm the ability of this method to highlight characteristic workflows.

The second contribution of our thesis concerns the dynamical monitoring of software use. We propose two methods – based on different representations of input actions – to address two distinct industrial issues: next action prediction and software crash risk detection. Both methodologies take advantage of the recurrent structure of Long Short Term Memory (LSTM) neural networks to capture dependencies among our sequential data as well as their capacity to potentially handle different types of input representations for the same data. To address the crash monitoring task, we propose to use feature vectors composed of significant actions contained in crash sessions as inputs of the LSTM – with the idea to take advantage of its ability to learn relevant past information to detect crash patterns. The method outperforms state of the art sequence classification methods. For a fair comparison, the latter are in addition fed with significant crash patterns. Given the history of user actions in the interface, our second monitoring method aims at predicting the next user action. The proposed recurrent neural network outperforms state of the art proactive user interface algorithms with standard one-hot vectors as inputs. Besides, we propose to feed the LSTM with action embeddings. This continuous representation performs better than one-hot encoded vector LSTM and its lower dimension reduces at the same time the computational cost. Experiments are conducted in order to determine the best strategy both on training sets obtained by explicit characteristics as well as on training sets obtained by unsupervised techniques.

All our approaches are demonstrated on logs from 50 different hospitals worldwide.

Keywords: frequent pattern mining; sequential data; representation learning; action embeddings; LSTM recurrent neural networks; session embeddings; clustering; software log files; crash signatures detection; risk monitoring; next action prediction; medical imaging software.

Contents

Introduction

RADIOLOGISTS USE MEDICAL IMAGING SOLUTIONS on a daily basis for diagnosis, surgery preparation or follow-up treatment. They play an important role in the process of patient treatment. Radiologists work closely with general practitioners or specialists and provide them with additional elements that are not detectable by clinical examination, thanks to their interpretation of images. Their analyses help dissipate doubts or on the contrary, confirm pre-existing diagnoses by detecting tumors, infections or fractures for example. Radiologists cover a large variety of specialties: cardiology, neurology, pulmonology, oncology, orthopedics, etc. using multiple imaging modalities (Computed Tomography, Magnetic Resonance Imaging, 3D X-ray, Positron Emission Tomography and Positron Emission Tomography - Computed Tomography). Today's technical advances in medical imaging represent a huge challenge for the radiologist workflow. With the decrease in image acquisition time, the amount of data to review increases. In other words, the number of patient exams increases, thus imposing shorter data interpretation and reporting times. Radiologists expect more advanced visualization applications to deliver effective patient therapy faster. This translates into more efficient and reliable image processing algorithms, but also an ever-improving user experience. While they constitute a significant time in the image review process, the optimization of processing algorithms such as automatic labeling or automatic segmentation is not the focus of our work. Our objective is the analysis of users workflows and their interaction with the software interface. Indeed, enabling to streamline diagnostic processes and simplifying repetitive tasks in user workflows is a permanent concern. Generally speaking, improving user experience is always a major line of the continuous effort to enhance the global quality and usability of software products. The overall software quality amelioration also involves reducing the number of defects and malfunctions which cause unpleasant interruptions for users. Obviously, the consequences of these disruptions are more or less severe depending on the field of use. In our case, software crashes are a crucial issue since patients' lives are at stake, especially in interventional radiology cases. Monitoring applications allow to record the evolution of various software and system parameters during their use and in particular the successive actions performed by the users in the software interface. These interactions may be represented as sequences of actions.

Based on this data, this work deals with two industrial topics: software crashes and software usability. Both topics imply on one hand understanding the patterns of use, and on the other developing prediction tools either to anticipate crashes or to dynamically adapt software in-

terface according to users' needs.

Identifying the root causes of crashes is essential in order to fix the underlying defects. While several factors such as the number of applications running, the memory availability or the dataset type are likely to cause crashes, we specifically focus on user workflows in this work. Numerous questions concerning the origin of crashes need to be addressed. Indeed, we do not know if crashes result from individual actions or from combinations of actions. Similarly, we do not know if their order has an impact. Besides, while the occurrence of a crash is explicitly recorded, we are not aware of the action(s) that triggered it. Answering these questions is even more challenging given the very high variability in the user sequences leading to crashes. The issues raised here are of real industrial interest insofar as they correspond to concrete application cases. Indeed, it happens that users or some particular hospitals complain about too many crashes. It is therefore important to be able to understand and target the origin(s) of the problem, or even to be capable of reproducing crashes, based on their logs. Some works were proposed on software bug mining [Aggarwal and Han, 2014]. They are mainly based on the analysis of the source code to highlight buggy program regions based on software behavior graphs. Unlike these works, we have at our disposal user workflows and we would like to highlight the actions in the interface that led to the crash. Other approaches consist in the mining of frequent patterns in software revision history [Livshits and Zimmermann, 2005]. However, in this case, they made the assumption that error patterns might be represented by itemsets for which the order of the events composing the pattern is not taken into account. This is precisely one of our research problems. Indeed, we aim at determining which type of pattern is the most appropriate to represent crash signatures, and for example determine whether the order of the actions composing the pattern is important or not. As no off-the-shelf solution to our specific issue currently exists, we draw extensively on methodologies used in pattern based sequence classification [Bringmann et al., 2009] where the principle is to use patterns as features of a classification method. An essential step of this methodology is the selection of the features which have to be as discriminative as possible of their class. While some works use the Fisher score [Lo et al., 2009], other design specific selection scores. The earliness of the pattern might be taken into account [Xing et al., 2008], as well as the proximity of the events composing the pattern [Zhou et al., 2016]. We propose to use a binomial test to determine which of 4 types of patterns (taking into account or not order and proximity) are the most appropriate to highlight crash patterns.

Similarly, the improvement of software usability through customization and adaptation of systems to each user's specific needs requires a very good knowledge of how they use the software. This is the general goal of software application users profiling [Schiaffino and Amandi, 2009]. Explicit information provided by the user as well as implicit information obtained by the recording of users actions is used to build groups of users. The purpose of obtaining such profiles differs depending on the field of application. For example, adaptive systems aim at providing different content to different users [Cawsey et al., 2007], whereas e-commerce applications take advantage of user profiles to recommend products that the customer is more

likely to buy given his preferences [Adomavicius and Tuzhilin, 2001]. Beyond the creation of groups of similar users, we aim at highlighting usage trends, which might correspond to different categories of users or not. This valuable information helps to assess the need to maintain rarely used options, to rethink the content of the user interface or to simplify repetitive actions. While user knowledge is easy when it comes to a partner hospital, it is less obvious to have an overview of the habits of use or the characteristic workflow of a random site. The recent possibility of collecting workflow data from users around the world raises the question of the automation of information extraction at a larger scale to make business decisions based on data analysis. Several works were proposed in order to cluster navigational paths coming from web sessions [Chaofeng, 2009]. In a way very analogous to our problem, they seek to group sessions in which usage patterns are similar. As these sessions are of different lengths one of the main challenges relies in the vector representation of the sessions. The standard method consists in representing sessions as vectors containing the frequency of occurrences of each web page consulted in a session [Xu and Liu, 2010] and is equivalent to the bag-of-words representation in text mining, widely used to perform document classification or clustering [Hotho et al., 2005]. This representation suffers from data sparsity and high dimensionality and therefore, recent works proposed distributed representations of sentences or documents [Le and Mikolov, 2014]. We propose to apply these lower dimensional representations to cluster user sessions and thus determine characteristic usage workflows.

Proactive management is a way to generally enhance the user experience whether in anticipating interruptions or in adapting the user interface in real time. Failure prediction systems are nowadays widely used in various applications [Salfner et al., 2010]. They enable to improve the availability of the system by anticipating potential disruptions. They generally solve a classification problem and aim at predicting whether a failure will occur in the near future. In the same spirit, proactive user interfaces use real-time predictions to anticipate users' needs and provide them with more natural interfaces. The concept of adaptive interfaces has been studied over the last decades [Browne, 2016]. The main reasons for adaptation being that end-users are heterogeneous. Moreover, usability criteria are not the same for all individuals. While some changes might be comfortable for some users, they might not be for others. Similarly, different users might not have the same preferences. Adaptive systems aim at satisfying each user needs to consequently increase their operational speed. Another important purpose of such systems is to reduce the learning time before efficient work might be achieved. Well designed adaptive interface fulfill all these requirements avoiding confusion that might be caused by badly planned adaptation systems [Gajos et al., 2006]. In this work we exclusively focus on the accuracy of the prediction algorithms leaving the design aspects to the ergonomy experts. We address 2 distinct industrial issues, namely crash risk monitoring as well as next action prediction. To tackle both issues, we take advantage of the recurrent structures of Long Short Term Memory (LSTM) neural networks. Fisrt introduced by [Hochreiter and Schmidhuber, 1997], LSTM are particularly appropriate to handle sequential data.

The manuscript is organized in 2 parts as follows. The first part of the proposed methods

aims to provide information for a better understanding of the software use. The second part is dedicated to real-time monitoring methods.

Part I :

- Chapter I.1 describes the data as well as the general mathematical formalism that will be used in this work. To conduct our experiments, we have at our disposal logs coming from 50 hospitals around the world. All experiments will be conducted on the same type of data. Specific datasets will be used depending on the applications.
- Chapter I.2 presents the methodology proposed to highlight which type of pattern best represents crash signatures. 4 types of patterns will be tested, taking into account order and proximity of the actions composing them. We propose to use a binomial test to determine significant patterns. Experiments are conducted on the logs of 5 systems having an important crash rate. The efficiency of the method is also demonstrated on a currently critical system that is the subject of a complaint.
- Chapter I.3 tackles the issue of workflow characterization. We propose to compare 3 types of inputs representation of the user sessions in different clustering algorithms. A cluster validity index is used to determine which configuration provides the best partition. Experiments are conducted on 2 systems dominated by the use of a particular application, our goal being to highlight typical workflows among a particular medical specialty. A process mining visualization tool is used to visualize and interpret the obtained session clusters.

Part II :

- Chapter II.1 provides a general formalism of sequence learning as well as state of the art on recurrent neural networks which are particularly efficient for this task. We focused on Long Short Term Memory recurrent neural networks and present the architecture which will be used to tackle the crash risk monitoring task (see Chapter II.2) as well as the next action prediction task (see Chapter II.3).
- Chapter II.2 deals with the monitoring of crash risk. To address this task, we propose to feed the LSTM with feature vectors composed of significant actions contained in crashes. The method is compared to state of the art classification algorithms fed with significant actions as well as significant crash patterns for a fair comparison. Experiments are conducted on 5 systems having an important crash rate.
- Chapter II.3 tackles next action prediction. The predictive system might take the form of a dynamic toolbar containing in real-time the k next most likely actions the user is going to need. For this purpose we propose to feed the LSTM with action embeddings. The method is compared to state of the art sequence prediction algorithms and proves to have the best performances. For this reason, it is likely to be implemented in future versions of the software, and thus we tested our method on all 50 systems. Some additional experi-

ments are conducted in order to determine the best training strategy, including training sets obtained by explicit characteristics as well as training sets obtained by unsupervised techniques.

We finally conclude by underlying the research and industrial perspectives of this work.

Part I

Data Mining for the Analysis of Software Sessions

Chapter I.1

Data & Formalism

In this study, we investigate practitioner workflows which are recorded in log files. This chapter describes the content of these log files as well as the general formalism we introduced. We also present the database which will be used to conduct our experiments.

I.1.1 Action Log Files

In our logs, one session corresponds to the analysis of one patient's series of images. In other words, each session contains all the necessary actions for the radiologist to provide his medical diagnosis. During these sessions, the physician's actions in the user interface are chronologically recorded in log files, one session after the other. These log files can be recovered via remote access to hospitals around the world. Each log covers several months of activity and contains a variable number of sessions depending on the hospital's workload. Besides, it is important to mention that hospitals purchase licenses corresponding to their medical specialties, therefore there is a great variability in the applications used in our database. As illustrated in Figure I.1.1, each session can be characterized by several attributes listed below:

- Hospital: each session is linked to a hospital whose name and location are known.
- Nature: as will be explained in section I.1.2, our log system allows us to distinguish normal sessions with a clean exit of the application from crashes in which the session was prematurely interrupted.
- Logged actions: the interface is split into different zones, and for some of those the actions are not logged. We know which zones have their actions correctly logged and which don't have.

However, the current logs do not allow to record the following information:

- Dataset: we have no information concerning the acquisition modality of the studied images (Computed Tomography, Magnetic Resonance Imaging, 3D X-ray, Positron Emission Tomography and Positron Emission Tomography - Computed Tomography) or the metadata associated with the image (number of volumes, size, voxel size, etc).

- User: some sessions might come from generalist practitioners while others might come from very specialized practitioners in a particular field of medicine implying specific application licenses. It is also important to notice that one log file might come from a working station used by several users as well as from a station belonging to one user only. We do not have that information.
- Missing actions: as explained previously, we know that certain types of actions do not appear in the current logs.

The medical application usually launched at the beginning of a session for the image analysis strongly influences the workflow of the session. Indeed the content of the user interface changes according to the application used. In the current log system only a few number of application executions are properly logged under their own names (the others appear under a generic name). Thus, when the application is taken into account in the analysis, we focus on the applications which are properly logged.

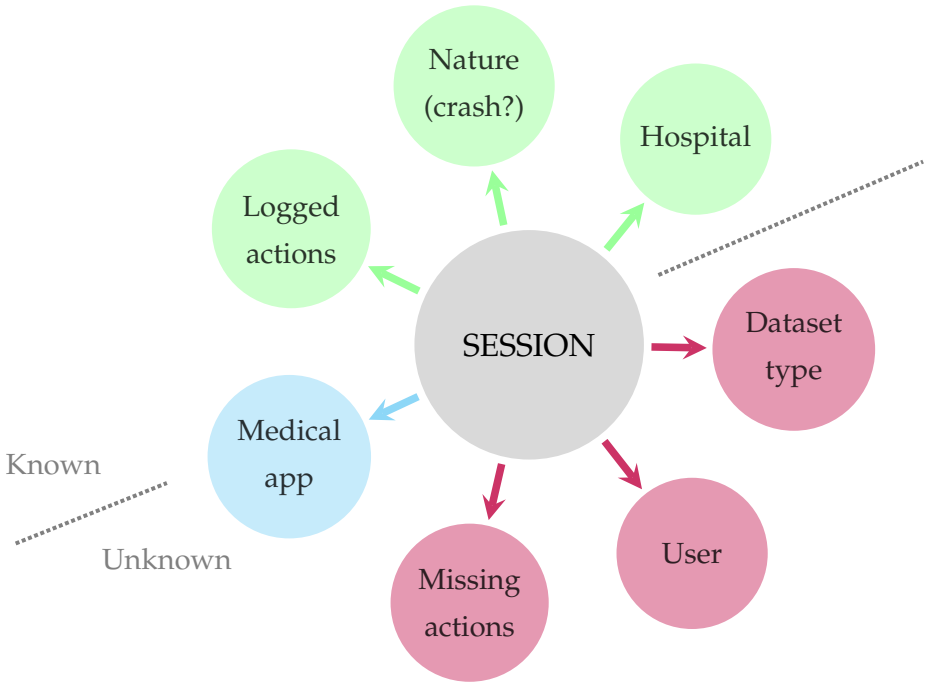

Figure I.1.1: ***User sessions characterization. Green information is known, red is not, blue is known in some particular cases.***

It should be noted that regulations in medical applications impose many privacy restrictions regarding patient data, thus making the retrieval of this type of data even more difficult. Indeed, confidential data protection requires relatively complex and time-consuming procedures to ensure that the recovery process of log files is completely secure and that they provide no critical information.

I.1.2 Formalism

The user mouse clicks recorded in log files refer to the corresponding function calls from the software source code. Therefore, the successive actions appear in the log files as strings of different lengths. To facilitate computations and analysis, we have introduced a formal language to normalize action representations into strings of the same length, see Table I.1.1.

Let Σ be an alphabet, that is to say a finite set of symbols, and D the dictionary assigning one user action to a symbol in Σ. A session of length n is an element of Σ^n. Let X be a session, it will thus be represented as a word on the alphabet Σ and will be denoted by $X = x_1x_2...x_n$, with each $x_i \in \Sigma$, an elementary user action.

In practice, considering the number of potential actions in the medical software of interest in this study, we choose without loss of generality to work with symbols composed of three letters, $\Sigma = \{\lll, \ggg, aaa, aba, ..., zzz\}$. The symbol $\lll$ stands for the beginning of a session, thus for all sessions $x_1 = \lll$. Likewise, $\ggg$ stands for a clean exit of the software application, that is to say the end of a session without crash. All the other symbols represent actions.

With these notations we can easily distinguish normal sessions from crashes: a session of length n is a normal session when $x_n = \ggg$, while if $x_n \neq \ggg$, it is a crash. The set of normal sessions will be denoted by $\mathcal{S}$ while $\mathcal{C}$ will represent the set of crash sessions. Finally, for a finite set Ω, $|\Omega|$ will denote its cardinal.

Σ	the finite set of symbols $\Sigma = \{\lll, \ggg, aaa, aba, ..., zzz\}$
D	the dictionary assigning one user action to a symbol of Σ
X	a session, a finite sequence of n symbols over Σ: $X = x_1x_2...x_n$ where $x_1 = \lll$ and $x_n = \ggg$ or $\lll$
$\mathcal{S}$	the set of normal sessions, where $x_1 = \lll$ and $x_n = \ggg$
$\mathcal{C}$	the set of crashes, where $x_1 = x_n = \lll$
$\|\Omega\|$	the cardinal of the finite set Ω
$\mathcal{D}$	the complete sequence database, and thus $\|\mathcal{D}\| = \|\mathcal{S}\| + \|\mathcal{C}\|$

***Table I.1.1:** Formalism.*

I.1.3 Database

To conduct our experiments we have at our disposal 50 log files, recorded in the same language, English in this case, coming from hospitals around the world equipped with working stations having the same software release. As can be seen in Table I.1.2, logs fulfilling the previous requirements, are mainly coming from India, the United States of America and Japan. Different subsets of the global database will be used to test the proposed solutions, depending on the purpose of the tested methods. Thus, logs with the highest crash rates will be used to test the methods related to crash analysis, whereas works dedicated on usability improvement will be tested on normal sessions only. Each solution will be tested on several systems in order

to ensure that the methods generalize well to all logs. Each dataset will be described in more detail in the experimental setup description of each experiment.

Country	Number of hospitals
INDIA	13
UNITED STATES OF AMERICA	8
JAPAN	5
EGYPT	4
KOREA (REPUBLIC OF)	4
POLAND	4
BELGIUM	2
ROMANIA	2
AUTRALIA	1
BANGLADESH	1
HUNGARY	1
IRELAND	1
NEPAL	1
SINGAPORE	1
SOUTH AFRICA	1
TURKEY	1
Total	50

*Table I.1.2: **Distribution of hospitals by country (50 log files).***

I.1.4 Logs Improvement

An important part of the initial work consisted in a precise inventory of the log file contents since these logs had never been analyzed before. Various improvements are continually being made to each new version of the software. For instance, the recording of keyboard shortcuts which constitute a significant part of the workflow of some users has only been added to the latest software versions. The time required for these improvements to be rolled out to customers and sufficiently used being too long, we were not able to work with the most advanced version of log files. We decided to work with previous logs with the advantage of having more sessions available. It is important to note that all the proposed methods in this thesis are completely adaptable to a richer log content and will certainly work all the better with the latter.

Chapter I.2

Crash Pattern Mining

A SPECIFIC PROBLEM potentially occurring while patient images are reviewed is a software crash during the practitioner workflow. This is all the more true in the case of interventional applications, for which patient safety is at stake. Therefore, reducing as much as possible the crash rate of medical applications is a major line of the continuous improvement effort regarding the global quality of medical software products. The motivation here is to detect crash signature to help the software development teams reproduce crashes and focus on specific program functions. Thus, in this chapter, we aim at determining the type of pattern which is the most appropriate to represent these signatures. We discuss the related work in section I.2.1. In section I.2.2.1, we define 4 types of patterns, according to 2 criteria: the order and the proximity of the actions composing them. We propose to use a binomial test, presented in section I.2.2.2, evaluating the number of occurrences of a crash pattern in crashes against its total number of occurrences in both crashes and normal sessions in order to determine which of these patterns are the most significant in crashes. Sequential pattern mining algorithms [Aggarwal and Han, 2014], well adapted to our problem, will be presented in sections I.2.3.1, I.2.3.2 and I.2.3.3. We conducted experiments on 5 systems having a particularly high crash rate to determine which type of pattern enables to best represent crash signatures. The results are given in section II.3.4.4. An application to a current investigation is presented in section I.2.5.

I.2.1 Research Problem

In addition to user actions in the software interface, several factors are likely to cause crashes: system status (number of running applications in the case of a server, memory availability), hardware characteristics or dataset type may be the root cause of crashes. However, because of the lack of consistent data, we exclusively focus on the user workflow in this thesis and make the assumption that crashes are due to user actions.

We aim at identifying the actions or the patterns of actions having a high probability to provoke crashes, in order to help development teams to focus on some specific functions of bug-prone actions. Furthermore these relevant crash signatures might be integrated to automatic tests in order to ensure the quality of the new software releases.

This issue might in the first instance be tackled using techniques that have been proposed for anomaly detection [Agrawal and Agrawal, 2015] or intrusion detection systems using signature detection systems [Patcha and Park, 2007]. A traditional signature detection system [Axelsson, 2000] aims at identifying an intrusion irrespective of any knowledge of what the normal situation looks like and requires a signature to be defined for all types of intrusions, making the detection of an unknown attack impossible and being therefore inapplicable in our case. On the contrary, anomaly detection systems compare activities against a normal behavior thus enabling to detect anomalies that have never been seen before. For instance, a database of normal short sequences of system calls may be built and an alarm triggered when the percentage of mismatch with a new sequence exceeds a certain threshold, considering it as an anomaly [Forrest et al., 1996, Hofmeyr et al., 1998]. These techniques aim at detecting abnormal behaviors in order to prevent them from harming systems in real time. In the same spirit we will propose in Chapter 6 a method enabling to anticipate crashes during the software use. The first goal here being to find the root causes of crashes, we took inspiration from pattern mining techniques [Aggarwal and Han, 2014].

The problem of frequent pattern mining has been widely studied and applied to a large variety of domains [Han et al., 2007]. Some works have in particular been proposed on software bug mining, see Chapter 18, Section 11 from [Aggarwal and Han, 2014]. One of the proposed approach is dedicated to the detection of non crashing bugs which are difficult to locate since no crashing point, hence no backtrace, is available [Liu et al., 2005]. Using software behavior graphs of program and classification, it is possible to detect suspicious code regions potentially containing logical errors. Always with the same objective, a method proposing to mine edge-weighted call graph was proposed to take into account the call frequency [Eichinger et al., 2008]. Both methods tackle the bug detection task using graph mining techniques, which are widely used for this task, the control flow of programs being often modeled in the form of call graphs, see Chapter 2, Section 4.3 from [Aggarwal and Wang, 2010].

Unlike these works, we have at our disposal user workflows that led to crashes coming from systems in use. The difficulty is to detect which patterns triggered the crashes, these combinations of actions being a priori very complex since they were not detected during the verification phase before the software deployment. The very high variability both in terms of content and of length of sessions ending with a crash (see section I.2.3) leads to a large number of questions. Indeed, we do not know if crashes result from individual actions or from combinations of several actions. In the case of several actions, we do not know their number or whether their order has an impact. Moreover, we do not know if their proximity has to be taken into account [Adam et al., 2016]. Similarly, we have no information about the exact moment in the session at which the crash is triggered.

In view of our problem, we draw extensively on methodologies used in pattern based sequence classification [Bringmann et al., 2009]. The general principle is to use patterns as features which can then be used as inputs of conventional classifiers [Cheng et al., 2007, Lo et al., 2009] or for

rule-based classification [Xing et al., 2008, Zhou et al., 2016]. The success of these methods generally relies on the step of patterns selection. Indeed, these patterns have to be discriminative and representative of the class to which they belong. As a large number of patterns is generally generated from the mining step, the selection might be based on the Fisher score [Duda et al., 2012]. Popularly used to evaluate the discriminative power of a feature, this score is defined as:

$$Fr = \frac{\sum_{i=1}^{c} n_i(\mu_i - \mu)^2}{\sum_{i=1}^{c} n_i \sigma_i^2}$$

where n_i is the number of data samples in class i, μ_i is the average feature value in class i, σ_i is the standard deviation of the feature value in class i and μ is the average feature value in the whole dataset. The Fisher score may be combined with a coverage threshold to select patterns [Cheng et al., 2007, Lo et al., 2009]. In this case, the coverage corresponds to the number of times each training instance is covered by the selected features. Another methodology proposes to evaluate the interestingness of a pattern [Zhou et al., 2016] taking into account its support (the number of sequences in which it appears – see section I.2.2.1) and its cohesion (measuring how close from each other the actions composing the pattern are – see section I.2.2.1). The earliness of features might also be taken into account to select patterns for early prediction [Xing et al., 2008].

I.2.2 Frequent Patterns

I.2.2.1 Types of Patterns

Let $X = x_1 x_2 \dots x_n$ be a sequence of n actions. We define 4 types of patterns of size k (containing k symbols) and their inclusion relationships into X. Let $\alpha = (\alpha_1, \dots, \alpha_k) \in \Sigma^k$ be a pattern.

k-itemset

We say that α is a k-itemset of X if for all $1 \leq i \leq k$, $\alpha_i \in X$. It simply implies that all symbols of the itemset belong to X, irrespective of the order. We will denote a k-itemset by $\{\alpha_1, \dots, \alpha_k\}$.

k-exact subsequence

We say that α is a k-exact subsequence of X if there exists i such that $\alpha_1 \alpha_2 \dots \alpha_k = x_i x_{i+1} \dots x_{i+k-1}$, that is to say that X contains the exact subsequence $\alpha_1 \alpha_2 \dots \alpha_k$. We will denote a k-exact subsequence by $\alpha_1, \dots, \alpha_k$.

k-cohesive subsequence

Let W be the sliding window size. We say that α is a k-cohesive subsequence of X if there exists $i, i_1, i_2, \dots, i_{k-1}$, with $i_j < W$ for all $1 \leq j \leq k-1$, such that:

$$\alpha_1 \alpha_2 \dots \alpha_k = x_i x_{i+i_1} x_{i+i_1+i_2} \dots x_{i+i_1+i_2+\dots+i_{k-1}}$$

Here, it means that X contains the sequence $\alpha_1 \alpha_2 \dots \alpha_k$, but the consecutive terms α_i, α_{i+1} may

be separated by up to $G = W - 2$ symbols, where G denotes the maximal gap allowed. We will denote a k-cohesive subsequence by $\langle \alpha_1 \alpha_2 \dots \alpha_k \rangle^G$.

For example, if $X_1 = \alpha_1\alpha_2\alpha_3$, $X_2 = \alpha_1\beta\alpha_2\delta\alpha_3$, $X_3 = \alpha_1\beta\gamma\alpha_2\delta\alpha_3$ and $X_4 = \alpha_1\beta\gamma\rho\alpha_2\delta\alpha_3$, the 3-cohesive subsequence with gap $G = 2$, $\langle \alpha_1\alpha_2\alpha_3 \rangle^2$, is included in X_1, X_2 and X_3 but not in X_4 since the maximal gap authorized between α_1 and α_2 is exceeded. Note that a cohesive subsequence of gap equal to zero simply corresponds to an exact subsequence.

k-subsequence

We say that α is a k-subsequence of X if there exist integers $1 \leq i_1 < i_2 < \cdots < i_k \leq N$ such that $\alpha_1 = x_{i_1}, \alpha_2 = x_{i_2}, \dots, \alpha_k = x_{i_k}$. In this case the order is taken into account, but two consecutive symbols of the subsequence might be separated by an indeterminate number of actions. We will denote a k-subsequence by $\langle \alpha_1 \alpha_2 \dots \alpha_k \rangle$.

Subpattern and Superpattern

If pattern α is included in pattern β, $\alpha \subset \beta$, and $\alpha \neq \beta$, pattern α is a *subpattern* of pattern β and pattern β is a *superpattern* of pattern α.

Support

Given a dataset composed of $|\mathcal{D}|$ sequences in total, the *support* of a pattern α, denoted $\sup(\alpha)$, is equal to the number of sequences in which it is contained. A pattern α, will be considered as *frequent* if it is contained in at least $\theta|\mathcal{D}|$ sequences, where $0 < \theta < 1$ is a user specified minimum support threshold which will be denoted by $\min_{\text{sup}}$:

$$\alpha \text{ frequent} \iff \sup(\alpha) \geq \min_{\text{sup}} |\mathcal{D}|.$$

Frequent Closed Pattern

Given a mininum support threshold $\min_{\text{sup}}$ and a dataset containing $|\mathcal{D}|$ sequences. If pattern α is frequent ($\sup(\alpha) \geq \min_{\text{sup}} |\mathcal{D}|$) and there exists no proper superpattern of α with the same support, i.e., $\nexists\, \beta$ such that $\alpha \subset \beta$ and $\sup(\alpha) = \sup(\beta)$, α is a *frequent closed pattern*.

Cohesion

A strict measure of the cohesion of a subsequence is given by the maximum gap allowed between two consecutive terms in the pattern, as defined in section I.2.2.1.

Another concept to evaluate the cohesion of a pattern was proposed in [Zhou et al., 2016] and measures the proximity of the events that make up the pattern. The length of the shortest interval of a subsequence α in a sequence X is defined as $W(\alpha, X) = \min\{i_e - i_s + 1 | i_s \leq i_e$ and α is a subsequence of $X_{i_s,i_e} = x_{i_s} \dots x_{i_e}\}$. The average shortest interval over a dataset $\mathcal{D}$ is then equal to:

$$\overline{W(\alpha)} = \frac{\sum_{X \in \mathcal{D}_\alpha} W(\alpha, X)}{|\mathcal{D}_\alpha|}$$

where $\mathcal{D}_\alpha$ denotes the subset of sequences of $\mathcal{D}$ containing the subsequence α and thus $|\mathcal{D}_\alpha|$, its cardinal. The cohesion of a pattern α wihtin the dataset $\mathcal{D}$ is computed as follows:

$$C(\alpha) = \frac{|\alpha|}{W(\alpha)}$$

where $|\alpha|$ stands for the length of the pattern α, corresponding to the number of actions in α. With this definition, $C(\alpha) \in [0, 1]$. Patterns having a cohesion close to 1 are highly cohesive. A cohesion close to 0 corresponds to patterns largely spread throughout the sequences.

I.2.2.2 Significant Patterns

To detect whether a pattern has a significant impact on causing crashes, we perform a binomial test. Let α be a pattern and Z_α the random variable equal to 1 if a session containing pattern α crashes and to 0 otherwise:

$$Z_\alpha = \begin{cases} 1, & \text{if crash} \\ 0, & \text{otherwise} \end{cases}$$

Z_α has a Bernoulli distribution with success parameter p_α, the theoretical probability of crash for a session containing α:

$$Z_\alpha \sim \mathcal{B}(p_\alpha).$$

We want to detect patterns for which the crash probability is significantly higher than the expected crash probability in the whole database, estimated by p_0:

$$p_0 = \frac{|\mathcal{C}|}{|\mathcal{C}| + |\mathcal{S}|}$$

where $|\mathcal{C}|$ and $|\mathcal{S}|$ stand for the total number of crashes and the total number of normal sessions respectively. Let:

- $\mathcal{C}_\alpha$ denotes $\{X \in \mathcal{C} \mid \alpha \subset X\}$ and $|\mathcal{C}_\alpha|$ the number of crash sessions containing the pattern α in the database,
- $\mathcal{S}_\alpha$ denotes $\{X \in \mathcal{S} \mid \alpha \subset X\}$ and $|\mathcal{S}_\alpha|$ the number of normal sessions containing the pattern α in the database,
- $\mathcal{D}_\alpha$ denotes $\{X \in \mathcal{D} \mid \alpha \subset X\}$ and $|\mathcal{D}_\alpha|$ the total number of sessions containing the pattern α in the database, $|\mathcal{D}_\alpha| = |\mathcal{C}_\alpha| + |\mathcal{S}_\alpha|$.

We consider that $|\mathcal{C}_\alpha|$ follows a binomial distribution of parameters $|\mathcal{D}_\alpha|$ and p_α:

$$|\mathcal{C}_\alpha| \sim \mathcal{B}(|\mathcal{D}_\alpha|, p_\alpha).$$

We consider the following hypotheses, where $\widetilde{p}$ is a threshold probability of crash:

$$\begin{cases} \mathcal{H}_0 : p_\alpha \leq \widetilde{p}; \\ \mathcal{H}_1 : p_\alpha > \widetilde{p}. \end{cases}$$

We take $\widetilde{p} = rp_0$, r depending on the dataset and for each pattern, we will test the null hypothesis by calculating the associated p-value:

$$p\text{-}value = \mathbb{P}_{\mathcal{H}_0}(|\mathcal{C}_\alpha| > |\mathcal{C}_\alpha^{observed}|).$$

Should the p-value exceed a given level τ (let's take $\tau = 1\%$). Otherwise, we might reject the null hypothesis, and conclude that p_α is greater than $\widetilde{p}$.

This representation enables to take into account both the crash probability of a pattern and its number of occurrences. We want to give more importance to patterns occurring several times with a moderate crash probability and exclude patterns occurring only very few times and having a very high crash probability. Indeed, evaluating $\hat{p}_\alpha$ as $|\mathcal{C}_\alpha| / |\mathcal{D}_\alpha|$ is not significant in many situations for which $|\mathcal{D}_\alpha|$ is too low.

I.2.3 Mining Algorithms

In this section we introduce the mining algorithms that will be used to conduct our experiments. The Closet algorithm will be used to mine frequent closed itemsets and the Bide algorithm to mine frequent closed subsequences. Cohesive subsequences might be mined with Gap-Bide, however, as will be explained later, our methodology does not require to mine this type of patterns. For this reason, we will only briefly describe Gap-Bide and we will detail Closet and Bide. Table I.2.1 summarizes the mining algorithm that might be used to mine each type of pattern.

Type of pattern	Mining Method
k-itemset	Closet [Pei et al., 2000]
k-exact subsequence	Gap-Bide [Li and Wang, 2008]
k-cohesive subsequence	
k-subsequence	Bide [Wang and Han, 2004]

***Table I.2.1:** Type of pattern and associated mining method.*

I.2.3.1 Closet

First introduced by [Agrawal et al., 1994], frequent itemset mining was developed for market basket analysis and customers buying habits analysis. By analyzing transactions, the objective was to find out which items are often purchased together. Many algorithms were proposed, to name a few: Apriori [Agrawal et al., 1994], FP-growth [Han et al., 2000] and Eclat [Zaki, 2000]. They all rely on the Apriori property: *a k-itemset is frequent only if all of its subitemsets are*

frequent (pattern α frequent $\iff \sup(\alpha) \geq \min_{\text{sup}} |\mathcal{D}|$*)*. FP-growth mines the frequent itemsets without candidate generation using a divide and conquer strategy. The first step consists in mining frequent itemsets of length 1 by scanning the dataset once. The search space is then partitioned in projected databases, one for each frequent 1-subitemset. Each database is then mined recursively. This method thus decomposes the complete dataset into much smaller sets, which makes it highly efficient. A major challenge of pattern mining is usually the huge number of frequent patterns, especially when the mininum support threshold is set low. Indeed, a frequent itemset contains a combinatorial number of frequent smaller subitemsets. To get around this phenomenon, closed frequent itemset mining algorithms were proposed such as AClose [Pasquier et al., 1999] or Closet [Pei et al., 2000] which is described below.

First of all, the notions of frequent item list and conditional database which are necessary to understand the algorithm will be defined. The principle of the algorithm is then provided and illustrated on an example.

Frequent Item List

Given a transaction database T, and a minimum support threshold $\min_{\text{sup}}$, the *frequent item list* denoted f_list corresponds to all frequent items in support descending order.

Conditional Database

Given a transaction database T and i, a frequent item in T, the *i-conditional database*, denoted $T|_i$ corresponds to the subset of transactions containing i, with all the infrequent items, item i and items following i in the f_list omitted.

Let X be a frequent itemset and j be a frequent item in X-conditional database, $T|_X$. The *jX-conditional database*, denoted $T|_{jX}$, is the subset of transactions in $T|_X$ containing j such that all local infrequent items (in X-conditional database), item j, and items following j in local frequent item list, f_list_X are omitted.

Algorithm

Given a transaction database T and a minimum support threshold $\min_{\text{sup}}$, the Closet algorithm (see Algorithm 1) outputs the complete set of frequent closed itemsets, FCI.

The set of frequent closed itemsets FCI is initialized to an empty set. The database is scanned to compute the frequent item list f_list. The *CLOSET* subroutine, takes as arguments: a frequent itemset, X, a transaction database, DB, a frequent item list f_list and the complete set of frequent closed itemsets, FCI. It is called for the first time with $X = \emptyset$, $DB = T$, f_list and FCI as inputs.

The subroutine is composed of 3 steps. It first tests if there exists a set of items in f_list occurring in all the transactions of DB. If so, this set, denoted Y is added to the current frequent itemset X to form a new itemset, $X \cup Y$. If this itemset is not a proper subitemset of any itemset in FCI with the same support, $X \cup Y$ is closed and therefore, added to FCI.

The remaining items from f_list are then used to compute corresponding conditional databases.

For each remaining item i from $\mathrm{f\_list}$ (in $\mathrm{f\_list} \setminus Y$), starting by the last one, if iX (the set composed of item i and items from X) is not a subitemset of any frequent closed itemset from FCI with the same support, the subroutine *CLOSET*(iX, $DB|_i$, $\mathrm{f\_list}_i$, FCI) is called, where $DB|_i$ corresponds to the i-conditional database with respect to DB and $\mathrm{f\_list}_i$ is the corresponding frequent item list.

Algorithm 1: Closet [Pei et al., 2000]

Inputs: Transaction database T and minimum support threshold $\mathrm{min_{sup}}$.

Output: The complete set of frequent closed itemsets, FCI.

Initilization:

1. Initialize FCI to $\emptyset$;
2. Scan database and compute frequent item list $\mathrm{f\_list}$;
3. Call CLOSET($\emptyset$, T, $\mathrm{f\_list}$, FCI).

Subroutine CLOSET(X, DB, $\mathrm{f\_list}$, FCI)

Arguments:

- X : the frequent itemset if DB is an X-conditional database, or $\emptyset$ if DB is T;
- DB : transaction database;
- f_list : frequent item list of DB;
- FCI : the set of frequent closed itemsets already found.

Method:

1. Let Y be the set of items in $\mathrm{f\_list}$ such that they appear in all the transactions of DB, add $X \cup Y$ to FCI if it is not a proper subitemset of some itemset in FCI with the same support;
2. Form conditional database for every remaining item in $\mathrm{f\_list}$ ($\mathrm{f\_list} \setminus Y$) and compute local frequent item lists for these conditional databases, if there is no remaining item in $\mathrm{f\_list}$, test if X is a closet frequent itemset;
3. For each remaining item i in $\mathrm{f\_list}$ starting from the last one, call CLOSET(iX, $DB|_i$, $\mathrm{f\_list}_i$, FCI) if iX is not a subitemset of any frequent closed itemset already found with the same support count. $DB|_i$ corresponds to the i-conditional database with respect to DB and $\mathrm{f\_list}_i$ is the corresponding frequent item list.

Given the example transaction database T in Table I.2.2 and a minimum support $\mathrm{min_{sup}}$ equal to 0.4, itemsets have to occur in at least 2 transactions to be frequent.

Transaction ID	Transaction
1	C, E, F, A, D
2	E, A
3	C, E, F
4	C, F, A, D
5	C, E, F

***Table I.2.2:** An example transaction database.*

The first step of the Closet algorithm consists in finding frequent 1-itemsets to determine the frequent item list f_list. In the running example f_list $= (C:4, E:4, F:4, A:3, D:2)$.

As there is no set of items in f_list appearing in all transactions of T, FCI remains empty.

The next step consists in the construction of conditional databases of the remaining items in f_list (in the running example, all items remain). We construct 5 conditional databases containing 5 non-overlap subsets:

- the ones containing item D (denoted $T|_D$);
- the ones containing item A but not D (denoted $T|_A$);
- the ones containing item F but not A, nor D (denoted $T|_F$);
- the ones containing item E but not F, A nor D (denoted $T|_E$);
- the ones containing only item C (denoted $T|_C$).

This results in the conditional databases provided in Table I.2.3.

$T\|_D$	$T\|_A$	$T\|_F$	$T\|_E$
C, E, F, A	C, E, F	C, E	C
C, F, A	E	C, E	C
	C, F	C	C
		C, E	
f_list$_D = (C:2, F:2, A:2)$	f_list$_A = (C:2, E:2, F:2)$	f_list$_F = (C:4, E:3)$	f_list$_E = (C:3)$

***Table I.2.3:** Conditional databases and corresponding frequent item lists for the running example.*

Starting from the last item in f_list, D in the example, we apply the subroutine *CLOSET*(D, $T|_D$, f_list$_D$, FCI). As $\{C, F, A\}$ appears in all the transactions of $T|_D$, we form $D \cup \{C, F, A\} = \{C, F, A, D\}$ which has a support of 2 and add it to FCI ($FCI = (\{C, F, A, D\} : 2)$). As there is no remaining item in f_list$_D$, there is no need to compute further conditional databases for $T|_D$.

The next item in f_list is item A, we apply the subroutine *CLOSET*(A, $T|_A$, f_list$_A$, FCI). As no set of items appears in all the transactions of $T|_A$, we form $A \cup \emptyset = \{A\}$. $\{A\}$ is a subitemset of $\{C, F, A, D\} \subset FCI$ but its support is equal to 3, whereras the support of $\{C, F, A, D\}$ is equal to 2, therefore $\{A\}$ is added to FCI, ($FCI = (\{C, F, A, D\} : 2, \{A\} : 3)$). Items C, E and F are remaining in f_list$_A$, meaning that $DB|_A$ can be further partitioned into three conditional databases ($T|_{FA}$, $T|_{EA}$, $T|_{CA}$). As $\{F, A\}$ is a subitemset of $\{C, F, A, D\}$ and has the same support, there is no frequent closed itemset containing F, A without containing D. There is no need to call CLOSET(FA, $DB|_{FA}$, f_list$_{FA}$, FCI). As $\{E, A\}$ is not a subitemset of any frequent closed itemset, we call CLOSET(EA, $T|_{EA}$, f_list$_{EA}$, FCI) with $T|_{EA} = C$ and f_list$_{EA} = \emptyset$.

As there is no item in f_list_{EA}, we form $\{E, A\} \cup \emptyset = \{E, A\}$, which is not a subitemset of any itemset from FCI, therefore, we add $\{E, A\}$ to FCI, ($FCI = (\{C, F, A, D\} : 2, \{A\} : 3, \{E, A\} : 2)$).

As there is no item in f_list_{EA}, the $T|_{EA}$ cannot be extented. We move to item F from f_list and call *CLOSET*(F, $T|_F$, f_list_F, FCI). C appears in all transactions from $T|_F$, we form $\{C, F\}$, which is included in $\{C, F, A, D\}$, but has a higher support, therefore we can add $\{C, F\}$ to FCI, ($FCI = (\{C, F, A, D\} : 2, \{A\} : 3, \{E, A\} : 2, \{C, F\} : 4)$). Item E remains in f_list_F, after constructing the conditional database $T|_{FE}$ we call subroutine CLOSET(EF, $T|_{EF}$, f_list_{EF}, FCI) and add $\{C, E, F\}$ to FCI, ($FCI = (\{C, F, A, D\} : 2, \{A\} : 3, \{E, A\} : 2, \{C, F\} : 4, \{C, E, F\} : 3)$).

The last step consists in the call of CLOSET(E, $T|_E$, f_list_E, FCI). C is included in all transactions of $T|_E$, but $\{C, E\}$ which has a support of 3 is a subitemset of $\{C, E, F\}$ and thus is not closed, however $\{E\}$ is a closed frequent itemset, we add it to FCI, ($FCI = (\{C, F, A, D\} : 2, \{A\} : 3, \{A, E\} : 2, \{C, F\} : 4, \{C, E, F\} : 3, \{E\} : 4)$).

Finally, there is no transaction in $T|_C$. C has a support of 4, but is a subitemset of $\{C, F\}$ which has the same support, therefore there is no frequent closed itemset containing only C.

The final set of frequent closed itemsets is $(\{C, F, A, D\} : 2, \{A\} : 3, \{A, E\} : 2, \{C, F\} : 4, \{C, E, F\} : 3, \{E\} : 4)$.

I.2.3.2 Bide

Several algorithms have been proposed to mine sequential patterns [Han et al., 2007] when there is no constraint regarding the number of actions between two consecutive terms of the pattern. Only the order of the actions matters. One of the most well-known methodologies consists in pattern-growth used by the PrefixSpan algorithm [Pei et al., 2004] for example which uses the pattern growth method. As for frequent itemsets mining, closed frequent subsequence mining algorithms were proposed such as CloSpan [Xifeng et al., 2003] and later Bide [Wang and Han, 2004]. Instead of returning all the frequent subsequences such as PrefixSpan, they return the complete set of frequent closed subsequences, defined previously.

We decided to mine frequent subsequences with the Bide algorithm because of its efficiency. Based on the pattern-growth method, Bide grows prefixes using a BI-Direction Extension pattern closure checking mechanism to mine frequent closed subsequences. In order to describe the algorithm provided in Algorithm 2, we recall a few definitions provided in the Bide original paper.

Sequence ID	Sequence
1	$CAABC$
2	$ABCB$
3	$CABC$
4	$ABBCA$

***Table I.2.4:** An example sequence database.*

Table I.2.4 shows an input sequence database, we set $\min_{\text{sup}}$ to 0.5, meaning that subsequences have to appear in at least 2 sequences to be frequent.

We will call prefix a subsequence of interest.

First instance of a prefix

Given a sequence X which contains a prefix of length 1, $\alpha = \alpha_1$, the exact-subsequence of X starting at the beginning of X to the first appearance of item α_1 in X is called the first instance of prefix α in X. Recursively, the first instance of a prefix of length $(k+1)$, $\alpha_1\alpha_2\ldots\alpha_k\alpha_{k+1}$ can be defined from the prefix $\alpha_1\alpha_2\ldots\alpha_k$ (where $k \geq 1$) as the exact-subsequence from the beginning of X to the first appaearance of item α_{k+1} which also occurs after the first instance of the prefix $\alpha_1\alpha_2\ldots\alpha_k$.

In the running example, provided in Table I.2.4, the first instance of AB in $CAABC$ is $CAAB$. The first instance of CB in $CAABC$ is also $CAAB$.

Projected sequence and projected database

Given a sequence X, containing a prefix $\alpha = \alpha_1\alpha_2\ldots\alpha_k$. The projected sequence X^α with respect to α corresponds to the remaining part of X after removing the first instance of α.

In the running example, the projected sequence of subsequence AB in sequence $CAABC$ is C.

Given a sequence database $\mathcal{D}$, the projected database $\mathcal{D}^\alpha$ with respect to prefix α corresponds to the complete set of projected sequences with respect to α.

In the running example, the projected database of prefix AB, denoted by $\mathcal{D}^{AB}$, is (C, CB, C, BCA).

Locally frequent items

The frequent items of a projected database $\mathcal{D}^\alpha$ of a prefix α, are called *locally frequent*. They have to be ordered in a lexicographical order, we take the alphabetical order.

In the running example, if the prefix is AB, its projected database is (C, CB, C, BCA) and $\{B:2, C:4\}$ is the set of locally frequent items.

According to the *Apriori* property, in order to find frequent subsequences, a prefix only needs to be grown using the set of its locally frequent items.

BI-Directional Extension closure checking scheme

Given a non-closed sequence $\alpha = \alpha_1\alpha_2 \dots \alpha_k$, there must exist at least one event x' which can extend α to form a new sequence α' having the same support. There are 3 possible extensions:

1. $\alpha' = \alpha_1\alpha_2 \dots \alpha_k x'$
2. $\alpha' = \alpha_1\alpha_2 \dots \alpha_i x' \alpha_{i+1} \dots \alpha_k$
3. $\alpha' = x'\alpha_1\alpha_2 \dots \alpha_k$

In the first case, x' is called a *forward extension* item and α' a forward extension sequence. In cases 2 and 3, x' occurs before α_k and thus is called a *backward extension* item and α' a backward extension sequence. When a new frequent prefix is obtained, pattern closure checking needs to be done in order to ensure that it is closed. Bide performs superpattern and subpattern checking using respectively forward extension and backward extension event checking and it is demonstrated that *if there is no forward extension item nor backward extension item with respect to a prefix α, this prefix α is closed, otherwise it is non-closed*.

Last instance of a prefix

Given a sequence X containing a prefix $\alpha = \alpha_1\alpha_2 \dots \alpha_k$, the last instance of α in X is the exact-subsequence from the beginning of X to the last appearance of the last item of α.

i-th last-in-last appearance with respect to a prefix

Given an input sequence X containing a prefix $\alpha = \alpha_1\alpha_2 \dots \alpha_k$, the i-th last-in-last appearance of α in X, denoted by LL_i is recursively defined as:

- the last appearance of α_i in the last instance of α in X, if $i = k$;
- the last appearance of α_i in the last instance of α in X while LL_i must appear before LL_{i+1}, if $1 \leq i < k$.

The i-th maximum period of a prefix

Given an input sequence X containing a prefix $\alpha = \alpha_1\alpha_2 \dots \alpha_k$, the i-th maximum period of α in X is defined as the exact-subsequence between the end of the first instance of prefix $\alpha_1\alpha_2 \dots \alpha_{i-1}$ in X and the i-th last-in-last appearance with respect to α, if $1 < i \leq k$. It is the exact-subsequence located before the 1st last-in-last appearance with respect to prefix α, if $i = 1$.

For example, given a prefix ABC and a sequence $X = C_1A_1A_2BC_2DA_3C_3E$. The last instance of ABC is $C_1A_1A_2BC_2DA_3C_3$. We can then compute:

- the 1st last-in-last appearance LL_1 which is equal to A_2;
- the 2nd last-in-last appearance LL_2 which is equal to B;
- the 3rd last-in-last appearance LL_3 which is equal to C_3.

Beside, we can compute the following maximum periods:

- the 1st maximum period MP_1 which is equal to C_1A_1;
- the 2nd maximum period MP_2 which is equal to A_2;
- the 3rd maximum period MP_3 which is equal to C_2DA_3.

Backward extension items

Given a prefix $\alpha = \alpha_1\alpha_2 \dots \alpha_k$. If $\exists i, 1 \leq i \leq k$, and item x' occurs in each of the i-th maximum periods of the prefix α in $\mathcal{D}$, item x' is a backward extension item.

The i-th last-in-first appearance with respect to a prefix sequence

Given an input sequence X containing a prefix $\alpha = \alpha_1\alpha_2 \dots \alpha_k$, the i-th last-in-first appearance of α in X is denoted as LF_i and recursively defined as:

- the last appearance of α_i in the first instance of α in X, if $i = k$;
- the last appearance of α_i in the first instance of α in X while LF_i must appear before LF_{i+1}, if $1 \leq i < k$.

i-th semi maximum period of a prefix

Given an input sequence X containing a prefix $\alpha = \alpha_1\alpha_2 \dots \alpha_k$, the i-th semi maximum period of α is defined as:

- the exact-subsequence between the end of the first instance of $\alpha_1\alpha_2 \dots \alpha_{i-1}$ in X and the i-th last-in-first appearance with respect α, if $1 < i \leq k$;
- the exact-subsequence in X located before the 1st last-in-first appearance with respect to α, if $i = 1$.

For example, given a prefix ABC and a sequence $X = C_1A_1A_2BC_2DA_3C_3E$. We have:

- the 1st last-in-first appearance LF_1 which is equal to A_2;
- the 2nd last-in-first appearance LF_2 which is equal to B;
- the 3rd last-in-first appearance LF_3 which is equal to C_2.

Beside, we can compute the following semi-maximum periods:

- the 1st semi maximum period SMP_1 which is equal to C_1A_1;
- the 2nd semi maximum period SMP_2 which is equal to A_2;
- the 3rd semi maximum period SMP_3 which is equal to $\emptyset$.

Backscan

The Backscan method enables to stop growing unecessary patterns if the current subsequence can not be closed.

Given a prefix $\alpha = \alpha_1\alpha_2 \dots \alpha_k$. If $\exists i, 1 \leq i \leq k$, and there exists an item x' which occurs in each of the i-th semi maximum periods of the prefix α in $\mathcal{D}$, the growth of α can be stopped.

In the running example, if $\alpha = B$, item A appears in each of the 1st semi-maximum period of α in the database, meaning we can stop mining frequent closed subsequences with subsequence B. However, if $\alpha = C$, there is no item which appears in each of the 1st semi-maximum periods of α, meaning we can not stop growing C.

	LF_1	SMP_1
$CAABC$	B	CAA
AB_1CB_2	B_1	A
$CABC$	B	CA
AB_1B_2CA	B_1	A

***Table I.2.5:** Last-in-first appearances and semi maximum periods of B in the database.*

	LF_1	SMP_1
C_1AABC_2	C_1	-
$ABCB$	C	AB
C_1ABC_2	C_1	-
$ABBCA$	C	ABB

***Table I.2.6:** Last-in-first appearances and semi maximum periods of C in the database.*

Algorithm

Algorithm 2 presents the Bide algorithm. The first step is to find the subsequences of length 1 that are frequent. A projected database $\mathcal{D}^{f1}$ is then built for each frequent 1-subsequence, $f1$. Each frequent 1-subsequence is then considered as a prefix and the Backscan method checks if it can be pruned. If not, the number of backward extensions items BEI is computed and the subroutine *BIDE* is called with the projected database $\mathcal{D}^{f1}$, the prefix $f1$, $\min_{\text{sup}}$, the backward extensions BEI and the set of frequent closed sequences FCS as arguments. This subroutine calls itself recursively and operates as follows. Projected database $\mathcal{D}^{\alpha}$ of prefix α is scanned to find its locally frequent items denoted LFI. Among these items, it then computes the number of forward extensions items FEI (having the same support as α in $\mathcal{D}$). If there is no forward extensions items nor backward extensions items BEI, α is a closed subsequence and is added to the set of frequent closed subsequences FCS. Each locally frequent item x' is then used to grow the prefix α to form a new prefix, denoted by $\alpha x'$, whose projected database $\mathcal{D}^{\alpha x'}$ is built. For each new prefix $\alpha x'$, the Backscan methods checks if it can be pruned. If not, the number of backward extensions items BEI is computed and the subroutine *BIDE* is called on the new projected database $\mathcal{D}^{\alpha x'}$, the newly formed prefix $\alpha x'$, $\min_{\text{sup}}$, backward extensions BEI and updated set of frequent closed sequences FCS.

To compute the frequent closed subsequences in the running example, we first find the frequent subsequences of length 1: $F1 = (A : 4, B : 4, C : 4)$. Table I.2.7 provides for each frequent subsequence of $F1$ its projected database.

$\mathcal{D}^A$	$\mathcal{D}^B$	$\mathcal{D}^C$
ABC	C	$AABC$
BCB	CB	B
BC	C	ABC
$BBCA$	BCA	A

***Table I.2.7:** Projected databases $\mathcal{D}^A$, $\mathcal{D}^B$ and $\mathcal{D}^C$.*

The first prefix to analyze is A. As there is no possible backscan for A in $\mathcal{D}^A$, we compute the number of backward extensions, BEI, which is equal to 0 and call $BIDE(\mathcal{D}^A, A, \min_{\text{sup}}, BEI, FCS)$, with $FCS = \emptyset$. The locally frequent items from $\mathcal{D}^A$ are $\{A : 4, B : 4, C : 4\}$. The number of forward extensions items FEI is equal to 3, meaning that we can not add A to FCS. We grow A with A and compute the new projected database $\mathcal{D}^{AA} = (BC)$. As there is no possible backscan for AA in $\mathcal{D}^{AA}$, we compute the number of backward extensions items, equal to 0, and call $BIDE(\mathcal{D}^{AA}, AA, \min_{\text{sup}}, BEI, FCS)$. There is no locally frequent item in $\mathcal{D}^{AA}$ and therefore $BEI + FEI$ is equal to 0, meaning that we can add AA to the frequent closed subsequences FCS. We then move to item B from the locally frequent items of $\mathcal{D}^A$ and grow A with B. The corresponding projected database is $\mathcal{D}^{AB} = (C, BC, C, BCA)$. As there is no possible backscan of AB in $\mathcal{D}^{AB}$, and no backward extension items, we call $BIDE(\mathcal{D}^{AB}, AB, \min_{\text{sup}}, BEI, FCS)$. The locally frequent items from $\mathcal{D}^{AB}$ are $\{B : 2, C : 4\}$. As the support of C is equal to 4, $FEI > 0$ and we can not add AB to FCS. We grow AB with B and compute the projected database $\mathcal{D}^{ABB} = (CA)$. There is no possible backscan of ABB in $\mathcal{D}^{ABB}$ and no backward extension item, we call $BIDE(\mathcal{D}^{ABB}, ABB, \min_{\text{sup}}, BEI, FCS)$. As there is no locally frequent item in $\mathcal{D}^{ABB}$ we add ABB to FCS (now $FCS = (AA, ABB)$) and move to item C from the locally frequent items of $\mathcal{D}^{AB}$. We grow AB with C and compute $\mathcal{D}^{ABC} = (A, B)$. There is no possible backscan of ABC in $\mathcal{D}^{ABC}$ and no backward extension item, we call $BIDE(\mathcal{D}^{ABC}, ABC, \min_{\text{sup}}, BEI, FCS)$. As there is no locally frequent item in $\mathcal{D}^{ABC}$, we can add ABC to FCS which is now equal to (AA, ABB, ABC). Following the same methodology on $\mathcal{D}^B$ and $\mathcal{D}^C$ will lead to the complete set of frequent closed subsequences

$FCS = (AA, ABB, ABC, CA, CABC, CB)$.

Algorithm 2: Bide [Wang and Han, 2004]

Inputs: an input sequence database $\mathcal{D}$, a minimum support threshold $\min_{\text{sup}}$.

Output: the complete set of frequent closed sequences, FCS.

$FCS = \emptyset$;
$F1$ = frequent 1-subsequences($\mathcal{D}$, $\min_{\text{sup}}$);
for $f1$ *in* $F1$ **do**
  $\mathcal{D}^{f1}$ = projected database($\mathcal{D}$, $f1$);
for *f1 in F1* **do**
  if *!Backscan(*$f1$, $\mathcal{D}^{f1}$*)* **then**
    BEI = backward extension check ($f1$, $\mathcal{D}^{f1}$);
    call BIDE($\mathcal{D}^{f1}$, $f1$, $\min_{\text{sup}}$, BEI, FCS);
return FCS;

Subroutine BIDE($\mathcal{D}^{\alpha}$, α, $\min_{\text{sup}}$, BEI, FCS);

Arguments:

- a projected sequence database $\mathcal{D}^{\alpha}$;
- a prefix sequence α;
- a minimum support threshold $\min_{\text{sup}}$;
- the number of backward extension items BEI;
- the complete set of frequent closed sequences FCS.

Compute:
LFI = locally frequent items($\mathcal{D}^{\alpha}$);
FEI = $|\{z \in LFI | \sup(z) = \sup_{\mathcal{D}}(\alpha)\}|$;
if $(BEI + FEI) == 0$ **then**
  FCS = FCS $\cup \{\alpha\}$;
for i *in* LFI **do**
  grow α with i to get a new prefix αi;
  $\mathcal{D}_{\alpha i}$ = projected database($\mathcal{D}^{\alpha}$, αi);
for i *in* LFI **do**
  if *!BackScan(*αi, $\mathcal{D}^{\alpha i}$*)* **then**
    BEI = backward extension check (αi, $\mathcal{D}^{\alpha i}$);
    call BIDE($\mathcal{D}^{\alpha i}$, αi, min_{sup}, BEI, FCS)

The Bide algorithm was implemented using the prefixspan Python [1] package.

I.2.3.3 Gap-Bide

As cohesive subsequences are included in subsequences, we only mined subsequences and evaluated the proximity of the actions composing them. However, an extension to the Bide

[1] https://pypi.org/project/prefixspan/

algorithm, namely Gap-Bide [Li and Wang, 2008] was proposed to mine frequent closed cohesive subsequence with gap constraints. It proposes two parameters M and G to characterize the gap constraints, where M stands for the minimal number of symbols that have to be contained between every two consecutive terms of the gap-constrained pattern and G refers to the maximal number of symbols possibly contained between two consecutive terms.

I.2.4 Methodological Tests

I.2.4.1 Data

Figure I.2.1 shows the crash rate for each system of our global database. Although all these systems have the same software release, the crash rate varies a lot from one system to another. To conduct our experiments, we used the 5 systems having a crash rate exceeding 3%. Systems above this threshold are particularly critical and generally require in-depth investigations.

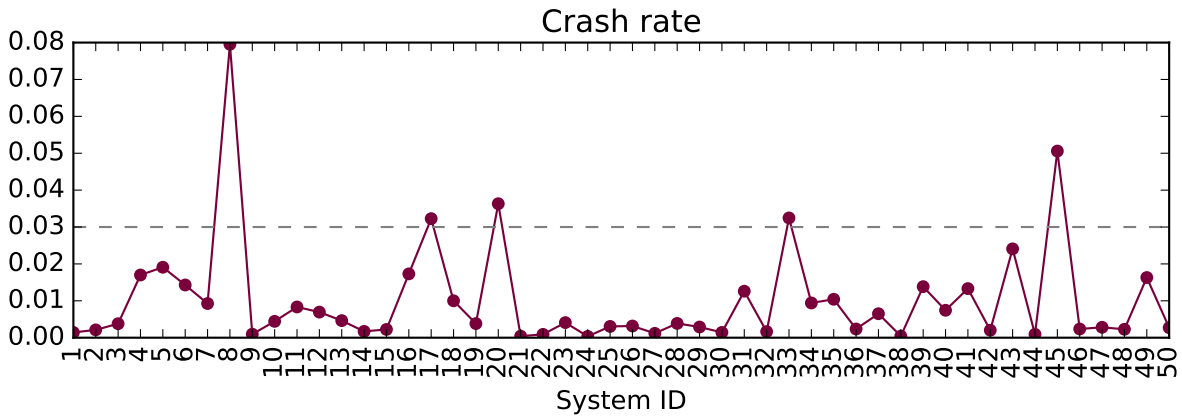

Figure I.2.1: Crash rate for each system of the database and in dashed line the critical threshold of 3%.

More details for each of these 5 systems are given in Table I.2.8. p_0 stands for the crash rate, $|\mathcal{D}|$ for the total number of sessions, corresponding to the sum of the number of crashes and the number of normal sessions ($|\mathcal{D}| = |\mathcal{C}| + |\mathcal{S}|$). Finally, $u_{a_c} = |\Sigma|$ stands for the number of unique actions in the crash sessions. This quantity differs from one system to another because the content of the interface and therefore the tools available to the user heavily depend on the medical applications used by the system in question.

System ID	p_0	$\|\mathcal{D}\|$	u_{a_c}
8	0.080	3445	89
17	0.032	4680	58
20	0.036	2368	61
33	0.032	2033	132
45	0.051	850	155

Table I.2.8: Crash rate p_0, total number of sessions $|\mathcal{D}|$ and number of unique actions in the crash sessions $u_{a_c} = |\Sigma|$ for each system selected for the analysis.

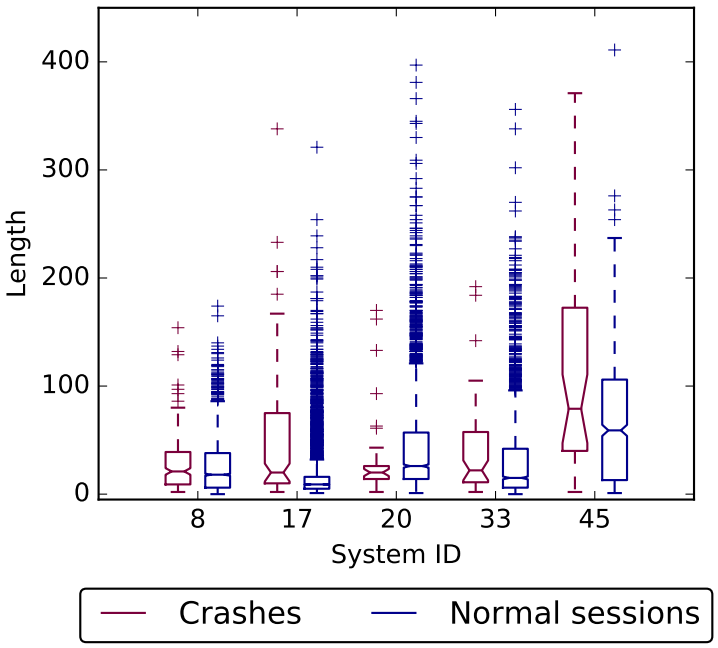

Figure I.2.2: Session length distributions for crashes and normal sessions.

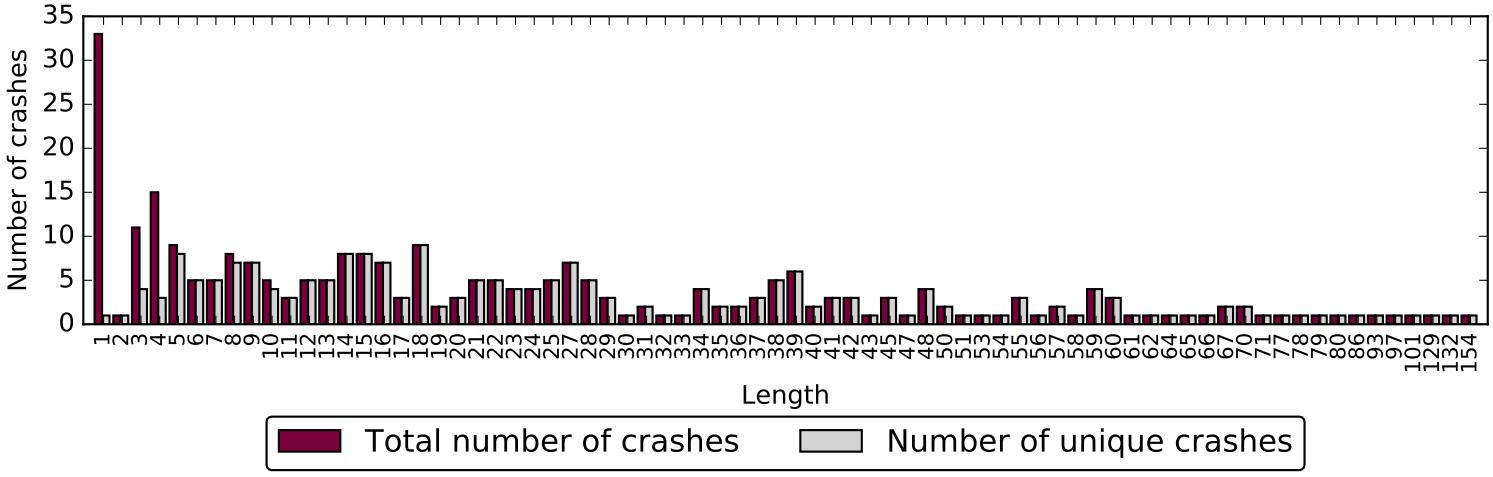

Figure I.2.3: Total number of crashes and number of different crashes as a function of session length for system 8.

The high variability of crashes in both content and length is illustrated in Figures I.2.2 and I.2.3. Figure I.2.2 shows for each system the length distribution of both crashes and normal sessions. While system 3 presents very short crashes, system 45 contains very long ones. Crashes may happen at the very beginning of a session as well as after multiple user actions. Figure I.2.3 shows, for system 8, the number of crashes according to the session length, as well as the number of different crashes of this length. Only some of the short crashes, containing 5 actions or less, consist of exactly the same workflow. All the other crash sessions are different. This trend is observed on all other systems.

I.2.4.2 Impact of Order

In this section we aim at determining if the order of the actions composing a crash pattern is decisive. For this purpose, we conducted 2 experiments, summarized in Table I.2.9. In Experiment 1 we first mine frequent subsequences and then test if corresponding itemsets are significant too. In Experiment 2 we do the opposite approach, we first mine frequent itemsets and then test if at least one of the corresponding subsequences has been detected as significant.

	Pattern Mined	Question
Experiment 1	Significant subsequences	Are corresponding itemsets significant too?
Experiment 2	Significant itemsets	Is one of the corresponding subsequence significant?

Table I.2.9: Experiments overview.

Experiment 1

To conduct the first experiment, we first mined frequent subsequences with the Bide algorithm and evaluate for different threshold probabilities the number of significant subsequences, at a significance level $\tau = 0.01$. The results are provided in Table I.2.10. As can be seen, the threshold probability to obtain only a few significant subsequences varies a lot from one system to another. In the case of system 8, $3p_0$ is enough to obtain 24 significant subsequences whereas a threshold of $10p_0$ is necessary to select 65 significant subsequences in the case of system 2.

System ID	p_0	$2p_0$	$3p_0$	$4p_0$	$5p_0$	$6p_0$	$7p_0$	$8p_0$	$9p_0$	$10p_0$
8	56994	3202	**24**	0	0	0	0	0	0	0
17	1674	1644	1463	1422	1407	1338	1150	555	157	**65**
20	190	168	152	133	99	49	25	18	3	**1**
33	26499	10353	5074	2953	1406	1311	31	31	**31**	0
45	46007	8076	3475	1329	309	162	**40**	0	0	0

Table I.2.10: Number of significant subsequences for each system mined with the Bide algorithm with $\tau = 0.01$, $\min_{\text{sup}} = 0.05$ and variable threshold probability rp_0, $r \in [\![1, 10]\!]$.

In order to test whether the order of the actions composing the significant subsequences are decisive, we propose to test if the corresponding itemset of a significant subsequence is signif-

icant too. To demonstrate our point, we considered the highest threshold probability which still provides significant subsequences. These configurations are in bold in Table I.2.10. For example, for system 8, we use the 24 subsequences obtained with a probability threshold equal to $3p_0$ since a highest value was too selective and does not return any significant subsequence. For each of the subsequences in bold in Table I.2.10, we tested if the corresponding itemset (containing the same actions, regardless of their order) is significant, at the same significance level $\tau = 0.01$. Results are provided in Table I.2.11. For 4 out of 5 systems (systems 17, 20, 33 and 45) the rate of corresponding significant itemsets is equal to 0. Besides, only 2 out of 24 corresponding itemsets are significant in the case of system 8. Table I.2.12 shows 2 examples of significant subsequences whose corresponding itemsets are not significant. While corresponding itemsets occur in either the same number or a higher number of crashes, they also appear in many more normal sessions, thus resulting in a higher *p-value*.

System ID	number of significant itemsets	number of tested subsequences	rate
8	2	24	0.083
17	0	65	0
20	0	1	0
33	0	31	0
45	0	40	0

***Table I.2.11:** Number of significant itemsets corresponding to significant subsequences for each system with $\tau = 0.01$. The tested itemsets correspond to the subsequences in bold in Table I.2.10.*

	significant subsequence			corresponding itemset		
System ID	$\lvert\mathcal{C}_\alpha\rvert$	$\lvert\mathcal{D}_\alpha\rvert$	*p-value*	$\lvert\mathcal{C}_\alpha\rvert$	$\lvert\mathcal{D}_\alpha\rvert$	*p-value*
example 1 (system 8)	19	47	0.008674	34	164	0.85
example 2 (system 33)	4	4	0.002518	4	7	0.12

***Table I.2.12:** Examples of non-significant itemsets corresponding to significant subsequences.*

Experiment 2

To conduct the second experiment, we first mined frequent itemsets with the Closet algorithm and evaluated for different threshold probabilities the number of significant subsequences, at a significance level $\tau = 0.01$. The results are provided in Table I.2.13. Again, the threshold probability to obtain only a few significant subsequences varies a lot from one system to another. As a k-itemset corresponds to $k!$ different k-subsequences, the number of determined significant itemsets is less important than the number of determined significant subsequences in Table I.2.10. In order to test the impact of the order of the actions, we check for each of the significant itemsets in bold if at least one of the corresponding subsquences was found in Experiment 1, at

the same threshold probability.

System ID	p_0	$2p_0$	$3p_0$	$4p_0$	$5p_0$	$6p_0$	$7p_0$	$8p_0$	$9p_0$	$10p_0$
8	408	15	**1**	0	0	0	0	0	0	0
17	348	308	265	228	143	60	**10**	0	0	0
20	17	14	13	10	7	3	3	**1**	0	0
33	105	7	4	2	1	**1**	0	0	0	0
45	216	27	6	**1**	0	0	0	0	0	0

Table I.2.13: ***Number of significant itemsets for each system mined with the Bide algorithm with*** $\tau = 0.01$, $\min_{\text{sup}} = 0.05$ ***and variable threshold probability*** rp_0, $r \in [\![1, 10]\!]$.

System ID	number of itemsets corresponding to a significant subsequence	number of tested itemsets	rate
8	1	1	1
17	10	10	1
20	1	1	1
33	1	1	1
45	1	1	1

Table I.2.14: ***Number of significant itemsets corresponding to a significant subsequence for each system with*** $\tau = 0.01$. ***The tested itemsets correspond to configurations in bold in Table I.2.13.***

As can be seen in Table I.2.14, all the significant itemsets were also found in significant subsequences.

Conclusion : Impact of Order

As almost all of the corresponding itemsets of significant subsequences are not significant and as for the majority of the significant subsequences at least one corresponding subsequence is significant, we conclude that the order of the actions making up a crash pattern is decisive. These 2 experiments enabled to discard the itemsets, which is the reason why we do not consider itemsets in the rest of this work.

I.2.4.3 Impact of Proximity

This section aims at testing whether the proximity of the actions composing a crash pattern is relevant or not. As we discarded the itemsets with the previous experiments, we want to evaluate which among exact subsequence, cohesive subsquence or subsequence is the most appropriate type of pattern to represent crash signatures. Indeed, these 3 representations take into account the order of the actions but differ concerning the proximity of the actions composing them. For this purpose we evaluate 2 measures on the significant subsequences in bold in Table I.2.10 : the maximum gap and the cohesion.

Maximum Gap

We evaluated for each significant subsequence determined in Table I.2.10 the maximum gap of actions contained between 2 consecutive actions composing the subsequence in the crash sessions in which they appear. Table I.2.15 shows the maximum gap among all the subsequences and all the crash sessions for each system. These maximum gaps are rather important, varying between 8 and 62. Figure I.2.4 shows the cumulative histogram of maximum gaps obtained with the 31 signficant subsequences from system 33. A maximum gap of 20 would only have detected 23% of the significant patterns. These results suggest that exact and cohesive subsequences are not suitable for our task.

System ID	maximum gap
8	43
17	55
20	8
33	35
45	62

***Table I.2.15:** Maximum gap found among the significant subsequences in bold in Table I.2.10 for each system.*

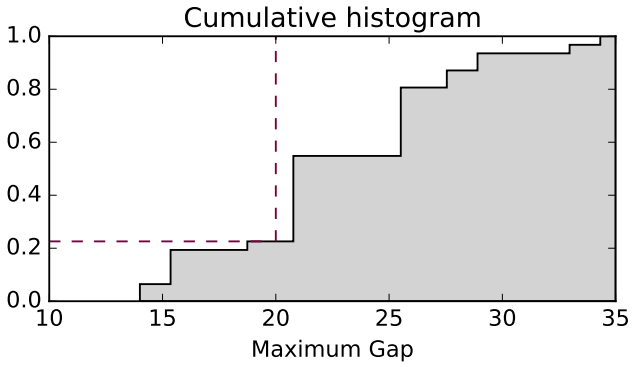

***Figure I.2.4:** Cumulative histogram of maximum gaps obtained for the 31 significant subsequences of system 33.*

I.2.4.4 Cohesion

Table I.2.16 shows the average cohesion of the set of significant subsequences (corresponding to the selected threshold probability in Table I.2.10). The average cohesion is lower than 0.4 for 4 out of 5 systems and equal to 0.55 in the case of system 20. These results indicate that the subsequences are composed of actions which are widely spread throughout sessions.

System ID	Average cohesion
8	0.38
17	0.11
20	0.55
33	0.3
45	0.18

*Table I.2.16: **Cohesion of the significant subsequences in bold in Table I.2.10 for each system.***

I.2.4.5 Results Summary

The first experiments consisted in testing the importance of the order of the actions composing a crash pattern. The goal here was to decide which among itemsets or subsequences are the most appropriate to represent crash signatures. Results suggested that the order of the actions is decisive and thus itemsets were discarded.

In order to evaluate the impact of action proximity making up a crash signature we proposed to evaluate the maximum gaps of significant subsequences. The cumulative histogram of maximum gaps obtained on system 33 indicates that gap constraints would not enable the detection of an important part of the significant subsequences. Besides, the low cohesion measures suggest that actions composing significant subsequences are spread throughout crash sessions.

We conclude that subsequences are the most appropriate pattern to represent crash signatures.

I.2.5 Application on a Critical Case

Context

At the time of this writing, this method has been used to investigate a particularly critical system having a crash rate reaching 6.8% (46 crash sessions and 629 normal sessions). This system has a recent version of the software and is not one of the 50 systems studied above. The major challenge in such a case is to be able to reproduce the crash in order to fix bugs. Even the users of the system in question are generally not able to explain which workflows lead to crashes. We applied the significant subsequences detection method described in this chapter to the logs coming from this system and it enabled to highlight some relevant crash signatures, two examples are given below.

Crash Signatures

Table I.2.17 shows the number of significant subsequences obtained for variable threshold probabilities. We conducted the investigation on the 12 subsequences obtained with a threshold of $2p_0$. 2 examples of crash signatures are given in Table I.2.18. Both of them were performed 3 times, systematically ending with a crash. While the example on the left contains 4 actions and is quite cohesive, the example on the right contains 17 actions which are rather spread throughout the crash sessions as indicated by a cohesion of 0.21. The signatures obtained enabled to focus on different workflows and were very useful to the investigation and the debugging

team.

p_0	$2p_0$	$3p_0$	$4p_0$	$5p_0$
3461	12	8	0	0

Table I.2.17: ***Number of significant subsequences for the studied system with the Bide algorithm with*** $\tau = 0.01$, $\min_{\text{sup}} = 0.01$ ***and variable threshold probability*** rp_0, $r \in [\![1, 5]\!]$.

'apps-launch vv', 'accept', 'reject', 'ctc/tracking/buildnextvol'	'apps-launch vv', 'tool-scalpel', 'vxwidget/vois/21', 'tool-auto select_toolkey', 'segmentation/autoselect/smallvessels', 'menu-main_reset_pointer', 'vxwidget/modifypanel2/8', 'segmentation/autoselect/remove', 'segmentation/autoselect/add', 'sliderview/model', 'sliderview/model', 'sliderview/model', 'tool-save state', 'vxwidget/savestatepanel/103', 'vxwidget/savestatepanel/101', 'menu-dynamicitem', 'ava/tracking/editcenter'
$k = 4$ $\lvert\mathcal{C}_\alpha\rvert = 3$ $\lvert\mathcal{S}_\alpha\rvert = 3$ $cohesion = 0.48$	$k = 17$ $\lvert\mathcal{C}_\alpha\rvert = 3$ $\lvert\mathcal{S}_\alpha\rvert = 3$ $cohesion = 0.21$

Table I.2.18: ***Examples of crash signatures obtained with*** $3p_0$ ***threshold probability and significance level*** $\tau = 0.01$.

I.2.6 Discussion

In this chapter we identified subsequences as the most appropriate pattern type to represent crash signatures among 4 types of patterns. As demonstrated on a critical industrial application, this method proves to be useful during in-depth investigations. However the fact that the crash signatures when reproduced do not crash systematically suggests that some crucial information is currently missing from log files. Indeed, the dataset details (number of volumes, size, acquisition modality) for example, which is known to be particularly relevant is notably missing.

In addition to being used in investigations, the most significant crash signatures might be used in automatic tests. Indeed, these high probability crash patterns are often complex and would not have been tested in the usual test routines.

Chapter I.3

User Workflow Characterization

USER BEHAVIOR MODELING is an important task and has many uses. Indeed, being able to characterize user workflows might enable to adapt user interface content, simplify repetitive tasks, reveal some features which are not used as expected and thus need to be investigated upon. More generally, the knowledge of trends of use provides precious insights for further marketing or business decisions. In this chapter we aim at grouping similar sessions and thus determine characteristic workflows. For this purpose we propose to compare different session representations as inputs of several clustering algorithms. We discuss the related work in section I.3.1. We present the data which will be used to conduct our experiments in section I.3.2. Section I.3.3 describes the session representations we propose to use as inputs of the clustering algorithms presented in section I.3.4. A preliminary work consisted in the clusterability assessment of our data which is detailed in section I.3.5. Sections I.3.6 and I.3.7 respectively present the selection of a cluster validity index and of the hyperparameters. Section I.3.8 presents the selection of the best configurations session representation – clustering algorithm and finally section I.3.9 is dedicated to the interpretation of the obtained clusters.

I.3.1 Research Problem

Understanding user behavior based on log files is a challenging task. Indeed, there is a significant variability in the workflows of different users to perform the same task [Dev and Liu, 2017]. This might be explained by several factors such as the experience of the user, his working habits and preferences, the difficulty of the case to be treated or the fact that several combinations of actions might lead to the same final result. Moreover, users might make mistakes thus adding unintended actions in their workflows corresponding to noise in our data. We propose to partition our sessions according to their similarities, making the assumption that the resulting clusters will represent characteristic workflows of our applications.

Works on web sessions clustering [Chaofeng, 2009] or clickstream clustering [Wang et al., 2016] are very close to our problem. Their goal is to highlight groups of similar navigational paths based on streams of hyper links or traces of user activities. The methods generally involve several common steps.

The first one concerns the pre-processing of the input data, so that it can be used in clustering algorithms. A standard way to represent web sessions is to transform them into feature vectors. Each unique page stands for a feature, and the corresponding web session vector will contain a 1 if the web page has been viewed during the session, a 0 otherwise [Dixit and Bhatia, 2015]. This process enables to represent sessions of variable lengths by vectors of the same size, here, the number of unique pages. In this representation, a new session is started when the time between two consecutive page accesses exceeds a given threshold. Another technique consists in analyzing website logs in a definite time interval. Each session is represented by the number of hits on each page during this time period [Xu and Liu, 2010]. Web sessions might also be represented by the relevance of each page based on a harmonic mean of the number of visits of the page and the time spent on it during the session [Sisodia et al., 2017].

This step is generally followed by the choice of a similarity measure between web sessions. Many distances have been used from sequence dissimilarity measures to classical distances. The sequence alignment measure [Wang and Zaïane, 2002], the Hamming metric [Dixit and Bhatia, 2015] or a distance based on the Levenshtein metric [Scherbina and Kuznetsov, 2004] might be used, as well as common distances such as the Euclidean distance, the squared Euclidean distance [Dixit and Bhatia, 2015] or the Cosine distance [Xu and Liu, 2010].

A wide variety of clustering algorithms have been proposed for different applications [Jain, 2010] over the last decades. Even if a various number of them have been applied to web sessions clustering, the most commonly used for this application is the partitional algorithm K-means or adapted versions of it [Wang and Zaïane, 2002, Xu and Liu, 2010, Poornalatha and Raghavendra, 2011]. Its simplicity of implementation, its convergence speed and the generally good quality of the resulting clusters are its main assets. To investigate the impact of various clustering methods, we also propose to test the quality of hierarchical clustering obtained with different linkage methods as well as spectral clustering which generally provides good results in non-convex clustering problems, see Chapter 14 in [Friedman et al., 2001].

The final step consists in the evaluation of the obtained partition to find the optimal one. Indeed, the clustering being an unsupervised learning procedure, there is no a priori knowledge of the optimal partition [Halkidi and Vazirgiannis, 2001]. Internal and external cluster validity indices have been proposed for this purpose [Brun et al., 2007]. Internal indices generally measure the separation between clusters (clusters should be clearly separated) and the cohesion within clusters (the points contained in the same cluster should be as close to each other as possible) to assess the partition quality. External criteria are based on known provided class labels and are therefore unusable in our case. The second approach, internal indices, evaluate the goodness of the partition without external information and is generally used to select the optimal number of clusters as well as the most appropriate algorithm. An extensive comparative study of validity indices has been made in [Arbelaitz et al., 2013]. We will present 3 of them, among the most commonly used: the Silhouette index, the Calinski–Harabasz index and the Dunn index [Ansari et al., 2011, Wiwie et al., 2015, Sisodia et al., 2017, Gu et al., 2017] in

section I.3.6.

Another approach draws inspiration from the work done in text mining, where document and word representations have been proposed for clustering tasks [Aggarwal and Zhai, 2012]. As in our problem, they have to deal with sentences or texts of different lengths. One of the most used fixed-length features in text classification or clustering is the bag-of-words representation which represents a text by a feature vector containing each term frequency. This representation has the advantage of a high interpretability and is thus widely used. For improved performance, the term frequency inverse document frequency (TFIDF) weighting scheme [Salton and Buckley, 1988] is often applied on the bag-of-words representation, and is a good baseline, in the case of clustering [Huang, 2008] or classification [Kim et al., 2017]. This weighting scheme takes into account the number of occurrences of a word in the entire corpus to readjust its count. If a word appears in a large number of documents, it will be considered as less important and its frequency will be reduced. An extension of the bag-of-words has been proposed, namely the term weighted inverse document frequency (TWIDF) [Rousseau and Vazirgiannis, 2013]. This document representation takes into account the proximity between words using graphes. As these representations can become very sparse and do not always capture semantics, distributed representations of documents have been introduced [Le and Mikolov, 2014]. Thanks to the way they are computed, these document vectors are able to learn the contextual information contained in texts and thus often outperforms bag-of-words representation for which the word order is not taken into account [Dai et al., 2014, Mijangos et al., 2017]. We propose to apply these 3 representations, bag-of-words with TFIDF, bag-of-words with TWIDF and distributed representations, to our user sessions to test which of them is the most appropriate to capture information among user actions for clustering.

I.3.2 Data

We conduct experiments on sessions for which the same medical application has been launched in order to avoid the clustering to group sessions at the application level which would not provide additional information. Thus, filtering sessions where the same application has been used should enable to detect characteristic workflows of a medical specialty.

Figure I.3.1 shows the total number of launches among the 50 files of our dataset. The general application launches have been removed since they sometimes correspond to specific application launches which were not properly logged. To be sure that we select only sessions in which a specific application was launched, we focused on applications for which we are sure they were correctly registered. As illustrated in Figure I.3.1, 4 applications are particularly used. We propose to conduct our experiments on the 2 most used, the method might then be reproduced on any log file or application for a specific analysis.

Figure I.3.2 shows the number of launches of each of the 4 most used applications for each system in the whole database. We decided to work with systems 30 and 50 since they correspond to the systems on which respectively app 1 and app 2 were the most used.

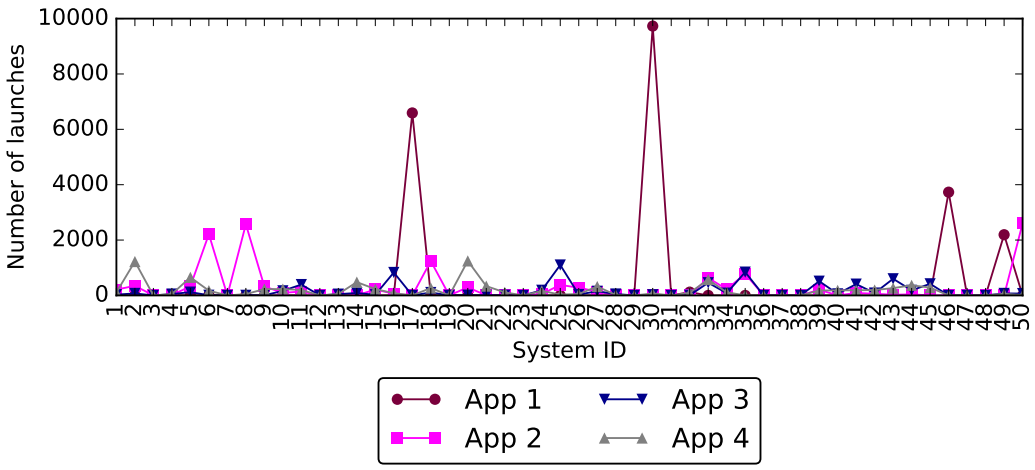

Figure I.3.2: Number of launches of each of the 4 most used applications, for each system.

App ID	System ID	$\lvert\mathcal{S}\rvert$
1	30	9433
2	50	2598

Table I.3.1: System ID corresponding to each studied app and corresponding number of sessions containing at least 4 actions.

Table I.3.1 shows the number of normal sessions $|\mathcal{S}|$ which will be used for clustering, for each selected system. For the reasons previously explained, we filtered only the sessions in which the studied application was launched. Moreover, we removed sessions containing 4 actions or less since they correspond to basic reviews with no interesting information on the user workflow.

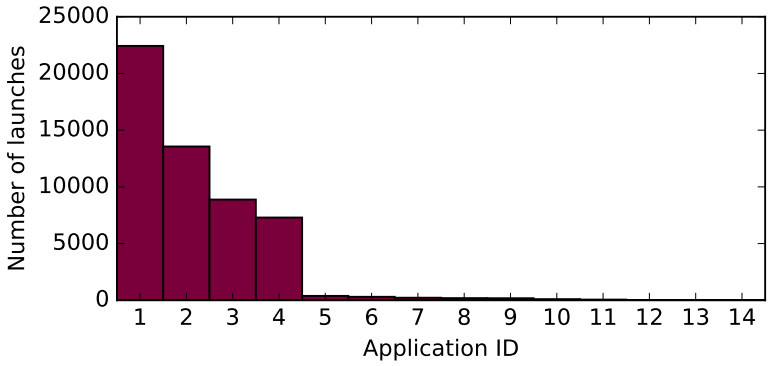

Figure I.3.1: Number of applications launches in the whole database. General application launches are not shown here.

I.3.3 Sequence Representation

For the reasons previously explained, user sessions contain different number of actions and can be of arbitrary length. To overcome this issue, we propose to transform user sessions using 3

different representations.

I.3.3.1 Bag-of-Actions with TFIDF

Let $\mathcal{D} = \{d_1, \ldots, d_n\}$ be a set of documents and $\mathcal{T} = \{t_1, \ldots, t_m\}$ the set of distinct terms occurring in $\mathcal{D}$. The Bag-of-Words representation of a document d is:

$$BoW(d) = (\mathrm{tf}(t_1, d), \mathrm{tf}(t_2, d), \ldots, \mathrm{tf}(t_m, d))$$

where $\mathrm{tf}(t, d)$ denotes the frequency of term t in document d. In order to give more importance to words appearing frequently in a small number of documents and less often in other documents, than words occurring in every document of the corpus, a weighting scheme tfidf is usually applied to the Bag-of-Words representation. The term-frequency times inverse document-frequency as defined in [Huang, 2008] weights the frequency of a term t in a document d according to the following equation:

$$\mathrm{tfidf}(t, d) = \mathrm{tf}(t, d) \times log \frac{|\mathcal{D}|}{\mathrm{df}(t)}$$

where:

- $\mathrm{df}(t)$ is the number of documents in which term t appears;
- $|\mathcal{D}|$ is the number of documents in the corpus.

We propose to apply the weighting scheme tfidf to our problem by replacing words by actions and documents by sessions. Let $\mathcal{S} = \{s_1, s_2, \ldots, s_N\}$ be a set of N user sessions and $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_{u_a}\}$ be the set of possible actions in the user interface, we denote $u_a = |\Sigma|$ its cardinal. By analogy, the weights of action occurrences will be calculated as follows:

$$w(\sigma, s) = f(\sigma, s) \times log \frac{|\mathcal{S}|}{\mathrm{sf}(\sigma)}$$

where:

- $f(\sigma, s)$ stands for the frequency of action σ in session s;
- $|\mathcal{S}|$ is the number of sessions;
- $\mathrm{sf}(\sigma)$ is the number of sessions containing action σ.

As a result, the Bag-of-Actions weighted according to the tfidf representation of a session s will be:

$$BoA_{\mathrm{tfidf}}(s) = (w(\sigma_1, s), w(\sigma_2, s), \ldots, w(\sigma_{u_a}, s)).$$

I.3.3.2 Bag-of-Actions with TWIDF

To capture relationships between words occurring close to each other, the graph-of-words model [Rousseau and Vazirgiannis, 2013] described in Chapter 2 proposes to represent documents as graphs in which each word is linked to all the successive words occurring within a sliding window. Thus, the frequency of a term in a document $\mathrm{tf}(t, d)$ is replaced by the weight of the node associated with the term t in the graph-of-words representation of document d. They propose to modify the *pivoted normalization weighting* version of the tfidf weighting scheme [Singhal et al., 1999] as follows:

$$\mathrm{twidf}(t, d) = \frac{\mathrm{tw}(t, d)}{1 - b + b \times \frac{|d|}{avdl}} * log \frac{|\mathcal{D}| + 1}{\mathrm{df}(t)}$$

where:

- $\mathrm{tw}(t, d)$ is the indegree of node t in the graph-of-words representation of document d;
- $|d|$ is the document length;
- $avdl$ is the average document length across the corpus;
- $|\mathcal{D}|$ is the number of document in the corpus;
- b is set to 0.003.

By analogy, we propose to improve session representations by capturing relationships between actions sharing the same context. The graph construction process is described in Figure I.3.3. The whole session is browsed, linking the current action by an edge to the $W - 1$ next following actions in the sequence, W being the length of the sliding window used to browse the user sessions. If the edge does not exist yet, a new edge is created. In this case we worked with directed unweighted graphs, since we are only interested in the indegree of nodes. In the example given in Figure I.3.3, the action *ufb* which would have a frequency of 1 in the tfidf weighting scheme will have a weight of 3, equal to its indegree in the twidf scheme.

Thus, the actions occurrences weights with twidf will be computed using a graph-of-actions, as follows:

$$w_g(\sigma, s) = \frac{tw(\sigma, s)}{1 - b + b \times \frac{|s|}{avsl}} \times log \frac{|\mathcal{S}| + 1}{\mathrm{sf}(\sigma)}$$

where:

- $\mathrm{tw}(\sigma, s)$ is the indegree of the node σ in the graph-of-actions representation of a session s;
- $|s|$ is the number of actions contained in s;
- $avsl$ is the average length of sessions across the dataset;
- $|\mathcal{S}|$ is the number of sessions;

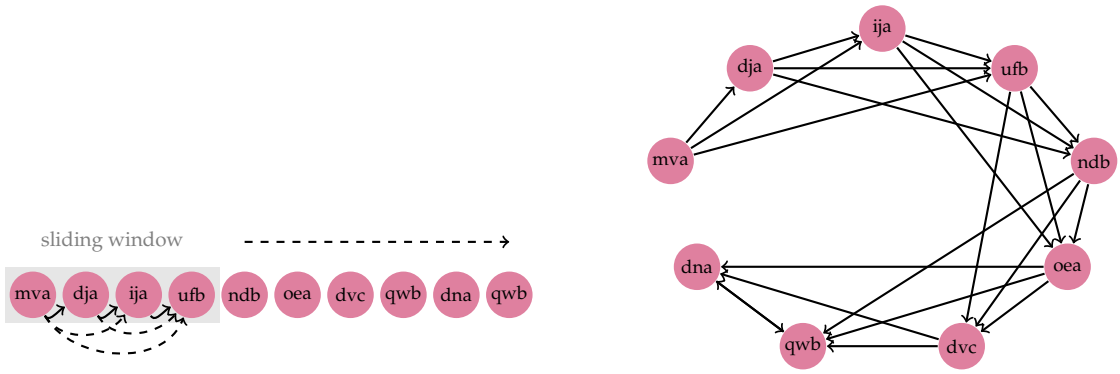

Figure I.3.3: ***Graph-of-Actions construction with a sliding window of length 4 from the example sequence on the left. The frequency of an action is replaced by the indegree of its corresponding node to compute the Bag-of-Actions representation with*** twidf ***weighting scheme.***

- b is set to 0.003 (as in [Rousseau and Vazirgiannis, 2013]).

Consequently, the Bag-of-Actions weighted according to the twidf representation of a session s will be:

$$BoA_{\text{twidf}}(s) = (w_g(\sigma_1, s), w_g(\sigma_2, s), \ldots, w_g(\sigma_{u_a}, s)).$$

I.3.3.3 Session Embeddings

Although Bag-of-Words based on the twidf weighting scheme in a way takes into account the relationship between words, this representation is sparse and grows with the size of the vocabulary. Distributed representations of words, which can be extended to documents, enable to overcome these limitations.

Doc2Vec

An approach to represent a document is to average the distributed representations of the words it contains, however this technique has proven to be less effective than document embeddings [Dai et al., 2014]. We propose to apply the Doc2Vec algorithm [Le and Mikolov, 2014] to tackle our clustering task. Although two architectures have been proposed to compute paragraph vectors with Doc2Vec, namely Distributed Memory model of Paragraph Vectors (PV-DM) and Distributed Bag-of-Words version of Paragraph Vector (PV-DBOW), we will only present the PV-DBOW architecture since it provides better results for our task. The PV-DBOW architecture, illustrated in Figure I.3.4, trains a feedworward network to predict words constituting the paragraph based on the corresponding paragraph id. If we apply this to our case, the model will work as follows. Let $\mathcal{S} = \{s_1, \ldots, s_N\}$ be a dataset of N sessions and $s_i = x_{i1} \ldots x_{in_i}$ be the sequence of n_i actions of the ith sequence. The PV-DBOW model trains a feedforward neural network to map a session s_i to the actions it contains. One-hot encoded sessions s_i with $i \in [\![1, N]\!]$ and actions x_{ir} with $r \in [\![1, n]\!]$ are respectively used as inputs (s) and outputs (y) of the following equations:

$$h = sW$$

$$y = \Phi(hW')$$

with E the embedding size, $W \in \mathbb{R}^{N \times E}$ are the input weights, $h \in \mathbb{R}^E$ the hidden layer and Φ the softmax function. $W' \in \mathbb{R}^{E \times u_a}$ represents the output layer. Paragraph vectors are trained using stochastic gradient descent to minimize the following loss (the gradient being obtained with backpropagration):

$$\mathcal{L}(W, W') = \sum_{i=1}^{N} \sum_{j=1}^{n_i} l(x_{ij}, \Phi(s_i W W'))$$

where l stands for the cross-entropy. At the end of the training process, final weights W will give the embeddings representation. Actions occurring less than frequency $\min_{\text{count}}$ will be ignored for training.

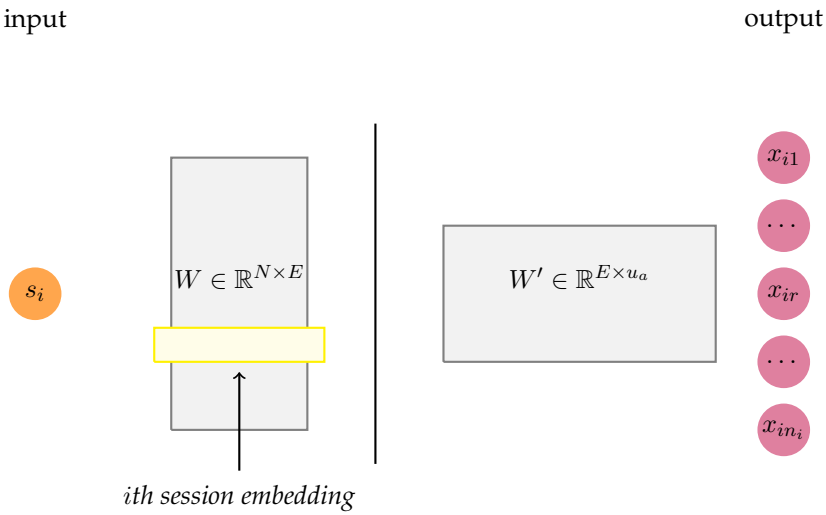

***Figure I.3.4:** Distributed Bag-of-Words version of Paragraph Vectors (PV-DBOW).*

I.3.3.4 Implementation

We used a Python feature extraction function to compute Bag-of-Actions with TFIDF weighting scheme and our own Python implementation of Bag-of-Actions with TWIDF weighting scheme, using NetworkX [Hagberg et al., 2008] to represent sessions as graphs. Embeddings were computed using the Gensim package [Rehurek and Sojka, 2010]. Unless otherwise specified, default PV-DBOW parameters were used and the network was trained during 5 epochs.

I.3.4 Clustering Algorithms

This section briefly describes each clustering algorithm we will use to discover user sessions clusters. Given a dataset $\mathcal{S}$ of N sequences represented as vectors in an $F-$dimensional space, the process of clustering $\mathcal{S}$ consists in partitioning $\mathcal{S} = \{s_1, s_2, \ldots, s_N\} \in \mathbb{R}^F$ into K disjoint groups: $C_K = \{c_1, c_2, \ldots, c_K\}$, with $c_k \neq \emptyset, \forall\, k$. $\mathcal{S} = \bigcup_{c_k \in C_K} c_k$ and $c_k \cap c_l = \emptyset, \forall k \neq l$. c_k is called a cluster. The centroid of a cluster c_k corresponds to its mean vector and is equal to:

$$\overline{c_k} = \frac{1}{|c_k|} \sum_{s_i \in c_k} s_i$$

where $|c_k|$ stands for the number of elements in cluster k. Besides, we will denote the Euclidean distance between objects s_i and s_j by $d_E(s_i, s_j)$.

I.3.4.1 K-means

K-means is one of the most widely used clustering algorithms. It aims at partitioning data into clusters by minimizing the sum of squared Euclidean distances between the data points and the centroid of the clusters as described in Algorithm 3.

Algorithm 3: K-means.

Input: The number of clusters K and the dataset composed of N elements $\mathcal{S} = \{s_1, s_2, \ldots, s_N\} \in \mathbb{R}^F$.

Steps:

1. Initialization of K centroids.
2. For each $i \in [\![1, N]\!]$, assign each element s_i to the closest centroid.
3. Compute the objective function also called inertia:

$$J = \sum_{k=1}^{K} \sum_{s_j \in c_k} d_E(s_j, \overline{c_k})^2$$

   Stop if J is below a given threshold ϵ.
4. Compute the centroid of each cluster with $\overline{C_K} = \{\overline{c_1}, \overline{c_2}, \ldots, \overline{c_K}\}$.
5. Repeat steps 2, 3 and 4 until $\overline{C_K}$ no longer changes.

We used a threshold $\epsilon = 0.0001$ and K-means++ initialization scheme [Arthur and Vassilvitskii, 2007], which spreads out the initial cluster centers. It initializes the centroids using the following steps:

1. Choose one center c_1, uniformly at random from $\mathcal{S}$.
2. Choose a new center c_i, in $\mathcal{S}$ with probability:

$$\frac{\zeta(s)^2}{\sum_{s \in \mathcal{S}} \zeta(s)^2}, \forall s \in \mathcal{S}.$$

$\zeta(s)$ denotes the shortest distance from point s to the closest center that has already been chosen.

3. Repeat step 2 until K centers have been chosen.

As K-means is highly dependent on the initialization of the centroids, each of the results presented corresponds to the best partition obtained in terms of inertia J over 100 runs.

I.3.4.2 Hierarchical Clustering

Agglomerative hierarchical clustering merges at each construction step the closest pair of clusters into larger and larger clusters. At the beginning, each point has its own cluster, hierarchical clustering process continues until all the the points are agglomerated together into one cluster. The obtained dendrogram might then be cut at different heights, corresponding to different numbers of clusters. Let c_i and c_j represent 2 clusters and $d_E(s, s')$ the pairwise observation distance between an element s in cluster c_i and an element s' in cluster c_j. We used the following linkage methods to compute the dissimilarity $d(c_i, c_j)$ between c_i and c_j:

- $d_{Single}(c_i, c_j) = \min_{s \in c_i, s' \in c_j} d_E(s, s')$;
- $d_{Average}(c_i, c_j) = \frac{1}{|c_i||c_j|} \sum_{s \in c_i} \sum_{s' \in c_j} d_E(s, s')$;
- $d_{Complete}(c_i, c_j) = \max_{s \in c_i, s' \in c_j} d_E(s, s')$;
- $d_{Ward}(c_i, c_j) = \frac{|c_i||c_j|}{|c_i|+|c_j|} d_E(\overline{c_i}, \overline{c_j})$.

Hierarchical clustering based on $Single$, $Average$ and $Complete$ linkages respectively minimizes the minimum, the average and the maximum distance between pairwise observations. Hierarchical clustering works slightly differently with the $Ward$ linkage and minimizes the sum of squared differences within all clusters. As described in Algorithm 4, the most similar clusters are successively merged using one of the above linkage methods.

Algorithm 4: Agglomerative hierarchical clustering.

Input: The number of clusters K and the dataset composed of N elements $\mathcal{S} = \{s_1, s_2, \ldots, s_N\} \in \mathbb{R}^F$.

Steps:

1. Find the most similar pair of clusters c_i and c_j using one of the linkage methods.
2. Merge clusters c_i and c_j.
3. Repeat steps 1 and 2 until only one cluster remains.

I.3.4.3 Spectral Clustering

Spectral clustering [Von Luxburg, 2007] uses graph partitioning and performs a low-dimensional embedding of the affinity matrix of input data points. This step is followed by a K-means algo-

rithm as described in Algorithm 5.

Algorithm 5: Spectral clustering.

Input: The number of clusters K and the dataset composed of N elements $\mathcal{S} = \{s_1, s_2, \ldots, s_N\} \in \mathbb{R}^F$, η a parameter to fix the number of neighbors to consider.

Steps:

1. Construct the η-nearest neighbor similarity graph by connecting node n_i corresponding to session s_i to the node n_j corresponding to session s_j if n_j is among the η nearest neighbors of n_i.
2. Compute the weighted adjacency matrix W, diagonal degree matrix D with the degrees $d_1, \ldots, d_N$ on the diagonal and Laplacian $L = D - W$.
3. Compute the first K eigenvectors $u_1, \ldots, u_K$ of L.
4. Let $U \in \mathbb{R}^{N \times K}$ be the matrix having $u_1, \ldots, u_k$ for columns.
5. Cluster the rows $y_i \in \mathbb{R}^K$, for $i \in [\![1, N]\!]$ using the K-means algorithm.

We selected $\eta = 5$ since it provided good results for both systems according to several cluster validity indices.

I.3.5 Clusterability

I.3.5.1 Hopkins Statistic

In order to assess the cluster tendency of our datasets, we propose to use the Hopkins statistic [Lawson and Jurs, 1990]. This test compares the nearest-neighbor distribution of uniformly randomly distributed artificial data points to that of randomly selected real points of our dataset. Let $\mathcal{Z}$ be a real dataset of N points. The Hopkins statistic is computed as follows:

1. Sample uniformly m points from $\mathcal{Z}$: $(p_1, p_2, \ldots, p_m)$.
2. Generate a simulated dataset $\mathcal{Z}_{rand}$ of m points: $(q_1, q_2, \ldots, q_m)$ drawn from a random uniform distribution, following the same range of variation as $\mathcal{Z}$.
3. For each $p_i \in \mathcal{Z}$, find its nearest neighbor $p_j \in \mathcal{Z}$ and compute the distance w_i between p_i and p_j.
4. For each $q_i \in \mathcal{Z}_{rand}$, find its nearest neighbor q_j in $\mathcal{Z}$ and compute the distance u_i between q_i and q_j.

The Hopkins statistic is obtained using the following formula:

$$H = \frac{\sum_{i=1}^{m} u_i}{\sum_{i=1}^{m} u_i + \sum_{i=1}^{m} w_i}.$$

To evaluate a consolidated value of H, the previous process is usually repeated over r runs and the average value of the statistic is then:

$$H_{av} = \frac{1}{r}\sum_{i=1}^{r} H_i.$$

A value of H_{av} close to 1 indicates the clusterability of the dataset. Indeed, if the dataset contains clusters, the value of $\sum_{i=1}^{m} u_i$ will be significantly higher than $\sum_{i=1}^{m} w_i$. On the contrary, if $\mathcal{Z}$ were uniformly distributed, $\sum_{i=1}^{m} u_i$ and $\sum_{i=1}^{m} w_i$ would be close and consequently H_{av} would be about 0.5. In other words, if the value of H_{av} is significantly higher than 0.5 we can reject the null hypothesis $\mathcal{H}_0$ ($\mathcal{Z}$ is uniformely distributed) and conclude that the dataset contains meaningful clusters.

I.3.5.2 Application to our Datasets

We selected m as the closest integer to $N/10$ as in [Banerjee and Dave, 2004] and ran the Hopkins statistic over $r = 1000$ repetitions, for each input representation, on each dataset. Bag-of-Actions weighted according to the TWIDF scheme were computed using a sliding window W of size 5 and session embeddings were computed using the PV-DBOW architecture with default parameters (embedding size E equal to 50, $\min_{\text{count}}$ equal to 5). Results are presented in Table I.3.2.

App ID	BoA_{tfidf}	BoA_{twidf}	Emb
1	0.99	0.99	0.91
2	0.98	0.98	0.91

Table I.3.2: Average Hopkins statistic H_{av} for each dataset and each input representation.

Table I.3.2 shows that all our datasets have a value of the Hopkins Statistic which is close to 1, it means that we can reject the null hypothesis and ensures that our datasets are significantly clusterable, regardless of the input representation.

I.3.6 Selection of Clustering Evaluation Indices

In order to evaluate the effectiveness of the input representations as well as the clustering algorithms, we propose to use an internal cluster validity index. In this section we detail 3 of the most commonly used indices [Wiwie et al., 2015, Gu et al., 2017]. These indices measure the cohesion of the clusters (distances within clusters or intra-variance) as well as the separation (distances between clusters or inter-variance). They differ in how they combine these 2 aspects.

I.3.6.1 Indices Definitions

The Silhouette index

For each observation the Silhouette index [Rousseeuw, 1987] computes the average distance between clusters using the average dissimilarity a between the observation and all the points of the cluster to which it belongs, as well as the minimum dissimilarity b between the observation and all the points from the clusters to which it does not belong. The Silhouette index value of a partition of K clusters is defined as:

$$Sil(C_K) = \frac{1}{N} \sum_{c_k \in C_K} \sum_{s_i \in c_k} \frac{b(s_i, c_k) - a(s_i, s_k)}{\max\{a(s_i, c_k), b(s_i, c_k)\}}$$

where:

- $a(s_i, c_k) = \frac{1}{|c_k|} \sum_{s_j \in c_k} d_E(s_i, s_j)$;
- $b(s_i, c_k) = \min_{c_l \in C_K \setminus c_k} \left\{ \frac{1}{|c_l|} \sum_{s_j \in c_l} d_E(s_i, s_j) \right\}$.

A value of the Silhouette index close to 1 means that observations are well-clustered whereas a value of -1 indicates poorly clustered observations.

The Calinski-Harabasz index

The Calinski-Harabasz index [Caliński and Harabasz, 1974] uses the global centroid of the dataset, defined as follows:

$$\overline{s} = \frac{1}{N} \sum_{s_i \in S} s_i.$$

To evaluate the quality of a partition of K clusters, it computes a ratio of Between Cluster Scatter Matrix (BCSM) and Within Cluster Scatter Matrix (WCSM):

$$CH(C_K) = \frac{BCSM}{K-1} \frac{N-K}{WCSM}.$$

where:

- $BCSM = \sum_{c_k \in C} |c_k| d_E(\overline{c_k}, \overline{s})^2$ evaluates the distance from the centroids to the global centroid,
- $WCSM = \sum_{c_k \in C} \sum_{s_i \in c_k} d_E(s_i, \overline{c_k})^2$ evaluates the distances from the points of a cluster to its centroid.

For compact and well separated clusters the between-clusters dispersion mean (BCSM) should be maximized and the within-cluster dispersion should be minimized, thus high values of Calinski-Harabasz indicate good partitioning.

The Dunn index

The Dunn index [Dunn, 1974] uses the minimum distance between each point of a cluster and all the other points from other clusters combined with the maximum cluster diameter as following:

$$D(C_K) = \frac{\min_{c_k \in C_K} \{\min_{c_l \in C_K \setminus c_k} \{\delta(c_k, c_l)\}\}}{\max_{c_k \in C_K} \{\Delta(c_k)\}}$$

where:

- $\delta(c_k, c_l) = \min_{s_i \in c_k, s_j \in c_l} \{d_E(s_i, s_j)\}$;
- $\Delta(c_k) = \max_{s_i, s_j \in c_k} \{d_E(s_i, s_j)\}$.

For a suitable partition of the dataset, the Dunn index should be maximized.

Since these 3 indices do not always behave in the same way, they will be compared on our data, see Section I.3.6.2 and the most appropriate will be selected to conduct our experiments.

I.3.6.2 Comparison of Indices and Selection

A problem with clustering validity indices is that they each capture specific aspects and do not all return the same optimal partition. We propose to compare the behavior of the 3 indices presented in Section I.3.6, namely the Silhouette score (Sil), the Calinski–Harabasz score (CH) and the Dunn index (D) on session embeddings of size 50 obtained with the PV-DBOW architecture. We illustrated on a specific example the potential antagonist conclusions that can be drawn according to the type of index considered. Figure I.3.5 shows 2-dimensional plots obtained with the t-SNE visualization algorithm [Maaten and Hinton, 2008] (see Appendix E) of the partitions obtained with K-means, Hierarchical Clustering with Single Linkage and Hierarchical Clustering with Complete Linkage. Each method was set to return 3 clusters. Table I.3.3 shows the number of sessions contained in each cluster, for each algorithm as well as the associated cluster validity indices.

	Cluster 1	Cluster 2	Cluster 3	*Sil*	*CH*	*D*
K-means	1428	933	237	0.50	4152.74	0.47
HC Single Linkage	4	1	2593	0.81	111.37	0.12
HC Complete Linkage	96	18	2484	0.60	936.05	0.60

***Table I.3.3:** Number of sessions contained in each cluster for each clustering algorithm and associated cluster validity index score.*

As illustrated, the Calinski–Harabasz score suggests that the best partition is obtained with the K-means algorithm. On the contrary, the Silhouette score gives the partition obtained with Hierarchical Clustering with Single Linkage as the best partition and the Dunn index is maximized when Hierarchical Clustering with Complete Linkage is used. As described previously, the Dunn index is based on extreme data points and the Silhouette score takes separation into account by evaluating the minimal average distances to all others clusters. Thus, both indices do not take into account all datapoints and behave poorly in the case of highly unbalanced clusters. On the other hand, the Calinski–Harabasz score evaluates the partition quality using the sum of the squares of the distances between the global centroid and each of the clusters centroid and the sum of the squares of the distances between the cluster centroid and all the data points contained in it. Calinski–Harabasz tends to prefer more balanced partitions and was therefore chosen to conduct our experiments.

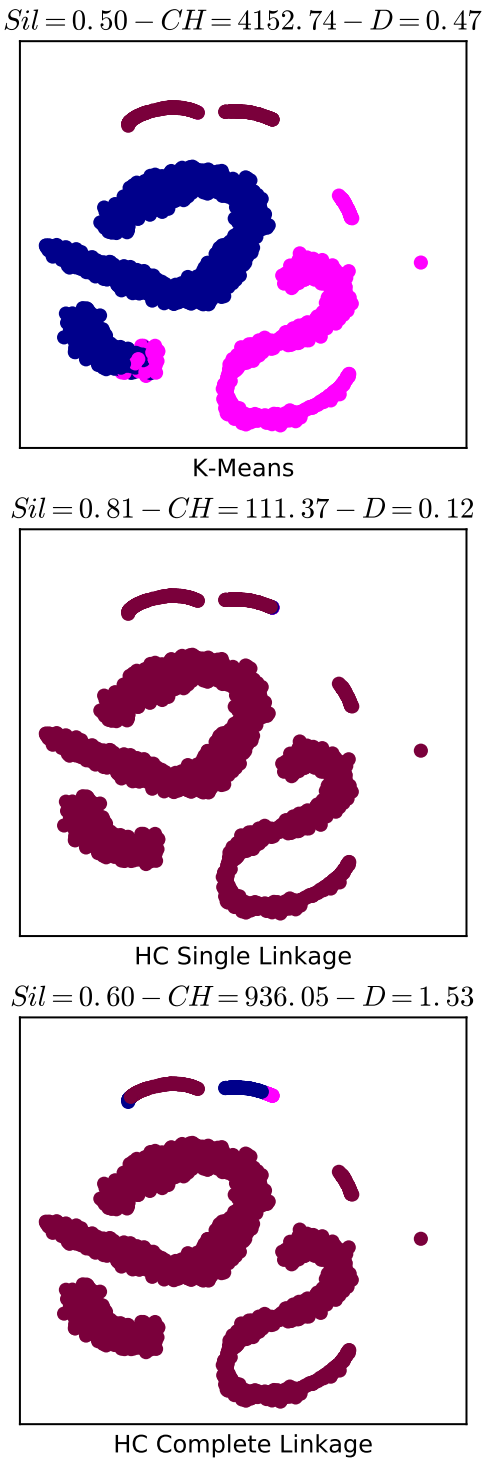

Figure I.3.5: Silhouette score (Sil), Calinski–Harabasz score (CH) and Dunn index (D) values in the case of K-means Clustering, Hierarchical Clustering with Single Linkage and Hierarchical Clustering with Complete Linkage obtained with embeddings as inputs. Plots were obtained using t-SNE [Maaten and Hinton, 2008].

I.3.7 Hyperparameters Selection

Bag-of-Actions with TWIDF weighting scheme and session embeddings require hyperparameters selection. This section presents the influence of some algorithmic parameters we tuned to improve the clustering quality.

I.3.7.1 Bag-of-Actions with TWIDF

Bag-of-Actions weighted according to the TWIDF scheme needs the window size W to be specified. To find the optimal window size, we used the K-means algorithm. Figures I.3.6 and I.3.7 show the maximum Calinski–Harabasz score obtained for partitions containing $k \in [\![2, K]\!]$ clusters:

$$\max_{k \in [\![2,K]\!]} \{CH(C_k)\}.$$

We used $K = 10$ and tested values of W in $[\![2, 10]\!]$ for both systems. For each k, K-means was run 100 times and only the best partition in terms of inertia was evaluated.

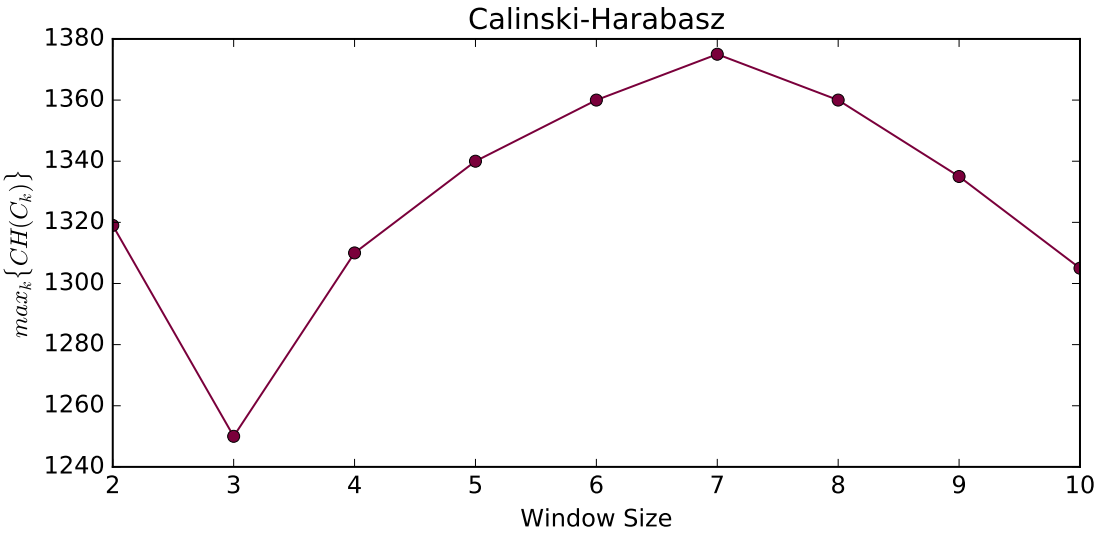

Figure I.3.6: ***Maximal Calinski–Harabasz score obtained for*** $k \in [\![2, K]\!]$ ***and TWIDF window size*** $W \in [\![2, 10]\!]$ ***in the case of system 30 (App 1) and the K-means algorithm.***

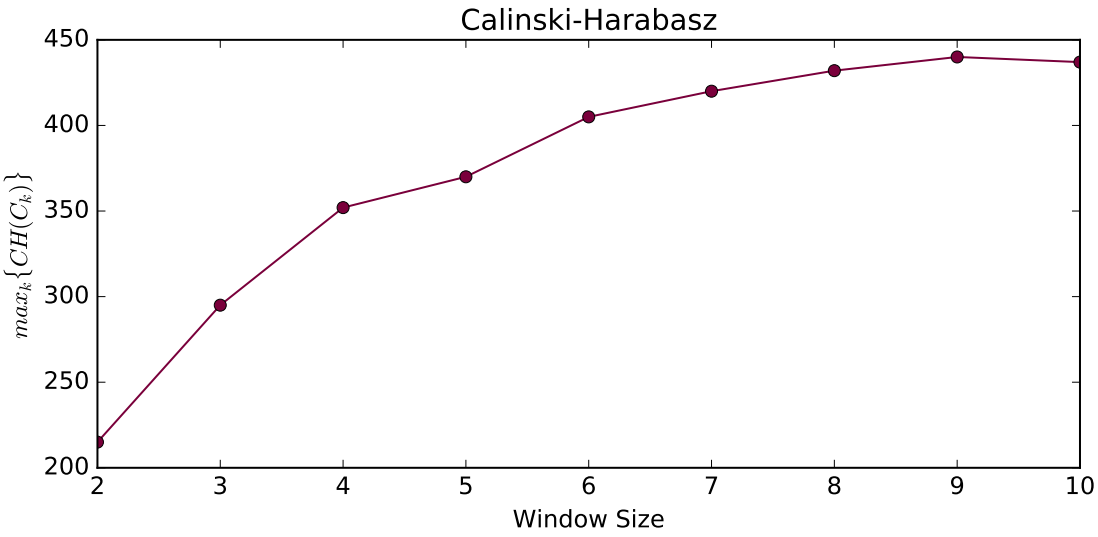

Figure I.3.7: ***Maximal Calinski–Harabasz score obtained for*** $k \in [\![2, K]\!]$ ***and TWIDF window size*** $W \in [\![2, 10]\!]$ ***in the case of system 50 (App 2) and the K-means algorithm.***

The maximum Calinski–Harabasz value is reached for a value of $W = 7$ for system 30 and for a value of $W = 9$ for system 50. We conducted experiments using $W = 7$ for both systems since results obtained on system 50 with $W = 7$ are quite equivalent to those obtained with $W = 9$. This value was tested for the other algorithms and provided good results.

I.3.7.2 Session Embeddings

Session embeddings computed with the PV-DBOW architecture of the Doc2Vec algorithm rely on several hyperparameters to be customized. We focused on the embedding size as well as on the $\min_{\text{count}}$ value representing the threshold under which words with lower frequency are ignored from the vocabulary. Again, Figures I.3.8 and I.3.9 show the maximum Calinski–Harabasz score obtained for partitions containing k clusters, with $k \in [\![2, K]\!]$ clusters.

We used $K = 10$, and tested the embeddings size E in $[50, 350]$ with a step of 50 and the $\min_{\text{count}}$ value in $\{1, 5, 10, 15, 20\}$. As previously, for each k, K-means was run 100 times and only the best partition in terms of inertia is evaluated.

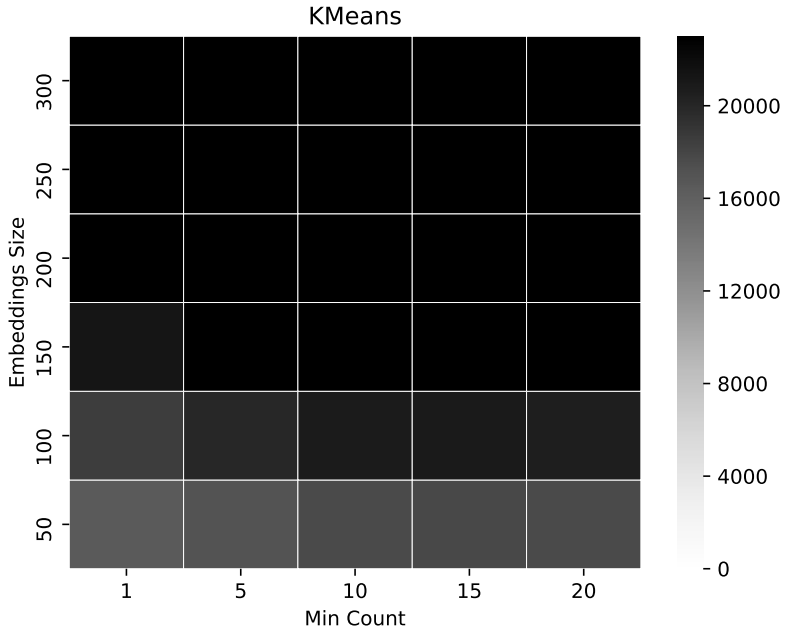

Figure I.3.8: ***Maximal Calinski–Harabasz score obtained with*** $k \in [\![2, K]\!]$, $\max_{k \in [\![2,K]\!]} \{CH(C_k)\}$ ***for embedding size*** E ***in*** $[50, 350]$ ***with a step of 50 and the*** min_{count} ***value in*** $\{1, 5, 10, 15, 20\}$ ***in the case of system 30 (App 1).***

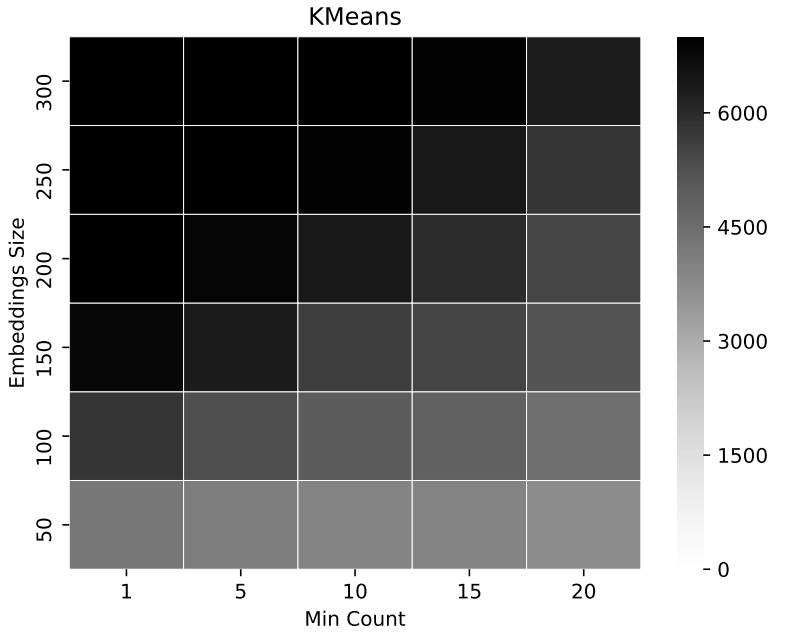

Figure I.3.9: ***Maximal Calinski–Harabasz score obtained with*** $k \in [\![2, K]\!]$, $\max_{k \in [\![2,K]\!]} \{CH(C_k)\}$ ***for embedding size*** E ***in*** $[50, 350]$ ***with a step of 50 and the*** min_{count} ***value in*** $\{1, 5, 10, 15, 20\}$ ***in the case of system 50 (App 2).***

Figures I.3.8 and I.3.9 suggest that an embedding size of 150 and minimum count of 1 pro-

vide results equivalent to those obtained with larger values and seem therefore appropriate. Appendix A shows the results obtained for the 4 other clustering algorithms which were evaluated. Individual hyperparameters selection might easily be re-run in the case of a specific analysis.

I.3.8 Tests of Representations & Clustering Algorithms

I.3.8.1 Experiments Overview

Our objective is to find the most appropriate session representation and clustering algorithm. For this purpose we will test each of the 3 input representations given in Table I.3.4 with each of the 5 clustering algorithms listed in Table I.3.5. Hierarchical Clustering with Single Linkage was not tested since preliminary experiments showed poor results and highly unbalanced clusters due the chaining phenomenon often occurring with this type of clustering.

Input Representation	Notation	Parameters
Bag-of-Actions with TFIDF	BoA_{tfidf}	-
Bag-of-Actions with TWIDF	BoA_{twidf}	$W = 7$
Embeddings with Doc2Vec	Emb	$E = 150$ $\min_{\text{count}} = 1$

Table I.3.4: Input representation, notations and parameters used for experiments.

Clustering Algorithm	Notation	Parameters
K-means	KM	K-means++ initialization. Best partition over 100 runs.
Hierarchical Clustering Linkage:		
- Average	HCA	
- Complete	HCC	
- Ward	HCW	
Spectral Clustering	SC	$\eta = 5$ Best partition over 100 K-means runs. K-means++ initialization

Table I.3.5: Clustering algorithms, notations and parameters used for experiments.

Each algorithm has been tested for number of clusters k with $k \in [\![2, 10]\!]$. Figures I.3.10 and I.3.11 respectively show the results obtained for system 30 (App 1) and system 50 (App 2). Each algorithm is followed by the value of k returning the highest value of the Calinski–Harabasz score, for Bag-of-Actions with TFIDF weighting scheme, Bag-of-Actions with TWIDF scheme and Embeddings, according to the following notation:

ALGORITHM $[\text{argmax}_k^{BoA_{\text{tfidf}}}\{CH(C_k)\}), \text{argmax}_k^{BoA_{\text{twidf}}}\{CH(C_k)\}, \text{argmax}_k^{Emb}\{CH(C_k)\}]$

I.3.8.2 System 30 (App 1)

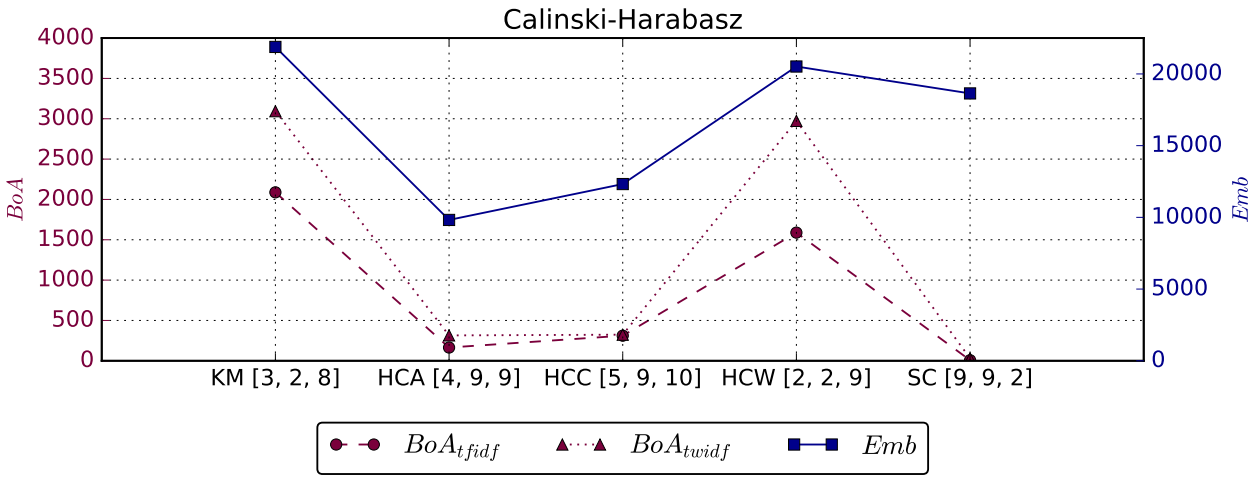

Figure I.3.10: ***System 30 (App 1). Comparison of the 3 input representations with 5 clustering algorithms. Brackets contain the value of k returning the highest value of the Calinski–Harabasz score for each input representation.***

I.3.8.3 System 50 (App 2)

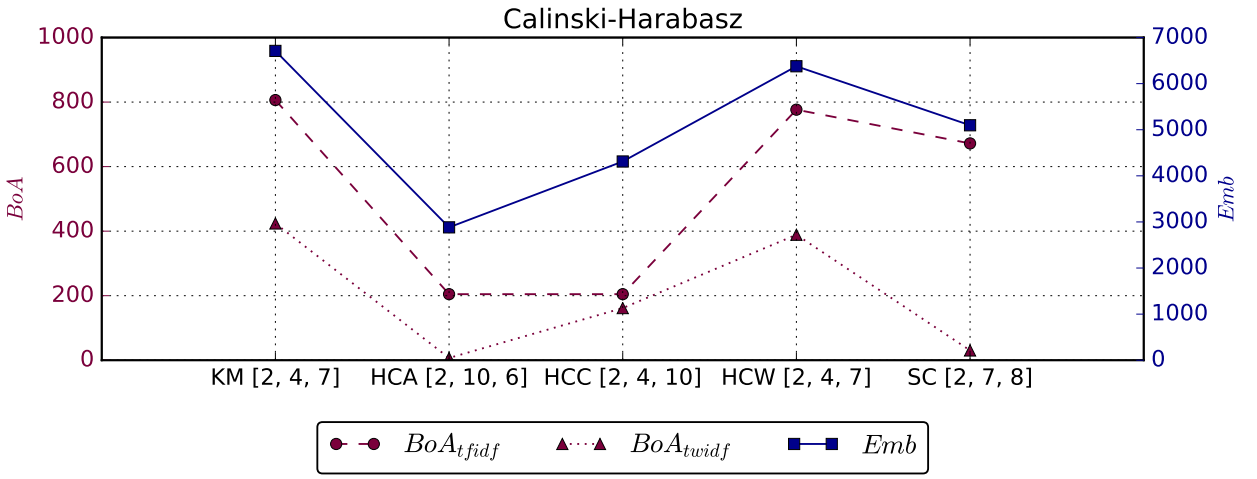

Figure I.3.11: ***System 50 (App 2). Comparison of the 3 input representations with 5 clustering algorithms. Brackets contain the value of k returning the highest value of the Calinski–Harabasz score for each input representation.***

While BoA_{twidf} provides better results than BoA_{tfidf} on system 30, it is the contrary on system 50. However, in both cases, session embeddings is the most appropriate representation and outperforms Bag-of-Actions based representations with all clustering algorithms. K-means

provides the best partition which is constituted of 8 clusters in the case of system 30 (App 1) and 7 clusters in the case of system 50 (App 2). The next section aims at interpreting the clusters obtained.

I.3.9 Cluster Analysis

I.3.9.1 2D Visualization

Figures I.3.12 and I.3.13 show the best partitions obtained for system 30 and system 50, with the K-means algorithm and session embeddings as inputs. The left figure shows the two-dimensional PCA plot while the right figure shows a two-dimensional t-SNE plot.

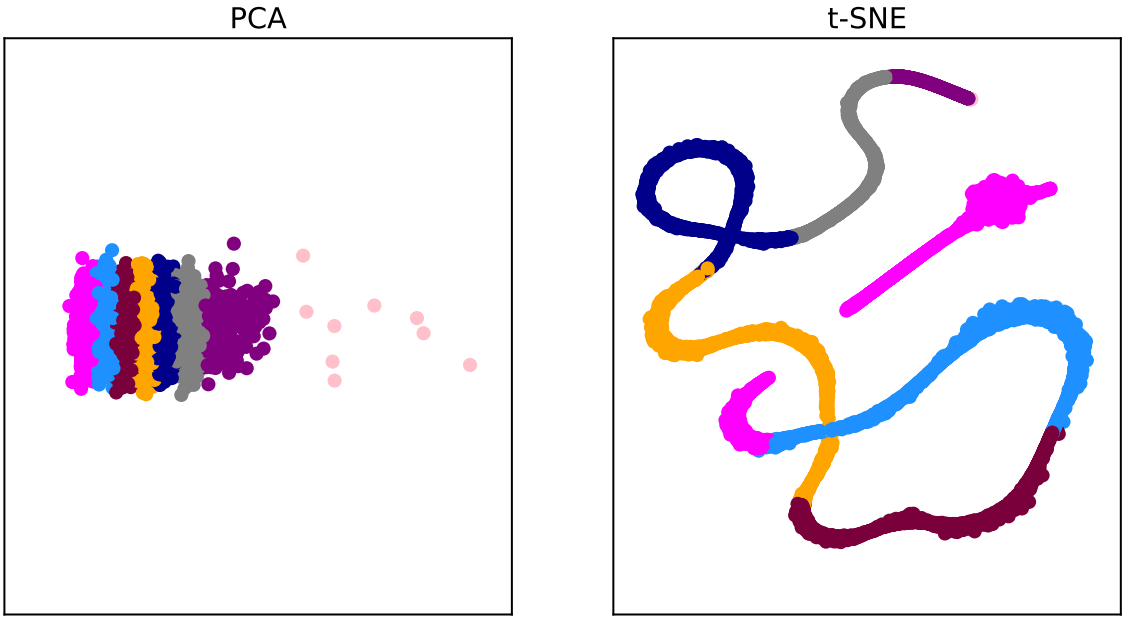

Figure I.3.12: 2-dimensional plots obtained with PCA on the left and t-SNE on the right for the best partition obtained with the K-means algorithm and session embeddings as inputs for System 30 (App 1).

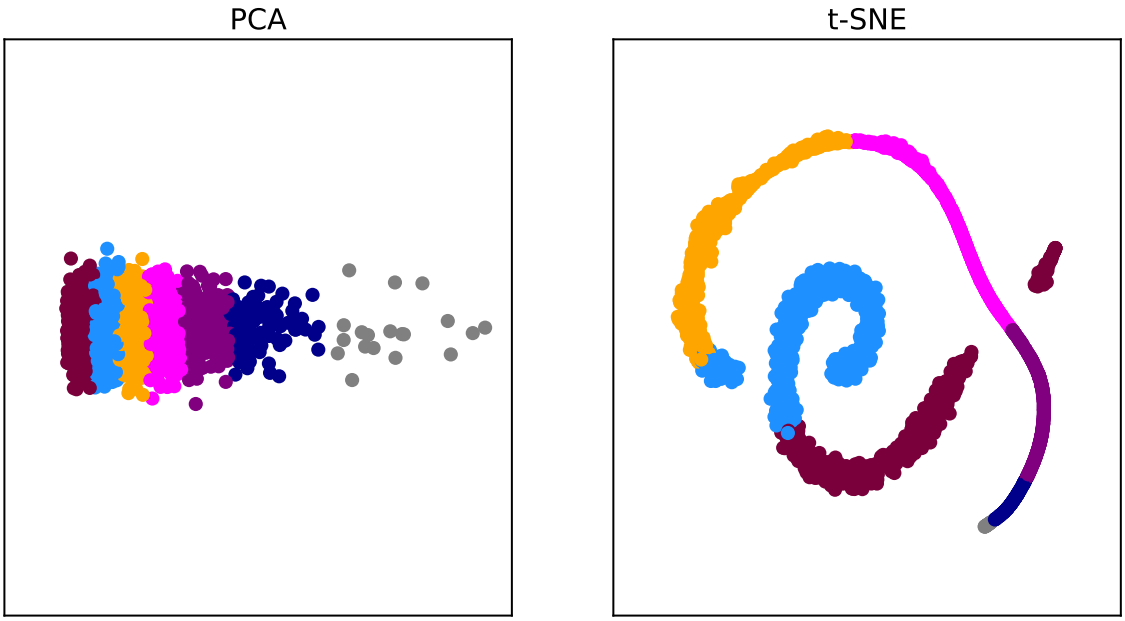

Figure I.3.13: ***2D plots obtained with PCA on the left and t-SNE on the right for the best partition obtained with the K-means algorithm and session embeddings as inputs for System 50 (App 2).***

I.3.9.2 Cluster Size and Session Length

Figures I.3.14 and I.3.15 show the session length distribution as well as the number of sessions contained in each cluster for system 30 and system 50 (the length of a session corresponding to the number of actions it contains). Cluster 0 corresponds to the statistics obtained with the whole dataset, for reference. In both cases, 1 or 2 very small clusters seem to contain particularly long sessions: cluster 8 contains 9 sessions in the case of system 30 and clusters 4 contains 18 sessions in the case of system 50. The other clusters contain a more balanced number of sessions.

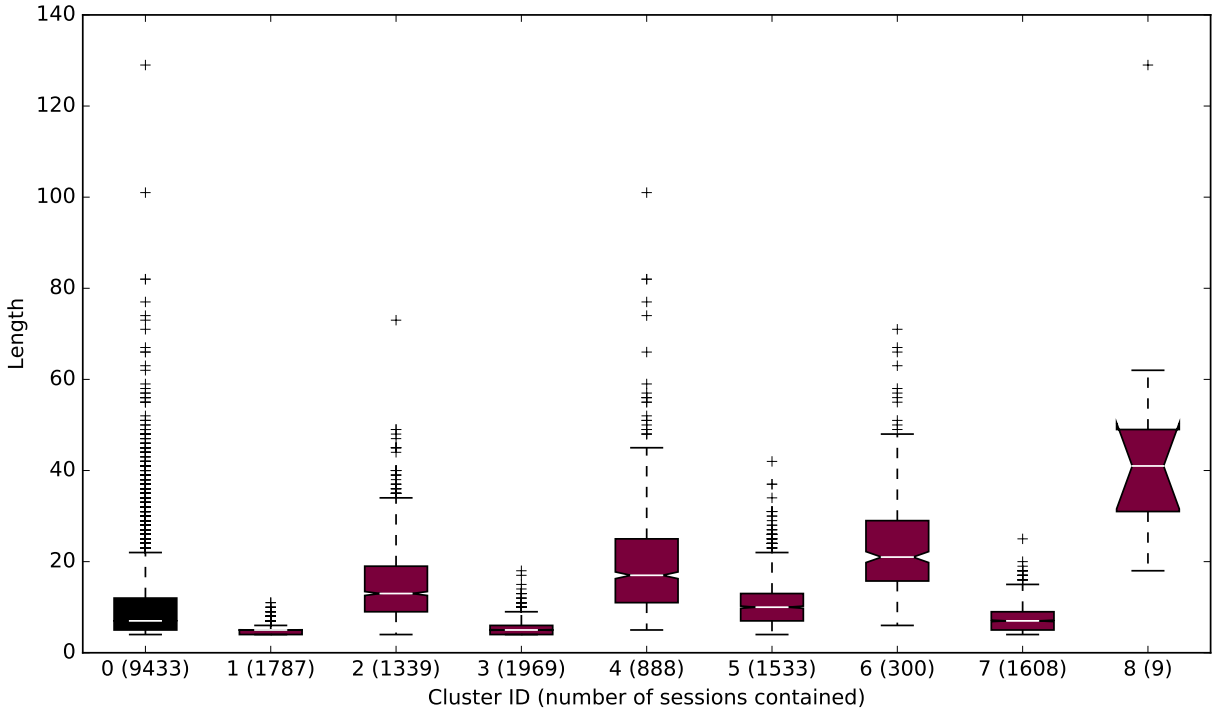

Figure I.3.14: System 30 session length distribution for each cluster. In parenthesis, the number of sessions contained in each cluster. Cluster 0 represents the whole dataset.

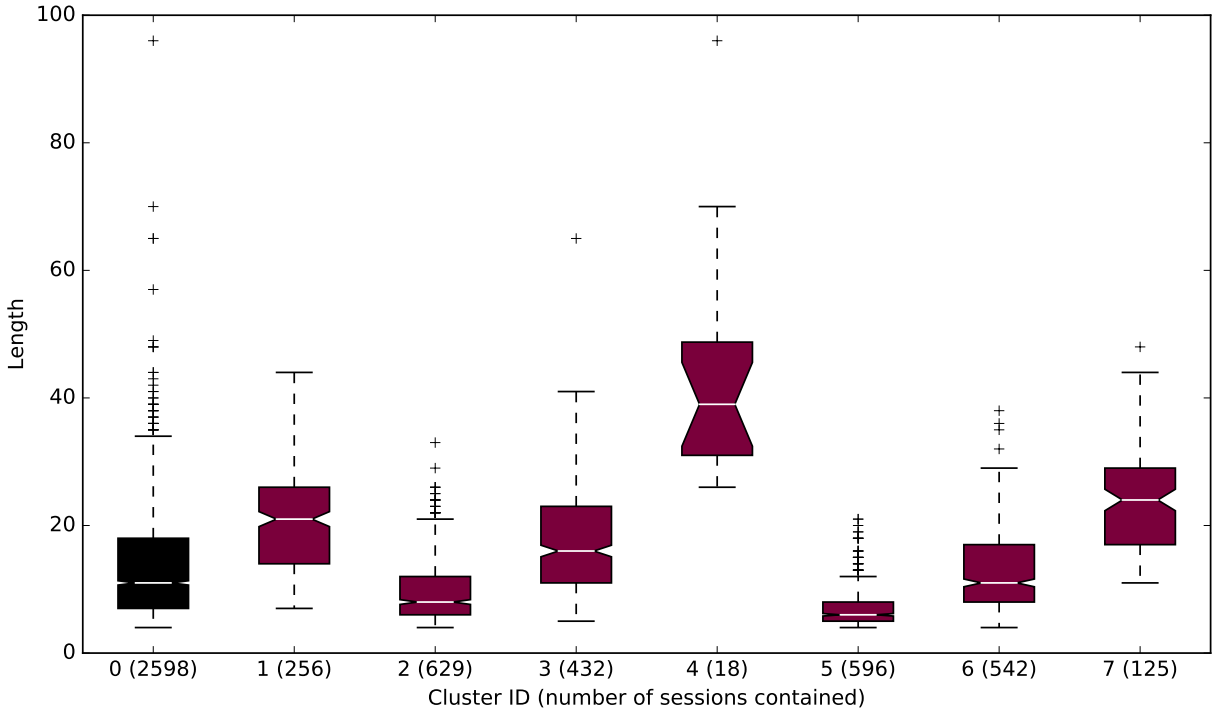

Figure I.3.15: System 50 session length distribution for each cluster. In parenthesis, the number of sessions contained in each cluster. Cluster 0 represents the whole dataset.

I.3.9.3 Workflow Characterization

We propose to characterize each cluster using a process mining visualization tool, namely the Inductive Visual Miner [Leemans, 2017] which is available in the ProM framework [1]. The principle is to represent all the sessions contained in a cluster by a process tree. Each node stands for a user action and the weighted edges represent transitions from one action to the other. Thus, this model enables to represent each workflow performed in a cluster from its beginning to its end using multiple branches in case of variations in the navigation path. To facilitate the visualization, only the most frequent nodes and paths are filtered (we take a threshold of 0.4 – meaning that events occuring in less than 40% of the sessions are discarded). We focused our analysis on the clusters obtained on system 30 in which an oncology application is used. Physicians use this application to find a tumor or to ensure that there is none. For this purpose they use tools dedicated to the navigation through images and the discovery of a tumor is generally characterized by the use of a measuring tool to determine its size. For this system in particular we observe 2 clusters: 1 and 3 containing very short sessions compared to the other clusters (see Figure I.3.14). Figure I.3.16 shows the characteristic workflows obtained for these 2 clusters. They seem to correspond to easy images reviews, since there is no loop in the workflows. They slightly differ from each other by the use of zoom (amm) and rotate (all) tools. Indeed, these analysis tools are most frequently used in cluster 3. Moreover, we observe that when these analysis tools are used, the discovery of the tumor, associated to the measure distance tool (amf) tends to be done at the beginning of the session (cluster 3) while it is rather done at the end when they are less used (cluster 1). This difference might correspond to two different users (we do not know how many users are working on the same system) or to the quality of the images the user is reviewing. Indeed, if the images were taken very precisely, the tumor might be straightforwardly detected and the remaining actions probably correspond to the verification of the absence of other anomalies, while otherwise some preliminary actions are needed to find the proper location of the tumor. Figure I.3.17 shows the process tree obtained for cluster 4. Loops can be observed meaning that the users repeat several steps during the images review. In fact, the used tool suggests that he compares an exam to a previous one (ald). These workflows probably correspond to follow-up exams. The same analysis performed on cluster 6 containing longer sessions shows that users of this clusters tend to not properly exit the current session and reload images probably from the next patient in the same session, which is not the recommended way to process. This type of information is very precious since it might help to know on which features to focus during user training for example.

[1] http://promtools.org

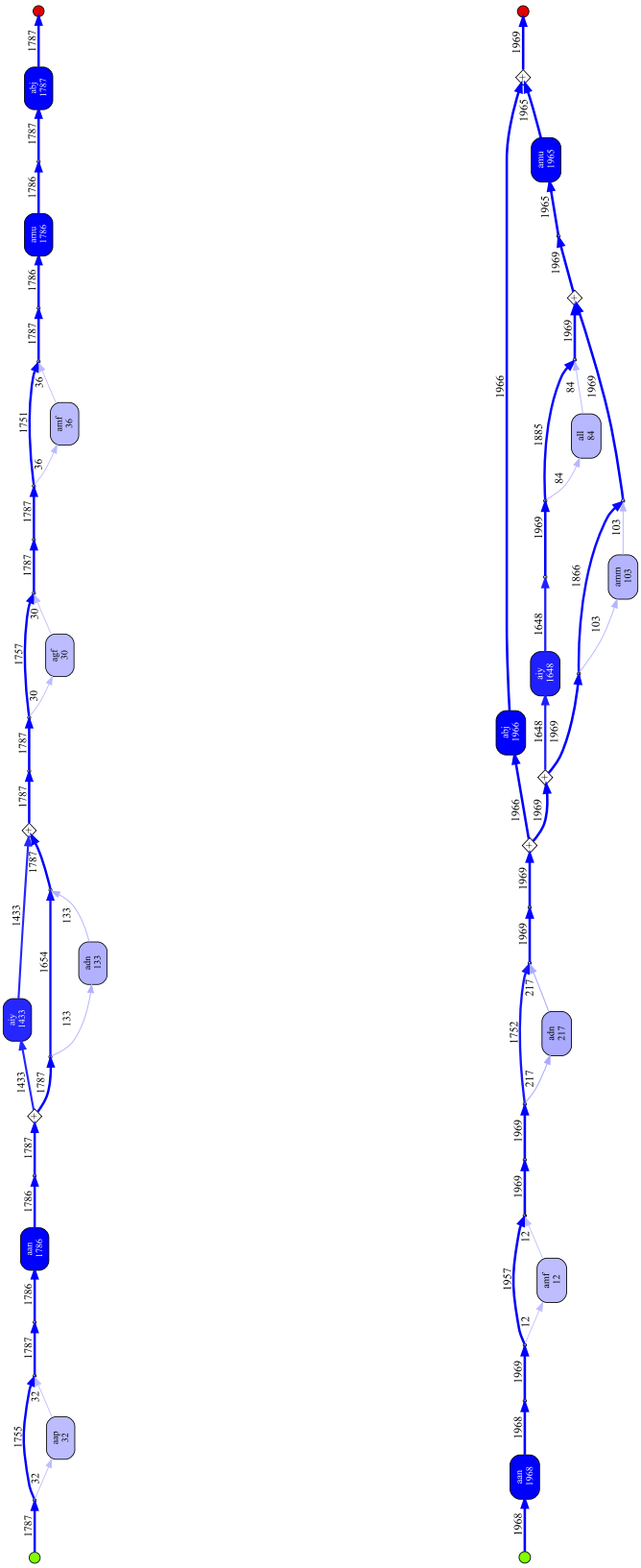

Figure I.3.16: System 30. Cluster 1 (on the left) and cluster 3 (on the right) workflow representations.

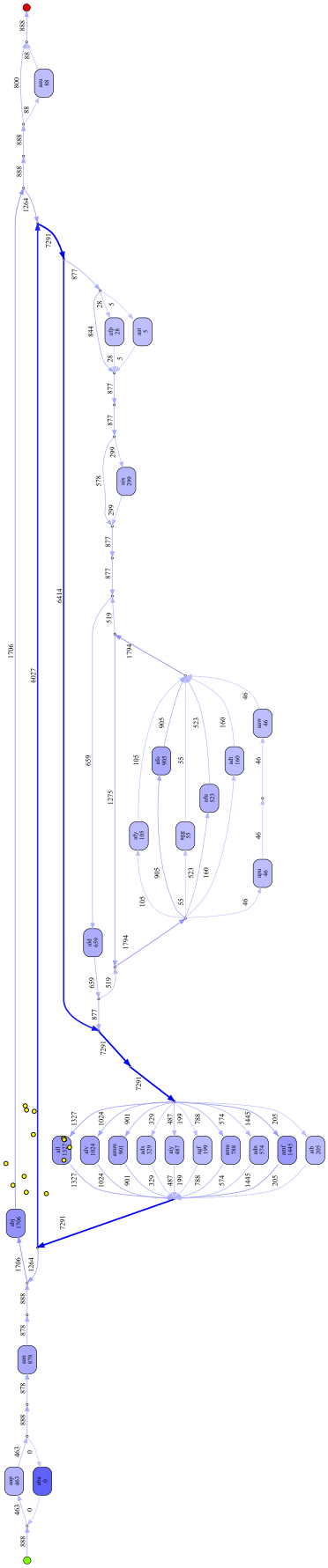

Figure I.3.17: ***System 30. Cluster 4 workflow representation.***

I.3.10 Discussion

In this chapter we tackled the user sessions clustering task. For this purpose we proposed to compare 3 different input representations coming from the text mining field, the first 2 being based on the well-known bag-of-words model whereas the last one consists in distributed representations. These different session transformations were tested with 5 clustering algorithms. Experiments conducted on 2 systems showed that in both cases, the K-means algorithm fed with session embeddings returned the best partitions. We used the Paragraph Vector based on Distributed Bag-of-Words architecture from the Doc2vec algorithm to compute session embeddings. Further improvements might involve the evaluation of other types of embeddings such as sequence to sequence autoencoders [Sutskever et al., 2014] or recent work based on deep clustering [Caron et al., 2018]. In depth cluster analysis using process mining visualization tools enabled to confirm the consistency of the different clusters although complementary information such as the images reviewed would help to improve this interpretation. The knowledge provided by this study is valuable in particular for application specialists who work closely with customers to improve the software. Indeed, this method is useful to have an overview of users' habits and from there, either to give them feedback in order to allow them to optimize their software use or to accordingly bring modifications to the software features and interface.

Part II

Dynamic Monitoring of Software Use

Chapter II.1

Sequence Learning

USER WORKFLOWS might be represented as sequential data and thus can be used to perform sequence prediction tasks. In this part we aim at dynamically monitoring software use. Given user history of actions in the interface, we will address two distinct industrial issues: software crash risk detection (see Part II, Chapter 2) and next action prediction (see Part II, Chapter 3). Both proposed methods take advantage of the recurrent structure of Long Short Term Memory neural networks to capture dependencies among our sequential data as well as their capacity to potentially handle different types of input representations for the same data.

II.1.1 Formalism

Let Σ be the set of possible actions in the user interface and u_a its cardinal. A user sequence of length n is an element of Σ^n. We will denote $X = x_1 x_2 x_3 \dots x_n$, with each $x_i \in \Sigma$ an elementary user action and $\Psi(X)$ a function over Σ^n onto Σ', a set of possible classes. Let u_c be the cardinal of Σ'. For a user sequence X of length n and for $r \in [\![1, n]\!]$, we will denote by X_r, the prefix sequence of length r: $X_r = x_1 x_2 \dots x_{r-1} x_r$. By abuse of notation we denote similarly the random variables in Σ or Σ^r and their actual realizations. As can be seen in Figure II.1.1, the problem of sequence prediction consists in computing the probability for each possible class in Σ' to be the next event class given an input prefix X_r. We will consider specifically two examples. In Chapter II.2 we take:

$$\Psi(X) \equiv S(X) = \begin{cases} 1, & \text{if session } X \text{ ends with a crash} \\ 0, & \text{otherwise} \end{cases}$$

and in Chapter II.3, we consider:

$$\Psi(X) \equiv \Psi_k(X) = x_k.$$

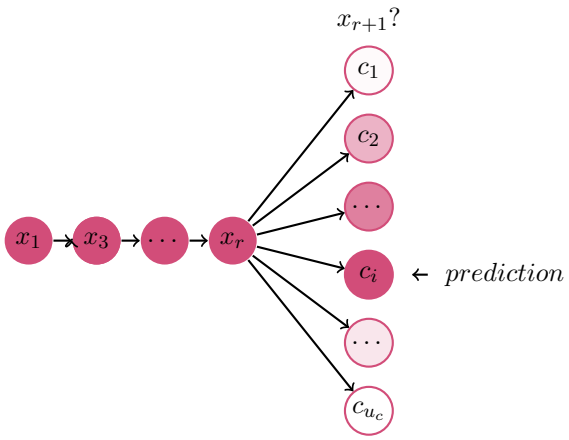

***Figure II.1.1:** Sequence prediction task.*

These values define an output probability distribution over Σ' where $\mathbb{P}(\Psi(X) = c_i|X_r)$ denotes the probability of class i to be the next event class given X_r. Thus in the case of crash monitoring, $\mathbb{P}(S(X)|X_r)$ denotes the probability that the session will crash given prefix X_r and in the case of next action prediction, $\mathbb{P}(\Psi_{r+1}(X)|X_r)$ denotes the probability for each user action to be the next one given prefix X_r.

The prediction corresponds to the event class having the highest value and is given by:

$$\hat{y}_r = \text{argmax}_{1 \leq i \leq u_c} \mathbb{P}(\Psi(X) = c_i|X_r).$$

This formulation also enables to predict the k most likely classes, as it might be of interest in some applications.

II.1.2 Recurrent Neural Networks

Numerous algorithms have been proposed to tackle the sequence prediction task [Gueniche et al., 2013]. Many of them make the Markovian assumption and only use the last events to compute a prediction. Recurrent neural networks [Jozefowicz et al., 2015] are composed of a recurrent hidden state which depends on the hidden state at the previous time, making them particularly well adapted to sequential data. Given a sequence $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_T)$, recurrent neural networks update the hidden state by iterating from $t = 0$ to T using the previous hidden state and the current input:

$$h_t = \begin{cases} 0, & \text{if } t = 0 \\ f_{\text{rec}}(h_{t-1}, \tilde{x_t}), & \text{otherwise} \end{cases}$$

where f_{rec} stands for a nonlinear function such as a logistic sigmoid with an affine transformation. However, recurrent neural networks are difficult to train since they suffer from vanishing

and exploding gradient problems [Bengio et al., 1994]. Therefore they are only able to capture short term dependencies. To overcome these difficulties, Hochreiter and Schmidhuber proposed the Long Short Term Memory (LSTM) architecture [Hochreiter and Schmidhuber, 1997]. This type of recurrent neural networks has proved to be particularly effective for the sequence learning task. Indeed, in addition to the usual hidden state present in standard recurrent neural network, an LSTM has a cell state in which it is able to store parts of the past information it learned as relevant while standard recurrent neural networks are not able to distinguish which information is useful and which information is not. The LSTM does so using three gates, namely the input gate, the forget gate and the output gate. Each of them is composed of a sigmoid layer which controls the memory that will be added or removed from the cell state. Although many LSTM variants have been proposed [Greff et al., 2016], we use the architecture with peephole connections [Gers and Schmidhuber, 2000]. Introduced by Gers and Schmidhuber, peephole connections correspond to connections from the cell to the gates and enable to better learn precise timings giving the cell a way to control the gates.

The LSTM model with one hidden layer is trained to map an input sequence $\tilde{x} = (\tilde{x}_1, \dots, \tilde{x}_T)$ to an output sequence $\tilde{y} = (\tilde{y}_1, \dots, \tilde{y}_T)$ from $t = 1$ to T by iterating over the following equations [Sak et al., 2014], the cell and the hidden states being initialized to zero vectors:

$$
\begin{aligned}
f_t &= \sigma(W_{fx}\tilde{x}_t + W_{fh}h_{t-1} + W_{fc}c_{t-1} + b_f) && \textit{forget gate} \\
i_t &= \sigma(W_{ix}\tilde{x}_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i) && \textit{input gate} \\
\tilde{c}_t &= \gamma(W_{cx}\tilde{x}_t + W_{ch}h_{t-1} + b_c) && \textit{new candidate} \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t && \textit{cell state} \\
o_t &= \sigma(W_{ox}\tilde{x}_t + W_{oh}h_{t-1} + W_{oc}c_t + b_o) && \textit{output gate} \\
h_t &= o_t \odot \gamma(c_t) && \textit{hidden state} \\
\tilde{y}_t &= \phi(W_{hy}h_t + b_y) && \textit{prediction}
\end{aligned}
$$

$\tilde{x}_t \in \mathbb{R}^{D_x}$, and $\tilde{y}_t \in \mathbb{R}^{D_y}$, where D_x and D_y are respectively input and output dimensions.

f, i, o respectively stand for the forget, the input and the output gates. c and h are respectively the cell and the hidden states which are of the same size as the gates: $f_t, i_t, o_t, c_t, h_t \in \mathbb{R}^H$, where H is the size of the hidden state of an LSTM unit.

The symbol W is used to denote weight matrices (e.g., W_{fx} is the matrix of weights from the forget gate to the input). $W_{fx}, W_{ix}, W_{cx}, W_{ox} \in \mathbb{R}^{H \times D_x}$, $W_{fh}, W_{ih}, W_{ch}, W_{oh} \in \mathbb{R}^{H \times H}$ and W_{fc}, W_{ic}, W_{oc} are diagonal weight matrices in $\mathbb{R}^{H \times H}$ for peephole connections. The b terms denote bias vectors (e.g., b_i is the input gate bias vector) and $b_f, b_i, b_c, b_o \in \mathbb{R}^H$. The matrix of weights from the hidden state to the prediction, W_{hy}, and the prediction bias, b_y, are respectively in $\mathbb{R}^{D_y \times H}$ and $\mathbb{R}^{D_y}$.

$\odot$ denotes the element-wise product. σ stands for the sigmoid function, γ is the cell input and output activation function, $\tanh$ here. ϕ denotes the network output activation used to obtain

the final prediction probability distribution, the softmax function in our case.

As described in the equations, the forget gate f, composed of a sigmoid layer will determine which information will be removed from the cell state. To do so, it looks at the previous hidden and cell states and the current input values ranging between 0 and 1 to decide whether the information should be removed or not. On the same scheme, the input gate i is responsible for deciding which information will be stored in the cell state. The current cell state is then obtained by discarding information the forget gate decided to remove from the previous cell state and by adding new information based on the filtering of the input gate regarding the current cell candidate. The output gate o then looks at the new cell state, the previous hidden state and the current input to select useful information to compute the new hidden state as well as the final prediction $\tilde{y}_t$. For classification problems, the model is generally trained to minimize average cross-entropy loss.

These mechanisms make LSTMs an effective solution for sequence prediction problems in data science industry: speech recognition [Graves and Jaitly, 2014], search query prediction on keyboard [Cao et al., 2017] or translation tasks [Wu et al., 2016] to cite a few.

II.1.3 Methodology

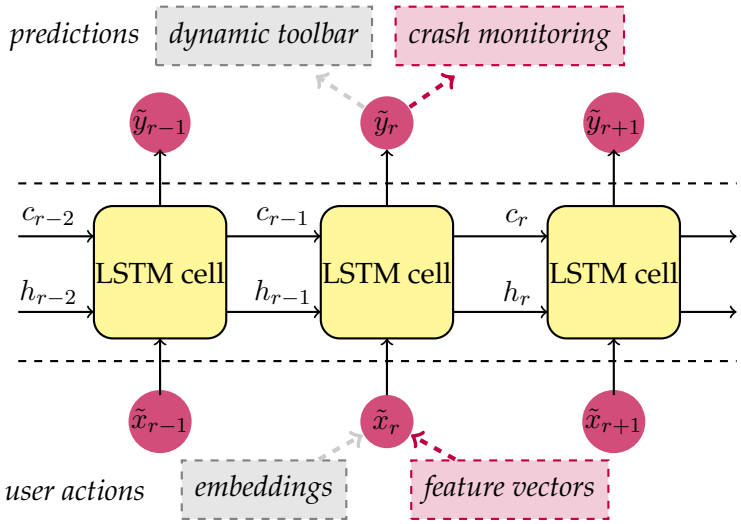

***Figure II.1.2:** The proposed LSTM neural network for user action prediction and crash monitoring.*

We propose to tackle two sequential problems using Long Short Term Memory neural networks [Adam et al., 2019]. The crash monitoring task, framed in pink, and the next action prediction task, framed in gray in Figure II.1.2. The specificity is that two different representations of the sessions will be used as inputs of the network: feature vectors for the crash monitoring task and action embeddings for the user action prediction task. Reasons for these choices, related work, as well as detailed methodologies to each application, namely crash risk detection and

user action prediction, will be presented in more details in the dedicated chapters.

Chapter II.2

Crash Risk Monitoring

ANTICIPATING software crashes might enable to save a precious amount of time in particular in the case of interventional applications, in which patient's safety is put at stake. In this chapter we introduce a real-time crash risk monitoring method, computing a crash probability at each new user action. Should this probability exceed a certain threshold, an automatic backup system could be triggered – thus allowing the user not to lose their work in progress. Several factors such as the system status (number of applications running, memory availability, dataset type, etc.) are likely to cause failures. In this work, we make the assumption that crashes are caused by patterns of practitioners' actions in the user interface. Motivated by the above, we turn to LSTM networks to dynamically monitor the sequence of user actions, by taking advantage of their recurrent structures and by adapting the representation of user sequences as LSTM inputs. Section II.2.1 presents the related work. The problem formulation is given in section II.2.2. Section II.2.3 presents the input representation. Experiments are conducted on 5 systems having an important crash rate. Results are provided in section II.2.4.

II.2.1 Related Work

To tackle the issue of crash prediction, we draw extensively on methodologies used for the sequence classification task [Xing et al., 2010], since we aim at labeling user sessions as normal sessions or sessions that might end with a crash, in real-time. We particularly focus on pattern based classification [Bringmann et al., 2009] which is adapted to our sequential action logs. The general principle is to use patterns as features which can then be used as inputs of classifiers. The method ordinarily involves the three following steps: pattern mining (detect frequent patterns), feature selection (select the ones with the best predictive power) and finally the classifier construction. These steps largely differ in existing works. Indeed, various kinds of patterns might be mined including itemsets [Cheng et al., 2007] as well as subsequences [Lo et al., 2009]. The cohesion of the patterns might also be taken into account. Thus, [Zhou et al., 2016] propose to define the interestingness of patterns by evaluating their support as well as the cohesion of the items making up the pattern. As we concluded in Part I, Chapter 2 that subsequences of user actions were more appropriate than itemsets (for which the order

of the actions is not taken into account) and cohesive-subsequences (for which a maximal gap constraint is imposed between consecutive actions of the subsequence), we propose to reuse subsequences as features of our classifiers. While the Fisher score combined with a coverage threshold to determine the most discriminative patterns might be used [Lo et al., 2009, Cheng et al., 2007], the earliness of the pattern is also taken into account to select features for early prediction [Xing et al., 2008] and only select features appearing frequently and early in the training sequences. We propose to select significant patterns among the crashes using the binomial test described in Part I, Chapter 2. The last step consists in the training of a classification method. This task may involve rule based classification such as in [Zhou et al., 2016, Xing et al., 2008] or state-of-the-art supervised learning classifiers, such as support vector machines or decision trees [Lo et al., 2009, Cheng et al., 2007].

Recent works also proposed to apply deep learning algorithms, and in particular LSTM networks to anomaly dectection [Du et al., 2017] and failure prediction [Zhang et al., 2016]. In the latter case, they use sequence alignment algorithm to perform pattern recognition among their system logs. These patterns are then fed to an LSTM which outperforms standard classification methods (Logistic Regression, SVM and Random Forest). Unlike what is proposed in this work, we suggest to only use patterns coming from the crash class as features, since they are the most discriminative, while [Zhang et al., 2016] use both normal and abnormal entries.

For our specific problem, we do not know precisely when a crash is triggered. Therefore, we propose to use feature vectors composed by significant crash probability individual actions as inputs of an LSTM neural network, making the assumption that its ability to learn long term dependencies will enable the detection of crash patterns.

II.2.2 Problem Formulation

Let Σ be the set of possible actions in the user interface and u_a its cardinal. A user sequence of length n is an element of Σ^n. We will denote $X = x_1x_2x_3 \dots x_n$, with each $x_i \in \Sigma$ an elementary user action. For a user sequence X of length n and $r \in [\![1, n]\!]$, we will denote by X_r, the prefix sequence of length r: $X_r = x_1x_2 \dots x_{r-1}x_r$. We denote by $S(X)$ the class of session X:

$$S(X) = \begin{cases} 1, & \text{if session } X \text{ ended with a crash;} \\ 0, & \text{otherwise.} \end{cases}$$

The objective is to determine a model $\mathcal{C}_r$:

$$\begin{aligned} \mathcal{C}_r \ : \ & \Sigma^r \to [0,1] \\ & X_r \mapsto \mathbb{P}(S(X) = 1 | X_r) \end{aligned}$$

where $\mathbb{P}(S(X) = 1|X_r)$ denotes the probability of session X to eventually crash, given that its prefix sequence of length r is X_r. The highest crash probability predicted during the session will be used for model evaluation (see also Figure II.2.1). The family of models $\mathcal{C} = (\mathcal{C}_r)_{r \geq 1}$ defines a dynamic model for crash risk detection.

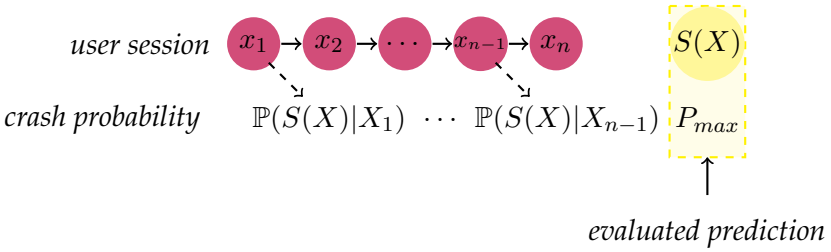

***Figure II.2.1:** Evaluation of crash risk monitoring: the highest crash probability predicted during the session will be used for the score computation.*

II.2.2.1 Loss

The $\mathrm{LSTM-FV}$ model is the dynamical model $(\mathcal{C}_r)_{r\geq 1}$, predicting at each user action i a crash probability $\tilde{y}_i$. For each session X, an elementary binary cross entropy loss is computed:

$$L_c(X) = -\sum_{i=1}^{n_X}(y_i \log \tilde{y}_i + (1-y_i)\log(1-\tilde{y}_i))$$

where n_X stands for the total number of actions in session X and:

$$y_i = \begin{cases} 1 \text{ if the } i\text{-th action is followed by a crash,} \\ 0 \text{ otherwise.} \end{cases}$$

Finally the network is trained by minimizing the complete loss $\mathcal{L}_c$:

$$\mathcal{L}_c = \sum_{X\in\mathcal{T}} L_c(X)$$

where $\mathcal{T}$ denotes the training set.

II.2.3 Input Representation

We propose two types of LSTM models for the crash risk detection task: one-hot encoded vectors and feature vectors.

II.2.3.1 One-hot Vectors

The first model, called $\mathrm{LSTM-1HOT}$, uses as inputs one-hot vectors whose dimension is equal to the number of possible elementary actions. Let $x \in \Sigma$ be an elementary action. We classically define its one-hot representation $x^{\mathrm{1HOT}} \in \{0,1\}^{u_a}$, with only one component different from 0, as follows:

$$x^{\mathrm{1HOT}_k} = \begin{cases} 1, \text{ if } x = a_k \\ 0, \text{ otherwise} \end{cases}$$

where a_k, for $k \in [\![1, u_a]\!]$ are all the possible elementary actions in Σ and x_k^{1HOT} the k-th component of the one-hot vector.

II.2.3.2 Feature Vectors

We also propose an LSTM model fed with feature vectors, $x^{\text{FV}} \in \mathbb{N}^{F_\tau}$, we call it LSTM − FV. This representation is inspired by pattern based sequence classification methods [Xing et al., 2010, Zhou et al., 2016]. The principle is to transform user sessions into a vector of features. At each new user action, feature vectors are incrementally updated, by counting the number of occurrences of each feature in the current prefix X_r (see Figure II.2.2).

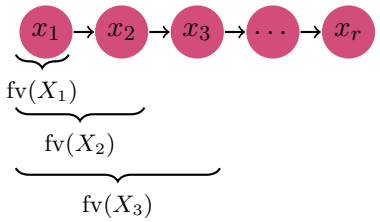

Figure II.2.2: Feature vector computation.

To select actions which will be used as features we propose to use the binomial test described in Part I, Chapter 2 to determine significant crash actions with a threshold τ. Here, only the actions having a *p-value* smaller than the significance level τ will be selected.

II.2.4 Experiments

II.2.4.1 Data

Table II.3.3 shows the crash rate for each system in the database. As in Part I, Chapter 2, we will test our method on the 5 systems having a crash rate higher than 3 %. Since systems with crash rates above this value are generally subject to complaints, the proposed solution could be implemented temporarily on these systems, pending the resolution of the bug.

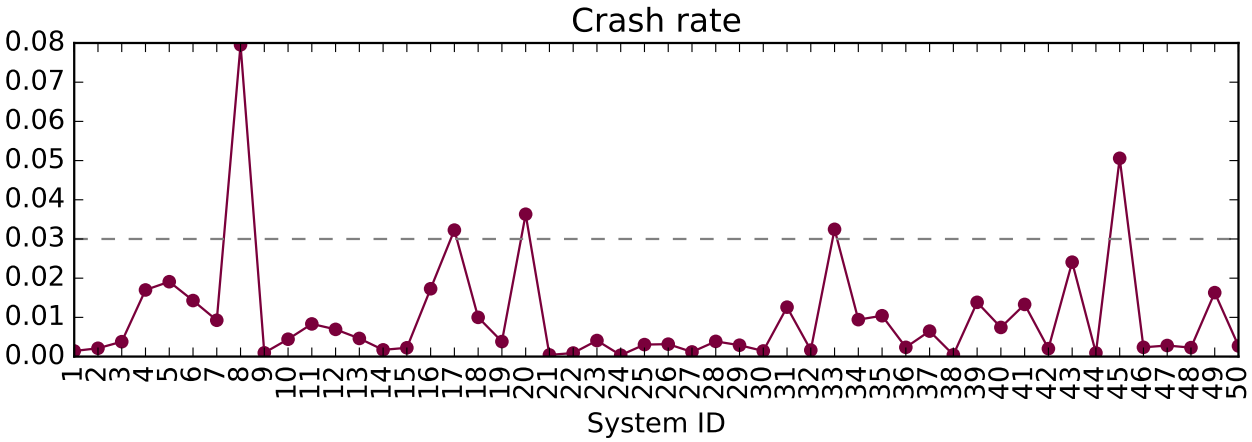

Figure II.2.3: Crash rate for each system of the database, in gray the critical threshold of 3 %.

As crashes occurring after a long range of actions are more disturbing for the users, we only kept normal sessions and crashes containing at least seven user actions to conduct our exper-

iments. Shorter sessions were removed since we are not particularly interested in monitoring crash risk for them. Table II.2.1 shows for each studied system the resulting number of sessions as well as the associated crash rate p_0. The crash rate remains slightly the same except for system 20, where it decreases to 1.6% while it was higher than 3% with all the sessions, indicating an important number of very short crashes.

System ID	$\lvert\mathcal{D}_7\rvert$	p_0
8	2746	7.5
17	3710	3.5
20	2032	1.6
33	1716	3.4
45	697	4.9

***Table II.2.1:** Total number of sessions containing 7 actions or more $\lvert\mathcal{D}_7\rvert$ for each studied system and associated crash rate.*

II.2.4.2 Baseline Methods

To assess the performance of $\mathrm{LSTM-FV}$ fed with feature vectors, we compare to state-of-the-art supervised learning methods including Support Vector Machine such as in [Lo et al., 2009, Cheng et al., 2007] and Random Forest classifier. Note that the Random Forests classifier was preferred to Decision Trees since it provided better results. For fair comparison to the LSTM network, which is able to handle long term dependency among sequential data, each baseline will be fed with 2 types of input representations.

Inputs

As for the LSTM network, baselines will be fed with feature vectors composed of elementary actions performed in crashes being significant at level τ. This type of inputs will be denoted FV. To make the comparison more meaningful, we propose a second type of inputs where crash patterns are added to the feature vector composed of elementary actions being significant. As for the elementary actions, only the patterns being significant are added to elementary actions to compose the features. This second type of inputs will be denoted FVP. This results in the following 4 baseline methods.

Support Vector Machine with Feature Vectors ($\mathrm{SVM-FV}$): a support vector machines model using feature vectors composed of elementary actions being significant.

Support Vector Machine with Patterns in Feature Vectors ($\mathrm{SVM-FVP}$): a support vector machines model using feature vectors composed of significant elementary actions and patterns.

Random Forest with Feature Vectors ($\mathrm{RF-FV}$): a random forest classifier using feature vectors composed of elementary actions being significant.

Random Forest with Patterns in Feature Vectors ($\mathrm{RF-FVP}$**):** a random forest classifier using feature vectors composed of significant elementary actions and patterns.

II.2.4.3 Performance Evaluation

ROC curves

As can be seen in Table II.2.1, the crash rate amounts to a few percents meaning that the sessions prematurely ended by a crash actually account for a very small portion of the total number of sessions. Many techniques were developed to tackle the challenge of skewed data in classification problems: sampling, ensemble methods, synthetic sample generation, etc. [Chawla, 2005]. As in our case not all errors are equal, we have decided to evaluate our results using the area under ROC curves [Fawcett, 2004]. We predict a crash when $P_{\max} = \max(\tilde{y_i}) \geq P_{\text{thresh}}$. The ROC curve is thus obtained by varying P_{thresh} from 0 to 1. ROC curves show the trade-off between the True Positive Rate, denoted TPR, indicating the rate of detected crashes and the False Positive Rate, denoted FPR, indicating the rate of normal sessions announced as crashes. FPR and the TPR are computed as follows:

- $\mathrm{FPR} = \dfrac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$
- $\mathrm{TPR} = \dfrac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$

where TP, FP, TN and TP respectively stand for the number of True Positives, False Positives, True Negatives and True Positives (see Table II.2.2 for details). In the general case, we want to avoid unnecessary backups for memory reasons in the case of normal sequences and we accept to suffer from some crashes. This translates into a low False Positive Rate and a satisfying True Positive Rate.

		Actual	
		Crash	**Normal Session**
Predicted	**Crash**	True Positive (TP)	False Postitive (FP)
	Normal Session	False Negative (FN)	True Negative (TN)

Table II.2.2: Confusion matrix.

In the case of interventional radiology, on the other hand, we would prefer a true positive rate close to 1.

II.2.4.4 Experimental Setup

We conducted experiments on each system individually, and used for each system the 80% oldest sessions for training and the 20% most recent for the test set. Results presented correspond to the average AUC obtained after 10 runs.

Baselines

In order to make the baselines as competitive as possible, we use a grid search approach. In order to select the significant actions we set the threshold probability to p_0 (the higher threshold

probability was too selective) and tuned the significance threshold τ with $\tau \in \{0.01, 0.02, 0.05, 0.1\}$, the penalty parameter C of the Support Vector Machines classifier with $C \in \{0.1, 10, 100, 1000\}$ and the maximum depth of the trees m in the Random Forest classifier with $m \in \{25, 50, 75, 100\}$. In the case of $\mathrm{SVM} - \mathrm{FVP}$ and $\mathrm{RF} - \mathrm{FVP}$ there is an additional hyperparameter to tune corresponding to the threshold probability rp_0 to select significant patterns. For that, we set the significance threshold to 0.01 and tuned the threshold probability rp_0 with $r \in [\![1, 10]\!]$. We mined the frequent subsequences from crash sessions with the Bide algorithm and set the support $\min_{\mathrm{sup}}$ to 5%.

We use a linear kernel in the Support Vector Machine classifier (other kernels were providing worse results) and 100 trees in the Random Forest classifier. We report the best results obtained on the test set.

The Support Vector Machine classifier with probability estimates and Random Forest classifier are the implementations of the Scikit-Learn Python package [Pedregosa et al., 2011].

LSTM

$\mathrm{LSTM} - \mathrm{FV}$ is trained and implemented in Tensorflow [Abadi et al., 2015] with the RMPSProp optimizer [Tieleman and Hinton, 2012]. The LSTM networks used with one-hot vectors and feature vectors have the same architecture. Hidden and cell states are initialized to zero at the beginning and reset at the beginning of every user session. We use minibatches of size $B = 32$. Network weights are initialized using a Glorot initialization procedure [Glorot and Bengio, 2010] and are unrolled for $T = 200$ steps. To prevent the LSTM network from over-fitting, we use early stopping [LeCun et al., 2015] and interrupt learning when the validation error (computed on 20% of the training samples) does not decrease for 10 epochs.

We selected the dimension of the hidden state as well as the significance threshold on the test set. The objective here is to provide default hyperparameters values which might be used in the future. The selection of the number of units and that of the significance threshold are presented in section II.2.4.5.

II.2.4.5 LSTM Tuning

We first selected the optimal dimension of the hidden state on $\mathrm{LSTM} - \mathrm{1HOT}$ and then the significance threshold τ providing the best results for $\mathrm{LSTM} - \mathrm{FV}$.

LSTM-1HOT

Table II.2.3 shows the average area under the curve obtained on the 5 studied systems for the dimension of the hidden state H varying between 25 and 300. As observed, the best results are obtained with $H = 50$ which is selected to conduct the following experiments.

H	25	50	100	150	200	250	300
Average AUC	0.681	0.683	0.662	0.661	0.646	0.662	0.652

Table II.2.3: ***Average area under the curve obtained on the 5 systems studied with*** $\mathrm{LSTM}-\mathrm{1HOT}$ ***and dimension of hidden state*** H ***varying between 25 and 300.***

LSTM-FV

Table II.2.4 shows the average area under the curve obtained on the 5 systems studied for a threshold probability equal to p_0 and a significance threshold τ varying between 0.01 and 0.1. The number of corresponding significant actions is provided in Appendix D. The best results are obtained with $\tau = 0.01$. This value will be used for the rest of the experiments.

τ	0.01	0.02	0.05	0.1
Average AUC	0.763	0.749	0.739	0.722

Table II.2.4: ***Average area under the curve obtained on the 5 studied systems with*** $\mathrm{LSTM}-\mathrm{FV}$ ***and significance threshold*** τ ***varying between 0.01 and 0.1.***

Threshold probability of $2p_0$ is too selective in the case of single actions and the very small number of significant actions does not allow the LSTM network to work as well as with features selected at threshold probability of p_0.

II.2.4.6 Results

Table II.2.5 shows the results obtained on the 5 systems with the 6 different methods. The baselines fed with additional patterns in feature vectors perform better than the baselines fed feature vectors composed of significant actions only. $\mathrm{LSTM}-\mathrm{FV}$ provides the best results on systems 8, 17 and 20 but it turns out that $\mathrm{LSTM}-\mathrm{1HOT}$ does slightly better on system 33. Moreover, the Random Forest Classifier fed with additional patterns is more efficient on system 45 for which very few sessions are available for training. On average, $\mathrm{LSTM}-\mathrm{FV}$ provides the best results. The LSTM network fed with one-hot encoded actions does not seem appropriate for this task.

For both systems containing the highest number of crashes (59 crashes in test for system 8 and 42 crashes in test for system 17), the $\mathrm{LSTM}-\mathrm{FV}$ model returns the best area under the curve. As explained previously, ideally, we would like our method to detect a satisfying rate of crashes without triggering too many false alarms. This means that we seek a model returning a True Positive Rate as high as possible in the region shaded in gray in Figures II.2.4 and II.2.5. As can be seen on Figure II.2.5, $\mathrm{LSTM}-\mathrm{FV}$ performs the best in the desired region. In this case, $\mathrm{LSTM}-\mathrm{FV}$ is able to detect 65% of the crashes while maintaining a False Positive Rate approximating 10%. However, Figure II.2.4 shows an example where the $\mathrm{LSTM}-\mathrm{FV}$ is not able to take a clear advantage over other methods. ROC curves indicate that only 40% of the crashes are anticipated when ensuring a false alarm rate of 20%.

System ID	SVM – FV	SVM – FVP	RF – FV	RF – FVP	LSTM – 1HOT	LSTM – FV
8	0.536	0.618	0.640	0.655	0.509	**0.666**
17	0.834	0.835	0.845	0.849	0.704	**0.877**
20	0.900	0.901	0.897	0.905	0.860	**0.909**
33	0.601	0.631	0.631	0.663	**0.740**	0.738
45	0.709	0.720	0.717	**0.731**	0.600	0.627

Table II.2.5: AUC for each of the studied systems. The best values obtained for each system are in bold.

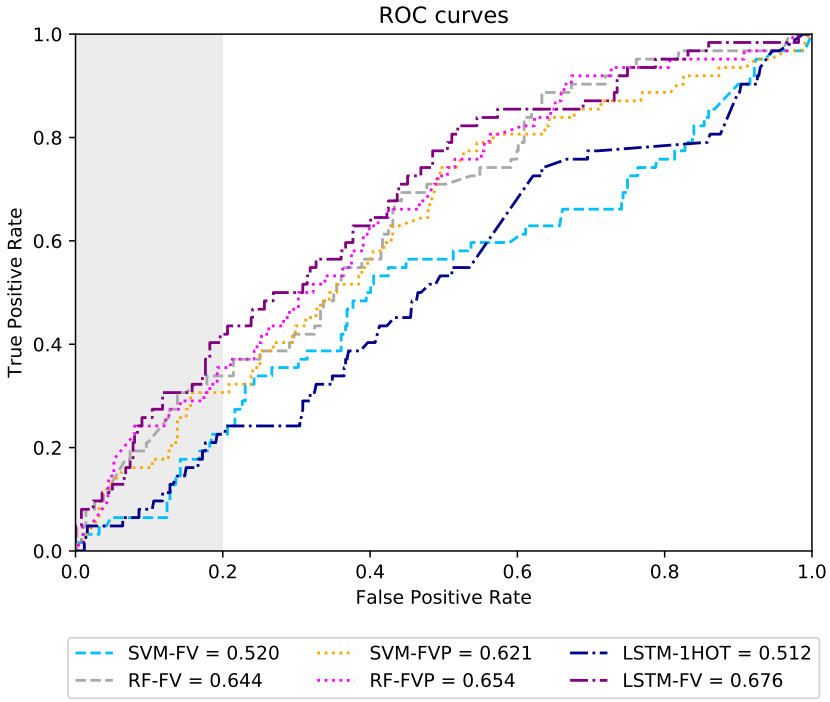

Figure II.2.4: ROC curves obtained for system 8. The gray area corresponds to the region where we want a high TPR.

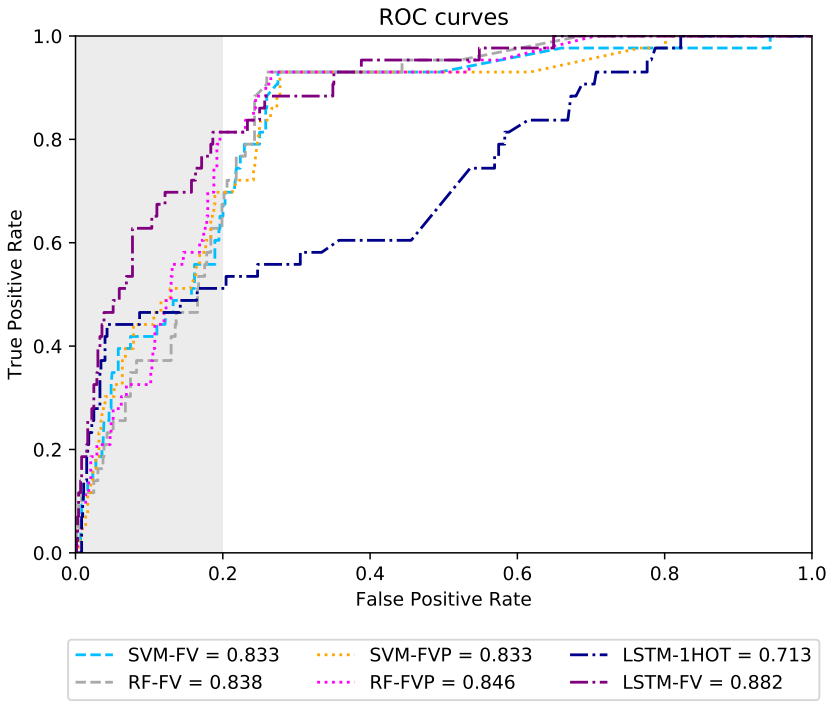

Figure II.2.5: ***ROC curves obtained for system 17. The gray area corresponds to the region where we want a high TPR.***

II.2.5 Discussion

To tackle the crash detection task, we proposed to feed an LSTM network with feature vectors composed of significant actions present in crashes. This representation outperforms on average baselines approaches. The average area under the curve obtained on the 5 systems is promising considering the scarcity of crashes in our datasets. As underlined by James Smith[1], "The dirty secret of software is that you cannot fix every single bug". Crash monitoring is thus a critical issue in all industrial codes.

Even if the priority is to understand the origin of the crashes and to highlight the crash signatures allowing to solve bugs, our future work will aim at increasing the performance of crash risk detection by combining user workflows and system parameters evolution logs.

[1]https://softwareengineeringdaily.com/2016/03/31/applic/

Chapter II.3

User Action Prediction

In order to speed up radiologists' workflow and diagnosis, software developers aim at providing them with an interface increasingly adapted to their patterns of use, despite the very strong variability in workflows which depend on their experiences and specialties. Our objective in this chapter is to develop methods for the prediction of next user action, which allows the enrichment of the interface with diverse features such as adaptive display or task automation [Adam et al., 2017a]. In our study, practitioners' workflows are structured around key actions which will be called *Tools* throughout the presentation of the method. This particular type of actions corresponds to the main steps of the patient images review. Indeed, the use of a *Tool* in the interface often triggers the appearance of more specific option panels. Our work focuses on the prediction of the next *Tool* that the user will use during his diagnosis. This prediction solution might take the form of a dynamic toolbar, proposing in real-time the k most likely next *Tools* needed by the user. The efficiency of the system is crucial if the software is used on a portable device with a reduced size user interface, which is more and more frequent. As a matter of fact, with fewer default options available, the prediction system must be as accurate as possible to avoid additional actions for the user to reach the *Tool* he intends to use. In many medical applications, the slightest time saving may be decisive. We present the related work in section II.3.1. The problem formulation is detailed in section II.3.2. The input representations are described in section II.3.3. We conducted experiments on 50 systems, whose results are provided in section II.3.4. Finally, we compare several training strategies in section II.3.5.

II.3.1 Related Work

Many sequence prediction algorithms have been provided in the Proactive User Interfaces research field [Hartmann and Schreiber, 2007]. They focus on online help, interface adaptation, content prediction or task automation, in order to simplify the interaction of the user with the interface via highlighted fields, hints, minimization of the interface to some specific functions, etc. For that purpose, an algorithm able to predict the next actions of the user based on his interaction history would be beneficial. Wide range of algorithms, mainly based on Markov

models have been proposed and applied to this domain. For instance, the *Incremental Probabilistic Action Modeling* method [Davison and Hirsh, 1998] makes the assumption that each action depends only on the last action and uses a first order Markov model which is updated after each prediction. Later, the *Jacobs Blockeel* [Jacobs and Blockeel, 2002] and *ActiveLeZi* [Gopalratnam and Cook, 2007] algorithms used mixed order approach to compute the most likely next action, whereas *FxL* [Hartmann and Schreiber, 2007] computes a prediction score by assigning different weights on different Markov order models. Quite recently, the *Frequency Vector* method [Moon et al., 2016] computes a prediction for the next user action by combining a frequency model with an embedded vector representation model. This method slightly improves *FxL* algorithm and outperforms *Incremental Probabilistic Action Modeling* algorithm as well as a recurrent neural network on datasets containing user behavior logs.

More generally, the prediction of the next item of a sequence is used in a wide variety of applications such as statistical language modeling, text analysis, web page recommendation, weather or time series forecasting, or customer purchases prediction [Begleiter et al., 2004]. State-of-the-art methods such as *Back-Off N-Gram Models* [Katz, 1987] or *All-Kth-Order-Markov models* [Pitkow and Pirolli, 1999] assume the Markovian property, that is to say that the next item depends only on the last occurring items. Hence, they do not use the whole information contained in the sequence.

Another approach consists in tackling the sequence prediction task as a classification problem using a multilayer perceptron for example [Zhao et al., 2017]. However, this type of technique still computes a prediction by only exploiting a fixed number of the previous actions.

To capture longer dependencies, we have focused on recurrent neural networks [Graves, 2013]. Lately applied to analogous sequence learning problems in the domain of Natural Language Processing (NLP), Recommender Systems or Web Search, they respectively perform next word prediction [Zaremba et al., 2014, Sundermeyer et al., 2012], purchased items prediction [Tan et al., 2016, Donkers et al., 2017] and user clicks on search engine results prediction [Borisov et al., 2016, Zhang et al., 2014]. In particular, we have worked with the Long Short Term Memory architecture, originally formulated by Hochreiter and Schmidhuber [Hochreiter and Schmidhuber, 1997]. This type of neural networks has proved to be particularly effective for this task. Indeed, in addition to the usual hidden state present in standard recurrent neural network, the LSTM network has a cell state in which it is able to store the parts of the past information it learnt as relevant. It does so using three gates, each of them composed of a sigmoid layer which controls the memory that will be added or removed from the cell state, as detailed in Part II, Chapter I, section II.1.2. To the best of our knowledge, LSTM recurrent networks have never been applied to the specific task of user action prediction. As we will present below, we propose a first model where the input action is encoded by a one-hot vector [Sundermeyer et al., 2012], which is the classic approach. This representation grows with the size of the vocabulary, in the case of NLP tasks for example, becoming computationally demanding. Moreover, the one-hot representation does not take into account possible similarities between

words. Distributed representations of words enable to overcome these limitations [Mikolov et al., 2013]. Thanks to the way they are computed, these lower-dimensional continuous vectors capture meaningful information about words. Thus, in the embedded space, words that often appear one after the other would be close to each other. Such representation is widely used in language modeling [Zaremba et al., 2014]. Similarly this distributed representation has been applied in the Recommender System field [Borisov et al., 2016, Greenstein-Messica et al., 2017] making the assumption that items occurring in the same context will also be spatially close in the embedded space. Here, we propose to apply this technique, which turns out to improve the performance of user action prediction task.

II.3.2 Problem Formulation

Let Σ be the set of possible actions in the user interface and u_a its cardinal. A user sequence of length n is an element of Σ^n. We will denote $X = x_1x_2x_3\dots x_n$, with each $x_i \in \Sigma$ an elementary user action. For a user sequence X of length n and $r \in [\![1, r]\!]$, we will denote by X_r, the prefix sequence of length r: $X_r = x_1x_2\dots x_{r-1}x_r$.

We consider a practitioner's session X of indefinite length and $X_r \in \Sigma^r$, the current prefix sequence of length r. The objective is to determine a model $\mathcal{A}_r$:

$$\begin{aligned}\mathcal{A}_r \ : \ & \Sigma^r \to [0;1]^{u_a} \\ & X_r \mapsto (\mathbb{P}(x_{r+1} = a_1|X_r), \cdots, \mathbb{P}(x_{r+1} = a_{u_a}|X_r))\end{aligned}$$

where a_i, for $i \in [\![1, u_a]\!]$, are all the possible elementary actions in Σ and $\mathbb{P}(x_{r+1} = a_i|X_r)$ denotes the probability of the $(r+1)$-th action in sequence X to be a_i, given that the prefix sequence of length r is X_r. The next action prediction at time r is denoted by $\hat{y}_r$ and thus given by

$$\hat{y}_r = \text{argmax}_{1\leq i\leq u_a} \left(\mathbb{P}(x_{r+1} = a_i|X_r)\right). \tag{II.3.1}$$

It may also be of interest to predict the k most probable actions, corresponding to the k highest probabilities given by $\mathcal{A}_r$. The family of models $\mathcal{A} = (\mathcal{A}_r)_{r\geq 1}$ defines a dynamic model for next action prediction.

As explained previously, we are only interested in predicting a particular type of actions, called *Tools*. The problem can be redefined as follows. Let Σ^T be the set of possible *Tools*, $\Sigma^T \subset \Sigma$, and let u_t be its cardinal. An example of user sequence could be denoted by $x_1x_2x_3^Tx_4\dots x_{r-2}^Tx_{r_1}x_r$, where $x_i^T \in \Sigma^T$. As described in Figure II.3.1, a prediction will be computed at each user action but only the predictions being made when a *Tool* is used (actions in yellow) will be evaluated (cf. Equation II.3.2). To each $x \in \Sigma$, we will associate its one-hot vector representation for an arbitrary ordering of the actions in our database, so that in the following each element in Σ will be considered as an element in $\{0;1\}^{u_a}$.

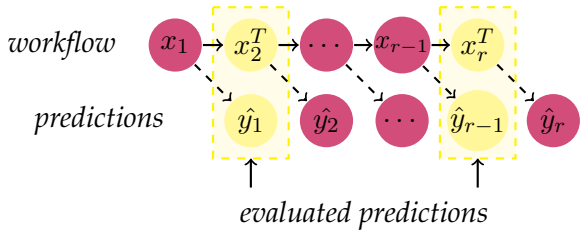

***Figure II.3.1:** Evaluation of user action prediction: actions in yellow represent Tools and thus are the only predictions we will focus on.*

II.3.2.1 Loss

Both the $\mathrm{LSTM}-\mathrm{1HOT}$ and $\mathrm{LSTM}-\mathrm{EMB}$ models are trained to minimize the following weighted average cross-entropy loss $\mathcal{L}$, which enables the LSTM network to only focus on the *Tools* prediction. For each session $X \in \mathcal{T}$, of length n_X, the elementary loss is given by:

$$L(X) = -\sum_{i=1}^{n_X-1} \delta_{x_{i+1}}(\Sigma^T) \sum_{j=1}^{u_a} x_{(i+1)j} \log \tilde{y}_{ij}$$

where n_X represents the number of actions in the session X and:

- $x_{(i+1)j}$ is the j-th component of the u_a-dimensional one-hot vector encoding the $(i+1)$-th action of session X;
- $\tilde{y}_{ij}$ is the j-th component of the u_a-dimensional probability distribution predicted at time step i by the LSTM (corresponding to model $\mathcal{A}_i$ defined in section II.3.2);
- $\delta_x(\Sigma^T) = \begin{cases} 0, & x \notin \Sigma^T; \\ 1, & x \in \Sigma^T. \end{cases}$

Finally, the complete loss on the whole training set is computed as follows:

$$\mathcal{L} = \sum_{X \in \mathcal{T}} L(X)$$

where $\mathcal{T}$ denotes the training set.

II.3.3 Input Representation

We propose two types of LSTM models for the next action prediction task: one-hot encoded vectors and action embeddings.

II.3.3.1 One-hot Vectors

The first one, called $\mathrm{LSTM}-\mathrm{1HOT}$, uses as inputs one-hot vectors whose dimension is equal to the number of possible elementary actions. Let $x \in \Sigma$ be an elementary action. We classically define its one-hot representation $x^{\mathrm{1HOT}} \in \{0,1\}^{u_a}$, with only one component different from 0,

as follows:

$$x_k^{\text{1HOT}} = \begin{cases} 1, \text{ if } x = a_k \\ 0, \text{ otherwise} \end{cases}$$

where a_k, for $k \in [\![1, u_a]\!]$, are all the possible elementary actions in Σ and x_k^{1HOT} the k-th component of the one-hot vector.

II.3.3.2 Embeddings

The second model, called $\text{LSTM} - \text{EMB}$, uses action embeddings as inputs: $x^{\text{EMB}} \in \mathbb{R}^E$ where E denotes the embedding size. The objective is to replace the one-hot encoding representation of actions, which is high dimensional and sparse with its corresponding distributed representation. To compute action embeddings we have used the *Continuous Bag-Of-Words* (CBOW) architecture [Mikolov et al., 2013]. Let $X = x_1 \dots x_n$ be a sequence of actions and x_r a target action, the CBOW model trains a feedforward neural network to map its context actions contained in a window of size w to the target action x_r. Pairs of actions (x_{r+c}, x_r) with x_{r+c} corresponding to averaged one-hot encoded vectors of actions contained in context $c \in [-w, w]$ are respectively used as inputs (x) and outputs (y) of the following equations:

$$h = xW$$

$$y = \phi(hW')$$

With E the embedding size, $W \in \mathbb{R}^{u_a \times E}$ is the hidden layer weight matrix, $h \in \mathbb{R}^E$ the hidden layer and ϕ the softmax function. $W' \in \mathbb{R}^{E \times u_a}$ represents the output layer. Final weights of W will give the embedding representation. As it was more efficient in our case, this architecture was preferred to the other possible Word2Vec architecture, namely *Skip-gram* which trains a feedforward neural network to predict context words based on a word contained in the context.

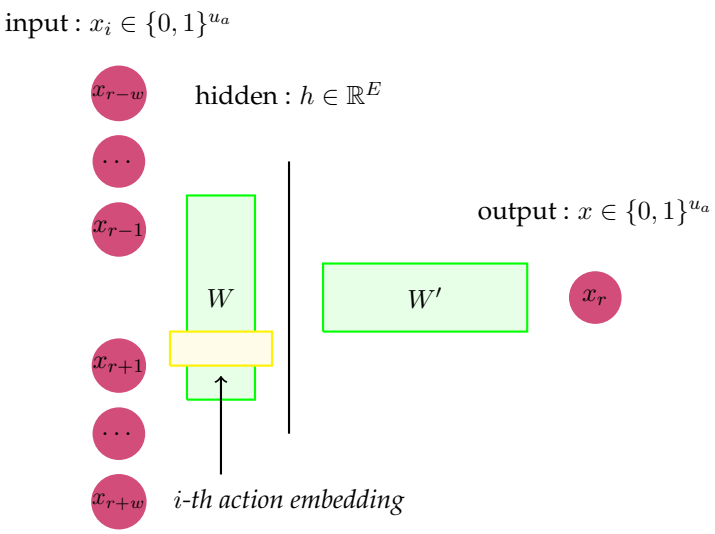

***Figure II.3.2:* CBOW model.**

As in our case there are only very few actions occurring in the test set which have never been performed in the training set, we assigned all the out-of-vocabulary actions to the same value in the one-hot encoded vectors and to an averaged vector of all distributed representations of actions in the case of embeddings. However, more sophisticated methods were proposed to handle unseen data representation. For example, embeddings of an unseen word might be induced by its nearest neighbors [Tafforeau et al., 2015].

II.3.4 Experiments

II.3.4.1 Data

We conducted experiments on all 50 systems from the database to ensure the effectiveness of the proposed methods in all types of cases. Table II.3.3 shows the number of normal sessions $|\mathcal{S}|$ for each system. Table II.3.4 shows the number of unique actions u_a and unique *Tools* u_t for each system. Finally, Table II.3.5 shows the number of performed actions N_a and *Tools* N_t for each system. These values are highly variable depending on the hospital. Indeed, the workload is not the same on all systems. While some systems are continuously used, others are occasionally used for specific cases. Besides, the content of the user interface changes according to the medical application which is launched for image analysis. As a result, generalist systems purchasing licenses of many applications are most likely to have a number of unique actions which is more important than a very specialized system.

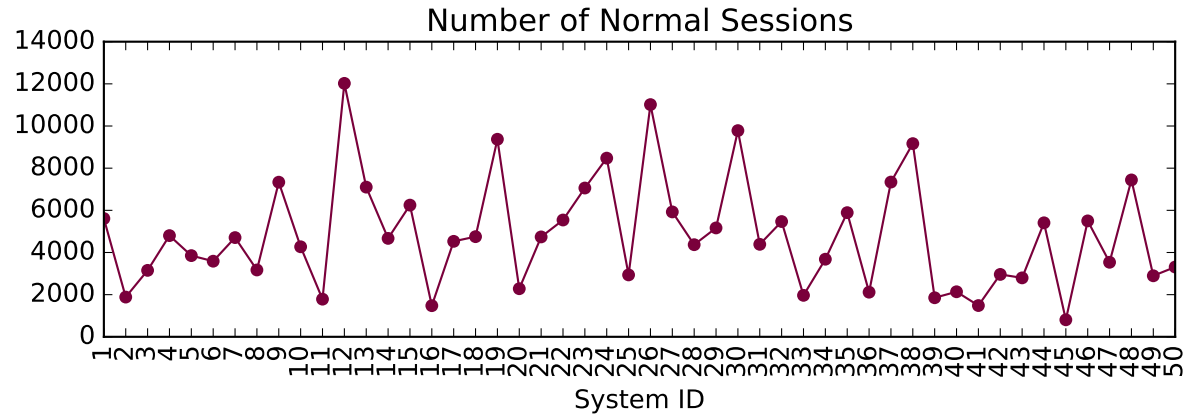

Figure II.3.3: Number of normal sessions for each system in the database.

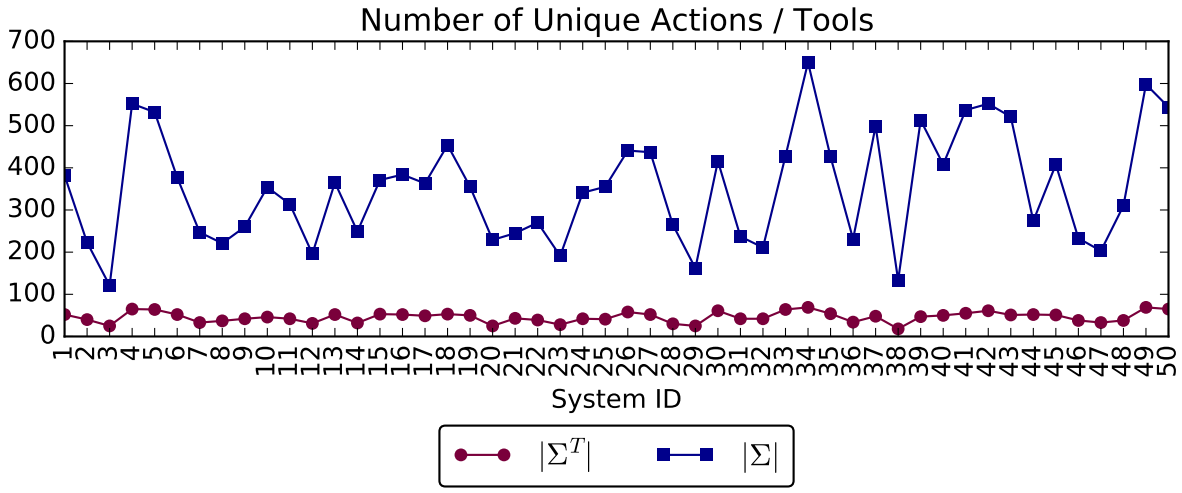

Figure II.3.4: Number of unique actions u_a and unique Tools u_t for each system in the database.

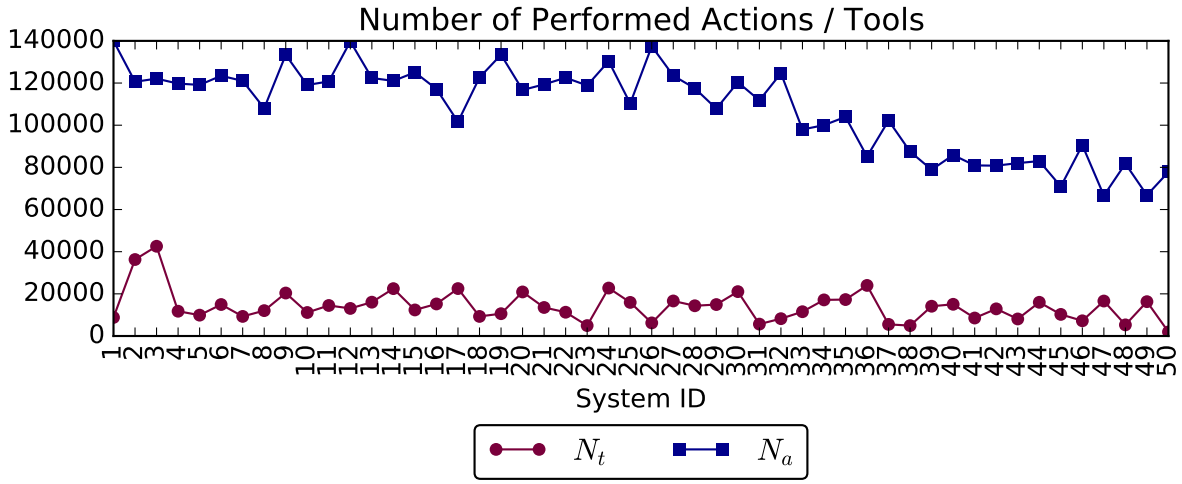

Figure II.3.5: Number of performed actions N_a and Tools N_t for each system in the database.

II.3.4.2 Baseline Methods

The prediction performance of the LSTM networks is compared to two other sequential prediction methods and to a Multilayer Perceptron neural network considered as a standard classification method. These baselines were selected among all the existing algorithms since they provided the best results.

FxL [Hartmann and Schreiber, 2007].

The FxL algorithm assumes that higher order Markov Chains yield more reliable results and computes a prediction score by combining different orders. It does so by multiplying the frequency of an action by the length of the context after which it occurred:

$$P(x_i|x_{i-n},\dots,x_{i-1}) = \frac{\text{score}(x_i)}{\sum_{x\in\Sigma}\text{score}(x)}$$

$$\text{score}(x_i) = \sum_{j=1}^{n} w(j)C(x_{i-j}\dots x_{i-1}x_i)$$

where:

- $C(x)$ is the number of times the exact subsequence x appears in the training set;
- x_i is the i-th action in the given sequence;
- $w(j)$ is the weight assigned to the Markov chain of order j, $w(j) = j$.

We chose FxL as baseline since it was the best performing method among the Proactive User Interfaces state-of-the-art algorithms [Hartmann and Schreiber, 2007].

Markov Chains with Backing-Off (MCBO) [Katz, 1987]

The MCBO model of order n uses the last n actions to predict the next action:

$$P(x_i|x_{i-n}\dots x_{i-1}) = \begin{cases} \dfrac{C(x_{i-n}\dots x_{i-1}x_i)}{C(x_{i-n}\dots x_{i-1})} & \text{if } C(x_{i-n}\dots,x_i) > 0 \\ P(x_i|x_{i-n+1}\dots x_{i-1}) & \text{otherwise} \end{cases}$$

where:

- $C(x)$ is the number of times the exact subsequence x appears in the training set;
- x_i is the ith action in the given sequence.

Whenever the path composed of the last n actions has not been observed in training, it backs off to the estimate corresponding to models with smaller histories, in this case, the path composed of the $n-1$ actions, etc. The backing-off enables to ensure a prediction after each action. Indeed, even if the path of the lowest order model has not been seen in the training, the model simply returns the most used action in the training. We chose this method since it was working particularly well on our data, providing results comparable to FxL.

Multilayer Perceptron (MLP) [Zhao et al., 2017]

For comparison, we have also implemented a fully connected MLP neural network composed of 2 hidden layers. We use a sigmoid activation function for the hidden layers and a softmax activation output layer. The MLP inputs consist of one-hot encoded vectors of the $n = 5$ last actions that were used.

Note that in order not to put these models at a competitive disadvantage, they were trained to predict the next *Tool* occurring after the last n actions (even if it occurs several actions afterwards) rather than the next action. The LSTM model is trained to predict next action regardless its type, the weighted loss naturally leads the LSTM to be efficient on *Tools*.

II.3.4.3 Performance Evaluation

The four models will be evaluated only on their ability to predict *Tools*. The prediction accuracy is computed by dividing the number of correct predictions by N_t^{te}, the total number of used *Tools* in the sessions of the test set:

$$Top_1 = \frac{1}{N_t^{te}} \sum_{1 \leq i \leq N_t^{te}} \mathbb{1}(y_i = \hat{y}_i) \tag{II.3.2}$$

where y_i is the observed used tool and $\hat{y}_i$ corresponds to the prediction, obtained by taking the argmax of the probability vector predicted by the LSTM model as detailed in Equation II.3.1. We also define a score considering a prediction as correct if one of the k actions predicted with maximal probabilities correspond to the true *Tool*. We denote by $\hat{A}_i^k$ the set of all these actions, obtained by taking the k highest probabilities in the the probability vector predicted by the LSTM network:

$$Top_k = \frac{1}{N_t^{te}} \sum_{1 \leq i \leq N_t^{te}} \mathbb{1}\left(y_i \in \hat{A}_i^k\right).$$

Indeed, our prediction system might take the form of a dynamic toolbar containing in real-time the k most likely next *Tools* needed by the user.

II.3.4.4 Experimental Setup

Training and Test Sets

We have conducted experiments on each system individually. As we work with ordered data, we have used for each system the 80% oldest sessions for training and the 20% most recent for the test set.

Model Averaging

To reduce generalization error, we perform model averaging. We divide the training set into $k = 10$ non-overlapping subsets. On trial i, the i-th subset is used as the validation set. The k models output distributions are averaged to compute final predictions.

Implementation

We use our own Python implementation for FxL and the Markov Chains with Backing-Off models. The predictions obtained for the best MCBO and FxL of order $1 \leq n \leq 7$, on the corresponding validation set of the 10 averaged models were used. Higher orders are too expensive at running time. The MLP as well as the LSTM network are defined and trained in Tensorflow [Abadi et al., 2015] with the RMPSProp optimizer [Tieleman and Hinton, 2012]. The MLP is composed of 2 hidden layers, each of size 100. The LSTM networks used with one-hot vectors and embeddings have the same architecture. Hidden and cell states are initialized to zero at the beginning and reset after every minibatch of size $B = 32$. The weights of both networks are initialized using Tensorflow initialization for weights [Glorot and Bengio, 2010] and are unrolled for $T = 100$ steps. They both have one layer of $H = 50$ units, see section II.3.4.5. The embeddings are computed using the Gensim [Rehurek and Sojka, 2010] Python package. The results presented are obtained with embedding size $E = 100$, and a context window of size $W = 20$, see section II.3.4.5.

Regularization

To prevent the networks (both MLP and LSTM) from over fitting we use early stopping [LeCun et al., 2015] and interrupt them from learning when the validation error does not decrease for 10 epochs.

II.3.4.5 LSTM Tuning

This section presents experiments conducted on 2 systems to select the dimension of hidden state in the LSTM network fed with one-hot encoded actions. Experiments on other systems showed the same tendency. The same LSTM architecture is then used to select the embeddings parameters.

LSTM-1HOT

Tables II.3.1 and II.3.2 show Top_1 prediction accuracy obtained with values of the dimension of hidden state H comprised between 25 and 300 for systems 1 and 2. The dimension of the hidden state impacts only slightly the performance score.

H	25	50	100	150	200	250	300
Top_1	0.586	0.590	0.587	0.589	0.581	0.589	0.571

***Table II.3.1:** Dimension of hidden state H and corresponding score Top_1 for system 1.*

H	25	50	100	150	200	250	300
Top_1	0.573	0.579	0.581	0.577	0.579	0.578	0.583

***Table II.3.2:** Dimension of hidden state H and corresponding score Top_1 for system 2.*

We decided to continue the experiments with $H = 50$ since this value seems to be a fair com-

promise for both systems.

LSTM-EMB

Figures II.3.6 and II.3.7 show the Top_1 score obtained with embedding size E between 50 and 200 and a context window size W between 5 and 50 for systems 1 and 2. We selected $E = 100$ and $W = 20$ to conduct our experiments.

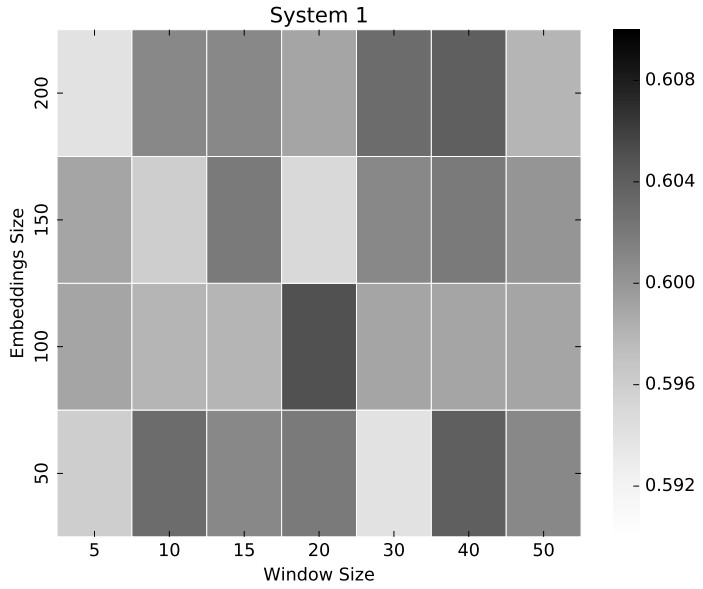

Figure II.3.6: $Top1$ ***score for system 1,*** $E \in [50, 100, 150, 200]$ ***and*** $W \in [5, 10, 15, 20, 30, 40, 50]$***.***

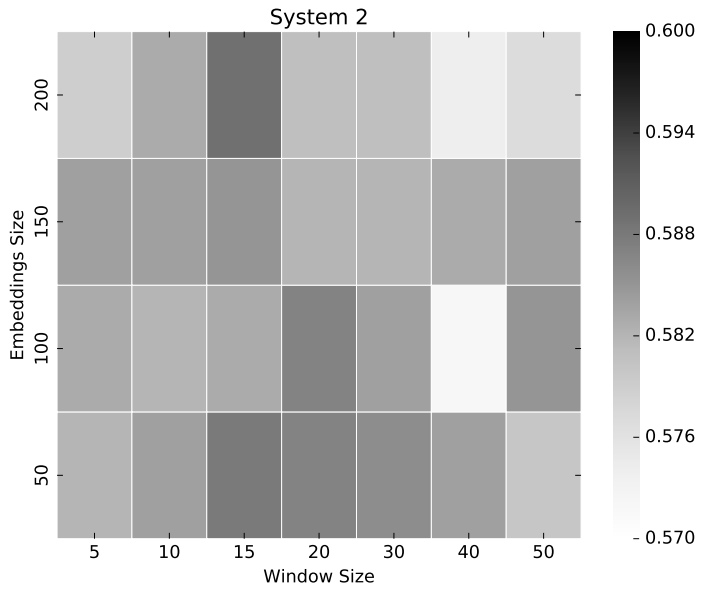

Figure II.3.7: $Top1$ ***score for system 2,*** $E \in [50, 100, 150, 200]$ ***and*** $W \in [5, 10, 15, 20, 30, 40, 50]$***.***

II.3.4.6 Results

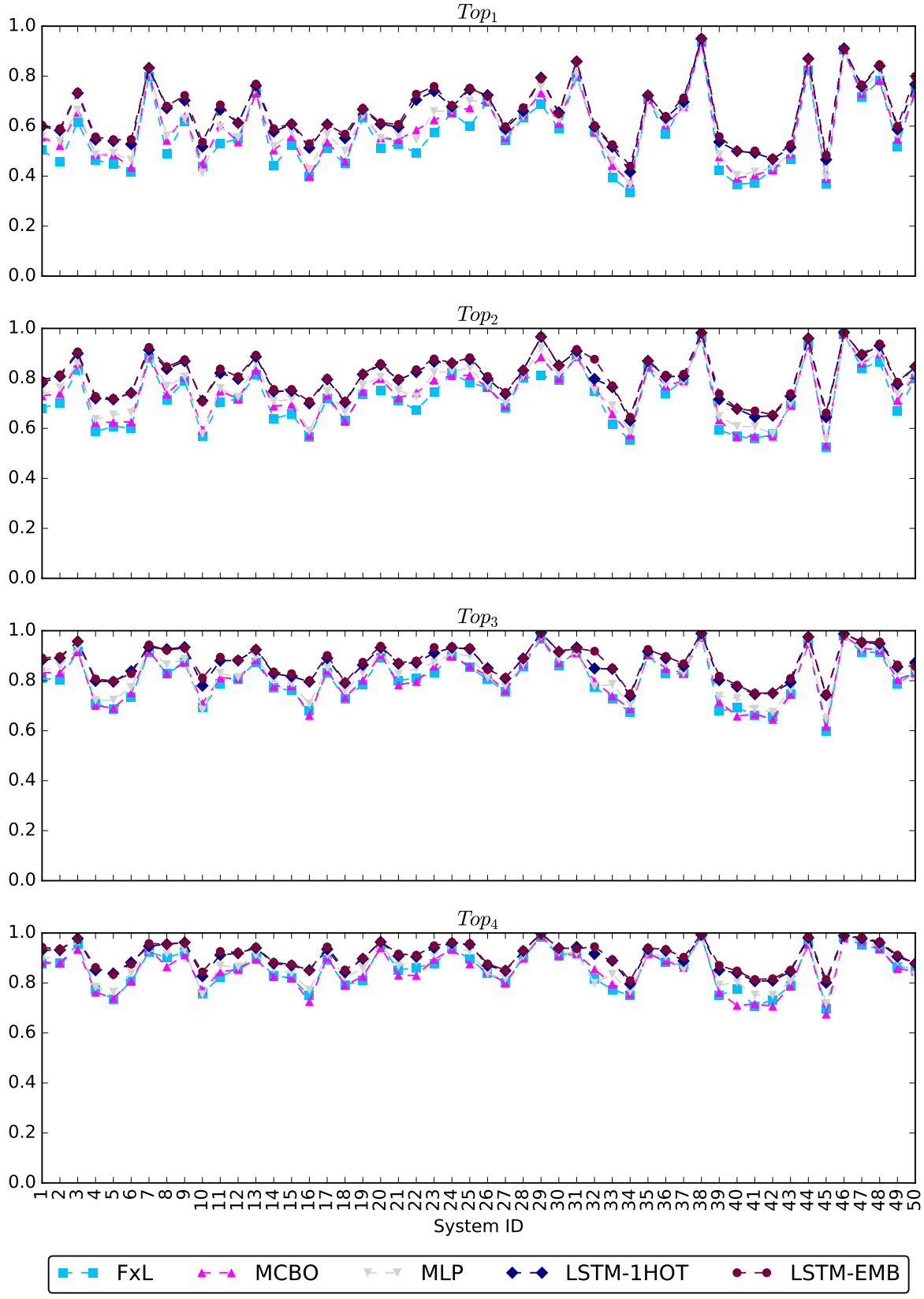

Figure II.3.8: $Top1$, $Top2$, $Top3$, *and* $Top4$ *prediction accuracy for each of the 50 systems.*

Figure II.3.8 shows the $Top1$, $Top2$, $Top3$, and $Top4$ predictions accuracy obtained individually on each system of the database. For the sake of robustness of the industrial solution and to save computational time, we used the same values of H, E and $\min_{\text{count}}$ for all the systems. A way to improve each system prediction accuracy would be to individually select the best values on each validation set.

The $\text{LSTM} - \text{EMB}$ fed with action embeddings, outperforms the other models on each dataset and provides consistently higher prediction accuracy over FxL in the case of Top_1 score all the more when the prediction score of FxL is low. Table II.3.3 shows the average gain from $\text{LSTM} - \text{EMB}$ over FxL for different intervals of values of the Top_1 score obtained with FxL. While the gain is equal to 3.8% when FxL returns prediction accuracy greater than 0.7, the gain is on average equal to 12.4% when FxL performance is lower than 0.4.

Top_1 Fx $<$ l Score	Average gain from $\text{LSTM} - \text{EMB}$ over FxL
<0.4	12.4 %
$(0.4, 0.5]$	12.3 %
$(0.5, 0.6]$	9.3 %
$(0.6, 0.7]$	6.0 %
>0.7	3.8 %

***Table II.3.3:** Average gain from* $\text{LSTM} - \text{EMB}$ *over FxL for different intervals of scores provided by FxL.*

Figure II.3.9 illustrates an example of the results that were obtained for system 2, on each individual *Tool*. $\text{LSTM} - \text{EMB}$ provides equivalent or better predictions than FxL for every *Tool*, and in particular, it is able to predict with better accuracy some rare *Tools*.

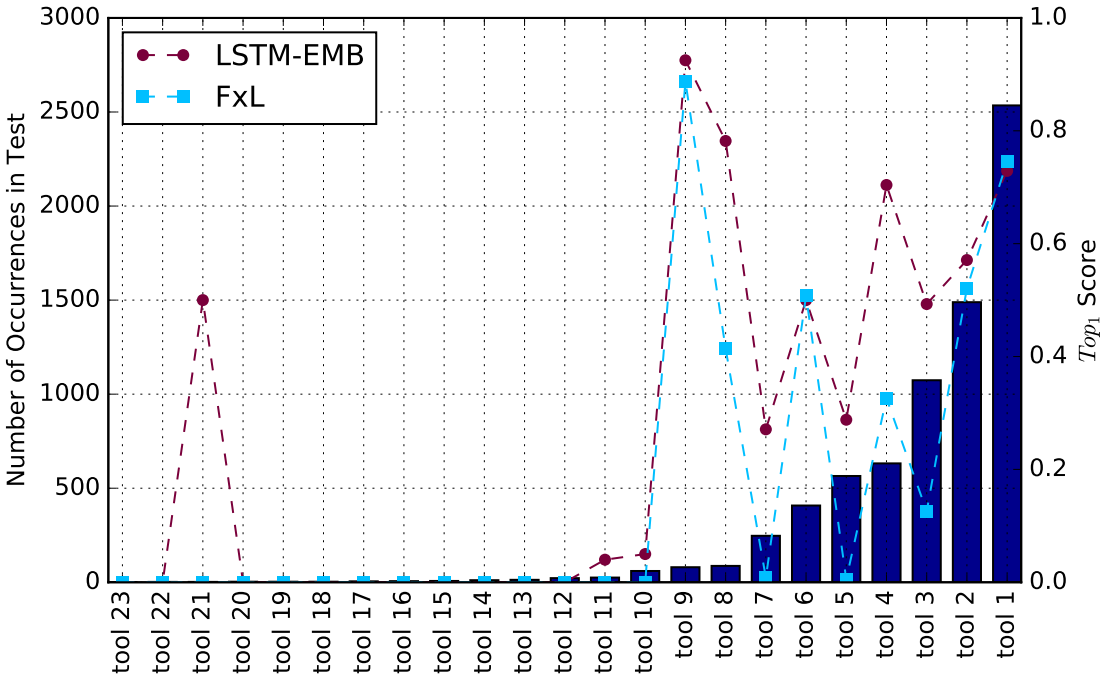

Figure II.3.9: Top_1 ***prediction accuracy for FxL and*** LSTM − EMB ***on each Tool for system 2.***

On average, the LSTM − EMB method outperforms FxL by 8.8% on the Top_1 score and by 5.7% on the Top_4 score (see Table II.3.4). It also does consistently better than the Markov Chains with Backing-Off (+ 6.3% on the Top_1 score and + 5.9% on the Top_4 score) and the Multilayer Perceptron neural network (+5.4 % on the Top_1 score and +4.1 % on the Top_4 score).

model	FxL	MCBO	MLP	LSTM − 1HOT	LSTM − EMB
Top_1	0.569	0.594	0.603	0.648	0.657
Top_2	0.725	0.745	0.759	0.806	0.814
Top_3	0.807	0.814	0.830	0.871	0.877
Top_4	0.855	0.852	0.871	0.906	0.911

Table II.3.4: ***Average results on the 50 systems for*** Top_1, Top_2, Top_3, ***and*** Top_4 ***predictions accuracy.***

Besides, it slightly improves the LSTM − 1HOT network by 0.9% on average on Top_1 score and by 0.6% on Top_4 score, using only 85% of the training and prediction time of LSTM − 1HOT, as can be seen in the Table II.3.5. As shown in Figure II.3.10, training time is reduced on each system with action embeddings instead of one-hot encoded vectors as inputs of the LSTM network. Note that the training time presented for each system in Figure II.3.10 and average training time presented in Table II.3.5 takes into account the embeddings training time and thus makes the LSTM network fed with action embeddings easier to deploy.

Model	Average Training Time for 1 Model (seconds)	Ratio
LSTM − 1HOT	349.24	1
LSTM − EMB	295.32	0.85

Table II.3.5: ***Average training time in seconds of LSTM models. Training time for*** LSTM − EMB ***includes embeddings computation with Gensim [Rehurek and Sojka, 2010].***

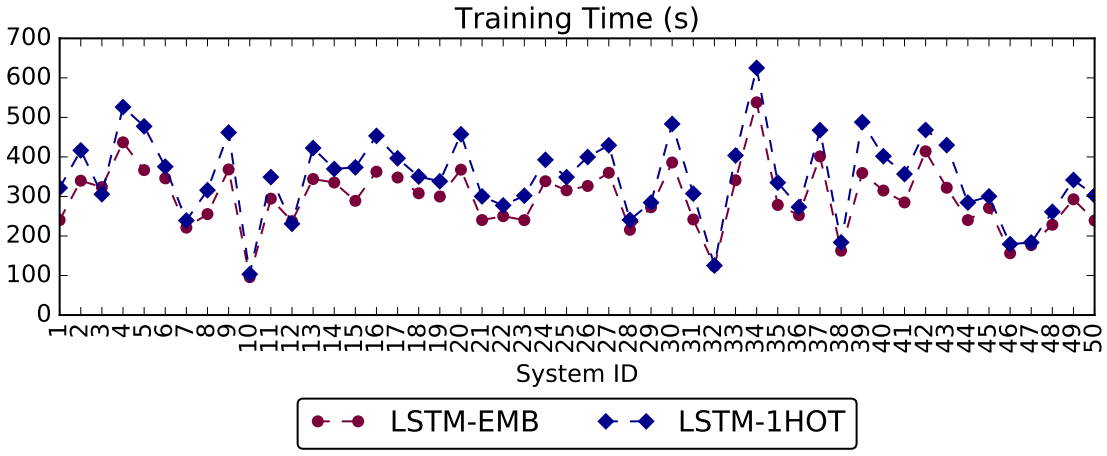

Figure II.3.10: ***Training time for 1 LSTM model. Training time for*** LSTM − EMB ***includes embeddings computation with Gensim [Rehurek and Sojka, 2010].***

As shown on Figure II.3.8, Top_1 prediction results present a high variability in the predictive efficiency of the 5 methods which is attenuated with Top_3 and Top_4 scores. To try to explain these differences we propose to visualize the correlation between Top_1 score obtained with LSTM − EMB and system characteristics we have at our disposal. Thus, Figure II.3.11 shows the scatter matrix plots of Top_1 scores obtained with LSTM − EMB on the 50 systems against the total number of sessions $|\mathcal{S}|$, the number of unique actions u_a and unique *Tools* u_t, their total number of performed actions N_A and the total number of performed *Tools* N_T.

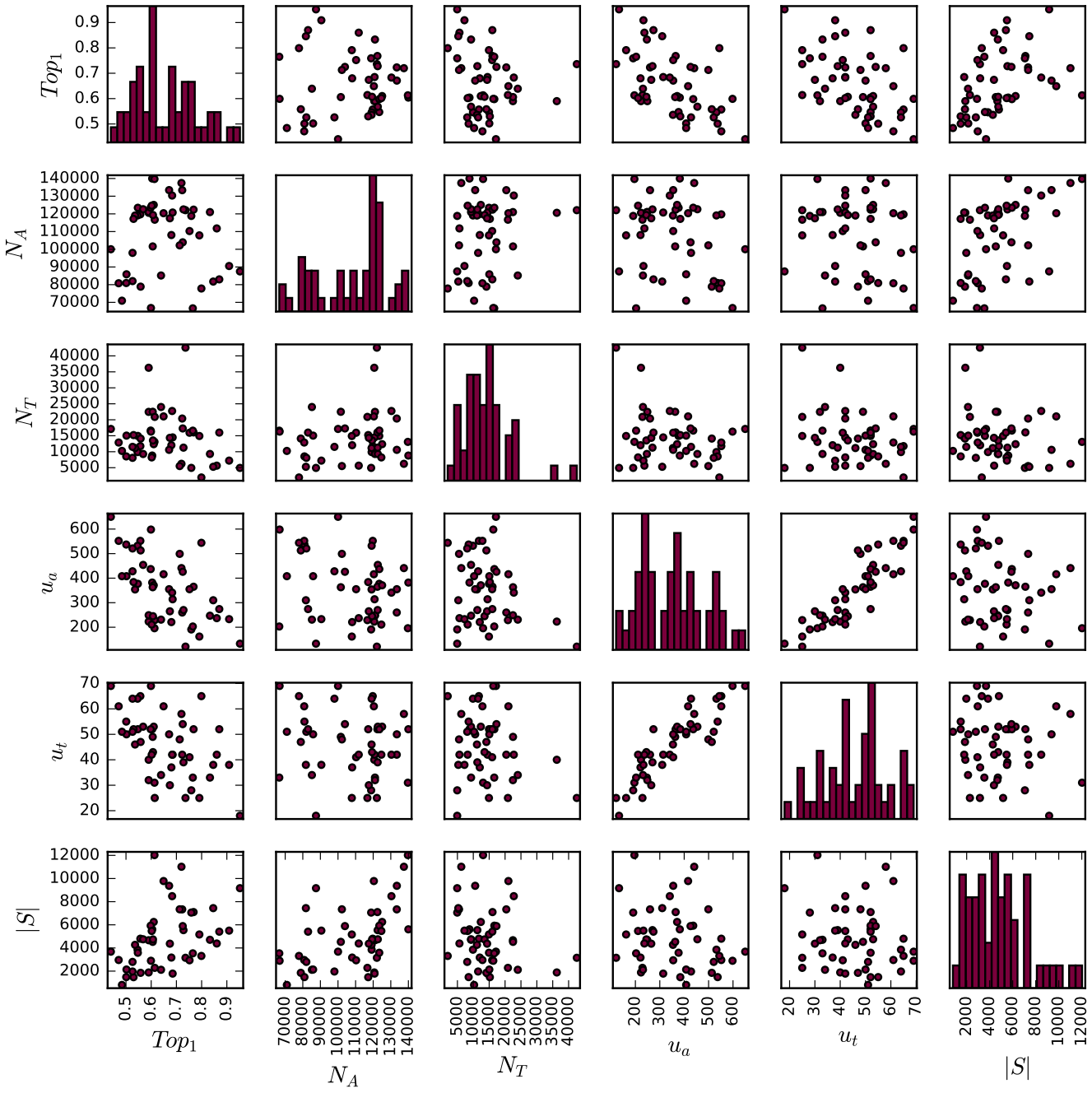

Figure II.3.11: *Top_1 scores obtained with* LSTM − EMB *on the 50 systems against the total number of normal sessions $|\mathcal{S}|$, the number of unique actions u_a and unique Tools u_t, their total number of performed actions N_A and the performed total number of Tools N_T.*

We notice that the Top_1 score tends to be higher when the number of unique actions u_a and the number of unique *Tools* u_t is small. Moreover, it increases when the total number of normal sessions $|\mathcal{S}|$ grows. This observation leads us to the next section concerning the best training strategy of our LSTM model. Since the prediction accuracy is improved with the number of sessions it is interesting to test if gathering sessions coming from different hospitals in the training set enables to improve the prediction accuracy.

II.3.5 Best Training Strategy

So far we trained one model per system. In this section we aim at identifying the best training strategy. Indeed, the model might benefit from additional training sessions or on the contrary it could be more efficient by being trained on a specific group of sessions. For this purpose we will test models trained on different training sets, some of them obtained by session characteristics which are known such as the medical application which is used, the hospital they come from or the corresponding country. Section II.3.5.1 presents the experiments conducted to test whether training sets based on such criteria enable to improve prediction accuracy. Another approach consists in using unsupervised clustering techniques to highlight groups of similar sessions to use for the training. This approach is detailed in section II.3.5.2.

II.3.5.1 Known Characteristics

In this section we consider two possible approaches. The first one consists of a global model that would not distinguish between different medical applications. To be the most efficient, should the model be trained:

- only on the hospital's own logs? The underlying assumption would be that user habits on a system are more pertinent to train the model than additional sessions from other systems containing more variability in user workflows.
- on several hospital logs coming from the same country? Here the idea is that radiologists from the same geographical region might tend to have similar habits of use (medical education, medical advance of the country, country health structure, country medical culture, etc.) and that additional sessions will help the model better generalize.

The second approach defines the training sessions according to the medical application. As for the global model, the same questions might be addressed in the case of a particular medical application.

Should we restrict the training set to a single hospital or enrich it with additional sessions in which this application has been launched in other hospitals?

As is shown in Table II.3.6 we have at our disposal 50 logs mainly coming from India (13 systems), United States of America (8 systems) and Japan (5 systems). We use logs coming from these 3 most represented countries to conduct our experiments. Table II.3.7 shows the system IDs corresponding to the systems which will be tested for each selected country.

Country	Number of hospitals
INDIA	**13**
UNITED STATES OF AMERICA	**8**
JAPAN	**5**
EGYPT	4
KOREA (REPUBLIC OF)	4
POLAND	4
BELGIUM	2
ROMANIA	2
AUTRALIA	1
BANGLADESH	1
HUNGARY	1
IRELAND	1
NEPAL	1
SINGAPORE	1
SOUTH AFRICA	1
TURKEY	1

Table II.3.6: ***Country distribution. We will use Indian, American and Japanese log files to conduct our experiments.***

Country	System ID
INDIA	1, 10, 12, 15, 18, 21, 22, 23, 26, 30, 31, 32, 46
UNITED STATES OF AMERICA	11, 16, 20, 25, 35, 39, 41, 45
JAPAN	3, 6, 34, 36, 40

Table II.3.7: ***System IDs of studied countries.***

Methodology

Each hospital sessions are divided into training (80% oldest sessions) and test (20% most recent sessions) sets. Each system will be tested separately on its own test set, with different training sets. We use the LSTM model fed with action embeddings described in section II.3.3.2. We denote by:

- $\mathcal{T}_i$, the set of training sessions of hospital i;
- $\mathcal{T}_i^k$, the subset of training sessions of hospital i in which application k was launched;
- H_C, the set of hospitals from country C and $|H_C|$ its cardinal;
- $\mathcal{T}_C = \bigcup_{i \in H_C} \mathcal{T}_i$, the set composed of all the training sessions from all the hospitals from country C;

- $\mathcal{T}_C^k = \bigcup_{i \in H_C} \mathcal{T}_i^k$, the subset composed of the training sessions from all the hospitals in country C, in which application k was launched.

And thus:

- $M_{\mathcal{T}_i}$ denotes the model trained on $\mathcal{T}_i$;
- $M_{\mathcal{T}_i^k}$ denotes the model trained on $\mathcal{T}_i^k$;
- $M_{\mathcal{T}_C}$ denotes the model trained on $\mathcal{T}_C$;
- $M_{\mathcal{T}_C^k}$ denotes the model trained on $\mathcal{T}_C^k$.

For each hospital of each selected country we first tested if the predictions would benefit of a model trained on additional sessions coming from the same country. For each group of hospital H_C we selected the 2 most used applications to compare the different training strategies in the case of a specific application. Table II.3.8 summarizes the experiments which will be conducted.

	global	1 application
1 hospital	$M_{\mathcal{T}_i}$	$M_{\mathcal{T}_i^k}$
group of $\lvert H_C \rvert$ hospitals	$M_{\mathcal{T}_C}$	$M_{\mathcal{T}_C^k}$

***Table II.3.8:** Experiments overview. We tested the models with C successively equal to India, USA, Japan.*

We used Top_1 score to evaluate global models:

$$Top_1 = \frac{1}{N_t^{te}} \sum_{1 \leq i \leq N_t^{te}} \mathbb{1}(y_i = \hat{y}_i).$$

We recall that N_t^{te} is the total number of used *Tools* in the sessions of the test set. We also define a specific score to evaluate the predictions made when the application k was launched:

$$Top_1^{App_k} = \frac{1}{N_t^{te^{App_k}}} \sum_{1 \leq i \leq N_t^{te^{App_k}}} \mathbb{1}(y_i = \hat{y}_i)$$

where $N_t^{te^{App_k}}$ denotes the number of *Tools* performed in test sessions where application k was launched.

Results: Indian Hospitals

Figure II.3.12 shows the Top_1 score obtained for each Indian system. The prediction acccuracy of models trained on each system individual logs is compared to the one obtained with a model trained on all the Indian systems. Hospitals 31 and 32 benefit from a model trained on additional sessions. Hospitals 1, 10 and 46 slightly have a better prediction accuracy with the model trained only on their own sessions. There is no noticeable difference for the other systems.

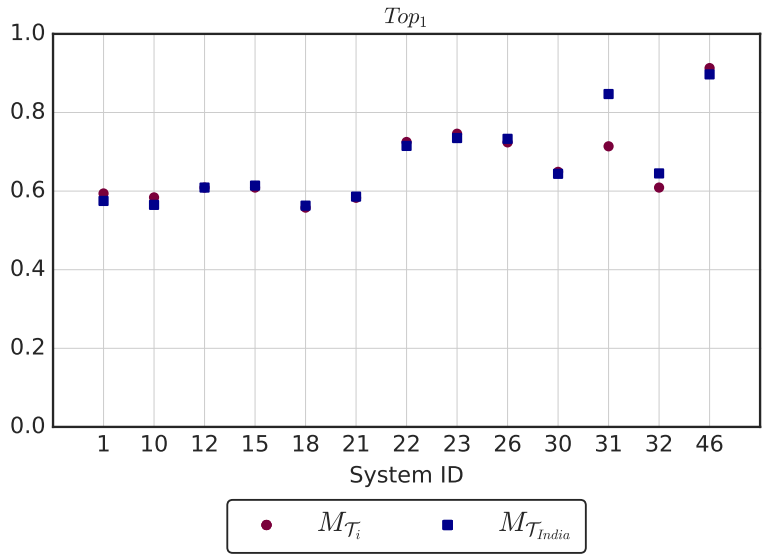

Figure II.3.12: *Top_1 score obtained for each Indian system. The performance of a model trained on each system own logs is compared to that of a model trained on all the sessions performed in Indian hospitals.*

Table II.3.9 shows the total number of applications runs for the group of Indian hospitals. General application runs were removed since they might correspond to specific applications that were not properly logged. Apps 1 and 2, which were the most executed, will be used to conduct the experiments. Details about the number of sessions used for training and test for each application, as well as the number of corresponding *Tools* are available in Appendix B.

App ID	Runs
1	**13586**
2	**2045**
3	399
4	1500

Table II.3.9: *Total number of application runs in Indian hospitals group H_{India}.*

Figure II.3.13 shows $Top_1^{App_1}$ and $Top_1^{App_2}$ prediction accuracy for each Indian system in the case of test sessions in which App_1 and App_2 were launched. Systems with no performance evaluation did not have enough sessions in which the concerned application was used to train the model. As can be observed, system 32 predictions for App_1 are the most accurate with the global model trained on the sessions coming from all Indian hospitals. However there are only 14 *Tools* in the test set. Besides, the model trained only on sessions where App_1 is launched completely fails to predict the next *Tool* due to the very small training dataset, marked with an asterisk. Concerning the predictions for App_2, except for system 18, the model trained only on each hospital's own sessions where the application was launched is less effective than the other models. The most appropriate model seems to be the global model trained on additional

sessions. Again, predictions made with the specific model, present very bad results for system 10 which has too few training sessions for the concerned application.

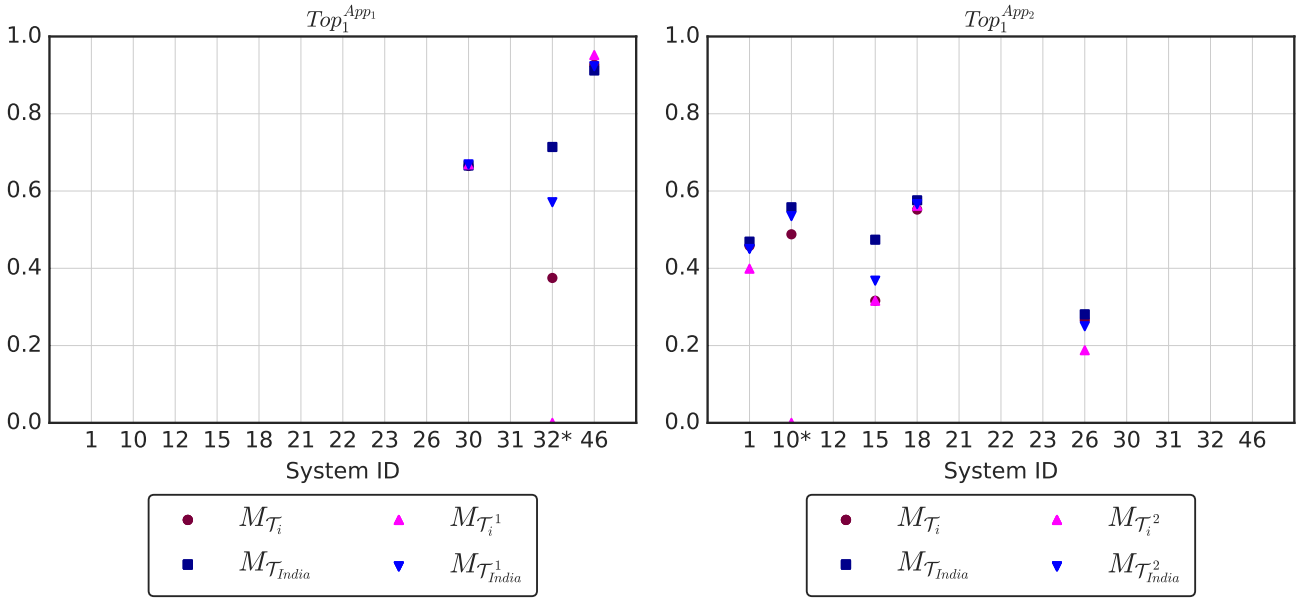

Figure II.3.13: *$Top_1^{App_1}$ and $Top_1^{App_2}$ scores obtained for each Indian system. The performance of global and specific models trained on each system own logs is compared to that of global and specific models trained on all the sessions performed in Indian hospitals. Systems having small but trainable datasets are signaled with an asterisk.*

Results: American Hospitals

Figure II.3.14 shows the Top_1 score obtained for each American system. The prediction acccuracy of models trained on each system individual logs is compared to the one obtained with a model trained on all the American systems. Only system 45 slightly benefits from the model trained on additional sessions.

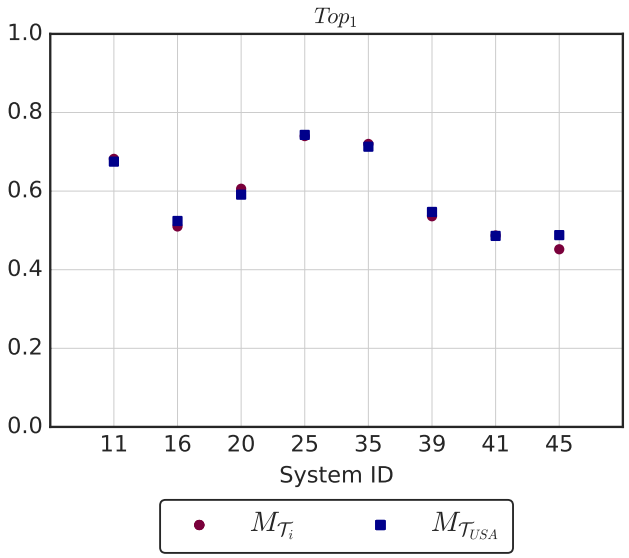

Figure II.3.14: *Top_1 score obtained for each American system. The performance of a model trained on each system own logs is compared to that of a model trained on all the sessions performed in American hospitals.*

Table II.3.10 shows the total number of runs of applications for the group of American hospitals. Apps 3 and 4 which were in this case the most executed will be used to conduct the experiments. As previously, details about the number of sessions used for training and test for each application, as well as the number of corresponding *Tools* is available in Appendix B.

App ID	Runs
1	2
2	**2207**
3	**4513**
4	1946

Table II.3.10: *Total number of applications runs in American hospitals group H_{USA}.*

Figure II.3.15 shows $Top_1^{App_3}$ and $Top_1^{App_2}$ prediction accuracy for each American system in the case of sessions where App_3 and App_2 were launched. While predictions of hospital 35 on sessions in which App_3 was launched are better when the model is trained only on its own sessions in which the application was launched, system 45 benefits from additional sessions. Regarding App_2, when the application is used on a system, the model trained only on sessions where it was launched systematically returns poorer predictions. Three systems (11, 20 and 39) take advantage from additional sessions where the application was used, two systems (16 and 25) have better predictions with a model trained on additional mixed sessions. Finally, two systems (35 and 41) have more accurate predictions when the model is trained only on their own logs.

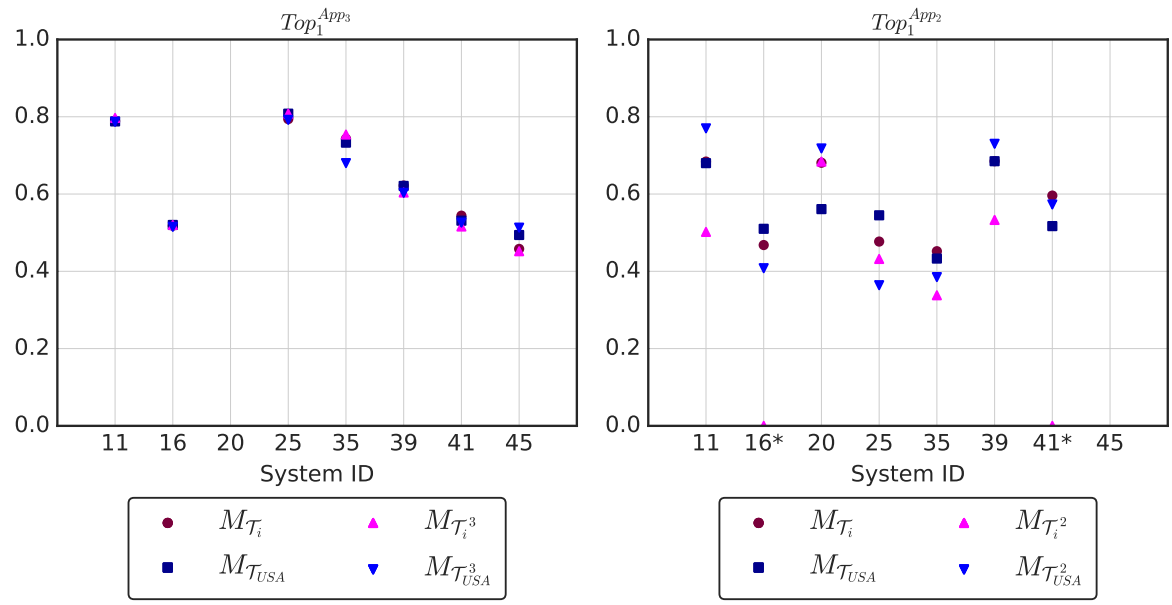

Figure II.3.15: *$Top_1^{App_3}$ and $Top_1^{App_2}$ scores obtained for each Indian system. The performance of global and specific models trained on each system own logs is compared to that of global and specific models trained on all the sessions performed in Indian hospitals. Systems having small but trainable datasets are signaled with an asterisk.*

Results: Japanese Hospitals

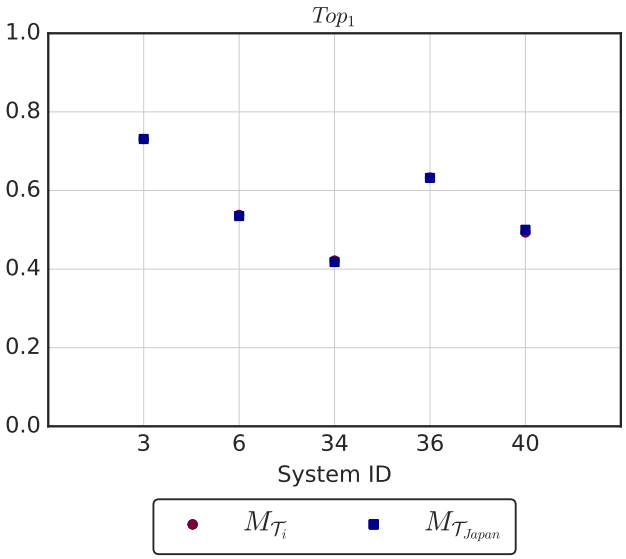

Figure II.3.16: *Top_1 score obtained for each Japanese system. The performance of a model trained on each system own logs is compared to that of a model trained on all the sessions performed in Japanese hospitals.*

Figure II.3.16 shows the Top_1 score obtained for each Japanese system. Again, the prediction accuracy of models trained on each system individual logs is compared to that obtained with a model trained on all the Japanese systems. For each system, the prediction accuracy is the same

whether the model is trained on hospital's own logs or on additional sessions. Table II.3.11 shows the total number of runs of applications on Japanese sessions. Apps 2 and 4 which were in this case the most executed will be used to conduct the experiments. More details about the number of sessions used for training and test for each application, as well as the number of corresponding *Tools* are available in Appendix B. Only 2 systems are concerned by App_2. While there is no difference between the 4 models for hospital 6, hospital 34 gets better prediction when the model is trained specifically on sessions where App_2 was launched. The predictions made when App_4 is launched are better for 2 systems (6 and 40) when the model is trained on their own sessions, hospital 34 benefits from additional sessions but has only few sessions where the concerned application is used.

App ID	Runs
1	11
2	**2434**
3	144
4	**466**

***Table II.3.11:** Total number of applications runs in Japanese hospitals group H_{Japan}.*

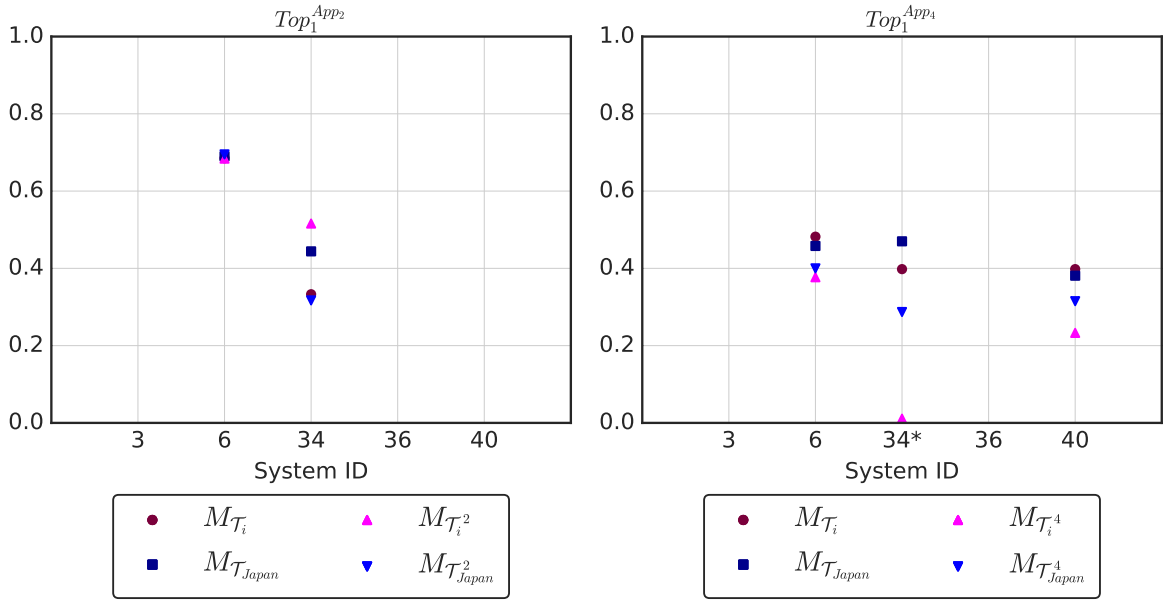

***Figure II.3.17:** $Top_1^{App_2}$ and $Top_1^{App_4}$ scores obtained for each Indian system. The performance of global and specific models trained on each system own logs is compared to that of global and specific models trained on all the sessions performed in Indian hospitals. Systems having small but trainable datasets are signaled with an asterisk.*

Discussion

Among the 26 hospitals studied, only 2 Indian systems benefit from additional sessions from other hospitals, in the case of the global score. Additional sessions involve a significantly heav-

ier training time and yet rarely improves significantly the prediction accuracy. Therefore, the models trained on hospitals own sessions are the most appropriate in this case.

Concerning specific scores, only some American hospitals and one Japanese hospital take advantage of the model trained only on sessions where a particular application was launched. For the others, it appears that the best prediction on specific sessions are obtained with global models.

In both cases, there is no real gain when the model is trained on sessions coming from other systems, meaning the variability from one system to another is too important and disrupts the LSTM recurrent neural network more than it improves it. Regarding the calculation cost generated by this solution we recommend to train one global model for each hospital, with its own data.

As explained previously, sessions can be characterized by 3 attributes (country, hospital and medical application). Experiments showed that specific training sets based on application runs do not enable to significantly improve prediction accuracy. This might be explained by the fact that application launches are not sufficiently representative to highlight groups of similar sessions among sessions corresponding to a specific application (see Figure II.3.18). We propose to use clustering techniques in an attempt to make a finer grouping of sessions to train the model on. Indeed, the obtained clusters might correspond to similar clinical cases inside a particular application or to users common preferences or habits of use which cannot be highlighted otherwise.

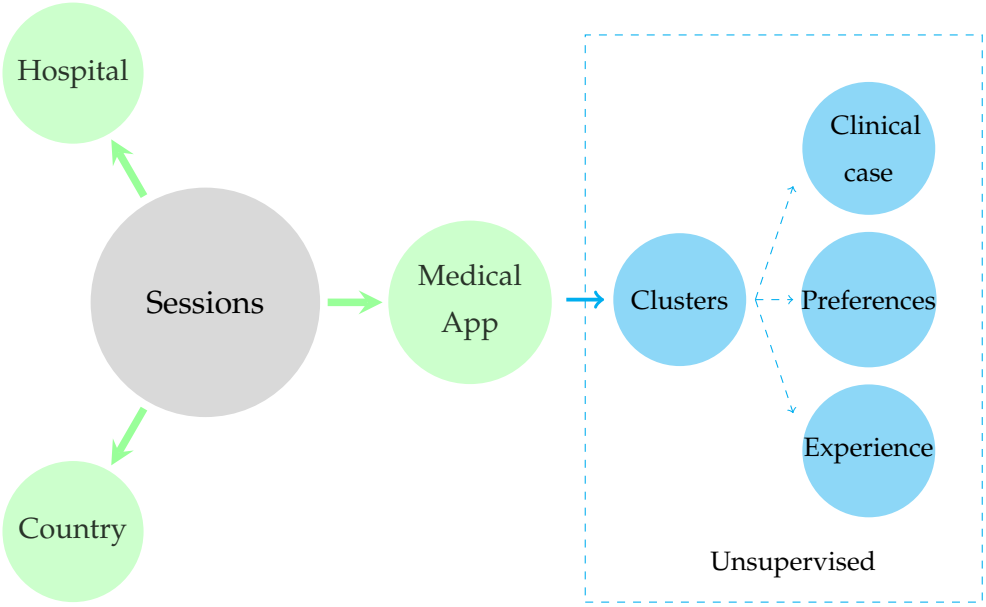

*Figure II.3.18: **User sessions characterization. Green characteristics are known, blue ones are not and may be highlighted with clustering techniques.***

II.3.5.2 Clustering to Improve Prediction

This section aims at testing whether the prediction accuracy might be improved by training the LSTM on groups of sessions obtained by clustering techniques since the previous experiments showed that clusters simply based on the type of applications runs were not particularly relevant. We conduct our experiments on the 2 systems (30 and 50) studied in Part I, Chapter 3 since they correspond to systems with an important number of sessions in which a specific application is launched.

Methodology

We propose to reuse session embeddings as inputs of the KMeans clustering algorithm, as this configuration provided the best results in Part I, Chapter 3. As presented in Figure II.3.19, a global model $M_{\mathcal{T}_i}$ trained on all the sessions from system i will be used to perform predictions from the beginning of the session until the s-th action is reached. From this action, predictions will be done by models independently trained on clusters. The reason for this choice is that we cannot determine in which cluster the session can be classified before enough actions were executed by the user. There is generally no specific user id allowing us to identify the proper cluster beforehand. We denote by M_{c_i} the model trained on the sessions from cluster c_i with $i \in [\![1, K]\!]$, K standing for the total number of clusters.

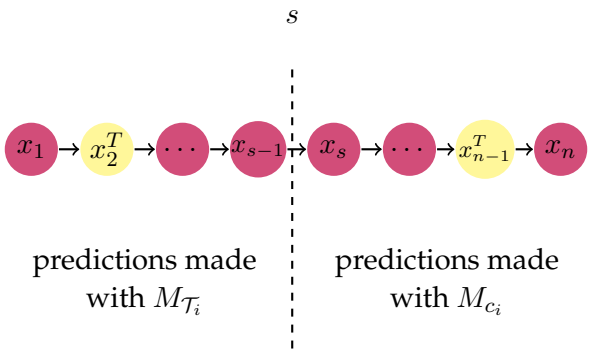

***Figure II.3.19:** Models used to perform prediction, before and after the switching value s. Actions in yellow correspond to Tools.*

We use a Random Forest classifier to dynamically assign the current prefix X_r to a cluster. As can be seen in Figure II.3.20, at each new user action, the prefix embeddings are inferred from the model used to compute session embeddings.

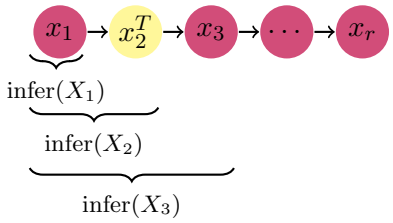

***Figure II.3.20:** Prefix inference for cluster assignment.*

The switching value s will be chosen as the smallest value providing a satisfying classification accuracy. To evaluate the classification we made a first experiment by clustering all the sessions where App_1, for system 30 and App_2, for system 50 were launched. After this step, all the prefixes may be labeled by the cluster number to which they belong. To test the classifier ability to reassign a prefix to the proper cluster, we use the prefixes of 80% oldest sessions as training set and the prefixes of the 20% remaining sessions as test set. We tested values of $s \in [\![5, 15]\!]$ and $k \in [\![2, 5]\!]$. Figures II.3.21 and II.3.22 show the results for systems 30 and 50, when $K = 2$. Results obtained for greater values of K are available in Appendix C. As they were less good, we decided to work with $K = 2$, unlike in Part I, Chapter 3 where the optimal number of clusters was determined with the Calinski–Harabasz score. The classifier ability to reassign a prefix to the corresponding cluster is better in the case of system 30. It is important to notice that the higher the value of s, the fewer predictions are concerned, prefixes are becoming longer and thus easier and easier to reassign to a cluster. As there is a significant gain for the system 50 (Figure II.3.22) in the classifier accuracy between s equal to 10 and s equal to 11, this value then remaining relatively the same between $s = 11$ and $s = 15$, we decided to work with $s = 11$ for both systems.

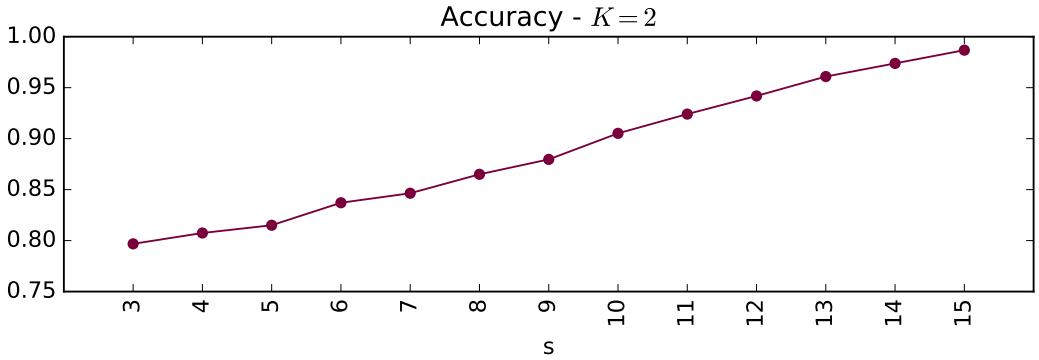

***Figure II.3.21:** System 30 evaluation of prefix assignment to clusters with $K = 2$ with Random Forest classifier and switching value $s \in [\![3, 15]\!]$.*

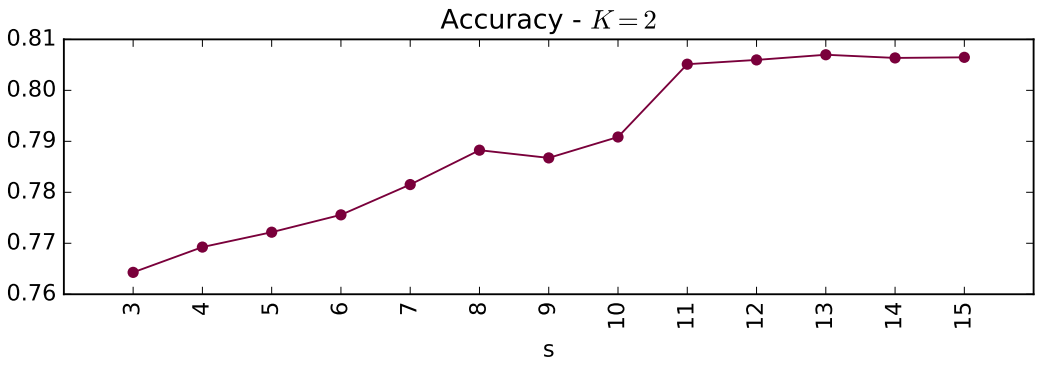

Figure II.3.22: ***System 50 evaluation of prefix assignment to clusters with*** $K = 2$ ***with Random Forest classifier and switching value*** $s \in [\![3, 15]\!]$.

Results of next action prediction based on clustering

Table II.3.12 shows the results obtained on systems 30 and 50. The prediction accuracy obtained by the LSTM trained on all the sessions $M_{\mathcal{T}_i}$ is compared to that obtained with models trained separately on 2 clusters. After the switching value, at each new user action, the current prefix is assigned to a cluster and the model which has been trained on this cluster is used to perform the prediction. Both training strategies are evaluated only on sessions where a specific application has been used (App_1 for system 30 and App_2 for system 50).

			Details		
System ID	$M_{\mathcal{T}_i}$	$M_{c_{i \in [\![1,2]\!]}}$	$M_{\mathcal{T}_i}$	M_{c_1}	M_{c_2}
30	0.666	0.643	0.693 (1994)	0.611 (18)	0.578 (1517)
50	0.855	0.869	0.857 (133)	0.933 (30)	0.8 (5)

Table II.3.12: ***Prediction accuracy of the model trained on the whole training set*** $M_{\mathcal{T}_i}$ ***and of the model trained on clusters*** $M_{c_{i \in [\![1,2]\!]}}$. ***Details provide the prediction accuracy obtained with each model and values between parentheses represent the number of Tools in the test set for each model.***

Training on clusters improves the prediction accuracy of system 50 but not that of system 30. If we analyze the predictions provided by each model more in details, we see that in the case of system 50, until the switching value, $M_{\mathcal{T}_{50}}$ provides the same prediction accuracy than on the whole test set, however, the predictions done with M_{c_1} on 30 *Tools* have an accuracy equal to 0.933 which improves the global accuracy. Here, the training on cluster 1 worked as expected and enabled to refine predictions of the ends of the sessions. In the case of system 30, the accuracy given by $M_{\mathcal{T}_{30}}$ until the switching value is reached is significantly higher than the global accuracy equal to 0.666, meaning that the prediction done at the end of the sessions by the global model are less accurate. However the model trained on clusters did not help the LSTM networks to improve their performances after the switching value, returning even worse

values than the global model.

II.3.6 Discussion

This chapter was dedicated to the next action prediction task. To tackle this issue, we proposed to take advantage of Long Short Term Memory recurrent neural networks and their ability to capture long range terms dependencies among user actions. As an alternative to the classic one-hot encoded inputs, we proposed to feed the LSTM network with action embeddings. This input representation enables the LSTM to outperform baseline approaches and to slightly improve LSTM performance taking as inputs one-hot encoded vectors. This lower dimensional distributed representation reduces at the same time the computational cost, making this method easier to deploy.

We tested different training strategies to determine the best method to train the LSTM network. Adding training sessions to a system's logs was not relevant, training on groups determined by the type of application was not relevant either. A final attempt was done by training the models on groups obtained by unsupervised clustering techniques, showing promising results for one of the tested system. We therefore recommend to use the logs specific to each system for the training of the model. An improvement in model performance could be achieved by using a larger number of sessions from the system in question, as suggested by the study of the correlation between the total number of sessions for each system and the Top_1 accuracy score.

So far, we only tested whether augmentating the dimension of the training set for one global system would improve the predictive performances. However, a very promising idea would also be to train jointly specific models for each system. A promising strategy for this purpose would be in the frame of hierarchical Bayesian neural networks [Joshi et al., 2017, Lacoste et al., 2018]: each system would have its own model with specific parameters but a joint distribution, as in classical Bayesian hierarchical models. More generally, when facing the prediction task for a new system, transfer learning methodologies [Pan et al., 2010] should be used in order to benefit from user sessions in slightly different contexts.

The proposed solution was tested on logs from 50 hospitals around the world, covering various medical specialties and proved to be effective on each of them. The prediction system returns an accuracy of 0.657 on average when one *Tool* is proposed and 0.911 when 4 *Tools* are proposed to the user, making this solution very attractive. Prototypes will be implemented to be validated with radiologists. Besides, the action prediction category might easily be expanded to other types of user actions, in addition to the *Tools*, keeping the same methodology.

While Long Short Term Memory neural networks have proven to be particularly powerful to solve sequence learning problems in recent years, some works claim that Attention networks are a very interesting alternative from a computational point of view. They propose to replace the recurrent nature of LSTM networks by self-attentions mechanisms [Yu et al., 2018]. This idea could be tested in our studycase.

Conclusion & Perspectives

This work addresses several critical issues regarding the global usability of a medical imaging software, based on the recorded history of user interaction with the interface described as elementary actions. Some solutions were proposed but do not always constitute definite answers and sometimes raise further questions. In this chapter we summarize the different contributions of this work. We then move to possible improvements and potential extensions.

Contributions

Crash Pattern Mining

We have studied the issue of software crashes from two perspectives. The first one aims at understanding the root causes of the crashes.

For this purpose we proposed to use a binomial test to determine which types of patterns are the most appropriate to represent crash signatures. We compared 4 types of patterns, taking into account proximity and/or order of the actions composing them. We concluded that subsequences of actions are the most appropriate for this task. This type of pattern takes into account the order of the actions, but does not impose any constraint on the number of actions comprised between 2 consecutive actions composing the pattern.

Contrary to our expectations, some determined crash signatures contain up to several dozen actions that are spread throughout the session. The initial hypothesis was rather that it was a few actions which triggered the crashes. These crash patterns are even more challenging to detect since they correspond to signatures that were not tested during the verification phase before the software deployment. While the primary purpose of this solution is to help resolve complaints, crash signatures could also be integrated into automatic tests for future versions. Although the crash signatures, when reproduced, do not always lead to crashes, they are of great help and make it possible to better target their origins.

Workflow Characterization

The overall improvement of the software interface requires a good knowledge of how it is used by its users. While we may have this information in the case of a close collaboration with a partner hospital, it is generally not possible to have this knowledge for all sites.

We proposed to use clustering algorithms in order to group similar sessions, making the as-

sumption that sessions contained in each cluster will represent one characteristic workflow. We compared 3 different session representations as inputs of several clustering techniques. We used a cluster validity index to determine the best partition. K-means algorithm fed with session embeddings obtained with Doc2Vec algorithm provided the best results. In order to visualize the sessions of each cluster we took advantage of a process mining tool.

This methodology enables to have an overview of the workflow trends for specific groups of users of a system. The information obtained in this way can be used for different purposes. For example, it makes it possible to highlight behaviors that were not expected. Sometimes the software is used differently from the way it was designed for. In these cases, it means either that the interface has to be accordingly modified or that the training courses given to users have to be readjusted. Besides, the observation of repetitive tasks may lead to simplifications or to the grouping of several actions into a single one.

Software Monitoring

As it is not possible to reduce the crash rate to zero, we propose to monitor the crash risk in real-time. The objective here was to compute a crash probability at each new user action. Should this probability exceed a certain threshold, an automatic backup system might be triggered, thus avoiding the user to lose his current work.

For this purpose we proposed to take advantage of the recurrent structure of Long Short Term Memory neural networks. We fed the LSTM with feature vectors composed of significant actions occurring in crash sessions. As we deal with highly imbalanced data, we evaluate our method using the area under ROC curves. Our method outperforms state-of- the-art classifiers fed with significant crash patterns in addition, for fair comparison. It is able to predict a satisfying rate of crashes while avoiding too many false alerts which would be quite disturbing in user workflows.

Another very attractive way to improve user experience is to provide them with adaptive interfaces, increasingly adapted to their preferences. By anticipating their needs in real-time, such interfaces enable to streamline user workflows and therefore improve their efficiency.

In our specific case, some actions in the interface are key actions in user workflows. We propose to focus on this category of actions called *Tools* in this work. The users might benefit from a dynamical toolbar containing in real time the k most likely *Tools* he his going to need. The prediction accuracy of such a dynamical toolbar is even more important for portable devices having a reduced screen size. Indeed, efficient predictions might avoid supplementary actions to reach the desired *Tool*. To tackle this task, we proposed to feed the LSTM with action embeddings. This method outperforms state-of-the-art sequence prediction algorithms as well as an LSTM fed with one-hot encoded vectors. We conducted experiments to determine the best training strategy. We concluded that the best training set is composed of each system's specific sessions, additional sessions from systems of the same country or in which the same applications was executed did not generally improve the prediction accuracy, suggesting that there is

a high variability in terms of use between users of different systems. For this reason, we also tested a training strategy where training sets are obtained by clustering on each system's specific sessions. The idea here was to refine the predictions with models trained on specific sets. This method has proven to be effective for one of the two systems studied.

Perspectives

Crash Pattern Mining

Even if the significant subsequences of user actions from crash sessions allow to better target the origin of crashes, the problem is not fully resolved. It is hoped that the enrichment of the log content will allow the determination of more accurate crash signatures. Besides, other areas of research might be exploited. For example, user workflow might be coupled to hardware characteristics or system status to highlight crash signatures as well as system configurations being more likely to lead to defects.

Workflow Characterization

In order to determine trends of use we compared three session representations as inputs of clustering algorithms. Further improvements might involve the comparison of embeddings computed with Doc2Vec to other types of sessions embeddings. For example, an extension of Doc2Vec, namely Doc2VecC [Chen, 2017] was recently proposed. They represent documents by averaging the embeddings of the words it contains. They apply a corruption mechanism making the embeddings of common and non-discriminative words close to zero. This method matches and sometimes outperforms Doc2Vec representations on document classification tasks. In these representations, the order of the actions is not taken into account. Even if in our application, there is a natural order in which actions are performed, inherent to the workflow, it might be interesting to evaluate the performance of embeddings obtained with sequence to sequence autoencoders [Sutskever et al., 2014]. With this methodology, a reccurrent neural network is trained to predict the next action at each time step. The hidden state at the end of the session is then used as the session embeddings. Deep clustering [Caron et al., 2018] reuses the principle of sequence to sequence autoencoders and proposes to cluster the embeddings at each iteration in the training and to dynamically readjust the embeddings values by adding a clustering metric evaluation in the global loss of the network. This approach should apply well to our case and might be worth explored.

Software Monitoring

The limited performance of the LSTM fed with feature vectors to predict crashes of the software suggests that there is room for improvement. Similarly, the dynamical toolbar containing in real time the k most likely next *Tools* will be all the more attractive with increased predictive performance. We took advantage of the recurrent structure of LSTMs which have proven to be very adapted for sequence learning, even though alternatives exist. For example, recent works proposed attention networks. They replace sequential processing by an attention mechanism able to learn which information is relevant given a particular context. These self-attention mecha-

nisms have some advantages over recurrent networks [Vaswani et al., 2017] such as reduced complexity by layer and they are easily parallelizable thus implying lower training costs.

Here again, other input represensations might be tested, for example the efficiency for this prediction task of embeddings computed with the GloVe algorithm [Pennington et al., 2014] might be compared to those obtained with Word2Vec. This algorithm computes word representations using a word-word cooccurrence matrix and a weighting function that assigns lower weights to distant pairs of words. This methodology is proving to be more effective for some datasets.

Besides, our solutions assume that the user has already used the software for some time. An important issue which still needs to be adressed is the prediction for new users who do not yet have a usage history. Many solutions have been proposed to tackle the cold start problem in particular in the recommender systems field [Lika et al., 2014]. Naive methods propose to recommend items on their popularity [Park and Chu, 2009], while warm start methods ask new users to rate a few items to compute first recommendations [Contardo et al., 2015].

Finally, we have not yet fully taken advantage of the fact that our software are deployed worldwide, in many different contexts. One technical reason for this was the fact that the data of a large number of systems were only available in the last stage of this work. Our first attempt was to consider a global model trained on the sessions of a whole country for example. Other more refined approaches based on transfer learning [Pan et al., 2010] seem more pertinent to tackle the diversity of contexts. For example we could imagine specific models for each specific system, but not trained independently, the frame of hierarchical Bayesian neural networks [Joshi et al., 2017, Lacoste et al., 2018] seems a very promising perspective in this regard.

Temporal Dimension

All of this work does not take into account the time dimension. However, more sophisticated models might be developed by adding this information. Some pattern mining algorithms that explicitely consider time have been proposed [Chen et al., 2003] and take into account the time intervals between two consecutive items. Such methods might be applied to tackle crash signatures detection.

In the same way, in the case of software monitoring we could consider models that not only predict the next event but also the date at which the event will occur [Xiao et al., 2017]. Other works use time to improve the predictions themselves [Li et al., 2017], arguing that time intervals between events carry important information about the sequence.

Conclusion

This work is in line with the trend towards the development of artificial intelligence tools for health, in particular for diagnosis assistance. Medical imaging solutions especially have been very successful, sometimes performing better than humans. However, one of the dangers of these approaches in a field such as medicine is the black box aspect and the lack of explainability of algorithms [Castelvecchi, 2016]. This results in the loss of control of the process by medicine. In our work, we solve this problem by providing a detailed understanding of the

radiologist's workflow and use this analysis to help gain efficiency by intervening at specific points of the process which remains fully controlled by the radiologists: we help but do not replace.

Appendices

Appendix A

Session Embeddings Hyperparameters Selection

We provide here the results obtained with the Agglomerative Hierarchical Clustering algorithm with Average, Complete and Ward linkages as well as with the Spectral Clustering algorithm. Each figure shows the maximal Calinski–Harabasz score obtained when varying the embedding size and the $\min_{\text{count}}$ value representing the threshold under which words with lower frequency are ignored from the vocabulary.

A.1 System 30 (App 1)

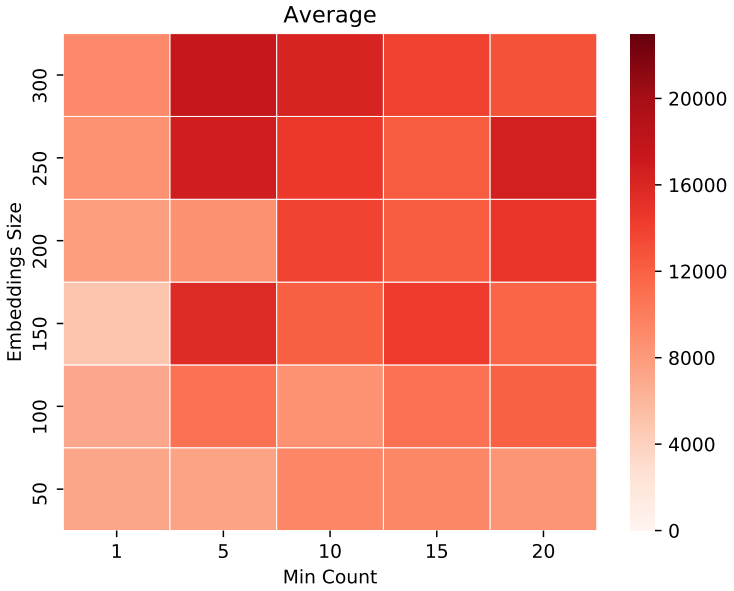

Figure A.1: ***Maximal Calinski–Harabasz score obtained with*** $k \in [\![2, K]\!]$, $\max_{k \in [\![2,K]\!]}\{CH(C_k)\}$ ***for embedding size*** E ***in*** $[50, 350]$ ***with a step of 50 and the*** min_{count} ***value in*** $\{1, 5, 10, 15, 20\}$ ***in the case of system 30 (App 1) and the Agglomerative Hierarchical clustering with the Average linkage algorithm.***

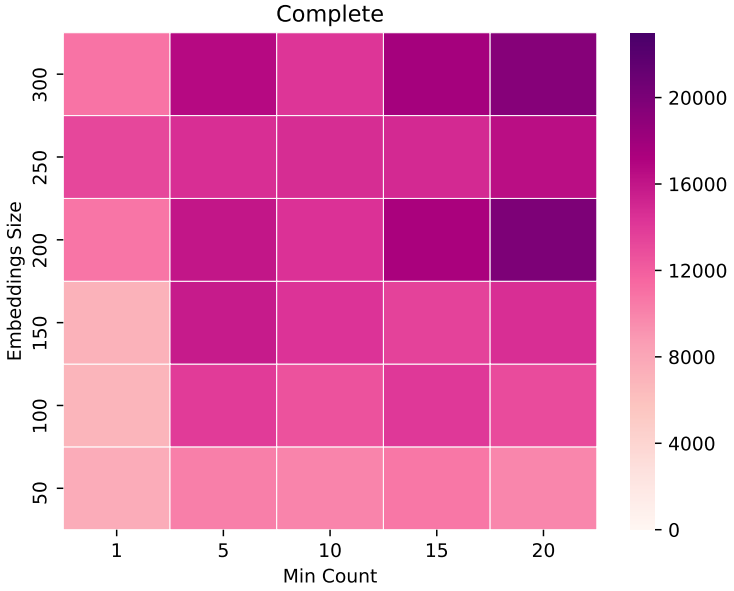

Figure A.2: ***Maximal Calinski–Harabasz score obtained with*** $k \in [\![2, K]\!]$, $\max_{k \in [\![2,K]\!]}\{CH(C_k)\}$ ***for embedding size*** E ***in*** $[50, 350]$ ***with a step of 50 and the*** min_{count} ***value in*** $\{1, 5, 10, 15, 20\}$ ***in the case of system 30 (App 1) and the Agglomerative Hierarchical clustering with the Complete linkage algorithm.***

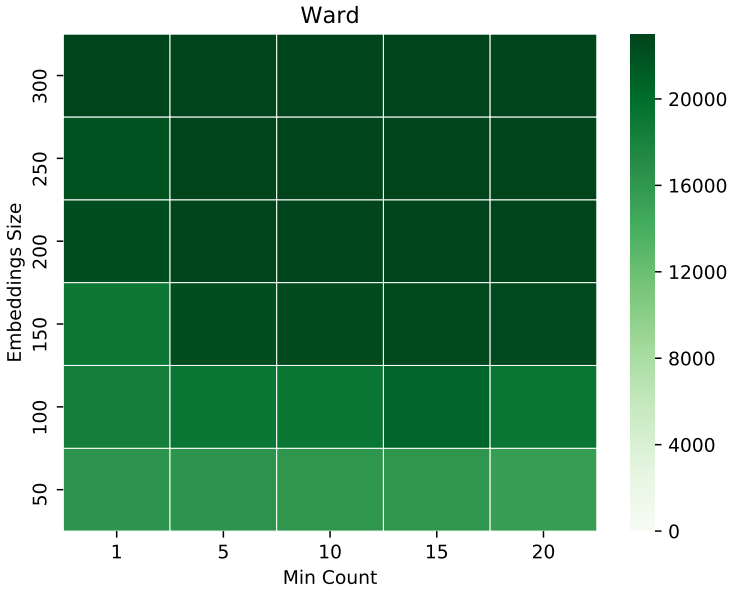

Figure A.3: ***Maximal Calinski–Harabasz score obtained with*** $k \in [\![2, K]\!]$, $\max_{k \in [\![2,K]\!]}\{CH(C_k)\}$ ***for embedding size*** E ***in*** $[50, 350]$ ***with a step of 50 and the*** min_{count} ***value in*** $\{1, 5, 10, 15, 20\}$ ***in the case of system 30 (App 1) and the Agglomerative Hierarchical clustering with the Ward linkage algorithm.***

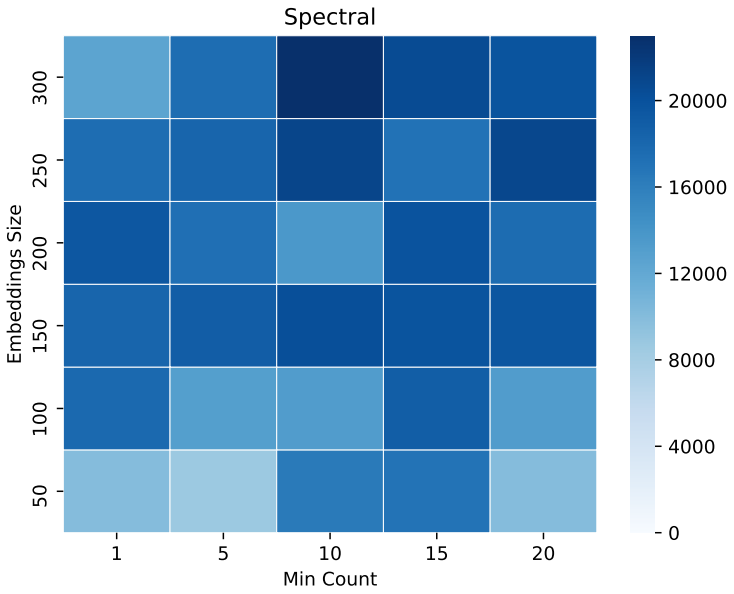

Figure A.4: Maximal Calinski–Harabasz score obtained with $k \in [\![2, K]\!]$, $\max_{k \in [\![2,K]\!]}\{CH(C_k)\}$ *for embedding size* E *in* $[50, 350]$ *with a step of 50 and the* min_{count} *value in* $\{1, 5, 10, 15, 20\}$ *in the case of system 30 (App 1) and the Spectral clustering algorithm.*

A.2 System 50 (App 2)

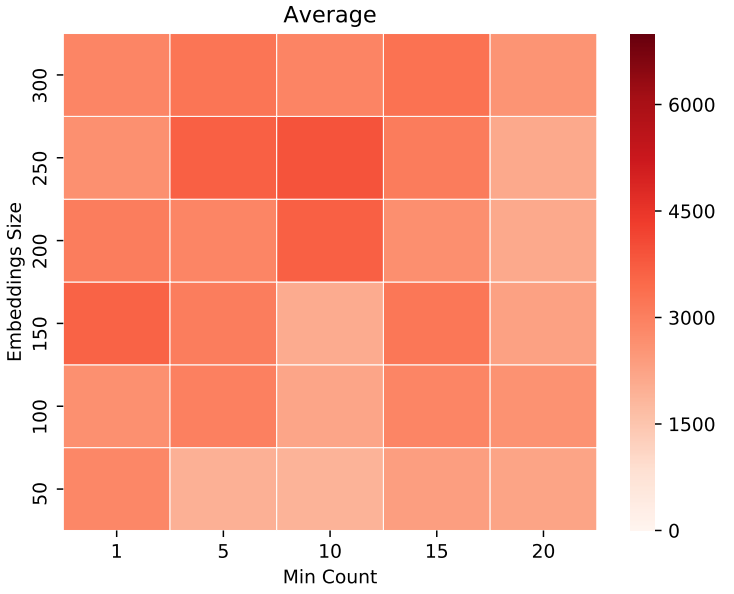

Figure A.5: Maximal Calinski–Harabasz score obtained with $k \in [\![2, K]\!]$, $\max_{k \in [\![2,K]\!]}\{CH(C_k)\}$ *for embedding size* E *in* $[50, 350]$ *with a step of 50 and the* min_{count} *value in* $\{1, 5, 10, 15, 20\}$ *in the case of system 50 (App 2) and the Agglomerative Hierarchical clustering with the Average linkage algorithm.*

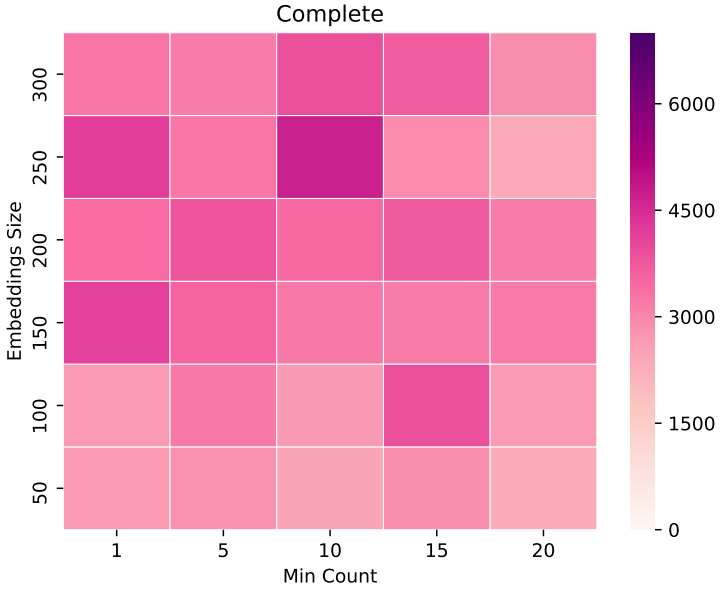

Figure A.6: ***Maximal Calinski–Harabasz score obtained with*** $k \in [\![2, K]\!]$, $\max_{k \in [\![2,K]\!]}\{CH(C_k)\}$ ***for embedding size*** E ***in*** $[50, 350]$ ***with a step of 50 and the*** min_{count} ***value in*** $\{1, 5, 10, 15, 20\}$ ***in the case of system 40 (App 2) and the Agglomerative Hierarchical clustering with the Complete linkage algorithm.***

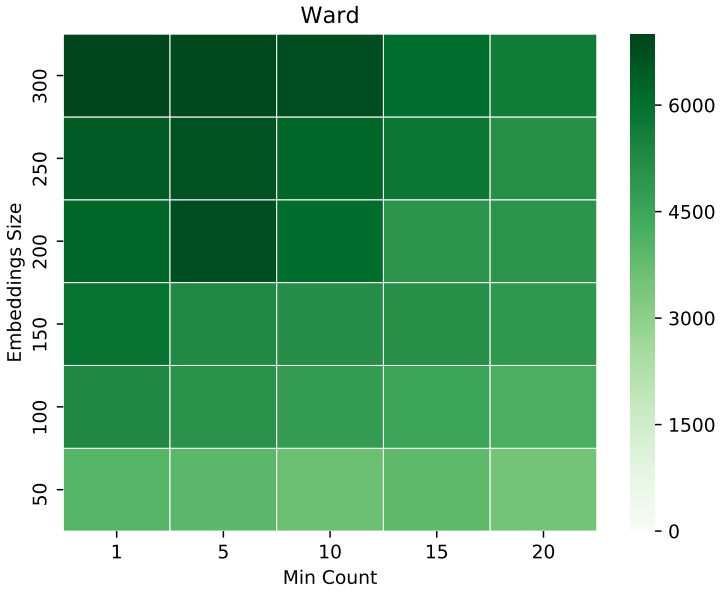

Figure A.7: ***Maximal Calinski–Harabasz score obtained with*** $k \in [\![2, K]\!]$, $\max_{k \in [\![2,K]\!]}\{CH(C_k)\}$ ***for embedding size*** E ***in*** $[50, 350]$ ***with a step of 50 and the*** min_{count} ***value in*** $\{1, 5, 10, 15, 20\}$ ***in the case of system 50 (App 2) and the Agglomerative Hierarchical clustering with the Ward linkage algorithm.***

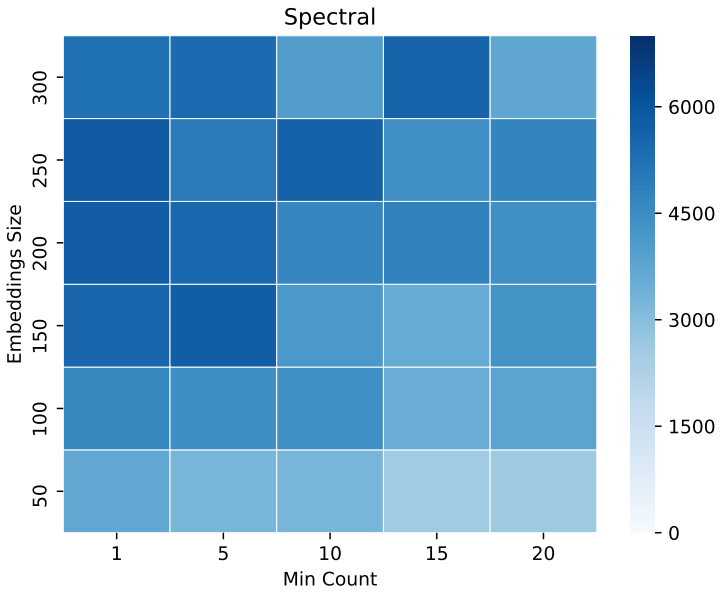

Figure A.8: ***Maximal Calinski–Harabasz score obtained with*** $k \in [\![2, K]\!]$***,*** $\max_{k \in [\![2,K]\!]}\{CH(C_k)\}$ ***for embedding size*** E ***in*** $[50, 350]$ ***with a step of 50 and the*** min_{count} ***value in*** $\{1, 5, 10, 15, 20\}$ ***in the case of system 50 (App 2) and the Spectral clustering algorithm.***

Appendix B

Tests to Define the Best Training Strategy

This appendix provides for each system of each studied group of hospitals (India, United States of America and Japan) the number of sessions used as training and test sets as well as the corresponding number of *Tools* for each of the 2 most executed applications.

B.1 Indian Systems

B.1.1 Sessions

	all sessions		App_1		App_2	
hospital	training	test	training	test	training	test
1	4488	1123	*2*	0	137	72
10	3416	855	*3*	1	31*	61
12	9616	2405	*0*	0	*0*	0
15	4997	1250	*0*	0	209	6
18	3800	951	*0*	0	1017	223
21	3791	948	*0*	0	*0*	0
22	4432	1109	*1*	0	*8*	2
23	5646	1412	*0*	0	*0*	0
26	8812	2204	*0*	0	238	39
30	7825	1957	5816	1380	*0*	0
31	3509	878	*0*	0	*0*	0
32	4374	1094	108*	11	*0*	0
46	4399	1100	3148	549	*0*	0

Table B.1: ***Number of training and test sessions for the whole dataset and specific subsets, for each Indian hospital. Too small training sets are shown in italics. Small but trainable training sets are signaled with an asterisk.***

B.1.2 Corresponding Number of Tools

	all sessions		App_1		App_2	
hospital	training	test	training	test	training	test
1	6932	1867	0	0	878	578
10	8984	2169	2	1	99	129
12	10322	2759	0	0	0	0
15	9794	2569	0	0	545	19
18	7311	1978	0	0	2356	795
21	10980	2554	0	0	0	0
22	9039	2239	1	0	0	0
23	3954	990	0	0	0	0
26	5070	1147	0	0	627	64
30	16999	4054	15853	3529	0	0
31	4679	961	0	0	0	0
32	6453	1768	70	14	0	0
46	5933	1282	5160	987	0	0

Table B.2: Number of Tools in training and test sessions for the whole dataset and specific subsets, for each Indian hospital.

B.2 American Systems

B.2.1 Sessions

	all sessions		App_3		App_2	
System ID	training	test	training	test	training	test
11	1428	358	322	66	107	33
16	1181	296	561	145	41*	15
20	1825	457	*7*	2	233	58
25	2349	588	880	216	292	84
35	4712	1178	695	142	577	186
39	1482	371	395	94	160	24
41	1186	297	340	58	54*	26
45	645	162	259	82	*0*	0

Table B.3: Number of training and test sessions for the whole dataset and specific subsets, for each American hospital. Too small training sets are shown in italics. Small but trainable training sets are signaled with an asterisk.

B.2.2 Corresponding Number of Tools

	all sessions		App_3		App_2	
Hospital	training	test	training	test	training	test
11	11549	2933	4326	929	652	228
16	12263	2948	8152	2030	200	49
20	17096	3805	42	64	1644	326
25	12446	3496	3480	1036	289	44
35	13658	3627	958	206	591	208
39	11238	2882	3881	1058	790	89
41	6814	1719	1599	482	115	89
45	7885	2365	3731	1416	0	0

Table B.4: ***Number of Tools in training and test sessions for the whole dataset and specific subsets, for each American hospital.***

B.3 Japanese Systems

B.3.1 Sessions

	all sessions		App_2		App_4	
hospital	training	test	training	test	training	test
3	2521	631	*0*	0	*0*	0
6	2869	718	1754	452	139	33
34	2944	737	208	13	76*	15
36	1693	424	*0*	0	*0*	0
40	1709	428	*1*	0	111	23

Table B.5: ***Number of training and test sessions for the whole dataset and specific subsets, for each Japanese hospital. Too small training sets are shown in italics. Small but trainable training sets are signaled with an asterisk.***

B.3.2 Corresponding Number of Tools

	all sessions		App_2		App_4	
hospital	training	test	training	test	training	test
3	33714	8862	0	0	0	0
6	11933	3006	452	1331	1077	413
34	13076	4052	637	126	482	181
36	19363	4634	0	0	0	0
40	12408	2633	23	0	1514	294

*Table B.6: **Number of Tools in training and test sessions for the whole dataset and specific subsets, for each Japanese hospital.***

Appendix C

Cluster Assignment

This appendix provides the classification accuracy according to the switching values with 3 and 4 clusters.

C.1 System 30

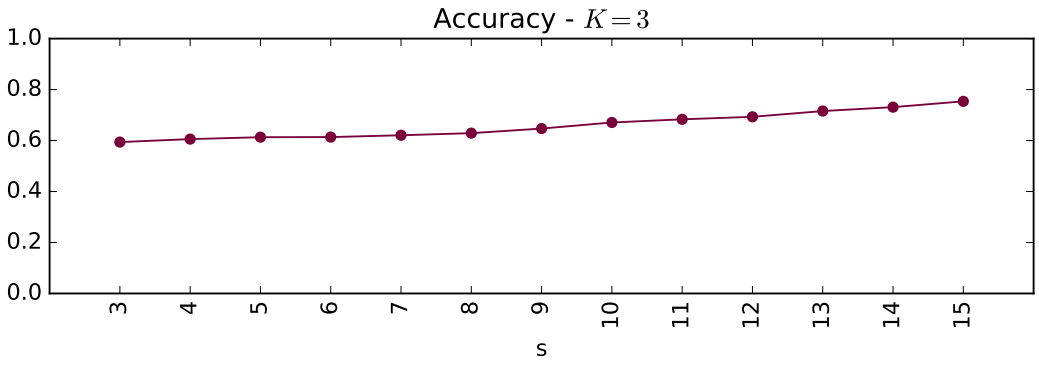

Figure C.1: ***System 30 evaluation of prefix assignment to clusters with $K = 3$ with Random Forest classifier and switching value $s \in [\![3, 15]\!]$.***

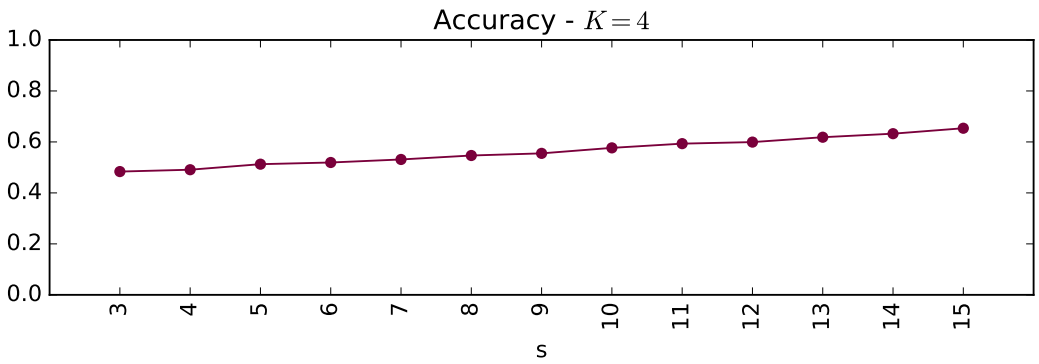

Figure C.2: ***System 30 evaluation of prefix assignment to clusters with $K = 4$ with Random Forest classifier and switching value $s \in [\![3, 15]\!]$.***

C.2 System 50

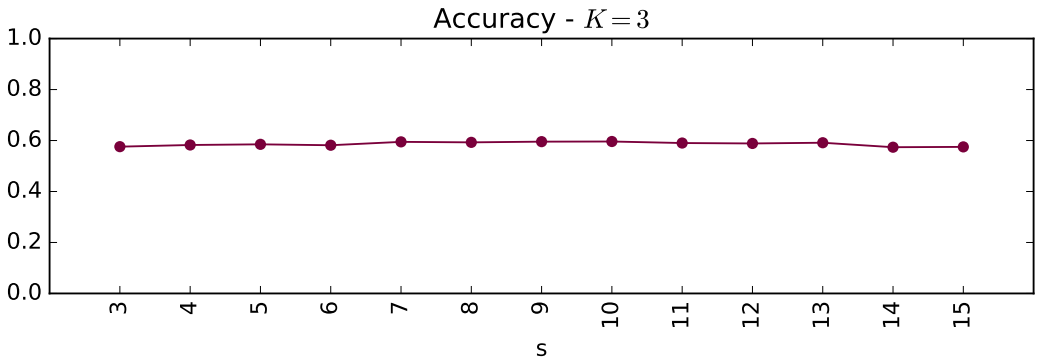

Figure C.3: ***System 50 evaluation of prefix assignment to clusters with $K = 3$ with Random Forest classifier and switching value $s \in [\![3, 15]\!]$.***

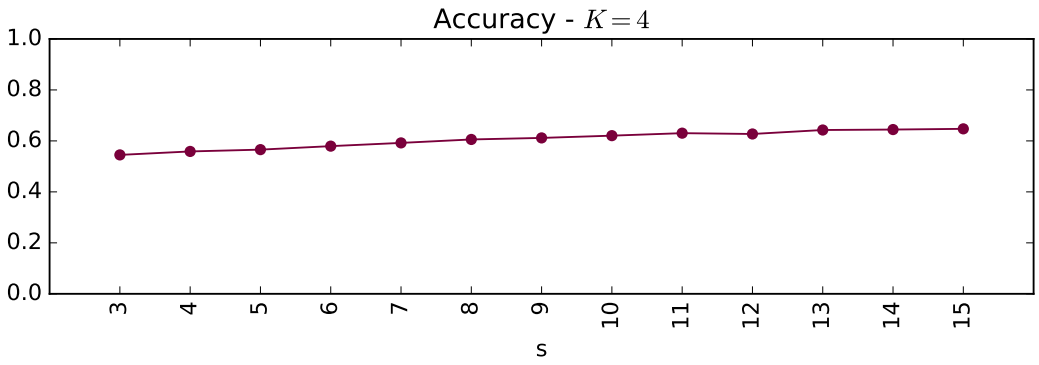

Figure C.4: ***System 50 evaluation of prefix assignment to clusters with $K = 4$ with Random Forest classifier and switching value $s \in [\![3, 15]\!]$.***

Appendix D

Significant Actions

To tackle the crash monitoring task, we propose to feed the LSTM with feature vectors counting the number of occurrences of significant actions from crash sessions. Table D.1 provides the number of significant actions at different significance threshold.

	τ			
System ID	0.01	0.02	0.05	0.1
8	21	22	24	29
17	26	26	28	30
20	17	20	23	23
33	20	23	31	41
45	18	21	32	35

Table D.1: Number of significant actions for significance threshold τ between 0.01 and 0.1.

Appendix E

t-distributed Stochastic Neighbor Embedding (t-SNE)

In order to represent our clustering results in two dimensions we took advantage of the t-SNE algorithm [Maaten and Hinton, 2008] that we will describe in this appendix.

Contrary to Principal Component Analysis, t-SNE is a non linear method dimensionality reduction algorithm. It enables to represent in 2 or 3 dimensions multi-dimensional data. The principle of t-SNE is to model similar objects from high-dimensiomal data by close points in 2 dimensions with a high probability. Similarly, dissimilar objects from high-dimensional data will be represented by distant points in 2 dimensions with a high probability. It does so by minimizing the Kullback-Leibler divergence between the joint probabilities of high-dimensional

data and low-dimensional representations, as described in Algorithm 6.

Algorithm 6: Simple version of t-Distributed Stochastic Neighbor Embedding [Maaten and Hinton, 2008].

Inputs: Dataset $X = \{x_1, x_2, \ldots, x_n\}$, cost function parameters: perplexity *Perp*, optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$.

Output: Low-dimensional data representation $Y^{(T)} = \{y_1, y_2, \ldots, y_n\}$.

begin

Compute pairwise affinities $p_{i|j}$ using:

$$p_{i|j} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

where σ_i is the variance of the Gaussian centered on point x_i (see details about σ_i selection with *Perp* in original paper);

Set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$;

Sample initial solution $Y^{(0)} = \{y_1, y_2, \ldots, y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$.

for $t = 1$ *to* T **do**

Compute low-dimensional affinities q_{ij} using:

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l} (||y_k - y_l||^2)^{-1}};$$

Compute gradient of the Kullback-Leibler divergence $\frac{\delta C}{\delta Y}$ between P and the Student-t based joint probability distribution Q using:

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1};$$

Set $Y^{(t)} = Y^{(t-1)} + \eta \frac{\delta C}{\delta Y} + \alpha(t)(Y^{(t-1)} - Y^{(t-2)})$.

Publications & Patents

This work resulted in the following publications:

- [Adam et al., 2016] C. Adam, A. Aliotti, P.-H. Cournède. Learning from user workflows for the characterization and prediction of software crashes. *In 2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 1023–1030.
- [Adam et al., 2019] C. Adam, A. Aliotti, F. D. Malliaros, P.-H. Cournède. Dynamic monitoring of software use with recurrent neural networks. *Submitted*.

Besides, two patents applications were submitted in the United States:

- [Adam et al., 2017a] C. Adam, A. Aliotti, T. Almecija. Adaptive User interface for medical software.
- [Adam et al., 2017b] C. Adam, A. Aliotti, T. Almecija, P.-H. Cournède. Determining and using a 'gold standard' for adaptive user interfaces.

Bibliography

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Adam et al., 2017a] Adam, C., Aliotti, A., and Almecija, T. (2017a). Adaptive user interface for medical software - Application in the United States.

[Adam et al., 2017b] Adam, C., Aliotti, A., and Almecija, T. (2017b). Determining and using a 'gold standard' for adaptive user interfaces - Application in the United States.

[Adam et al., 2016] Adam, C., Aliotti, A., and Cournède, P.-H. (2016). Learning from user workflows for the characterization and prediction of software crashes. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 1023–1030.

[Adam et al., 2019] Adam, C., Aliotti, A., Malliaros, F. D., and Cournède, P.-H. (2019). Dynamic monitoring of software use with recurrent neural networks. *Submitted*.

[Adomavicius and Tuzhilin, 2001] Adomavicius, G. and Tuzhilin, A. (2001). Using data mining methods to build customer profiles. *Computer*, 34(2):74–82.

[Aggarwal and Han, 2014] Aggarwal, C. C. and Han, J. (2014). *Frequent pattern mining*. Springer.

[Aggarwal and Wang, 2010] Aggarwal, C. C. and Wang, H. (2010). *Managing and mining graph data*. Springer.

[Aggarwal and Zhai, 2012] Aggarwal, C. C. and Zhai, C. (2012). A survey of text clustering algorithms. In *Mining text data*, pages 77–128. Springer.

[Agrawal et al., 1994] Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499.

[Agrawal and Agrawal, 2015] Agrawal, S. and Agrawal, J. (2015). Survey on anomaly detection using data mining techniques. *Procedia Computer Science*, 60:708–713.

[Ansari et al., 2011] Ansari, Z., Azeem, M., Ahmed, W., and Babu, A. V. (2011). Quantitative evaluation of performance and validity indices for clustering the web navigational sessions. *WCSIT*.

[Arbelaitz et al., 2013] Arbelaitz, O., Gurrutxaga, I., Muguerza, J., Pérez, J. M., and Perona, I. (2013). An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1):243–256.

[Arthur and Vassilvitskii, 2007] Arthur, D. and Vassilvitskii, S. (2007). K-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics.

[Axelsson, 2000] Axelsson, S. (2000). Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Department of Computer Engineering, Chalmers University.

[Banerjee and Dave, 2004] Banerjee, A. and Dave, R. N. (2004). Validating clusters using the Hopkins statistic. In *Proceedings IEEE International Conference, Fuzzy systems*, volume 1, pages 149–153.

[Begleiter et al., 2004] Begleiter, R., El-Yaniv, R., and Yona, G. (2004). On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, 22:385–421.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[Borisov et al., 2016] Borisov, A., Markov, I., de Rijke, M., and Serdyukov, P. (2016). A neural click model for web search. In *Proceedings of the 25th International Conference on World Wide Web*, pages 531–541. International World Wide Web Conferences Steering Committee.

[Bringmann et al., 2009] Bringmann, B., Nijssen, S., and Zimmermann, A. (2009). Pattern-based classification: a unifying perspective. *In A. Knobbe J. Furnkranz (Eds.), LeGo'09, Proceedings of the ECML PKDD 2009 Workshop 'From Local Patterns to Global Models'*.

[Browne, 2016] Browne, D. (2016). *Adaptive user interfaces*. Elsevier.

[Brun et al., 2007] Brun, M., Sima, C., Hua, J., Lowey, J., Carroll, B., Suh, E., and Dougherty, E. R. (2007). Model-based evaluation of clustering validation measures. *Pattern recognition*, 40(3):807–824.

[Caliński and Harabasz, 1974] Caliński, T. and Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27.

[Cao et al., 2017] Cao, J., Greenberg, A., Sharma, A., Su, Y., Nicholas, K., Mohsin, M., Jurewicz, J., Huang, W., Sharifi, M., Sidhom, B., et al. (2017). Search query predictions by a keyboard. US Patent 9,720,955.

[Caron et al., 2018] Caron, M., Bojanowski, P., Joulin, A., and Douze, M. (2018). Deep clustering for unsupervised learning of visual features. *In European Conference on Computer Vision*.

[Castelvecchi, 2016] Castelvecchi, D. (2016). Can we open the black box of AI? *Nature News*, 538(7623):20.

[Cawsey et al., 2007] Cawsey, A., Grasso, F., and Paris, C. (2007). Adaptive information for consumers of healthcare. In *The adaptive web*, pages 465–484. Springer.

[Chaofeng, 2009] Chaofeng, L. (2009). Research on web session clustering. *Journal of Software*, 4(5):460–468.

[Chawla, 2005] Chawla, N. V. (2005). Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 853–867. Springer.

[Chen, 2017] Chen, M. (2017). Efficient vector representation for documents through corruption. *5th International Conference on Learning Representations*.

[Chen et al., 2003] Chen, Y.-L., Chiang, M.-C., and Ko, M.-T. (2003). Discovering time-interval sequential patterns in sequence databases. *Expert Systems with Applications*, 25(3):343–354.

[Cheng et al., 2007] Cheng, H., Yan, X., Han, J., and Hsu, C.-W. (2007). Discriminative frequent pattern analysis for effective classification. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 716–725. IEEE.

[Contardo et al., 2015] Contardo, G., Denoyer, L., and Artières, T. (2015). Representation learning for cold-start recommendation. In *ICLR 2015 Workshop*.

[Dai et al., 2014] Dai, A. M., Olah, C., and Le, Q. V. (2014). Document embedding with paragraph vectors. *NIPS Deep Learning Workshop*.

[Davison and Hirsh, 1998] Davison, B. D. and Hirsh, H. (1998). Predicting sequences of user actions. In *Notes of the AAAI/ICML 1998 Workshop on Predicting the Future: AI Approaches to Time-Series Analysis*, pages 5–12.

[Dev and Liu, 2017] Dev, H. and Liu, Z. (2017). Identifying frequent user tasks from application logs. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, pages 263–273. ACM.

[Dixit and Bhatia, 2015] Dixit, V. S. and Bhatia, S. K. (2015). Refinement and evaluation of web session cluster quality. *International Journal of System Assurance Engineering and Management*, 6(4):373–389.

[Donkers et al., 2017] Donkers, T., Loepp, B., and Ziegler, J. (2017). Sequential user-based recurrent neural network recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 152–160. ACM.

[Du et al., 2017] Du, M., Li, F., Zheng, G., and Srikumar, V. (2017). Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298. ACM.

[Duda et al., 2012] Duda, R. O., Hart, P. E., and Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

[Dunn, 1974] Dunn, J. C. (1974). Well-separated clusters and optimal fuzzy partitions. *Journal of cybernetics*, 4(1):95–104.

[Eichinger et al., 2008] Eichinger, F., Böhm, K., and Huber, M. (2008). Mining edge-weighted call graphs to localise software bugs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 333–348. Springer.

[Fawcett, 2004] Fawcett, T. (2004). ROC graphs: Notes and practical considerations for researchers. *Machine learning*, 31(1):1–38.

[Forrest et al., 1996] Forrest, S., Hofmeyr, S. A., Somayaji, A., and Longstaff, T. A. (1996). A sense of self for Unix processes. In *Proceedings IEEE Symposium on Security and Privacy*, pages 120–128. IEEE.

[Friedman et al., 2001] Friedman, J., Hastie, T., and Tibshirani, R. (2001). *The elements of statistical learning*, volume 1. Springer series in statistics New York, NY, USA.

[Gajos et al., 2006] Gajos, K. Z., Czerwinski, M., Tan, D. S., and Weld, D. S. (2006). Exploring the design space for adaptive graphical user interfaces. In *Proceedings of the working conference on Advanced visual interfaces*, pages 201–208. ACM.

[Gers and Schmidhuber, 2000] Gers, F. A. and Schmidhuber, J. (2000). Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 189–194. IEEE.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.

[Gopalratnam and Cook, 2007] Gopalratnam, K. and Cook, D. J. (2007). Online sequential prediction via incremental parsing: The active Lezi algorithm. *IEEE Intelligent Systems*, 22(1).

[Graves, 2013] Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

[Graves and Jaitly, 2014] Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *International Conference on Machine Learning*, pages 1764–1772.

[Greenstein-Messica et al., 2017] Greenstein-Messica, A., Rokach, L., and Friedman, M. (2017). Session-based recommendations using item embedding. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces*, pages 629–633. ACM.

[Greff et al., 2016] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2016). LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*.

[Gu et al., 2017] Gu, X., Angelov, P. P., Kangin, D., and Principe, J. C. (2017). A new type of distance metric and its use for clustering. *Evolving Systems*, 8(3):167–177.

[Gueniche et al., 2013] Gueniche, T., Fournier-Viger, P., and Tseng, V. S. (2013). Compact Prediction Tree: A lossless model for accurate sequence prediction. In *ADMA (2)*, pages 177–188.

[Hagberg et al., 2008] Hagberg, A., Swart, P., and S Chult, D. (2008). Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Laboratory, Los Alamos, NM (United States).

[Halkidi and Vazirgiannis, 2001] Halkidi, M. and Vazirgiannis, M. (2001). Clustering validity assessment: Finding the optimal partitioning of a data set. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 187–194. IEEE.

[Han et al., 2007] Han, J., Cheng, H., Xin, D., and Yan, X. (2007). Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86.

[Han et al., 2000] Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. In *ACM SIGMOD record*, volume 29, pages 1–12. ACM.

[Hartmann and Schreiber, 2007] Hartmann, M. and Schreiber, D. (2007). Prediction algorithms for user actions. In *In Hinneburg, A., editor, Proceedings of Lernen Wissen Adaption*, pages 349–354.

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

[Hofmeyr et al., 1998] Hofmeyr, S. A., Forrest, S., and Somayaji, A. (1998). Intrusion detection using sequences of system calls. *Journal of computer security*, 6(3):151–180.

[Hotho et al., 2005] Hotho, A., Nürnberger, A., and Paaß, G. (2005). A brief survey of text mining. In *LDV Forum-GLDV J. Comput. Linguistics Lang. Technol.*, volume 20, pages 19–62.

[Huang, 2008] Huang, A. (2008). Similarity measures for text document clustering. In *Proceedings of the sixth New Zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56.

[Jacobs and Blockeel, 2002] Jacobs, N. and Blockeel, H. (2002). Sequence prediction with mixed order Markov chains. In *Proceedings of BNAIC'02-Belgian-Dutch Conference on Artificial Intelligence*, pages 147–154.

[Jain, 2010] Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern recognition letters*, 31(8):651–666.

[Joshi et al., 2017] Joshi, A., Ghosh, S., Betke, M., Sclaroff, S., and Pfister, H. (2017). Personalizing gesture recognition using hierarchical bayesian neural networks. In *30th IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2017)*.

[Jozefowicz et al., 2015] Jozefowicz, R., Zaremba, W., and Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350.

[Katz, 1987] Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing*, 35(3):400–401.

[Kim et al., 2017] Kim, H. K., Kim, H., and Cho, S. (2017). Bag-of-Concepts: Comprehending document representation through clustering words in distributed representation. *Neurocomputing*, 266:336–352.

[Lacoste et al., 2018] Lacoste, A., Oreshkin, B., Chung, W., Boquet, T., Rostamzadeh, N., and Krueger, D. (2018). Uncertainty in multitask transfer learning. *arXiv preprint arXiv:1806.07528*.

[Lawson and Jurs, 1990] Lawson, R. G. and Jurs, P. C. (1990). New index for clustering tendency and its application to chemical problems. *Journal of chemical information and computer sciences*, 30(1):36–41.

[Le and Mikolov, 2014] Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32, pages 1188–1196.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[Leemans, 2017] Leemans, S. (2017). Inductive visual miner manual. Technical report.

[Li and Wang, 2008] Li, C. and Wang, J. (2008). Efficiently mining closed subsequences with gap constraints. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pages 313–322. SIAM.

[Li et al., 2017] Li, Y., Du, N., and Bengio, S. (2017). Time-dependent representation for neural event sequence prediction. *arXiv preprint arXiv:1708.00065*.

[Lika et al., 2014] Lika, B., Kolomvatsos, K., and Hadjiefthymiades, S. (2014). Facing the cold start problem in recommender systems. *Expert Systems with Applications*, 41(4):2065–2073.

[Liu et al., 2005] Liu, C., Yan, X., Yu, H., Han, J., and Philip, S. Y. (2005). Mining behavior graphs for backtrace of noncrashing bugs. In *Proceedings of the Fifth SIAM International Conference on Data Mining*, pages 286–297.

[Livshits and Zimmermann, 2005] Livshits, B. and Zimmermann, T. (2005). DynaMine: finding common error patterns by mining software revision histories. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 296–305. ACM.

[Lo et al., 2009] Lo, D., Cheng, H., Han, J., Khoo, S.-C., and Sun, C. (2009). Classification of software behaviors for failure detection: a discriminative pattern mining approach. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 557–566. ACM.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research*, 9:2579–2605.

[Mijangos et al., 2017] Mijangos, V., Sierra, G., and Montes, A. (2017). Sentence level matrix representation for document spectral clustering. *Pattern Recognition Letters*, 85:29–34.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *International Conference on Learning Representations Workshop*.

[Moon et al., 2016] Moon, C., Medd, D., Jones, P., Harenberg, S., Oxbury, W., and Samatova, N. F. (2016). Online prediction of user actions through an ensemble vote from vector representation and frequency analysis models. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 90–98. SIAM.

[Pan et al., 2010] Pan, S. J., Yang, Q., et al. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

[Park and Chu, 2009] Park, S.-T. and Chu, W. (2009). Pairwise preference regression for cold-start recommendation. In *Proceedings of the third ACM conference on Recommender systems*, pages 21–28. ACM.

[Pasquier et al., 1999] Pasquier, N., Bastide, Y., Taouil, R., and Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. In *International Conference on Database Theory*, pages 398–416. Springer.

[Patcha and Park, 2007] Patcha, A. and Park, J.-M. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks*, 51(12):3448–3470.

[Pedregosa et al., 2011] 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 et al., 2000] Pei, J., Han, J., Mao, R., et al. (2000). Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, volume 4, pages 21–30.

[Pei et al., 2004] Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., and Hsu, M.-C. (2004). Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on knowledge and data engineering*, 16(11):1424–1440.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. (2014). Glove: global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[Pitkow and Pirolli, 1999] Pitkow, J. and Pirolli, P. (1999). Mininglongestrepeatin g subsequencestopredict worldwidewebsurfing. In *Proc. USENIX Symp. On Internet Technologies and Systems*, page 1.

[Poornalatha and Raghavendra, 2011] Poornalatha, G. and Raghavendra, P. S. (2011). Web user session clustering using modified K-means algorithm. In *International Conference on Advances in Computing and Communications*, pages 243–252. Springer.

[Rehurek and Sojka, 2010] Rehurek, R. and Sojka, P. (2010). Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*.

[Rousseau and Vazirgiannis, 2013] Rousseau, F. and Vazirgiannis, M. (2013). Graph-of-word and TW-IDF: new approach to ad hoc IR. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 59–68. ACM.

[Rousseeuw, 1987] Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65.

[Sak et al., 2014] Sak, H., Senior, A., and Beaufays, F. (2014). Long Short-Term Memory recurrent neural network architectures for large scale acoustic modeling. In *Interspeech*, pages 338–342.

[Salfner et al., 2010] Salfner, F., Lenk, M., and Malek, M. (2010). A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)*, 42(3):10.

[Salton and Buckley, 1988] Salton, G. and Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523.

[Scherbina and Kuznetsov, 2004] Scherbina, A. and Kuznetsov, S. (2004). Clustering of web sessions using Levenshtein metric. In *Industrial Conference on Data Mining*, pages 127–133. Springer.

[Schiaffino and Amandi, 2009] Schiaffino, S. and Amandi, A. (2009). Intelligent user profiling. In *Artificial Intelligence An International Perspective*, pages 193–216. Springer.

[Singhal et al., 1999] Singhal, A., Choi, J., Hindle, D., Lewis, D. D., and Pereira, F. (1999). AT&T at TREC-7. In *7th Conference on Text Retrieval*, pages 239–252.

[Sisodia et al., 2017] Sisodia, D. S., Verma, S., and Vyas, O. P. (2017). Augmented intuitive dissimilarity metric for clustering of web user sessions. *Journal of Information Science*, 43(4):480–491.

[Sundermeyer et al., 2012] Sundermeyer, M., Schlüter, R., and Ney, H. (2012). LSTM neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[Tafforeau et al., 2015] Tafforeau, J., Artières, T., Favre, B., and Bechet, F. (2015). Adapting lexical representation and OOV handling from written to spoken language with word embedding. In *Interspeech 2015*.

[Tan et al., 2016] Tan, Y. K., Xu, X., and Liu, Y. (2016). Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pages 17–22. ACM.

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-RMSPROP: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

[Von Luxburg, 2007] Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416.

[Wang et al., 2016] Wang, G., Zhang, X., Tang, S., Zheng, H., and Zhao, B. Y. (2016). Unsupervised clickstream clustering for user behavior analysis. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 225–236. ACM.

[Wang and Han, 2004] Wang, J. and Han, J. (2004). BIDE: Efficient mining of frequent closed sequences. In *Proceedings. 20th International Conference on Data Engineering*, pages 79–90. IEEE.

[Wang and Zaïane, 2002] Wang, W. and Zaïane, O. R. (2002). Clustering web sessions by sequence alignment. In *Database and Expert Systems Applications, 2002. Proceedings. 13th International Workshop on*, pages 394–398. IEEE.

[Wiwie et al., 2015] Wiwie, C., Baumbach, J., and Röttger, R. (2015). Comparing the performance of biomedical clustering methods. *Nature methods*, 12(11):1033.

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

[Xiao et al., 2017] Xiao, S., Yan, J., Farajtabar, M., Song, L., Yang, X., and Zha, H. (2017). Joint modeling of event sequence and time series with attentional twin recurrent neural networks. *arXiv preprint arXiv:1703.08524*.

[Xifeng et al., 2003] Xifeng, Y., Jiawei, H., and Afshar, R. (2003). CloSpan: Mining Closed Sequential Patterns in Large Data Base. In *Proc. of the 3rd SIAM Int'l Conf. on Data Mining. San Franciso, USA*.

[Xing et al., 2008] Xing, Z., Pei, J., Dong, G., and Yu, P. S. (2008). Mining sequence classifiers for early prediction. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pages 644–655. SIAM.

[Xing et al., 2010] Xing, Z., Pei, J., and Keogh, E. (2010). A brief survey on sequence classification. *ACM SIGKDD Explorations Newsletter*, 12(1):40–48.

[Xu and Liu, 2010] Xu, J. and Liu, H. (2010). Web user clustering analysis based on kmeans algorithm. In *2010 International Conference on Information Networking and Automation (ICINA)*.

[Yu et al., 2018] Yu, A. W., Dohan, D., Luong, M.-T., Zhao, R., Chen, K., Norouzi, M., and Le, Q. V. (2018). QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension. *ICLR*.

[Zaki, 2000] Zaki, M. J. (2000). Scalable algorithms for association mining. *IEEE transactions on knowledge and data engineering*, 12(3):372–390.

[Zaremba et al., 2014] Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

[Zhang et al., 2016] Zhang, K., Xu, J., Min, M. R., Jiang, G., Pelechrinis, K., and Zhang, H. (2016). Automated IT system failure prediction: A deep learning approach. In *International Conference on Big Data*, pages 1291–1300. IEEE.

[Zhang et al., 2014] Zhang, Y., Dai, H., Xu, C., Feng, J., Wang, T., Bian, J., Wang, B., and Liu, T.-Y. (2014). Sequential click prediction for sponsored search with recurrent neural networks. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1369–1375.

[Zhao et al., 2017] Zhao, Y., Chu, S., Zhou, Y., and Tu, K. (2017). Sequence prediction using neural network classiers. In *International Conference on Grammatical Inference*, pages 164–169.

[Zhou et al., 2016] Zhou, C., Cule, B., and Goethals, B. (2016). Pattern based sequence classification. *IEEE Transactions on Knowledge and Data Engineering*, 28(5):1285–1298.

université PARIS-SACLAY
ÉCOLE DOCTORALE
INTERFACES
Approches Interdisciplinaires : Fondements, Applications et Innovation

Titre: Analyse des Séquences d'Usage d'Applications Médicales

Mots clés: exploration de motifs fréquents; représentations pour l'apprentissage; représentations d'action; réseaux de neurones récurrents LSTM; clustering.

Résumé:
Les radiologues utilisent au quotidien des solutions d'imagerie médicale pour le diagnostic. L'amélioration de l'expérience utilisateur est un axe majeur de l'effort continu visant à améliorer la qualité globale et l'ergonomie des produits logiciels. Les applications de monitoring permettent en particulier d'enregistrer les actions successives effectuées par les utilisateurs dans l'interface du logiciel. Ces interactions peuvent être représentées sous forme de séquences d'actions. Sur la base de ces données, ce travail traite de deux sujets industriels: les pannes logicielles et l'ergonomie des logiciels. Ces deux thèmes impliquent d'une part la compréhension des modes d'utilisation, et d'autre part le développement d'outils de prédiction permettant soit d'anticiper les pannes, soit d'adapter dynamiquement l'interface logicielle en fonction des besoins des utilisateurs. Tout d'abord, nous visons à identifier les origines des crashes du logiciel qui sont essentielles afin de pouvoir les corriger. Pour ce faire, nous proposons d'utiliser un test binomial afin de trouver les patterns fréquents et nous comparons différents types de patterns afin de déterminer celui qui est le plus approprié pour représenter les signatures de crash. L'amélioration de l'expérience utilisateur par la personnalisation et l'adaptation des systèmes aux besoins spécifiques de l'utilisateur exige une très bonne connaissance de la façon dont les utilisateurs utilisent le logiciel. Afin de mettre en évidence les tendances d'utilisation, nous proposons de regrouper les sessions similaires. Nous comparons trois types de représentation de session dans différents algorithmes de clustering. La deuxième contribution de cette thèse concerne le suivi dynamique de l'utilisation du logiciel. Nous proposons deux méthodes – basées sur des représentations différentes des actions d'entrée – pour répondre à deux problématiques industrielles distinctes : la prédiction de la prochaine action et la détection du risque de crash logiciel. Les deux méthodologies tirent parti de la structure récurrente des réseaux LSTM pour capturer les dépendances entre nos données séquentielles ainsi que leur capacité à traiter potentiellement différents types de représentations d'entrée pour les mêmes données.

Title: Pattern Recognition in the Usage Sequences of Medical Apps

Keywords: frequent pattern mining; representation learning; action embeddings; LSTM recurrent neural networks; clustering.

Abstract:
Radiologists use medical imaging solutions on a daily basis for diagnosis. Improving user experience is a major line of the continuous effort to enhance the global quality and usability of software products. Monitoring applications enable to record the evolution of various software and system parameters during their use and in particular the successive actions performed by the users in the software interface. These interactions may be represented as sequences of actions. Based on this data, this work deals with two industrial topics: software crashes and software usability. Both topics imply on one hand understanding the patterns of use, and on the other developing prediction tools either to anticipate crashes or to dynamically adapt software interface according to users' needs. First, we aim at identifying crash root causes. It is essential in order to fix the original defects. For this purpose, we propose to use a binomial test to determine the frequent patterns and compare several types of patterns to determine the most appropriate to represent crash signatures. The improvement of software usability through customization and adaptation of systems to each user's specific needs requires a very good knowledge of how users interact with the software. In order to highlight the trends of use, we propose to group similar sessions into clusters. We compare 3 session representations as inputs of different clustering algorithms. The second contribution of our thesis concerns the dynamical monitoring of software use. We propose two methods – based on different representations of input actions – to address two distinct industrial issues: next action prediction and software crash risk detection. Both methodologies take advantage of the recurrent structure of LSTM neural networks to capture dependencies among our sequential data as well as their capacity to potentially handle different types of input representations for the same data.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France