



Université de Haute-Alsace

École Doctorale Mathématiques, Sciences de l'Information et de l'Ingénieur
(MSII, ED 269)
Institut de Recherche en Informatique, Mathématiques, Automatique et Signal
(IRIMAS, EA 7499)

Machine Learning Algorithms for Behavior Prediction in Cloud Computing Architectures

Algorithmes d'apprentissage automatique pour la prédiction de comportement dans les architectures de Cloud Computing

Thèse

préparée par

Matías Ezequiel CALLARA

présentée pour obtenir le grade de
Docteur de l'Université de Haute-Alsace
Discipline : Génie Informatique

soutenue publiquement le 23 Septembre 2019 devant le jury composé de :

Dr. HDR, Armelle Brun, Université de Lorraine (rapporteure)
Pr. Abderrafiaa Koukam, Université de Technologie de Belfort-Montbéliard (rapporteur)
Pr. Michel Hassenforder, Université de Haute-Alsace
Dr. Christophe Corne, Systancia (invité)
Pr. Patrice Wira, Université de Haute-Alsace (directeur de thèse)

Dedicada a mi Alala.

Résumé

Les solutions de cloud computing fournissent des applications, des capacités de stockage et des ressources de calcul via un réseau. Les multiples avantages de ces technologies les ont rendues omniprésentes. Aujourd'hui, la virtualisation permet aux utilisateurs d'accéder à des applications gourmandes en ressources à partir d'une grande variété de périphériques tels que smartphones, tablettes, ordinateurs portables, ordinateurs de bureau, car ils n'ont pas besoin d'être installés ou exécutés localement. Cependant, des temps de chargement de plusieurs secondes de ces services peuvent compromettre l'expérience des utilisateurs. L'idée est d'analyser le comportement des utilisateurs pour prévoir l'utilisation des applications et réduire considérablement leur temps de lancement en les chargeant à l'avance.

Un logiciel de virtualisation collecte et centralise les actions des utilisateurs dans une base de données. En particulier, les données contiennent un identifiant unique pour chaque application et des horodatages indiquant quand elles ont été utilisées. Dans ce travail, nous avons décrit comment utiliser les horodatages et les variables qualitatives pour le profilage des utilisateurs en générant des plongements basés sur différentes caractéristiques comportementales ; par exemple, la fréquence relative d'utilisation des applications. Nous avons ensuite appliqué et comparé les résultats de différentes techniques de réduction de dimensionnalité (par exemple, ACP, Isomaps, LLE et t-SNE) pour créer des visualisations éloquentes des plongements. Nous avons montré comment identifier des utilisateurs similaires et créer des groupes en mesurant la distance qui les sépare et en appliquant différentes techniques de classification (telles que les k plus proches voisins et le partitionnement spectral). Toutes ces techniques nous ont permis de créer des interfaces graphiques qui améliorent et simplifient le travail de l'administrateur système.

Nous avons également proposé un pipeline de modélisation et de prédiction du comportement des utilisateurs qui réduit le temps de lancement des applications. Le pipeline proposé est composé d'une chaîne de cinq modules. Le premier module identifie les fréquences principales dans le comportement de l'utilisateur en analysant les pics du périodogramme et sélectionne la fréquence la plus pertinente à prendre en compte dans le modèle de l'utilisateur. Cette dernière est utilisée dans le second module pour construi-

ire un modèle probabiliste en estimant la fonction de densité de probabilité de son activité. Dans ce contexte, nous avons notamment décrit les avantages des approches bayésienne et fréquentiste, et en particulier des méthodes paramétriques et non paramétriques, telles que l'estimation de la densité par noyau. Le troisième module contrôle la qualité du modèle de l'utilisateur au fil du temps, à mesure que de nouvelles données arrivent. Le quatrième module génère une prédiction sous la forme d'intervalles de temps dans lesquels nous nous attendons à ce que l'utilisateur lance une application. Enfin, le cinquième module optimise le temps d'application de lancement pour un groupe d'utilisateurs, définit au préalable par l'administrateur système.

Notre approche présente plusieurs avantages. Il convient parfaitement aux clients disposant de peu de données (par exemple, de nouveaux clients), car il permet d'introduire une connaissance du domaine ou des conditions définies. Pour avoir de bonnes performances, il n'est pas nécessaire que les données soient partagées par les clients (en conformité aux normes en matière de politiques de confidentialité des données). Il permet aussi de choisir entre l'approche d'estimation par lots ou en ligne et le temps de calcul augmente de façon linéaire avec le nombre d'utilisateurs. L'architecture modulaire de notre solution la rend robuste, facile à débuguer et permet une amélioration progressive du système. La solution reste générale et peut également être appliquée à la prédiction d'autres types d'événements (l'heure et la date des événements étant la seule condition nécessaire au bon fonctionnement de notre pipeline).

Après avoir testé les performances du pipeline avec des données réelles, nous avons pu, avec une initialisation par défaut des paramètres, diviser par deux le temps de lancement moyen d'un groupe de 13 s à $6,5$ s avec au maximum 20% de consommation de ressources additionnelles, et si nécessaire, même atteindre $2,6$ s avec au maximum 40% de consommation de ressources additionnelles. Le pipeline est également accompagné d'une interface utilisateur graphique qui permet à l'administrateur système de régler l'ajustement entre la réduction souhaitée du temps de chargement et le coût de calcul supplémentaire généré par les applications préchargées.

Enfin, la solution proposée a été mise en œuvre et entièrement intégrée dans un logiciel de virtualisation d'applications déployé auprès de clients du monde entier et testée à plusieurs reprises dans des conditions réelles pour valider les performances.

Mots clés

1. Prédiction de comportement 2. Modélisation utilisateur 3. Analyse du comportement des utilisateurs 4. Cloud computing 5. Estimation par noyau 6. Technique d'estimation adaptative 7. Extraction de motifs temporels 8. Prédiction des lancements d'applications 9. Performance de prédiction 10. Ingénierie des données

Abstract

Cloud computing solutions provide applications, storage capabilities, and computational resources through a network. The multiple advantages of these technologies have made them ubiquitous. Today, virtualization lets users access resource-intensive applications from a wide variety of devices like smartphones, tablets, laptops, desktop computers since they do not need to be installed or run locally. However, loading times of several seconds of these services can jeopardize the users' experience. The idea is to analyze the user behavior to predict the utilization of applications and significantly reduce its launching time by loading them in advance.

A virtualization software collects and centralizes the users' actions in a database. In particular, the data contains a unique identifier for each application and timestamps that indicate its launching time. In this work, we described how to use the timestamps and categorical variables to profile the users by generating embeddings based on different behavioral characteristics; for instance, the relative frequency of the applications' utilization. We applied and compared the results of different dimensionality reduction techniques (e.g., PCA, Isomaps, LLE, and t-SNE) to create compelling visualizations of the embeddings. We showed how to identify similar users and create groups by measuring the distance between them and applying different clustering techniques (such as k-nearest neighbors and spectral clustering). All these techniques allowed us to create better graphical interfaces that improved and simplified the work of the systems administrator.

We also proposed a modeling and user behavior prediction pipeline that reduces the launching time of applications. The proposed pipeline is composed of a chain of five modules. The first module identifies the size of the periodic patterns in the user behavior by analyzing the peaks in the periodogram and selects the most relevant frequency to be considered in the user model. The second module constructs a probabilistic user model by estimating the probability density function of the user activity. We described the advantages of applying Bayesian and frequentist approaches, including both parametric and non-parametric methods, such as Kernel Density Estimation (KDE). The third module controls the quality of the user model over time as new data arrives. The fourth module generates a prediction in the form of time intervals in which we expect the user to launch an application. Fi-

nally, the fifth module optimizes the launching application time for a group of users that the administrator of the system can define.

Our approach shows several advantages. It is well suited for customers with scarce data (e.g., new customers) since it allows to introduce domain knowledge or set conditions. It does not require the data to leave the customers to perform well (complying with high standards on data privacy policies). It allows to change the estimation approach to switch from batch to on-line training and scale linearly as the number of user increases. The modular architecture of our solution makes it robust, easy to debug, and allows for progressive improvement of the system. The solution remains general, and it can also be applied to the prediction of other types of events (with the time and date of the events being the only requirement for our pipeline to work).

After testing the performance of the pipeline with real-world data, we found that under a default initialization of the parameters, we can half the average launching time of a group from 13s to 6.5s with at most 20The pipeline was also accompanied by a graphical user interface that allowed the system administrator to adjust the trade-off between the desired loading time reduction and the extra computational cost generated by the pre-loaded applications.

Finally, the proposed solution was implemented and fully integrated into an application virtualization software, deployed to customers around the world, and repeatedly tested under real conditions to validate the performance.

Keywords

1.Behavior prediction 2.User modeling 3.User Behavior Analytics (UBA) 4.Cloud computing 5.Kernel Density Estimation (KDE) 6.Adaptive estimation technique 7.Temporal patterns extraction 8.Prediction of application launches 9.Prediction performance 10.Data engineering

Contents

Résumé i

Abstract iii

Introduction 1

1 Cloud computing and virtualization 7
- 1.1 Cloud computing . . . 7
  - 1.1.1 Cloud deployments models . . . 8
  - 1.1.2 Cloud service models . . . 8
- 1.2 Virtualization . . . 9
  - 1.2.1 Desktop and application virtualization . . . 9
  - 1.2.2 Server-Based Computing (SBC) . . . 10
  - 1.2.3 Virtual Desktop Interface (VDI) . . . 11
- 1.3 Cloud computing and energy management . . . 11
- 1.4 Advantages and drawbacks of cloud computing and virtualization . . . 12
- 1.5 Systancia's solution: AppliDis . . . 14
- 1.6 AppliDis' architecture . . . 14
  - 1.6.1 Server components . . . 14
  - 1.6.2 Client component . . . 16
  - 1.6.3 AppliDis' load balancing . . . 18
- 1.7 Login and application launching management . . . 19
  - 1.7.1 Phase 1: Login . . . 19
  - 1.7.2 Phase 2: Launching an application . . . 19
  - 1.7.3 Phase 3: Utilization of the application . . . 20
- 1.8 AppliDis Booster . . . 20
  - 1.8.1 Profile virtualization . . . 22
  - 1.8.2 Pre-loading . . . 22
  - 1.8.3 Post-loading . . . 22
  - 1.8.4 AppliDis Booster performance measures and statistics display . . . 23
- 1.9 AppliDis databases . . . 23

1.10 Summary 25

2 Approaches for User Modeling (UM) and Behavior Prediction (BP) 27
2.1 Introduction 27
2.2 Data for UM and BP problems 28
2.3 Relevant approaches to UM and BP 29
2.3.1 Time series analysis and signal processing 29
2.3.2 Expert systems and Bayesian networks 32
2.3.3 Association rule mining 32
2.3.4 Sequential pattern mining 33
2.3.5 Probabilistic models 35
2.3.6 Deep learning models 36
2.3.7 Other models 38
2.4 Applications of UM in the industry 39
2.4.1 Faster application launching 39
2.4.2 Personal digital assistants 40
2.4.3 Recommendation systems 41
2.4.4 Online social networks and media 42
2.5 Limits of predictability 44
2.6 Conclusion 45

3 Exploratory Data Analysis (EDA) and user profiling 47
3.1 Database exploration 47
3.1.1 Shortlisted tables 47
3.1.2 Proposed new table `UtilisateurLogins` 48
3.1.3 Enhancement of the existing table `useruseappli` . . . 49
3.1.4 Other Booster tables 49
3.1.5 Improving the data quality 50
3.2 EDA of the user activity 50
3.2.1 Session logins 52
3.2.2 Application launches 54
3.2.3 Database visualization tool 56
3.3 Feature engineering 56
3.3.1 Categorical variables 57
3.3.2 Timestamps 58
3.3.3 Continuous variables 58
3.3.4 Relative features 59
3.4 Entropy of the behavior 59
3.4.1 Entropy 60
3.4.2 Mutual information 62
3.4.3 Behavior entropy analysis 62
3.5 Learning representations and user profiling 64
3.5.1 Eigenbehaviors 64

3.5.2 User profiling . . . 66
3.5.3 Clustering . . . 67
3.6 Discussion . . . 71
3.7 Conclusion . . . 71

4 User modeling and behavior prediction in cloud computing architectures 73
4.1 Introduction . . . 73
4.2 User modeling and statistical decision theory . . . 74
4.2.1 Statistical experiment . . . 74
4.2.2 Objective . . . 75
4.2.3 Decision rule (estimator) . . . 75
4.2.4 Loss function . . . 75
4.2.5 Risk function . . . 75
4.2.6 Evaluating the estimator . . . 75
4.3 Behavior prediction . . . 77
4.3.1 Prediction: Linking probabilities with intervals . . . 78
4.3.2 Prediction: Use modes . . . 81
4.3.3 Applying the proposed pipeline: Controlling the average application start-up time . . . 81
4.3.4 The behavior periodicity . . . 84
4.3.5 Analysis of periodicity (defining T) . . . 85
4.4 Modeling the application launches . . . 87
4.4.1 Modeling with a discrete distribution: Categorical distribution . . . 88
4.4.2 Modeling applications with a continuous distribution: Kernel Density Estimation (KDE) . . . 91
4.4.3 Model selection - defining h . . . 93
4.4.4 Incorporating a PID controller to improve results . . . 94
4.5 Results . . . 95
4.5.1 User model evaluation . . . 95
4.5.2 Summarizing the user model . . . 99
4.6 Implementation in production . . . 100
4.6.1 C++ implementation . . . 101
4.6.2 Controlling the algorithm from AppliDis: Booster GUI 101
4.6.3 C++ simulator implementation . . . 103
4.7 Future work . . . 106

Conclusion 107

List of acronyms 111

Bibliography 126

Introduction

Human behavior is predictable. Although at first, this assertion could sound somewhat striking, once we think about what implies it becomes almost trivial. As human beings, we are subject to constraints, and we practice certain activities with some regularity. The patterns or structures that are present to some degree in our behavior make us (in some measure) predictable. However, the natural next question is: How can we take advantage of this structure? Along this thesis, we will work not only on trying to answer this question but on how we can do it by learning from data. This last activity is at the center of different fields like statistics, data mining, statistical learning, pattern recognition, and Machine Learning (ML). We will use the term *data science* to refer to this particular interdisciplinary activity that intersects and gather all of them.

While a data science project can focus purely on research, our objective is to develop a data-driven software product. However, different companies have also faced the challenge of combining pure research and software development. To facilitate this endeavor, in 1996, a group of companies created the Cross-Industry Standard Process for Data Mining (CRISP-DM) [36]. This standard can be seen as a *process model* of a data science product life cycle, or as a *methodology* that describes the most common phases in a data science project.

Fig. 1 shows the phases of the CRISP-DM methodology and their relationship. In the diagram, the arrows that connect the phases represent the most common sequential order between them. The big external arrows show that this process is iterative. E.g., after the release of the first version of the analytic product, the main insights and takeaways can be used to improve the product. Translating these insights into a new improved version of the product can generate a new project that will itself follow the CRISP-DM process. To gain a better understanding of the whole model, we describe one by one its main phases.

Business understanding

The first phase focuses on understanding the problem from the business point of view and translating it into an ML problem. To do so, we need to

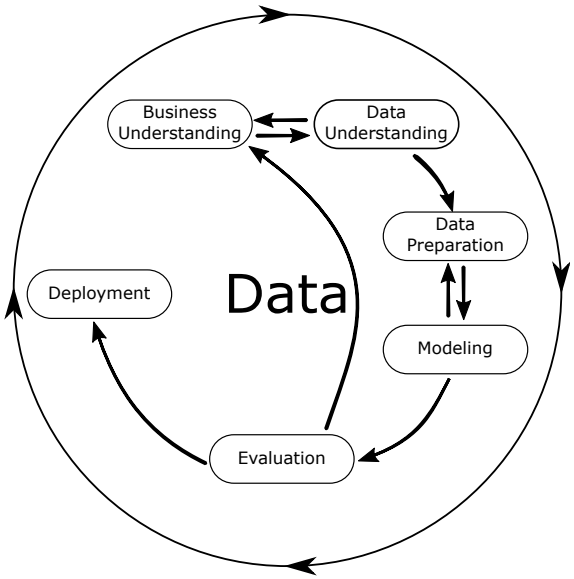


Figure 1 – CRISP-DM methodology (based on figure in [36])

state the objective in business terms and identify the success criteria. This will require understanding the business terminology, the associated risks of the decisions that solutions will support or generate, assumptions, trade-offs, existing solutions, and possible benchmarks.

Data understanding

Once the problem is framed, we need to gather and inspect the data. Since the data documentation can be poor or non-existent, getting in contact with data engineers and business experts can help to generate a good description of the data. Once we have a well-documented dataset, we can perform Exploratory Data Analysis (EDA) [129] by sampling data to construct tables and different kinds of visualizations. This exploration will allow to generate insights and hypothesis while facilitating quality assessment of the data (e.g., detecting missing values and outliers).

Data preparation

Data preparation encompasses all the tasks that need to be performed to convert the raw data into an analytical dataset that will be feed to the model. Some of these tasks are data cleansing, which consists in removing outliers and filling missing values, feature selection, and feature engineering, which includes decomposing features (e.g., in categorical, timestamps, numeric), transforming or aggregating features.

Modeling

During the modeling phase, candidate algorithms are selected, trained, and fine-tuned by adjusting their hyperparameters. Since, at this point, we may need to modify features from the analytical dataset, this phase can lead back to the data preparation phase.

Evaluation

The validated trained models need still to be evaluated from the business point of view. This is to verify that they achieve the business objective that has been set during the business understanding phase.

Deployment

The selected models that pass the evaluation phase will then be delivered to the final customer, who will use them regularly (applying the model on new data). This phase will usually require creating user interfaces that make the models usable by the final customer. It can also require code refactoring to reduce the complexity of the code, making maintenance easier, or improving the execution performance. Finally, changes in the phenomena that produce the data (or even in the software or databases that interact with it) can make the performance of the model decrease over time. This is why it is crucial to create monitoring functions and maintain the models (for instance, by retraining the models with a certain periodicity [135]).

Since the different version of the data-driven application that result from the work of this thesis has been developed following this methodology, the chapters of this manuscript follows the order of the CRISP-DM process model. Chapter 1 summarizes most of the business understanding, chapter 2 provides the theoretical context of the techniques and models applied, chapter 3 details the data understanding, data preparation and the first set of models, chapter 4 develops the ideas and results of the modeling, evaluation and deployment phases.

Objective of this work

This research work is the product of a public-private partnership research between the Systancia company and the IRIMAS (Institut de Recherche en Informatique, Mathématiques, Automatique et Signal) laboratory from the Université de Haute-Alsace in the context of a CIFRE thesis. In French, CIFRE stands for Conventions Industrielles de Formation par la Recherche (Industrial Agreements for Training through Research). The CIFRE subsidizes French companies that employ doctoral students and directly involve

them in research partnerships with state laboratories.

This thesis proposes methods to automatically learn behavior models of the users of a cloud computing service and predict when they will initiate sessions or launch applications. This allows to reduce the loading times while keeping the energy cost associated with these activities as reduced as possible. This improves the user experience when accessing remote applications and services. The results of this research have been implemented in Systancia's main virtualization product AppliDis Fusion that is deployed to customers with thousands of users around the world.

Publications and authors' contribution

The research activities in this thesis were presented in the following publications:

Posters presented in scientific meetings

- M. Callara, "Identifying the periodicity of mixed periodic patterns for user behavior prediction" in Machine Learning Summer School, Arequipa, Peru, 2-13 August 2016.
- M. Callara and P. Wira, "Machine learning algorithms for behavior prediction and resources optimization in cloud computing architectures," in Journée Doctorale Sciences Exactes à l'Université de Haute-Alsace, Mulhouse, France, 2 June 2017.

Conference papers published in national conferences with review and proceedings

- M. Callara and P. Wira, "Machine learning pour l'analyse de comportements et la classification d'utilisateurs," in Congrès National de la Recherche des IUT (CNRIUT'2017), Auxerre, France, 4-5 May 2017. [26]

Conference papers published in international conferences with review and proceedings

- M. Callara and P. Wira, "User behavior analysis with machine learning techniques in cloud computing architectures," in International Conference on Applied Smart Systems (ICASS 2018), Medea, Algeria, 24-25 November 2018. [27]
- M. Callara and P. Wira, "A probabilistic learning approach for predicting application launches in cloud computing architectures," In IEEE/SICE International Symposium on System Integration (SII 2019), Paris, France, 14-16 January 2019. [29]

- M. Callara and P. Wira, "A data-driven approach for user behavior prediction to boost productivity and sustainability of data centers and cloud-supported working environments," In European International Conference on Transforming Urban Systems (EICTUS 2019), Strasbourg, France, 26-28 June 2019. [28]

Article submitted to international journals

- M. Callara and P. Wira, "Machine learning paradigms for user behavior modeling: An overview." Journal of Applied Computer Science Methods, submitted, 2019. [30]

Organization of this thesis

This thesis is organized as follows.

Chapter 1 describes the state of the art of cloud computing and virtualization technologies. Indeed, the main cloud computing models are listed, and the principle of desktop virtualization and application virtualization are detailed with a focus on server-based computing and virtual desktop interfaces. Their advantages and drawbacks are investigated, and the link between cloud computing and energy management is also discussed. The main components of Systancia's AppliDis solution are fully detailed: The global architecture with the server and client software components' functionalities, the integrated load balancing system, the login and application launching management process, the Booster module and the involved databases.

Chapter 2 reviews the main approaches to User Modeling (UM) and Behavior Prediction (BP) and, in particular, the methods suited for the type of data related to the problem described in this work. In this chapter, we can also find a discussion about the use and implementation of these methods in the industry and their limitations. Limits of the user behavior's predictability are briefly tackled.

Chapter 3 describes the necessary preliminary work that enabled the development of the thesis and presents our proposed approach to user profiling. This means that the enhancements of existing tables that are necessary to allow the implementation of classification and prediction algorithms are highlighted. Features and entropy of the behavior are introduced. Furthermore, learning representations and user profiling strategies are fully developed. Results in user profiling and behavior clustering are presented and discussed.

Chapter 4 presents our proposed approach to user behavior prediction and describes its complete implementation in a specific cloud computing architecture. After presenting the user modeling and statistical decision theory that are used, a user behavior prediction strategy is proposed. The behavior periodicity is extracted by a self-adaptive algorithm. Then, it is used

and applied for modeling application launches through a distribution defined by a probability density function. Modeling is achieved by a discrete, i.e., categorical, distribution, or by a continuous distribution that relies on the kernel density estimator. Model selection techniques are proposed; they rely on the rule-of-thumb method and the cross-validation method. A PID controller has also been inserted in the prediction process to improve the results by compensating for the prediction biases of the probability estimators. Implementation in production and results obtained in a cloud computing architecture under real conditions are presented, and the performances are discussed. The launching of applications by users and even group of users has been automatized. Additionally, tools for the cloud computing architecture administrator have been proposed for monitoring and tuning purposes.

Finally, the Conclusion provides an overall summary of the thesis, and the main contributions are discussed. Some recommendations and perspectives are also provided.

An Appendix section recapitulates the definitions of the acronyms that have been used along this document.

Chapter 1

Cloud computing and virtualization

In the last 20 years, general and professional users have embraced a new paradigm related to the existence, operation, and management of the resources that make up the Information Technology (IT) infrastructures they use every day. The advances in communication technologies have made cloud computing ubiquitous, and today it is used in a tremendous variety of contexts, for instance, to store email securely or to run complex applications, process images, play games, or watch movies.

1.1 Cloud computing

According to [95], cloud computing [73, 140, 67] is a model for enabling "on-demand network access to a shared pool of configurable computing resources". The definition is based on five characteristics:

- On-demand self-service. The user can provision herself the resources as she requires them.
- Broad network access. The resources are accessed over the network through a wide variety of clients.
- Resource pooling. Resources are assigned dynamically according to different criteria established by the service provider.
- Rapid elasticity. From the user point of view, the resources seem to be unlimited, providing the freedom of rapidly augmenting or decreasing the amount of consumed resources at any moment in time.
- Measured service. The consumption of the resources can be tracked at different levels of granularity, providing expenditure control to the user.

Cloud Type	provisioned for	owned, managed and operated by	installed
Private cloud	single organization	organization, third party or combination	on or off premises
Community cloud	community	subset of the community, third party or combination	on or off premises
Public cloud	general public	cloud provider	cloud provider premises

Table 1.1 – Types of cloud deployments.

1.1.1 Cloud deployments models

According to the deployment model we can classify the types of clouds as public, community (a group of organizations with shared concerns) and private. A detailed comparison is shown in Table 1.1 [95].

By sharing data and applications between the different described types of clouds, we can create a fourth type of cloud known as a hybrid cloud.

1.1.2 Cloud service models

Following the Service-Oriented Architecture style of software design, cloud service providers can offer independent components of functionalities that address specific concerns. These components are known as services. This paradigm is based on the idea of seeing the services as a way of approaching a particular organizational need and getting a desired outcome. The service should be self-contained and let its users consume it without any knowledge of the internal mechanisms that make it work. The three standard models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [95, 73, 140].

Software as a service (SaaS)

One of the most common ways of using software involves paying a fixed price for a perpetual license that provides use and redistribution rights that then allows the user to install the software in her infrastructure to finally work with it. In the SaaS service model [106], the perpetual license is replaced by another type of license based on a different pricing model (e.g., a regular fee or the freemium model) and the cloud provider delivers the required infrastructure and installation. The user only has control over the software settings.

Platform as a service (PaaS)

In this service model, the cloud provider delivers a cloud platform (this is a set of defined networks, servers, storage, and computing resources). The user can then use it to deploy her software [32].

Infrastructure as a service (IaaS)

The cloud provider provisions processing, storage, and networks. In this case, the user can set the operating systems, storage, deploy applications and select networking components, and specialized hardware.

Major providers of public cloud services

Most of the cloud users today interact directly or indirectly with public cloud services. Some of the major providers today are Amazon, Google, Microsoft, and IBM. Amazon Web Services (AWS) started offering public cloud services on March 19, 2006, with Simple Storage Service (S3). During that year, they also added products like Amazon Simple Queue Service (SQS) and Amazon Elastic Compute Cloud (EC2). Google Cloud Platform started on April 7, 2008, with the Google App Engine, a PaaS product that lets the user run her applications on the Google platform. On February 1, 2010, Microsoft joined the cloud business launching Windows Azure (lately renamed Microsoft Azure on March 25, 2014). Finally, on June 4, 2013, IBM acquires SoftLayer creating IBM Cloud, helping them to gain a better position in the market. All of them have progressively incremented the number of services, and today, we can find equivalent products between them in diverse categories as IaaS, PaaS, ML, virtual reality, developer tools, and Internet of Things (IoT).

1.2 Virtualization

One of the key concepts of the cloud computing paradigm is virtualization [80]. This term is used with two different meanings, one related to the idea of abstraction and the other related to the idea of encapsulation. In hardware virtualization, a resource is decoupled (or abstracted) from the physical hardware by generating a logical view. E.g., one physical server can be used to generate several "virtual" servers, each one with their independent configuration (which is known as server virtualization). The same can be done with a hard drive (storage virtualization) or other hardware. On the other hand, there is a kind of virtualization related to the concept of encapsulation, as in the case where we separate an application from the underlying operating system (application virtualization) or when we detach a desktop environment from the physical client (desktop virtualization).

1.2.1 Desktop and application virtualization

Both desktop and application virtualization relies on the client-server model. In which we can identify two components, one that provides a resource (the server) and the other that requests it (the client). The communication between them will be established on a network.

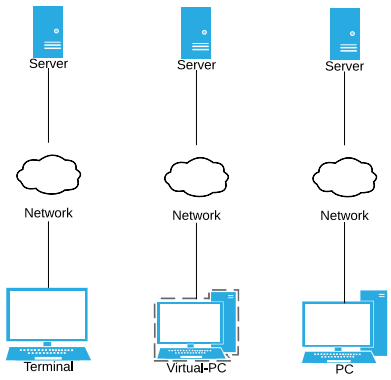


Figure 1.1 – Computer network diagrams showing instances of the server-based computing model.

There are two main ways of providing desktop (and in term application) virtualization to the clients, which are the Server-Based Computing (SBC) and the Virtual Desktop Interface (VDI).

1.2.2 Server-Based Computing (SBC)

In the SBC model [101], the server can host multiple client sessions in the operating system. When a client requests a session, the server can initiate it and present a desktop interface to the client, as if it were running on its side.

One example of a software that uses the SBC model is Microsoft's Remote Desktop Services (RDS), formerly Terminal Services, which allows the user to initiate/take control of a session on a remote computer (the server) or a virtual machine (in a server). The request of the session is sent via the client known as Remote Desktop Connection, which uses the Remote Desktop Protocol to establish the communication with the server. Once the communication is established, the server provides to the client the desktop interface, and the client transmits to the server the encrypted data of keyboard and mouse (as well as other resources on the client side like USB devices).

In this model, the applications that the user wants to run are implemented, controlled, supported, and functioned in the application server. The user interacts with a virtual representation in the terminal (client workstation). This representation is generated with data that is transmitted on the network. Since the application runs on the application server, it only needs to be compatible with the operating system of the server. In Figure 1.1, we see that different kinds of clients can be used to connect to a server.

1.2.3 Virtual Desktop Interface (VDI)

Instead of sessions, the Virtual Desktop Interface (VDI) [101] approach consists of offering the client a whole independent operating system instance for each user. This is the reason that makes it an excellent solution when we need to test an application in a specific environment (specific operative system, CPU, RAM) because we do not need to have a physical computer with this setting. On the other hand, it will usually consume more resources and licenses.

To summarize, while both SBC and VDI rely on the server-client model, SBC provides one session on the operating system of the server, while VDI can provide individual instances of different operating systems.

1.3 Cloud computing and energy management

Any cloud service (private, public, or hybrid) will require electrical energy to function. The energy efficiency of this system will be a quantity that relates the useful power (consumed in memory and computations) and the total power input, where the difference between these two values will be due to the energy loss in the form of heat. A significant part of this loss is usually produced at the power supply that converts AC voltage to low DC voltage. As a consequence, extra energy is consumed to keep its temperature under safety thresholds and to ensure the correct functioning of these systems. Since cloud computing servers usually share the same premises, we can apply energy management policies that would make no sense in the context of a single personal computer. Machine learning techniques have shown to be effective in improving these policies [50].

As explained in [85], energy management techniques mainly focus on two goals limiting the maximum power consumption to maintain the reliability and improving energy efficiency. Virtualization can produce benefits in both aspects because it allows quick and easy redistribution and accumulation of workloads improving the utilization rates of the servers [109].

Some energy management policies rely on an accurate model of the energy consumption of the servers. In [142], the authors proposed a model in which parameters can be automatically recovered from historical data of the utilization of the servers. Other policies consider static characteristics of the servers like CPU processing capabilities [104]. The virtualization component of the power consumption of the servers can also be integrated into the model of the server power consumption, as proposed by [143].

1.4 Advantages and drawbacks of cloud computing and virtualization

To let the reader get a better understanding of the applicability of these technologies, we list different aspects to consider when using cloud architectures.

Capabilities

Since the application servers can have a great amount of resources, we can assign to a client machine a big amount of computational and memory (disk and RAM) resources. This will let us run applications that would have been impossible to run in a standard desktop machine. The same principle applies to special hardware (e.g., GPUs, which is a requirement of deep learning models).

Economy of resources

When each employee has an individual machine with a significant amount of resources available, the utilization factor is usually low. An application server allows us to improve the utilization factor by allocating resources as they are needed (on-demand). Other source of economy is related to the use of big files or databases that can be shared among different users avoiding replications and saving disk space.

Collaboration and co-working

Some specific applications for the cloud, allow their users to work on the same file at the same time by finding ways to resolve in an online fashion the collisions that are produced by the modifications introduced by the users on the same file.

Compatibility

Desktop Virtualization allows the user to run applications on different operating systems avoiding compatibility issues. This is also useful in the context of software testing, where we require to set up a system with a particular software configuration to test the compatibility of a version or reproduce a bug.

Virtualization

Not all software or hardware can be virtualized. This is the case of applications that cannot be successfully isolated from their environment.

Maintainability

The system maintenance is much easier when all the resources are centralized. It requires less downtime and reduce the impact on the productivity of the users.

Licensing

The same principle of economy of hardware applies to software. Since usually the amount of people working at the same time with the same application is much lower than the total amount of people that use the application, we can satisfy the demand of the software with a smaller amount of licenses.

Data Security

All the data can be duplicated and distributed in servers in different physical locations to reduce the probability of permanently losing data. The frequency, at which we will save the data, can be set either at every modification that the user produces, generating what we know as Continuous Data Protection (CDP) or at fixed time intervals generating near-continuous backups.

On the other hand, if the data of several users is gathered in one physical unit, a failure will affect all of them.

Stronger measures on data security could also be put in place to avoid security attacks. This same strength could easily become a weakness since if once a person gains unauthorized access to the server, all the data can be compromised.

Data Privacy

As we will see along this work, much can be said about an organization if we exploit the traces that result from the interaction of it with a cloud system. Data privacy is a subject that is particularly relevant in the context of cloud computing since independently of the type of deployment (see section 1.1.1), organizations will be in general exposing data to the cloud service provider either directly or indirectly.

Learning curve

The use of tools on the cloud could require learning a big amount of material. The learning curve could become steep. AppliDis try to solve this by generating intuitive interfaces that simplify the task of working with these systems.

Complexity

The complexity of the system requires expert knowledge that could be difficult for enterprises to found and motivates companies to offer different components of the cloud as a service.

Latency

The high latency associated with physical distance can prevent the use of some applications.

1.5 Systancia's solution: AppliDis

Systancia is a French software company whose main product, AppliDis Fusion, provides a private cloud computing solution. The choice of the word "fusion" in the name of AppliDis is since the product offers a mix of SBC and VDI functionalities.

When an organization buys AppliDis Fusion, it will be installed by the administrator of the infrastructure, who is usually a member of the organization. Along the installation process, the administrator will have the responsibility of making different decisions that will impact the experience of the users. Some of these decisions will be related to the architecture of the system.

At any moment, the administrator can use the administration console to:

- make new applications available to the users by publishing them in AppliDis,
- control the access right of the users or group of users,
- change configuration options and
- find useful statistics that help to manage the system.

1.6 AppliDis' architecture

Since AppliDis use a client-server model as described in the section 1.2.1 to provide the SBC and VDI services. We will describe first the different main types of servers we can find in a general installation of the product and, finally, the four types of clients.

1.6.1 Server components

Management server

The management server is one of the fundamental components of AppliDis because it acts as a coordinator of all the other components of the system.

Its main functions are to:

- grant access to the administrator portal and the user portal,
- centralize the information collected by the application server,
- manage the Balancing Load and the statistics of the application servers,
- manage the published applications, users and access rights, and
- manage the replications of the SQL ODBC bases.

An AppliDis installation should always have a management server.

Application server

The application server is the one in charge of hosting the different users' sessions. Its main functions are:

- Manage the published applications.
- Inform the change of state to the management server when the application server is started by sending an XML Query.
- Inform the state of the application server (load, number of open sessions) to the Management Server.

While the basic installation of AppliDis requires only one Application server, it is possible to install several and create a silo. This is a set of application servers that share a common group of deployed applications. The main advantage is to avoid overloading a server by distributing the load. Silos also increment the reliability of the system because it increments the probability of finding a server where the application can be run, even if a server fails or need to be maintained.

Gateway server

The function of the Gateway Server is to enable authorized users that connect from an external network to communicate (through RDP) with the Application Server(s). There are two options: either the AppliDis client can indicate to the Gateway Server the reference of the Administration Server or the Gateway can query the Management Server to which Administration Server to connect.

Finally, it is worth noticing that one server can perform all these different roles. This is especially the case in small organizations that have a small basic infrastructure.

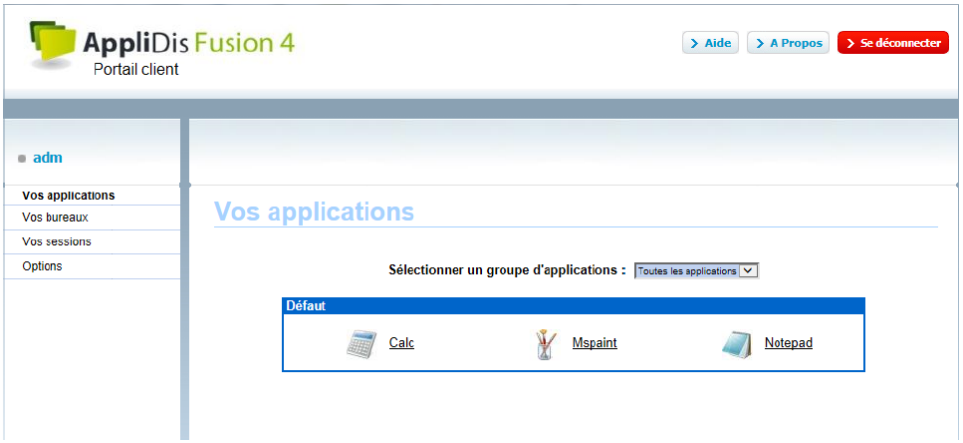


Figure 1.2 – Example of the Web Portal AppliDis client. On the center of the screen, icons of the available applications.

1.6.2 Client component

The user of AppliDis can choose between four different ways of starting an AppliDis session.

Web portal AppliDis client

The easiest way of accessing AppliDis is by using a web browser and installing a plugin (called "Client AppliDis Internet") that allows the user to access the Web Portal, where a list of her available applications (published applications) will appear, as shown in Figure 1.2. The applications will open in a seamless mode.

Desktop AppliDis client

Another way of using AppliDis consists in installing a Desktop client on the computer. This will usually run automatically and display the icons of the available applications on the desktop, as shown in Figure 1.3, and the user can open them as if they were installed on the computer. At least until version Fusion 4, logged-in users are not visible for the administrator in the Admin Console.

In Figure 1.3, we see a screen capture of the display of a logged-in AppliDis user. Even though it resembles a regular Windows desktop, the icons on the left side of the screen are not common Windows application icons but AppliDis Desktop client icons. They indicate which applications are available for that user on the application server. Finally, the green light in the taskbar on the right bottom corner indicates that the user is logged in, and the client has successfully connected to the server.

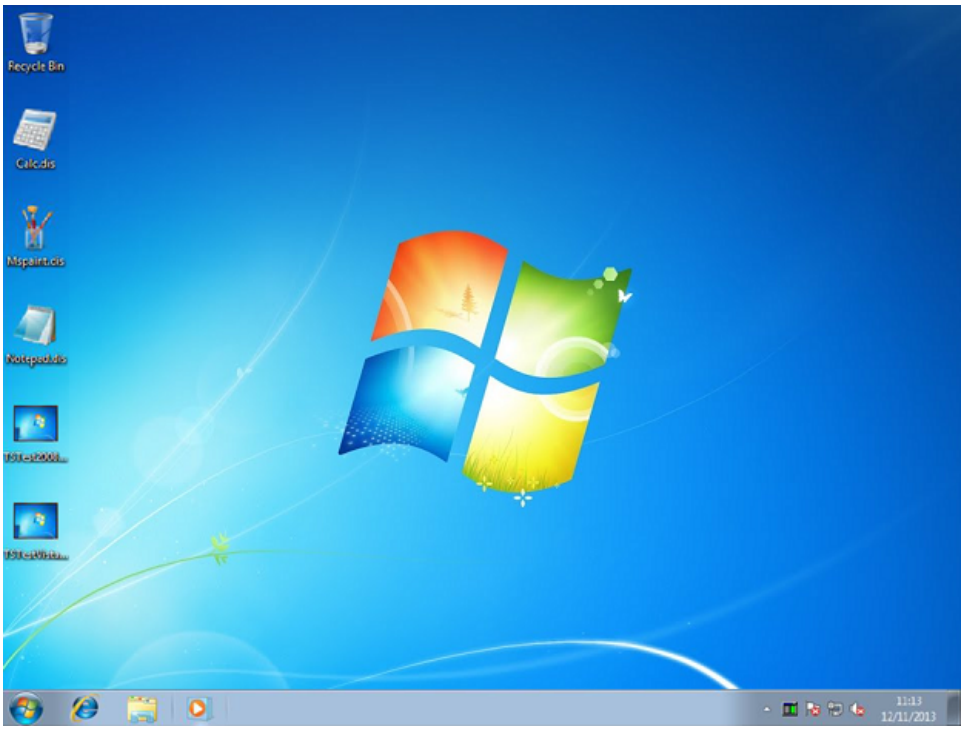


Figure 1.3 – Example of the AppliDis Desktop Client. On the left part of the screen, the icons of the available applications. In the task bar, the icon of the AppliDis Desktop showing a green light.

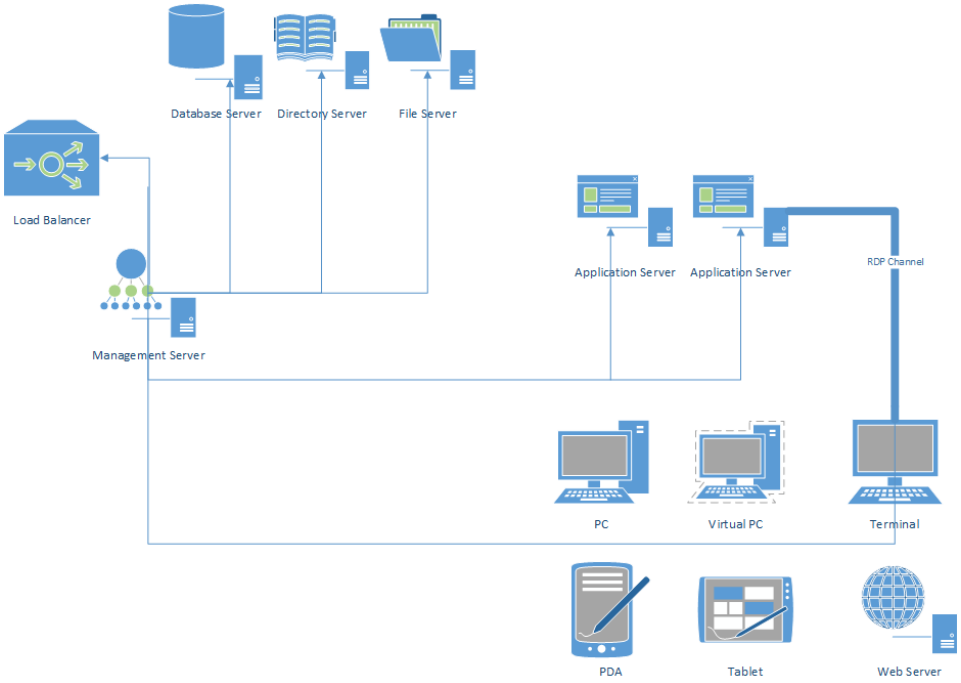


Figure 1.4 – Components of the AppliDis Server.

In Figure 1.4, we can see an instance of a functional AppliDis architecture that includes most of the elements we described in this section.

AppliDis command-line launcher

Finally, the user can launch applications with a client accessible via a command-line interface by providing the name of the application that has been published in AppliDis as well as the login data.

1.6.3 AppliDis' load balancing

After the user informs AppliDis the intention of launching an application, the load balancer will be in charge of selecting the best available application server to run it.

The priority ranking to select the best available application will be based on:

1. Static characteristics of the servers: These are the characteristics related to the hardware setup that do not change over time, e.g., the installed CPU and the total RAM.

2. Dynamic characteristics of the servers: There are the characteristics that do change over time, e.g., the utilization factor of the CPU and

RAM. These are updated every 30 seconds (overwriting the old records).

3. Open sessions on the servers: The amount of sessions and, in particular, in which servers the user has currently opened sessions.

4. Priority ranking of the servers according to the note generated by the load balancer.

1.7 Login and application launching management

Between the four client options described in section 1.6.2, the desktop AppliDis client and the web portal are the most popular ones. In Figure 1.5, we show the interactions between the main components of the architecture [139] during the login and launching process of an application using these clients. We describe in detail the elements of each phase in the following subsections.

1.7.1 Phase 1: Login

Client start, login request: The login starts when the user chooses a client to send a login request to the management server. By default, the desktop client is launch during the Windows startup, and it automatically sends the login request while the web portal should be reached via the browser and is the user who needs to enter their credentials (username and password).

Login verification, login validation: Once the management server receives the request, it starts the user's login verification process by communicating with the domain controller that will let validate the login. In a Windows domain network, the authentication and authorization of users rely on Microsoft Windows' Active Directory (AD) service.

Groups verification, groups list: Next, the management server will query the database and recover the groups that correspond to the user. Generation of the application menu: With this list, the management server will be able to build the list of all the available applications for the user.

Display of the application menu: Finally, the list of available applications will be displayed in the browser or will appear as icons on the desktop, along with the green light login status indicator, as shown in Figure 1.3.

1.7.2 Phase 2: Launching an application

Clicking over application: The application launching process starts with the user clicking the icon of the desired application. This will make the client send an application request to the management server.

License verification: The management server verifies again that the user has the access rights to the application by verifying the list of contracts and the available licenses.

Balancing load: The management server sends a request to the balancing loader to verify the availability of the application servers and rights of the servers to execute the requested application. If more than one application server is available, the balancing loader will use the statistics of them to determine and select the best and send a launching notification to the client. Once the application server is chosen, it sends a launching notification to the client.

Request forwarding: The client establish a communication channel with the selected application server, and it will forward the launching notification to the application server.

login, authentication: The user will be logged into the application server using the active directory authentication. Session start, application start: Then, a session will be initialized on the application server, and the application will be launched. Then the server sends the graphical interface to the client where a component that is running in the background will crop and show to the user only the parts of the interface that show the application and discard the rest.

1.7.3 Phase 3: Utilization of the application

Session: The application is now ready to use. Once the client access the application, the statistics of the launched application will be registered.

Working with more than one application simultaneously If the application can be opened in the same application server that is currently running a session for the user. The part of the login and session start of the Figure 1.5 can be avoided, resulting in the process we show in Figure 1.6. It is possible to set the load balancer to assign priority to the application server that already has an open session.

The statistics related to the login and application process are updated after every login and application launch in the database. At regular time intervals, the statistics are used to compute the load status of the system. The load balancer uses the load status to assign a note to each application server generating a priority ranking. Since the ranking is not updated in real-time, a mechanism avoids the overload of an application server if several applications are launched in a very short period.

1.8 AppliDis Booster

As we have seen in Section 1.7, the process of launching an application is complex and can take several seconds to be completed. This is especially the case for the launching of the first application. If the secondary application is launched in the same server as the first one, the waiting time of a secondary application will be shorter because the user do not need to wait for the

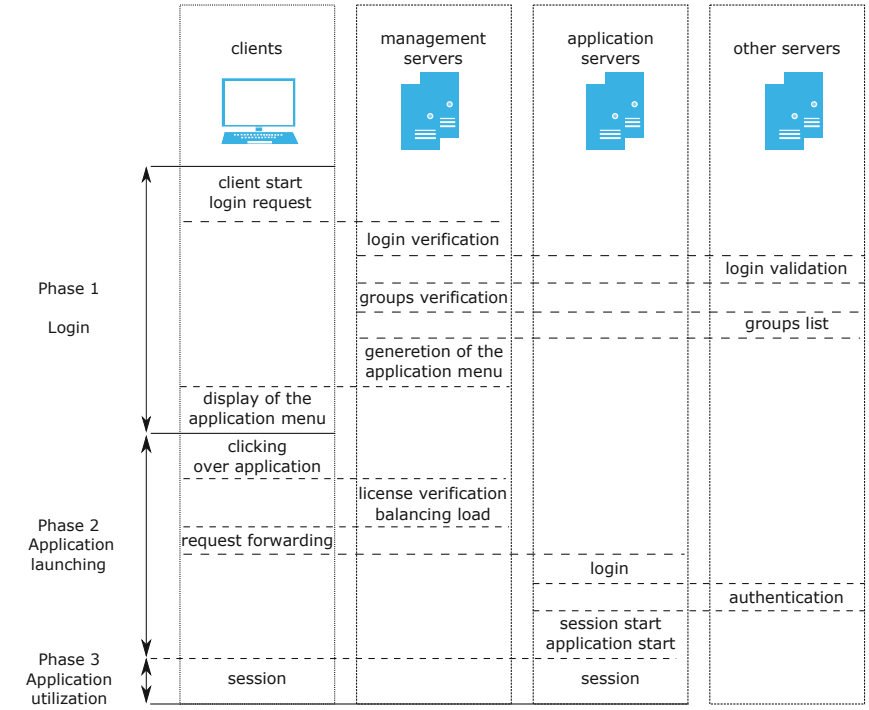


Figure 1.5 – Login and application launching process for the first application of the session.

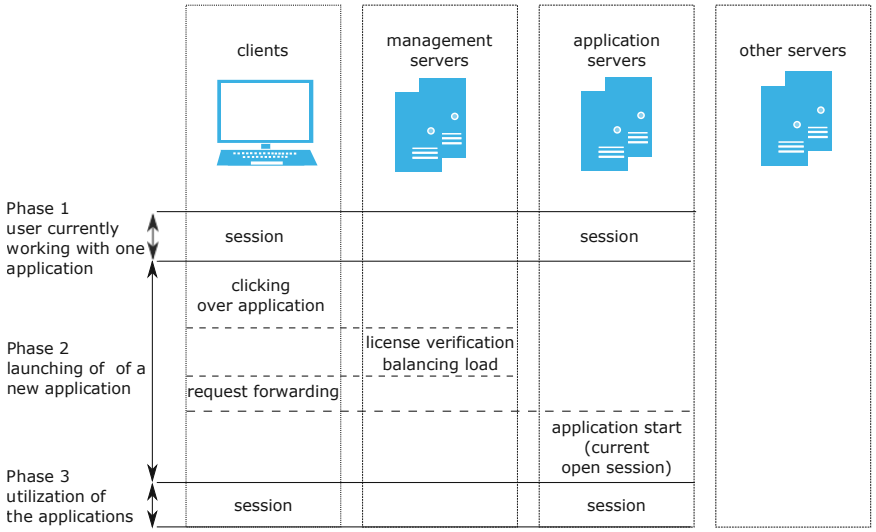


Figure 1.6 – Application launching process for supplementary application in a session.

RDP channel between the client and the server to be established and for the generation of the new session in the application server.

When a group of application servers has the same published applications, we say that it is a homogeneous group (also known as silo). As the quantity of applications servers that compose the silo increase, the probability of the load balancer choosing another application server for launching a secondary application on a different server also increases (since the number of application servers that can be chosen will be greater).

AppliDis Fusion 4 already has three different features that try to improve the delay in the application launching process. The settings of all of them can be controlled via the Booster Control Panel shown in Figure 1.7.

1.8.1 Profile virtualization

To run the application on an application server, we first need to have an initiated Remote Desktop Session on that Application Server. If it has not been initiated, AppliDis will first start the Remote Desktop Session, and the associated profile data will be downloaded to that particular application server. This is one of the most time-consuming processes in the whole launching sequence.

The profile virtualization creates "ghost files", empty versions of the original profile files, on the Application Server, where the Remote Desktop Session is initiated to avoid the time-consuming activity of downloading the original files, and the real files are downloaded just when they are needed.

1.8.2 Pre-loading

After the user authentication is completed, the pre-loading component will automatically start a "ghost" session in one of the application servers. By doing that, if the user now launches an application that is present on that application server, there will be a reduction on the launching time since the session will be already initiated. The "ghost" sessions usually are set to remain open a short amount of time (a few seconds) to avoid overloading the application servers.

1.8.3 Post-loading

When an application is closed, a "ghost" application is launched to keep the session active and ready to continue if the user decides to reopen the application. The component in charge of performing this task is known as AppliDisKeepAlive.

All the Booster options can be set either at the level of individual users or groups. The silo configuration, described in section 1.6.1 can also have an impact in the boosting of sessions due to the fact that if the users' applications are shared by a bigger amount of application servers, there will be a

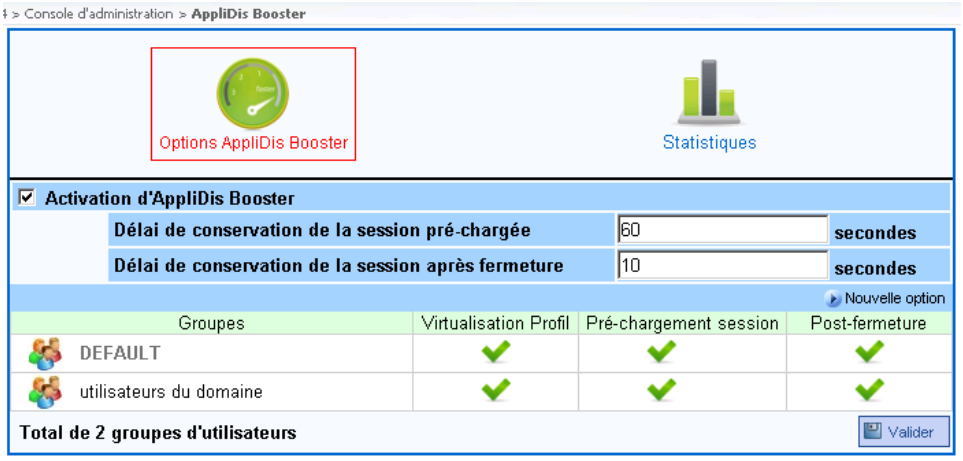


Figure 1.7 – Original control panel of Booster.

bigger probability of the load balancer choosing a server where a new session needs to be initiated.

1.8.4 AppliDis Booster performance measures and statistics display

The time savings of applying any combination of the AppliDis Booster components are displayed in an aggregated way according to a group of users to the system administrator in the Administration console, as shown in Figure 1.8. To visualize the savings, one user must have at least logged in once without using AppliDis Booster.

1.9 AppliDis databases

It is necessary to define what is known as the main AppliDis database to complete a successful installation of AppliDis. This database store all the necessary data for the correct functioning of AppliDis (e.g., list of applications, or which user can access them) as well as the data that will feed the functions that generate the statistics of use (date and time of the launching of an application).

The main AppliDis database has a master role. Other secondary databases, known as slaves, can be used to improve data availability via replication. AppliDis guarantee the replication and the maintenance of the data coherence. In general, new records are inserted first in the master and then replicated in the slaves' databases. AppliDis databases are, in general, SQL databases.

We have presented all the components of the AppliDis architecture. We now focus on the way these components are related from the database point of view. AppliDis use contracts to establish relationships between all the

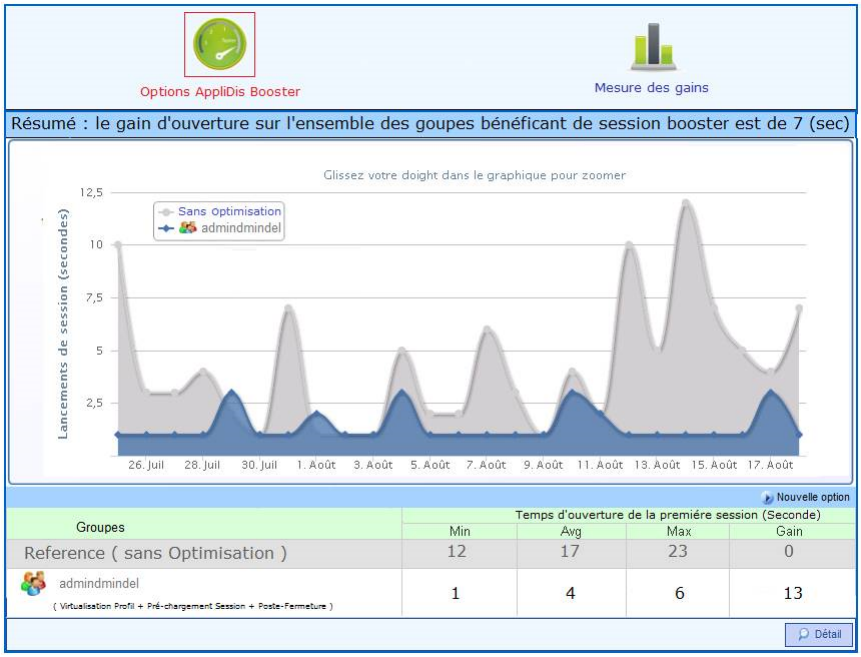


Figure 1.8 – Original Statistics Panel of Booster.

main components of the system. A contract is a way of grouping User(s), App(s), App Server(s), and VM host(s). Let us first consider one of the simplest cases, which is the one-to-one relationship. E.g., if we want to state that User 1 has the right to access App 2, we will create a contract on the database. The same contract could also be used to limit the amount of servers in which the App 2 for User 1 could run. This is, instead of looking for all the servers where the app is available, we can limit it to, for example, Server 3. Now the contract would link User, App, and Server. Similarly, we can establish a many-to-many relationship by using Groups. This option is available for Apps and Users.

AppliDis' update process

There are two main ways of modifying AppliDis. The first one is with the launch of a new AppliDis version and the second one with the distribution of a hotfix. A hotfix usually gathers several modifications and bug fixes, and the word "hot" refers to the fact that customers usually install these patches in systems that are currently running. During the project, the most significant update was going from AppliDis Fusion 4 to AppliDis Fusion 5. In general, the biggest changes will be introduced in a new version, and minor changes will be introduced in the form of a hotfix.

In the case of a modification to the database (e.g., adding a column to a table), the process involves first implementing the modification, then it will

go through the testing stage where possible compatibility issues are verified, and automatic tests check the presence of bugs. Finally, a package is generated that will be made available to the customer to update their systems. Good practices of Customer Relationship Management (CRM) were crucial to guarantee that decision-makers on the customer side had all the needed information (advantages and technical details) to install the released hotfixes and keep their system up-to-date. In particular, it is important to notice that the availability of complete data was directly related to the amount of customers installing the hotfix and how early they did it. To automatically and unequivocally define which are the modifications that are available in the database and decide which are the components of the analysis that can be run. We look for a field defining the AppliDis version and id of the last hotfix. Since we were not able to find it in the database, we proposed to add it. Then, in the Online AppliDis' Databases Visualization Tool, each modification that was proposed was listed along a way of verifying if it was present in the system. This way, the tool let us the user to quickly determine which set of analysis could be performed in the database.

During the project, we introduced modifications to the database, and different components of AppliDis, Booster being the most important.

1.10 Summary

In this chapter, we have introduced the reader to the concepts of cloud computing and virtualization. We showed their benefits and how the industry and, in particular, Systancia have built their services and products around them while trying to reduce the impact of the drawbacks. To gain a better understanding of one of such products, we have chosen to study in-depth AppliDis, the private cloud computing solution of Systancia. Starting with a description of the roles of the main components of the architecture (server and client side), we continued with an analysis of the interaction between them while logging in and launching an application (the main functionality of the product). We explained that a critical component of the user's experience is the launching time and detailed Systancia's solution to this problem (Booster). Since, during the thesis, we will analyze ML algorithm that will exploit data collected by AppliDis to improve the launching time, we explained the role of the database while the update process of AppliDis will let us better understand how the proposed improvements along this work were integrated to the product. Finally, since the behavior of the users is linked to the activity they develop, we carry out a customer analysis and segmentation.

Chapter 2

Approaches for User Modeling (UM) and Behavior Prediction (BP)

In this chapter, we review the progress of the User Modeling (UM) and Behavior Prediction (BP) techniques. It is organized as follows. In Section 2.2, we investigate the different kinds of data that we can encounter and the related approaches. In Section 2.3, we describe and review the most relevant set of methods that are: designed for the particular type of data we are interested in (sequential data) and for modeling the main features of the process that generate it (user behavior). In Section 2.4, we discuss how these techniques have been applied in the industry. Finally, in Section 2.5, we analyze the effect of the entropy in the behavior and how it affects the performance of any method.

2.1 Introduction

A User Model (UM) is a representation of the user (or group of users) of a computer system. As explained in [76], the first applications of user models are traced to the development of natural dialogue systems [105, 39, 8, 110]. These systems replicated the human behavior in a conversation and tried to improve the users' experience (e.g., assisting the user, answering questions, facilitating transactions, or teaching).

UM, as a discipline, focus on gathering data to construct models that can be exploited to build an adaptive system. These models can be useful by themselves since, for example, will let us classify our users, or we can use them as an intermediary step to solve other tasks, like making predictions about the user. Naturally, to tackle these tasks, researchers have been trying to apply techniques from other fields which focus is learning from data like statistics, pattern recognition, data mining, and ML [30].

Although, as explained by [145], ML researchers used to be confronted with issues like:

1. availability of large data sets,
2. need of labeled data,
3. capacity of the algorithm to adapt to a changing behavior ("concept drift"), and
4. computational complexity.

Nowadays, we have sensors, computing power, and storage capacity to produce data at very high rates (big velocity) and store enormous bodies of data (big volume). Crowd-sourcing Internet marketplaces, like Amazon Mechanical Turk, make easy to contract human workers to label datasets for a reasonable price, and many behavior-related problems can be framed as non-supervised problems [72] (avoiding the need of labeled data). Techniques that let us train algorithms in an online fashion, making it possible to detect and adapt to changes in the behavior. Finally, the development of specialized software (like the CUDA library [100]) and hardware (like GPUs, TPUs [71]) for processing ML related computations combined with the availability through cloud computing services, let us tackle problems that were not accessible in the past.

2.2 Data for UM and BP problems

The characteristics of the data will have a big impact on the framing of the problem. For instance, since behavioral data is inherently sequential in most cases, we will generally found algorithms that will require sequential data as inputs. As explained in [82], sequential data consist of a set of records ordered according to some index. A special case is temporal data (time series) where records are ordered according to time.

Temporal data can be obtained by sampling a signal at a particular rate (e.g., collecting temperature measurement each hour), or it can be the result of recording an event as it happens (e.g., recording a timestamp when a user logs into a computer system). Time series usually require the observations to have a fixed sampling frequency. It is also important to notice that in any case, there will always be a limit on the precision associated with the time measurement of an event (for instance, timestamps generally have a precision to the millisecond).

We usually associate big data with a dataset that have a huge amount of samples. However, this is not the only relevant characteristic. Big data can also be high dimensional. This means that we can potentially find cases from a dozen to hundreds (e.g., medical diagnosis) or thousands of

features (e.g., videos from surveillance cameras or functional magnetic resonance imaging) for each event. Another important aspect is the velocity of the data. For instance, a high-speed camera can record an astonishing amount of high-resolution frames in a second. So, as we see, the device (e.g., laptop, smartphone, or smartwatch) and the sensor will have a great impact on the characteristics of the data being captured.

When events are combined with a duration, we get intervals (e.g., the use of an application). Although there are algorithms specially crafted for interval-based data, which allow the extraction of temporal patterns [55], the relational algebra proposed by [7] let express relationships between intervals, use them for reasoning and extend algorithms that were initially designed for events.

In some cases, the learning algorithms can be modified to deal either with desegregated or aggregated data (e.g., when analyzing the eigenbehavior of students or groups of students [44]). In some cases, the data is already aggregated when it is collected (e.g., queries of search engines [107]).

2.3 Relevant approaches to UM and BP

This section review different approaches to user modeling and behavior prediction and other related methods that will be of interest in the development of this thesis. Although it is sometimes difficult to establish clear temporal boundaries to the development of certain methods, based on chronological order of major advancements and picks in popularity, we decide to group and present them in the following order:

- Time series analysis and signal processing
- Expert systems and Bayesian networks
- Association rule mining
- Sequential pattern mining
- Probabilistic models
- Deep Learning
- Other methods

2.3.1 Time series analysis and signal processing

Time series analysis will try to separate the noise from the patterns (in our case, this is the user behavior from the randomness).

We can also think of a time series as the realization of a stochastic process X, which is a sequence of random variables $\{X_t\}_{t\in\mathcal{T}}$, indexed by $\mathcal{T}$

where $\mathcal{T} = 1, ..., N$. In this case, we cannot assume that samples will be independent and identically distributed. For instance, for a given user, we would expect the number of applications open at time t to be related in some way to the number of applications open at time $t + 1$. We know that the covariance is a way of measuring a linear dependence between two random variables. In a time series, the autocovariance

$$C(t, s) = cov(X_t, X_s) = \mathbb{E}[(X_t - \mathbb{E}[X_t])(X_s - \mathbb{E}[X_s])], \tag{2.1}$$

will measure the linear dependence between two points (s,t) of the time series. We can also use the autocovariance to define the Autocorrelation Function (ACF)

$$\rho(t, s) = \frac{C(t, s)}{\sqrt{C(s, s)C(t, t)}}. \tag{2.2}$$

If the distribution of the random variables do not change over time, we say that the process is stationary. As a consequence, the mean, the variance over time, and the autocovariance will be constant. In our example, if the behavior of the user changes over time, this process will be nonstationary. Since several techniques rely on the assumption of the series being stationary, we can apply different transformations to the original series to respect this assumption. The most common one is based on differences between subsequent observations. This is to generate a new time series

$$Y_t = X_t - X_{t-1}. \tag{2.3}$$

We can apply this technique several times until getting a stationary series, but usually, one time suffice. The second one is to fit a curve and model the residuals to remove the trend of the data. Finally, logarithmic transformations can help with non-homogeneous variance over time.

As explained in [113], we can differentiate two complementary approaches for time series analysis, the time domain approach, and the frequency domain approach.

Time domain approach

The time-domain approach models a given value of a time series as a parametric function of its past values. To capture the linear dependence between a value and the p past values, we can use an autoregressive model of order p, AR(p):

$$X_t = \delta + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + \varepsilon_t \tag{2.4}$$

where X_t is the time series, ε_t is white noise, and

$$\delta = (1 - \sum_{i=1}^{p} \phi_i)\mu \tag{2.5}$$

with μ denoting the process mean.

On the other hand, a Moving Average model of order q also known as MA(q) which is defined as

$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} \tag{2.6}$$

use the past white noise error terms $\varepsilon_t, \varepsilon_{t-1}, \ldots, \varepsilon_{t-q}$ to model the of the series at time t.

As proposed by [21], we can use use the ACF to define the appropriate order q of the MA model and the Partial Autocorrelation Function (PACF) to choose the order of p of the AR model and combine them in what is known as an ARMA(p,q) model. Other models that fall into the time domain approach category are the Autoregressive Integrated Moving Average (ARIMA), multivariate ARIMA, Generalized Autoregressive Conditional Heteroskedasticity (GARCH) [18] , and state-space models [107].

Frequency domain approach

The frequency-domain approach focus on finding periodic variations in the time series. The Fourier transform represents the original time series (signal) as a linear combination of complex sinusoids.

In particular, the normalized Discrete Fourier Transform (DFT) of a sequence $X(n)$, $n = 0, 1, \ldots, N-1$ is a sequence of complex numbers $x(k)$:

$$x(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} X(n) e^{-2\pi i \frac{kn}{N}} \text{ , } k = 0, 1, \ldots, N-1 \tag{2.7}$$

The weight of each of these complex sinusoids let us separately evaluate the variance associated with each periodicity.

The periodogram

$$P(k) = \|x(k)\|^2, \; k = 0, 1, \ldots, \left\lceil \frac{N-1}{2} \right\rceil, \tag{2.8}$$

let us estimate the energy of each frequency.

The spectral analysis studies the profile of the signal over the frequency domain. The power spectral density can be estimated trough the Periodogram and trough the Autocorrelation Function. The frequency approach usually need long time series.

In the ML context, these tools could be used to generate features and compare time series through a metric (distance function). E.g., in [136],[137] we can find the application of the periodogram and autocorrelation to discover relevant periods in aggregated behavior of search engines queries, "bursts" of activity, use the feature for classifications via dendrograms. Defining metrics between time series let also apply k-spectral clustering techniques as in [147] to twitter data, or for spatio-temporal prediction [112].

The methods from time analysis and the ones of signal processing are intimately related. This means that while in statistics (time series analysis), we study a stochastic process and its realization, in signal processing, we study a signal. The methods and the mathematics remains almost the same, with just some adjustments (e.g., normalization).

2.3.2 Expert systems and Bayesian networks

Although there are different definitions of an expert system [128, 68], we will focus on the one that says that they are computer systems that solve a complex problem that requires expert knowledge [47]. These systems will need to perform inference (reasoning to arrive at conclusions from evidence) to accomplish this task, usually dealing with uncertainty and indicating the best decisions. We will also expect these systems to perform at a comparable level (or even better) than a human expert, either replacing or supporting him/her in the decision process.

While the first approaches to expert system consisted of hand-coding a set of fact-based rules and heuristics (knowledge engineering) [46], during the 1980s, the use of probabilistic graphical models become a popular in the design of expert systems [66].

Along with the most popular types of graphical models we find, what is known as, Bayesian Networks (BN) [103], which provides a way of representing the decomposing a joint probability distribution. Some of the most iconic applications of BNs were in medical diagnosis with the Pathfinder project (that applied first Naive Bayes models and later similarity networks [77] during Pathfinder IV) and troubleshooting [22].

Today, BNs are still in use, and we can find applications in user modeling [99] and behavior prediction like, for instance, in online shopping prediction [78].

2.3.3 Association rule mining

In the context of commercial applications, [4] studied the problem of finding relationships between products in a shopping list, for instance, products that are usually acquired together. The objective was to find what are called association rules [89] of the type $X \Rightarrow Y$ (where X is known as the

antecedent, and Y is the consequent), meaning that if the customer buy a set of items X, then the customer buy item Y.

The sales data can be organized in the form of what is known as a transaction database. This is a set of transactions made by different customers where each transaction is the set of items that the customer bought in a particular visit to the shop. Since a non-empty set of items is also know as an itemset, transactions are also itemsets.

Usually, the rules will need to respect certain conditions to be relevant. In general, the antecedent will need to have a support greater than a certain threshold where the absolute support of an itemset X, $\text{supp}(X)$, with respect to a set of transactions T is defined as the number of transactions in T that contain the itemset X. The relative support is defined as the absolute support divided by the total number of transactions in T (the cardinality of T).

Other common properties that are used to define the relevant rules are:

- Confidence [4]: The confidence of a rule, $X \Rightarrow Y$, with respect to a set of transactions T is defined as

$$\text{conf}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X)} = \frac{P(X,Y)}{P(X)} = P(Y|X) \tag{2.9}$$

- Lift: It measures how many times more often X and Y occur together than expected if they were statistically independent.

$$\text{lift}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X) \times \text{supp}(Y)} \tag{2.10}$$

- Conviction [24]: It has been proposed as an improvement of the confidence since indicates how dependent is the consequent on the antecedent (being directional). High conviction values indicates high dependence and takes the value 1 if the items are independent.

$$\text{conv}(X \Rightarrow Y) = \frac{1 - \text{supp}(Y)}{1 - \text{conf}(X \Rightarrow Y)}. \tag{2.11}$$

2.3.4 Sequential pattern mining

In many different applications (e.g., DNA sequencing, commerce, and weblogs), datasets can be structured in a particular way, known as a sequence database.

A sequence database is an ordered set of sequences, where each sequence is an ordered list of itemsets. Table 2.1 shows a sequence database that contains four sequences where each sequence is formed by a combination of different amounts of item lists in sequential order.

ID	Sequence
1	({1,3,8},{5},{7,9})
2	({2},{1},{2,3,9},{3,8})
3	({1},{3},{8})
4	({9},{5},{4,6,7,9})

Table 2.1 – An example of a sequence database.

In the case of the AppliDis applications' and sessions' logs, we can generate a sequence database if we structure the IDs of the applications launched during a session as an itemset and generate a sequence for each user with the data of all their sessions.

Similarly to the case of the support of an itemset for the association rule mining, we can define the support of a sequence, we will need first to define the concept of containment. We say that a sequence $s_a = (a_1, a_2, \ldots, a_n)$ is contained in a sequence $s_b = (b_1, b_2, \ldots, b_m)$ if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_n \subseteq b_{i_n}$. Now, we can define the support of a sequence, $supp(s_a)$, with respect to a sequence database S as the number of sequences in S that contain s_a.

Then a frequent sequence (or frequent sequential pattern) can be defined as a sequence for which $supp(seq)$ is greater or equal to a certain threshold $minsup$. This is that the sequence is contained in at least $minsup$ sequences.

A sequential pattern would be maximal when it is not included in another frequent sequential pattern. Sequential pattern mining, as introduced in [6], is the task of finding all frequent sequences in a sequence database. There are two main approaches. The apriori-based approach identifies the sequential patterns by building up a set of candidates (starting from singletons) and pruning the set by enforcing the support constraint at each step. It presents a high time and space complexity since it usually creates a big number of candidates and requires to scan the whole database each time we increase the size of the patterns. If additional constraints (like on the maximum or minimum size of the gaps between elements) need to be applied, then a generalized version of this algorithm, named GSP (Generalized Sequential Pattern) [5], can be used. To reduce the space complexity, a second approach known as pattern growth based, narrow the search space by projecting and dividing the original sequence database at each step. In recent years, tackling bigger databases have been possible thanks to the introduction of more efficient algorithms such as CM-SPADE, CM-SPAM [48], FCloSM, and FGenSM [83] and algorithms that can benefit of parallel computing [49].

2.3.5 Probabilistic models

We say that a model is deterministic if every state of the model is uniquely determined by its parameters and initial conditions. These kinds of models are well suited when we can measure those initial conditions, and we have a good understanding of the process we want to model. However, when modeling behavior, measurements are usually noisy, and we have no perfect knowledge of the underlying process. So to deal with this uncertainty, probabilistic models, incorporate random variables and probability distributions. In this section, we review different techniques to model-specific features of the processes related to human behavior. A very important one is that these processes will present recurrent patterns over different intervals of time (e.g., days, weeks, and months). This can be captured with the use of different Probability Density Functions (PDFs) [34] over different circular manifolds (e.g., the circle, the n-dimensional sphere or the torus)[3]. For instance, Kernel Density Estimation technique (KDE) can be combined with a circular distribution (the wrapped normal) as basis functions to model the behavior of smartphone users [81]. This method also allows for the use of other basis functions as von Mises, circular uniform, wrapped Cauchy, or wrapped Lévy. Sometimes the need for circular distributions can be avoided, as shown in [120], where a tangent space over the n-dimensional sphere and a Dirichlet process are combined to keep the method tractable. Since in this work, we will study the opening of sessions or the launching applications, we will also be interested in the ability of modeling events that seem to happen at a random rate. This is, for instance, the case of phone calls [57] that can be described with Poisson processes, although it is also important to consider that behavior can sometimes present non-Poisson characteristics [138].

We expect sequential data to show a correlation between observations that are close in time. In the most general case, we can think of a particular value in the sequence as depending on all the values that come before it. This means that given a sequence $\{x_1, \ldots, x_N\}$ we can factorize the joint PDF as

$$P(x_1, \ldots, x_N) = \prod_{n=1}^{N} P(x_n | x_1, \ldots, x_{n-1}). \tag{2.12}$$

However, conditioning on all the previous observations can be intractable very fast. So instead, we can assume that each value depends only on the previous N observations:

$$P(x_1, \ldots, x_N) = P(x_1) \prod_{n=2}^{N} P(x_n | x_{n-1}), \tag{2.13}$$

This model is known as Markov Chain [93] and, in particular, if $N = 1$ we call it a first-order Markov chain. Markov Chains have been successfully applied to behavioral models, with web utilization [150] being a clear example, and it can also be integrated with other methods as clustering and association rules [74].

If, for each observation, we add a latent variable and take away the Markov property from the observations and put it on the latent variables, we get a State Space Model (SSM)[43]. A Hidden Markov Model (HMM) [38] can be seen as an SSM where the latent variables are discrete or as a mixture model [25] where the choice of the mixture component from where the next observation will be drawn depends on the component that have been previously selected. Since training this kind of models can be expensive, incremental training approaches for HMMs [75], deal with the problem of retraining. On the other hand, when both latent and observed variables are Gaussian, we get what is known as a Linear Dynamical System (LDS) [16].

2.3.6 Deep learning models

Traditional ML algorithms learn to map manually engineered features to the output. In this section, we will discuss algorithms that will also be able to learn to extract the appropriate features from the input. In general, these algorithms belong to the subfield of representation learning, which contains deep learning, where several levels of features with different levels of abstraction are learned [52]. Some of the more relevant deep learning algorithms can be mapped to one of the following families:

Multilayer Perceptrons (MLP)

The Multilayer Perceptrons (MLP) [58] are feedforward artificial neural networks that have an input layer, one or several hidden layers, and an output layer. A network with just one hidden layer would be considered a shallow network, while the depth of the network will increase with the number of hidden layers. The neurons that compose a hidden layer will usually be connected to all the neurons in the previous layer (fully-connected layer) and apply non-linear activation functions (e.g., logistic function, hyperbolic tangent, or rectified linear units). The combination of the several levels of layers and the non-linear activation function will make the network capable of learning complex non-linear functions to solve classification and regression problems.

Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNN) [84] are feedforward artificial neural networks that apply a special kind of layer known as convolutional layers. While in a fully-connected layer, a neuron is connected to all the neurons in

the previous layer, in a convolutional layer, a network is connected to a small set of neurons in the previous layer (known as the receptive field). A set of parameters (weights and bias) shared by the neurons will be used to generate what is known as a feature map. A convolutional layer will be composed of a set of feature maps.

Autoencoder (AE)

Autoencoders (AE) [60] are unsupervised generative models based on the feedforward neural network that tries to reconstruct the input after compressing it in its hidden layers. The learned features can then be used in different applications.

Restricted Boltzmann Machine (RBM)

Restricted Boltzmann Machines (RBM) [61] are unsupervised generative models based on a network composed by a visible layer and a hidden layer which are fully connected. While Boltzman Machines allow for connection between the neurons in the same layer, RBMs do not. This makes them suitable for applying a particular training algorithm named contrastive divergence [33]. RBMs can be stacked to build a deep model (since we say that a model gets deeper as the number of layers increase).

Recurrent Neural Network (RNN)

While the MLP only allowed for feedforward connections, Recurrent Neural Networks (RNNs) [122] allow cycles in the connections between neurons. This makes them especially suitable for modeling sequential data since they can accept input of variable length. However, vanilla RNNs suffer from what is known as the vanishing or exploding gradient problem.

This is, the exponential explosion or decay of the gradient when propagating trough time [62, 12]. To deal with this problem, some architectures as the Long Short-Term Memory (LSTM) [63] or the Gated Recurrent Unit (GRU) [37], use a memory state and gates to act on it and allow the gradient to flow unchanged.

RNNs can also deal with complex probability distributions when combined with energy-based models as the RBM [20, 121].

Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GAN) [53] are unsupervised generative models that use two neural networks, one known as the generator network that generates samples and another known as the discriminator network which task is to tell apart samples from the dataset and samples produced by the generator. Since the task of the generator is to increase the error

rate of the discriminator when both networks are trained to become better into their adversarial task, they can achieve an equilibrium point, and the generator can learn an approximation of the probability distribution of the dataset.

Deep Reinforcement Learning (RL)

In the decision theory framework [108], we generate a probabilistic model of reality, and we decide which action to take (the decision) optimizing a Loss function. In this sense, the problem is static. On the other hand, the Reinforcement Learning (RL) framework [123, 114] incorporate two relevant features (characteristics). The first is that it is dynamic. Each decision (action) takes us to a new state where we take a new action (decision). The second is that since we do not know the transition model, we will learn it from the data. That makes this framework ideal for solving problems as in [126].

2.3.7 Other models

We briefly describe some work related to the user modeling and behavior prediction tasks that apply other relevant models that have not been described in the preceding sections.

- Logistic Regression [34], this algorithm has been applied in the context of social media to model the probability of the user accessing multimedia content [141]. The prediction of this model was then combined with other models to decide which and when the content should be prefetched by combining data from the user and the state and evolution of the internet connectivity (i.e., speed of the connection and cost of downloading data). It has also been applied to the analysis of online purchases, modeling the probability of a user buying a product given a log of the user activity [91].

- Support Vector Machines (SVM) [134], this classification algorithm has been applied, for instance, to predict purchasing behavior [124].

- Decision Trees (DT) [131], in particular, classification decision trees, have been applied to predicting clicks on adds in the context of social networks [59].

- Ensembles, which are the family of methods to combine different models (e.g., Logistic Regression, SVM, DT, and others) to improve the quality of the prediction. For instance, it has been used for the prediction of commercial transactions [19].

2.4 Applications of UM in the industry

2.4.1 Faster application launching

The notion of making a resource available before it is needed is known as prefetching. In the context of computer systems, data is a resource that can take a long time to retrieve from hard drives. So instead, a smaller and faster memory, named cache can be used to prefetch data. Intel Turbo Memory is a technology presented in 2005 and made available in 2007 that relied on caching data into a flash memory to allow faster boot-up, faster application launches, and power-saving since it avoided spinning the hard drive disks when accessing applications. Microsoft also developed a similar caching technology based on flash memory named ReadyBoost and another based on hybrid drives named ReadyDrive. Both of these technologies, relied on SuperFetch [65] a prefetching system, developed for Windows Vista and Windows OS operative systems, that exploited the user behavior (e.g., the user most used applications according to weekday and time of the day). While Falcon [146], for Windows Phone systems, was designed for learning behavioral patterns of mobile phone users.

The problem of predicting the next application launch in the context of mobile phones is closely related to the one we are trying to tackle. Some important similarities are that 20-second launching delay is not uncommon [146], which is close to the estimates Systancia has of average time for applications in the cloud.

Some relevant differences are that smartphones provide a big set of built-in sensors like microphones, cameras, light detector, GPS, accelerometers, gyroscopes, and even WiFi antennas that can be used as sensors. The UM and BP algorithms can exploit all the data generated from these sensors. In our case, the available data sources are much more restricted.

From the application point of view, the mean interaction time of the user with the applications is significantly shorter in mobiles phones. Mobile Applications tend to be used in "burst". A mobile phone will usually be idle until some event "ignite" an active period (like an incoming SMS) that is then chained with the use of some applications before going back to idle. For a greater interval of time, certain applications are download and use very frequently for a short interval of time and then "forgotten" by the user. The energy and memory cost of caching an application on a mobile phone is not negligible. Thus this should also be considered in a utility function when evaluating the performance of false-positive predictions.

Falcon extract different features that will be used for the prediction. Since the activity is expected to appear in the form of burst or as they call them "sessions", the idea is to classify applications in two categories, trigger applications (that start the activity) and follower applications (that tend to be used after). Location is estimated by minimizing the distance to the

center of the geolocation traces. Another feature that is computed is the burst, computing the starting point and duration of the activity burst for each application with a wavelet detection algorithm.

These features feed a decision function that, in turn, feeds an optimization algorithm (0-1 Knapsack problem) to optimize the utility function that takes into consideration the energy and memory cost. The algorithm is trained tabula rasa for each user and requires two months of activity to yield good results.

2.4.2 Personal digital assistants

One of the first massively distributed personal digital assistant was the one included in Microsoft Office 97. This personal assistant was required to exploit data to find the correct answer to the queries of the user related to the use of the Office suite products. Behind the early development of the necessary algorithms was the Lumière Project, which had several objectives, as explained in [64]. The approach was based on a Bayesian Networks. These networks helped the assistant to reason about the actions of the user, the answer for the queries, and to build a user model. This model evolved by acquiring new data from the applications and system events.

The Microsoft Office assistant provided passive assistance in the sense that it acted upon requirement via a query from the user. Today, several companies provide proactive assistants meaning that they can predict and provide useful assistance even before it is required by the user. Some of them are:

- Microsoft: Cortana (Windows 8).
- Apple: Siri Suggestions (Apps suggestion for Iphone). [102] A menu for smartphones similar to siri suggestions.
- Google: Google Now, replaced by Google Assistant. [54]
- Facebook: M.
- Amazon: Alexa (used in the Echo product).
- Samsung S voice.

Most of the services that these companies provide are not linked to a specific device. This means that, for instance, we can access Facebook from a laptop, desktop computer, or a mobile phone. This lets them build a general profile of the user not necessary linked to a specific device and improve the performance of the prediction. To make this possible, they also need to store the generated profile in the cloud.

Since most of these companies also provide a suite of products, is from the combination of different facets of the user behavior that they will be able to provide richer results. Next, we explore some of the data sources they use.

Companies that do provide an email service (like Microsoft and Google) had historically exploited the content of the email to provide targeted advertisement. However, companies that do not provide an email service can still benefit from email content if they store it in their devices (as in the case of Apple). The latest generation of assistants can now use this data to detect dates and codes that indicate plans, flights, and even mail packages to display it, for instance, in the calendar service or launch associated services like maps. Email can also be used to train language models customized to the user and propose short answers for incoming emails. Microsoft and Google both have the possibility of generating suggestions based on the search activity in their internet browsers and their search engines. Mobile operating systems also apply some form of recommendation system to choose what news to display to the user.

One of the most exploited data sources across different predictive task is location data, which can be extracted from different sensors. Human behavior is strongly correlated with location. One of the most direct applications is to generate a location profile to predict where the user is heading next and show useful information like the traffic state in the usual route or the estimated arrival time to destination. These systems can also classify the locations and identify common categories like home or work. This data can be combined with accelerometer data to identify activities like sleeping or exercising. Activity detection is also enhanced by comprehensive data sources like specialized gadgets that can measure precise location, heart rate, or sleeping patterns. Since most of this specialized gadgets prose the use of their applications, operative systems like Apple iOS, propose an application to gather all these data coming from different sources. Location data also feed application utilization prediction. On this last task, some of the assistants will also exploit third-party application data.

As we have seen, the richness of the data these companies can gather make them possible to generate incredibly detailed models of their user, which they can, in turn, use to generate accurate predictions.

2.4.3 Recommendation systems

According to [2], although related techniques can be traced to other fields, recommender systems appeared as a subfield in the mid-1990s with work in collaborative filtering. The problem could be informally stated as, based on the ratings that users assigned to different items and maybe other data, estimate the ratings that the users will assign items that they did not rate. In turn, these estimations could be used to rank the items and recommend them to the users. We can identify three general approaches. The content-

based approach compares the item we want to rate to other items that the user has already rated. The collaborative recommendation approach looks for the ratings of other users to similar items. Finally, the hybrid approaches combine the benefits of the former two approaches (for instance, defining a method for combining the estimation of the two separate approaches). More in-depth descriptions of the recommender system approaches can be found in different general surveys [17, 15].

One well-known application of recommender system is predicting movie ratings. Netflix, a media service provider, held a competition between 2006 and 2009 to improve the accuracy of their recommendation system [11]. The datasets of the competition and other related (like the Large Movie Review Dataset [92]) are still used for benchmark purposes. Another very relevant industrial application that provides interesting datasets and fosters the development of this subfield is the e-commerce (e.g., Amazon [116] or Alibaba), where we need to propose products to the customer based on their activity and products they like or buy.

In recent years, deep learning techniques have also been applied to recommender systems, improving their results [149].

2.4.4 Online social networks and media

The common characteristic of online social networks and media is that they allow users to share content and participate in social networking. Just to name some categories [86]:

- Social networks: Facebook, Twitter, Linkedin, Google +.
- Blogging and Microblogging Platforms: WordPress, Blogger, Tumblr.
- Wikis: Wikipedia, Encyclopedia of Mathematics.
- Rating and Reviews: Yelp, TripAdvisor.
- Instant messaging: Whatsapp, Facebook Messenger, Viber, Skype.
- Sharing sites: Instagram, Flickr, Youtube.
- Question and Answer Platforms: Quora, Yahoo! Answers.

All these platforms possess millions of users that create a huge amount of data and enormous social graphs. They also present a big variety of different problems that can be tackled with ML (e.g., UM and profiling [1], BP [51], sentiment analysis, recommendation, malicious activity detection [90]). We can say that a significant part of the latest advancement in the field of ML had come from researchers working for online social networks and media companies. Next, we will cover some interesting application in the context of UM.

The development of user profiling algorithms is based on exploiting different data sources that can be separated into two categories. On the one hand, we have the data provided by the user (e.g., demographic data, comments, posts, likes, rates,status updates, answers, searches, tags) and on the other hand the data that corresponds to the user activity which results from the use of the platform (e.g., chosen content, time spent, mouse activity and eye movement).

One of the key element of any online social network is the relationship between the users. A common representation of the connection between users is to construct what is known as a social graph, where the nodes represent users and the edges represent some relationships that the platform allows between users [144] (e.g., Facebook friendship can be represented with an undirected edge, while the concept of following another user on Twitter can be represented with a directed edge). Social graphs, for instance, can also include places, events, organizations, or other entities as nodes, and edges can then represent endorsement of the user to a certain organization or other actions like visiting a place.

These social graphs allow researchers to study the profile of the users and a wide variety of phenomena related to their behavior. Many of the interesting characteristics that we can measure in the social graphs are quantities that have been defined in the context of graph theory. Some of them are:

- Degree distribution[130]: which is the distribution of the number of relationships that each user has.
- Homophily: that describes to which degree the users' present relationships with users with similar profiles.
- Multiplexity: which describes the strength of the relationship between users.
- Density: which is the ratio between the quantity of relationships and the quantity of all possible relationships between the users.
- Distance and shortest path between users: which is the number of users between two users.
- Cliques: which is the number of groups of users that are pairwise connected between each other.
- Structural cohesion: which is the number of users that we need to eliminate to disconnect two groups of users.

All the characteristics of the graph that we have presented can be computed over the graph at a given moment in time. However, we can also consider the evolution of the graph over time [94].

- Evolution of the structure over time.
- Velocity of the changes.
- Which properties are preserved.
- Finding features that help to predict future relationships.
- In which degree a user influence other users.

A social graph for N users can be encoded as a square matrix A of size N, such as the element A_{ij} is equal to one if there is an edge between the user i and the user j and zero otherwise. This matrix is known as the adjacency matrix which we can combine with the degree matrix (a diagonal matrix D of size N such as A_{ii} is equal to the degree of the user i) to construct what is known as the Laplacian matrix $L = D - A$. This last matrix is useful for the application of the spectral clustering techniques [79] that let us find communities in the social graph.

2.5 Limits of predictability

Two important factors will affect the performance of any behavior prediction algorithm. The first one is the amount of structure that there is in the behavior of the subject that we are studying (or the amount of randomness, since the combination of the structured behavior and the random behavior will give us the behavior we observe). The second one is the capacity of the algorithm itself to find this structure.

We can focus on the first factor by assuming that we have an algorithm that will be able to exploit all the structure present in the behavior and establish an upper limit for the performance of our prediction algorithm. Instead of trying to predict the amount of structure, we can estimate a lower bound to the amount of randomness by measuring the entropy of different probability distribution of the user behavior.

[118] used a database of the geographical location of 50000 cellphone users for three months to established a limit to the predictability of these users independently of the algorithm applied for the prediction. The result showed that the distribution of maximum potential predictability picked at approximately 93%, which means that for the users in that database, independently of the technique we use, the distribution of the minimal amount of wrong location predictions for a user has a mode at 7%.

As the estimation of the entropy is directly linked to the amount of structure in the behavior that we can capture, they computed three different values of entropy capturing different relationships between the locations the user visited (and therefore different amount of structure). The first one (and the one capturing the smallest amount of structure) considered only

the total amount of locations that the user visited during the three months of the study and assumed a uniform distribution over all the locations. The second one replaced the uniform distribution for the empirical distribution (relative frequency) of the locations. The last one considered not only the total amount of locations but the sequential relationship between them by applying a method based on the Lempel-Ziv compression algorithm. This approach is also similar to the applied by other authors [69, 88] that studied location data from GPS sensors.

Considering our problem, this would mean that we can establish three different measures of entropy. One, considering the total amount of applications launched by the user. A second one, considering the relative frequency of the launched apps. And finally, a third one, considering the sequential relationship between the use of the different applications.

One difference we would notice between the locations and the applications is that while the user can only be in one place at a time, a user can launch several applications at the same time. A naive way of solving this difference would be to generate a new symbol to represent the state of a specific combination of applications.

A critical point in the estimation of the entropy of the users via a sample is that we do not have any guarantee that the user has visited all the locations she will ever visit. We can improve the estimation of the entropy if we try to add an estimation of the locations that the user will still visit even though we did not observe that in our sample [132]. The problem of estimating the entropy of the user behavior in the cloud will be explored in Chapter 4.

All the discussed estimation tried to established an upper bound to the predictability without taking into consideration the capacity of the prediction algorithms. For instance, [119] studied the limits of the predictability in particular for Markov models using location captured by Wi-Fi and [87] shows that the mutual information between symbols in a sequence decays exponentially with the distance for Markov processes while only logarithmically for deep architectures, what explain why deep learning methods can present and advantage for sequential predictions.

Finally, in all the studies discussed before in this section, the location of the user was model as a discrete sequence of locations. In [118], the fact that the location is continuous is captured by modeling human mobility as a continuous-time random-walk.

2.6 Conclusion

From the devices and sensors that capture the data to the computational resources that process it, today, new technologies allow us to overcome big barriers and apply techniques to UM and BP that were not at reach a couple of decades ago.

While in the past, we mostly rely on divide-and-conquer approaches, some new methods can deal with highly complex problems and get better results when trained end-to-end. On the other hand, these methods will usually require a big amount of data. So it is also essential to keep in mind the value of being able to deal with small dataset which are still very common in some industrial applications as our.

Since there is a wide variety of techniques that can help to improve the user experience and they do not share a common framework or directly translate to our problem, it is not trivial to establish a baseline. This is a significant difference when comparing, for instance, to image classification problems where there are standard datasets and performance measures that are widely accepted and used in the community. This is why in this work we will propose a solution to this problem.

The enormous economic value in behavior modeling and prediction motivates big companies to invest heavily in the development of new methods and related technologies and make them, at last, an important development driver. However, even though the technological advances in the field are astonishing, they are maybe not as deep as the cultural changes that make devices and sensors ubiquitous today. Human behavior has been, without any doubt, affected by these technologies, and it seems that this is still a process in progress.

Chapter 3

Exploratory Data Analysis (EDA) and user profiling

Following the order of the phases of the CRISP-DM process model (shown in Figure 1), in Chapter 1 the business understanding phase was completed by framing the business problem. Now, this chapter develops the next two phases of the model, the data understanding, and data preparation phases.

First, in Section 3.1, we will analyze the structure of the database that is used to gather data to then explore the data itself in Section 3.2. While exploring the data, we will highlight the need for transforming the original data to apply different ML algorithms. We will propose some useful transformations in Section 3.3. Then, in Section 3.4, we will introduce the definition of entropy and other related concepts that will prove fundamental for our understanding of behaviors of users and for generating a measure of distance that will help us generate user profiles in Section 3.5.

3.1 Database exploration

The first part of the exploratory work was to inspect all the tables of the AppliDis SQL database and shortlist the ones that potentially included any data related to the activity of the users.

3.1.1 Shortlisted tables

We started from analyzing the database of a customer with AppliDis Fusion 4 and we shortlisted 11 from a total of 78 tables:

- `useruseappli`
- `app`
- `machine`

- machine2
- utilisateurs
- grpapp
- grpuser
- contratlocatif
- liaisgrpapp
- liaisgrpuser
- liaissrvgpusr

For each table, the columns and values were inspected. During this process, all tables and columns were renamed to keep a naming convention that would make it easier to identify.

As a result of the exploratory work and as to be able to recover interesting data for the study of the behavior and the use of the prediction algorithms introduced in Booster, we proposed modifications to the database.

The list of tables added to the database during the project are the following:

- UtilisateursLogins
- boosterappstat
- boosterpredictionoptions
- usagepredictionparam
- sendstatbooster

The objectives of these new tables are explained in the following subsections.

3.1.2 Proposed new table UtilisateurLogins

Since any application can be launched without a session, the first proposed modification was to create a table named UtilisateurLogins to track the sessions of the users. This allows to determine the start and finish date of the session (moment in which the user opened and closed the web portal or connected or disconnected from the desktops client), the type of client used to start the session and finally, the IP address of the device as a way of keeping track of the different devices used by the user.

These modifications were documented in Systancia's internal reports:

- ADIS-10319 added the `UtilisateursLogins` table to the database, which allows us to record when a User started an AppliDis Sessions.

- ADIS-15618 added *LogoutDate* (end of the AppliDis Session) to UtilisateursLogins to compute the duration of the Sessions.

- ADIS-15375 added two new columns to the table `UtilisateursLogins`, $PredictedLogoutDate$, and $IspredictedSession$ to distinguish between sessions started by the user and sessions possibly started by Booster.

- ADIS-15625 added a new encoding to the column $ConectionType$, to recover the AppliDis client used to start the session.

3.1.3 Enhancement of the existing table `useruseappli`

This table contains the data related to the application launches. We noticed that the applications launched by Booster Preloader or that remained opened with Booster Postloader where not distinguished from the applications launched by the user. In particular, Booster uses an application named `keepalive` that refers to the function of keeping the session "alive". Although it was possible to identify the `keepalive` application by its application ID, we decided to add a column to encode information about which Booster mode was responsible for launching it.

These modifications were documented in internal reports:

- ADIS-12978 added the *keepalive* column that lets us distinguish applications launched by the user from applications launched by different Booster components.

- ADIS-15652 improved the *keepalive* column encoding system.

3.1.4 Other Booster tables

In order to keep track of the Booster settings chosen by the administrator of ApplDis over time. We created a set of tables (`boosterappstat`, `boosterpredictionoptions`, `usagepredictionparam`, `sendstatbooster`).

This modifications were documented in internal reports:

- ADIS-15408 added `sendstatbooster` table to save the hyperparameters of the models used by Booster Prediction.

- ADIS-15375 added `boosterappstat` table to save Booster basic statistics (e.g., the total amount of time gain due to Booster).

3.1.5 Improving the data quality

As a result of the study of the database structure and data, we found improvement opportunities that were exploited to guarantee the data quality. The application of best practices simplify the development of new algorithms and avoid bugs that could be potentially difficult to spot. For instance, we noticed that the `useruseappli` table used a date convention in which, while an application is open, the end date takes the value of start date minus one day. E.g., if an application is opened on 2016-02-09 15:50:12.520, while it is open, the end date takes the value 2016-02-08 15:50:12.520. This is not a good practice since any data analyst that is not aware of this way of encoding information could easily miss compute statistics of the end date. In this case, the use of a "non-available" value is recommended, because it avoids the use of incorrect values in the computation of statistics.

Some other minor bugs that affected the data quality were documented in internal reports and fixed:

- ABVLR-16009 showed that after closing the session from the desktop client, the logout date field was not updated.
- ABVLR-16010 showed that after closing the browser window of the web client, the logout date field was not updated.
- ABVLR-16012 showed that when connecting from web client, the machine name field was not correctly populated.

3.2 EDA of the user activity

As to gain insights about the user activity we will explore the user session logins and applications launches gathered in the `SessionLogins` and `ApplicationLaunches` tables.

Figure 3.1 shows the relationship between these two tables and the other tables of the database utilizing an entity-relationship diagram.

We notice that even though an application is always open inside a session, this is not reflected in the data model. This would be possible by adding, for instance, the foreign key *SessionId* into the `ApplicationLaunches` table. In the case of a UML Diagram, this would be reflected by a composition relationship, meaning that the application launch could not exist without a session.

Although, we can try to recover this relationship from the data, by assigning each application at the current open session, this approach will only work if there are not simultaneous open sessions for a given user (which is not the case since a user can connect to AppliDis from different devices at the same time). It is also important to notice that since the administrator of the system can erase data from the database to free memory space at any

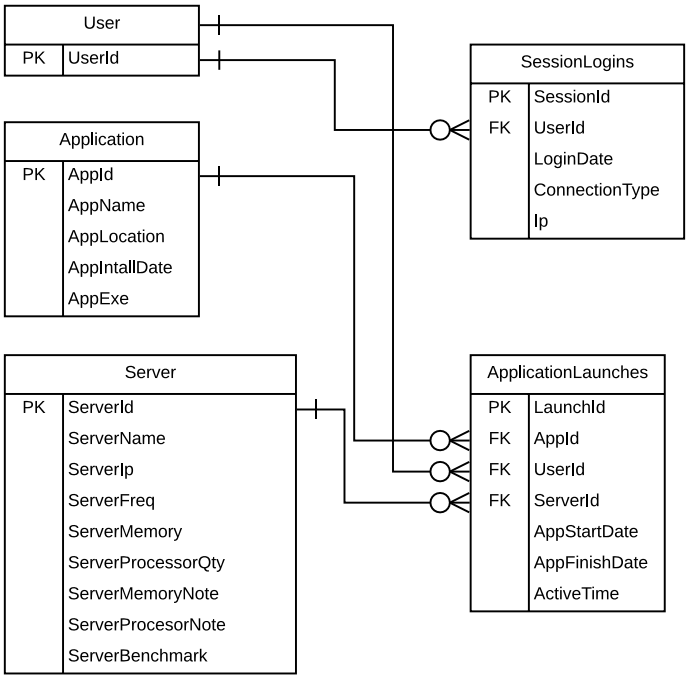


Figure 3.1 – Entity-relationship diagram of the data.

SessionId	UserId	LoginDate	ConnectionType	MachineName	Ip
15	876	2015-02-09 14:38:55.400	2	SOI8001	125.0.2.28
16	876	2015-02-09 14:39:29.890	2	SOI8001	125.0.2.28
497	159	2015-02-10 09:54:35.100	2	SOIXP216	125.0.2.48
699	876	2015-02-10 14:47:42.740	2	SOI8001	125.0.2.28
959	876	2015-02-11 08:35:14.350	2	SOI8001	125.0.2.28

Table 3.1 – Selection of instances from the `SessionLogins` table.

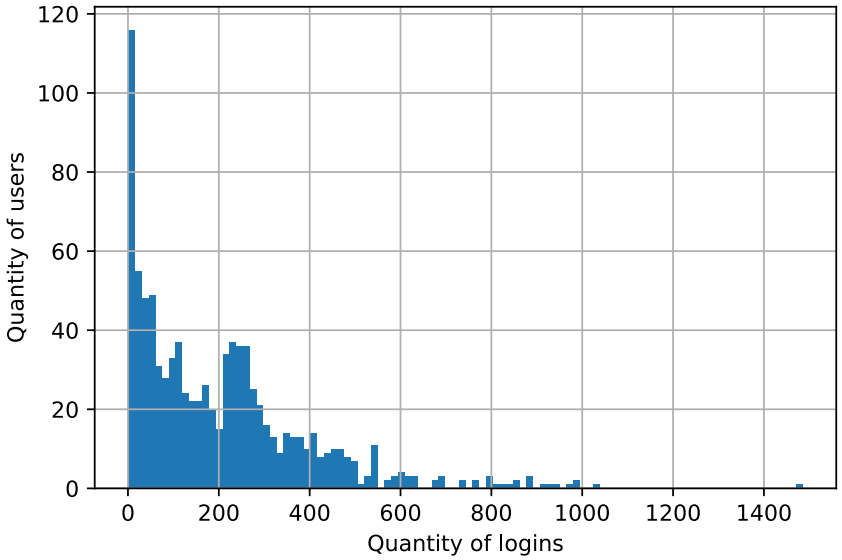


Figure 3.2 – Histogram of the quantity of logins by user.

time will make impossible to us to tell at a certain moment if the user is really not active (i.e., she has no open sessions at a given moment).

3.2.1 Session logins

In Table 3.1, we show 5 instances taken from the total of 187091 records in the `SessionLogins` table from the Hospital Center X. The data points were collected over 366 days from 2016-02-09 to 2015-02-09.

The table contains the sessions logins of 927 users. Figure 3.2 shows the histogram of the number of logins by user ($mean = 200$ and $median = 162$) and we see that it has a long tail. The user with the biggest amount of logins has a total of 1485. Since this quantity is much higher than the average (more than 7 times higher), we suppose that these user credentials are being shared by several people.

Combining the features Day, $Month$, and $Year$, we can gain some in-

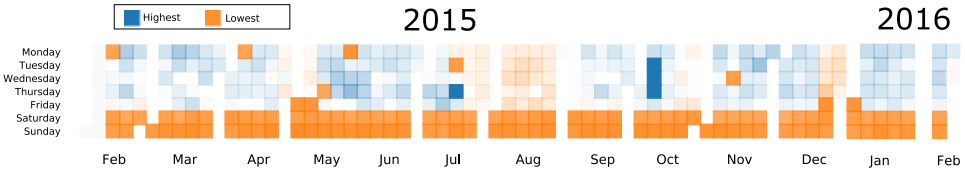


Figure 3.3 – Quantity of logins by day of the year.

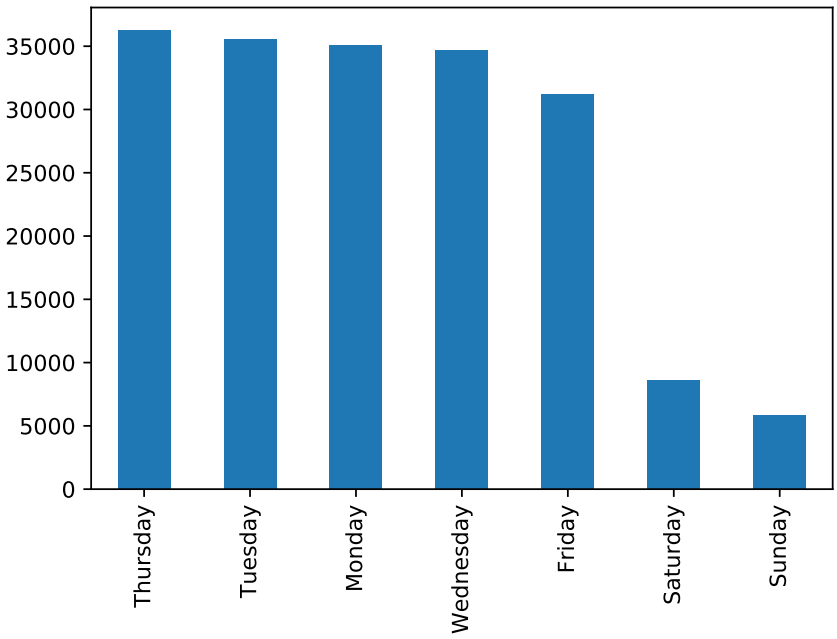


Figure 3.4 – Quantity of logins by weekday.

sights about the monthly behavior. In Figure 3.3, we observe that the busiest month is October, while August, with 30% fewer logins than October, is the month with the lowest activity level. A lower level of activity can also be observed in the last days of December. This is consistent with the holiday periods in the northern hemisphere, and we observe a similar pattern for different customers of AppliDis independently of the type of organization. We also observe specific days with low levels of activity, which are consistent with french bank holidays (e.g., 2015-11-11).

The analysis of the weekly quantity of logins in Figure 3.4 shows a clear difference between weekends and working days.

The same analysis but by hour of day, in Figure 3.5, shows that the daily behavior is bimodal, with a strong peak in the morning and the second one after midday. This bimodality seems to be present in all the days of the week, as shown in Figure 3.6, although the low-level activity of the weekend

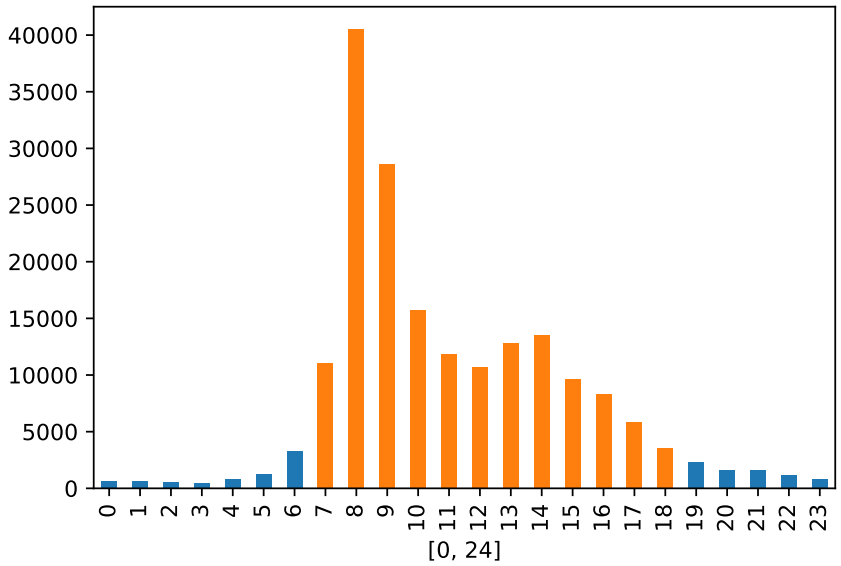


Figure 3.5 – Quantity of logins for the 24 hours of the day where common working hours are marked in orange.

does not let us verify this assumption.

As we have shown, for all the different analyzed periods (monthly, weekly, and daily), we can confirm the presence of expected behaviors. These are holiday periods (like bank, summer, Christmas, and new year's holidays), weekends, and associated customs (e.g., leaving early on Fridays), and finally, lunch breaks.

3.2.2 Application launches

Now we focus our attention on the application launches, in particular, the `ApplicationLaunches` table of two customers that belong to different industry segments. The first one is a software development company (Systancia), and the second one is the same hospital that we used for the logins analysis (Hospital Center X). Some instances for this last customer have been selected and shown in Table 3.2. Although no application can be launched without a server (to run on), we found that for Hospital Center X, about 8.9% of the applications launched have a $ServerId$ equal to None, while for Systancia all application launches have a corresponding $ServerId$ value.

To better understand the complexity of the predictive task, we investigate the amount of users, applications, and servers. The results for both customers are summarized in Table 3.3.

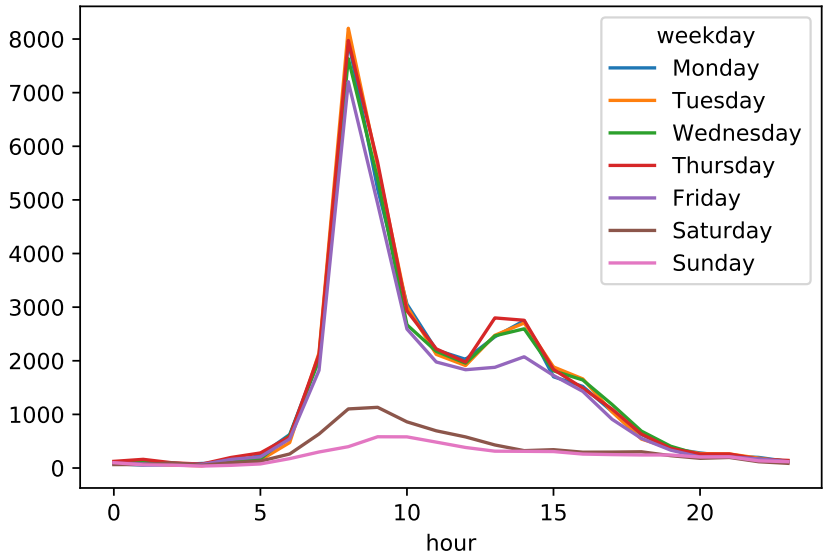


Figure 3.6 – Comparison of the logins quantity for the 7 days of the week for the Hospital Center X.

LaunchId	AppId	UserId	ServerId	AppStartDate	AppEndDate	ActiveTime
0	28	5	68	2014-10-09 14:47:50.930	2014-10-09 19:43:03.067	17713
1	188	174	43	2014-10-09 14:47:52.417	2014-10-09 15:27:17.730	2365
2	139	146	41	2014-10-09 14:48:01.587	2014-10-09 16:16:56.433	5335
3	102	899	41	2014-10-09 14:48:12.227	2014-10-09 15:06:00.733	1068
4	115	922	None	2014-10-09 14:48:14.240	2014-10-09 17:04:42.733	8188

Table 3.2 – First 5 instances the table `ApplicationLaunches` of Hospital Center X. We observe the presence of None values.

	Hospital Center X	Systancia
Quantity of Users	765	84
Quantity of Apps	108	24
Quantity of Servers	37	3
Quantity of Launches	384251	15709
Quantity of days of data	229	545

Table 3.3 – Comparison between the tables `ApplicationLaunches` of Hospital Center X and Systancia.

3.2.3 Database visualization tool

The results of the EDA of these two databases proved to be useful also for the development process of the new Booster components. Since we needed to apply it regularly to other databases, we designed and implemented a web application known as the Database Visualization Tool. For its development, different frameworks (i.e., D3 and Bootstrap) were used, while the back-end was mostly developed in Python. Finally, a set of SQL queries to generate clean views of the data were developed.

Some of the mean features of the tool are:

- Remote connectivity: that allows to connect to any remote AppliDis SQL database without the need of previously installing the AppliDis Client.
- Compatibility verification: tat allows to verify the presence of tables that have been implemented during the project and of interest for different user behavior analysis.
- Activity calendar: that allows us to visualize in a calendar format (similar to Figure 3.3) that let us quickly determine the interval of time for which data was available and detect any anomalies in the database (e.g., periods with low or no activity).
- Custom queries: that allows us to visualize the results of customs queries on the browser.

3.3 Feature engineering

Most of the data in the tables of the database correspond to values that we can measure directly (e.g., the IDs of the entities, the timestamp). Each of these measurable characteristics are known as *features*. In particular, the ones we have described are known as *raw features*, while the ones that result from transforming or combining raw features are known as *extracted features*. In a relational database, each column of a table represents a feature, and the values, in each row, are samples.

The process of extracting new features is known as *feature engineering*. This part of an ML project is crucial because:

- the performance of the model depends on the inputted features,
- some models rely on features that respect certain conditions, and
- training some models without previously precomputing features can be excessively expensive.

According to the variable data type, we can distinguish different transformations.

3.3.1 Categorical variables

Categorical or discrete variables take values from a countable set (or categories). E.g., the feature $Weekday$ can take values in the set {"Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"}). Since algorithms, in general, need to deal with numeric data types. We need to find a way of transforming these chains of characters. We notice that in our problem, most of the data we gather is categorical (e.g., $UserId$, $AppId$, $ServerId$, IP, and $MachineName$). Although in practical business problems, categorical data is ubiquitous, it is difficult to find techniques to deal with this kind of data types in the literature. Next, we discuss some techniques, with progressively higher complexity, that let transform categorical non-numeric data to numeric values.

The simplest way of achieving this last objective is by assigning integer values to each element of the set. In our example of the $Weekday$, for instance, we can assign values from 1 to 7 to each weekday. However, this assignment depends on an arbitrary decision. A completely different valid option would have been to choose values $\{5, 2, 1, 3, 10, 87, 13\}$. The problem with this representation is that, in general, algorithms will interpret the magnitude of the values as information about the relationship between the categories. In this last example, the algorithm could interpret that $Monday$ with value 5 "is half" $Friday$ with value 10.

One way of solving this issue is by using a "one-hot encoding". This is replacing each category with a vector, with as many components as categories, a value of 1 in only one component, and zero in the others. E.g., we would associate $Monday$ with the vector $(1, 0, 0, 0, 0, 0, 0)$. With this representation, each vector is independent of each other and do not share any information. The problem now is that for features with a big amount of categories our vectors would get big (e.g., for the feature "MachineName" we would need one component for each different name in the database). At the same time, if we generate our vectors, and later a new machine name appears, we would need to increment the size of all our vectors (and modify our model).

The third problem is that sometimes, the categories do have a relationship with each other that we would like to model and exploit. E.g., notice that although we can model the sequence order between the weekdays from $Monday$ to $Sunday$ by assigning values from 0 to 6, it would be impossible also to capture the fact that $Sunday$ comes before $Monday$ just with a scalar. On the other hand, we can capture this relationship by using a vector with two components, like in Figure 3.7.

While in the last example, the relationship we wanted to model was easy enough to design it by hand, in the general case, we would like our algorithms to learn these representations itself. In the Section 3.5, we will discuss some techniques that let the model learn the transformations of the raw data.

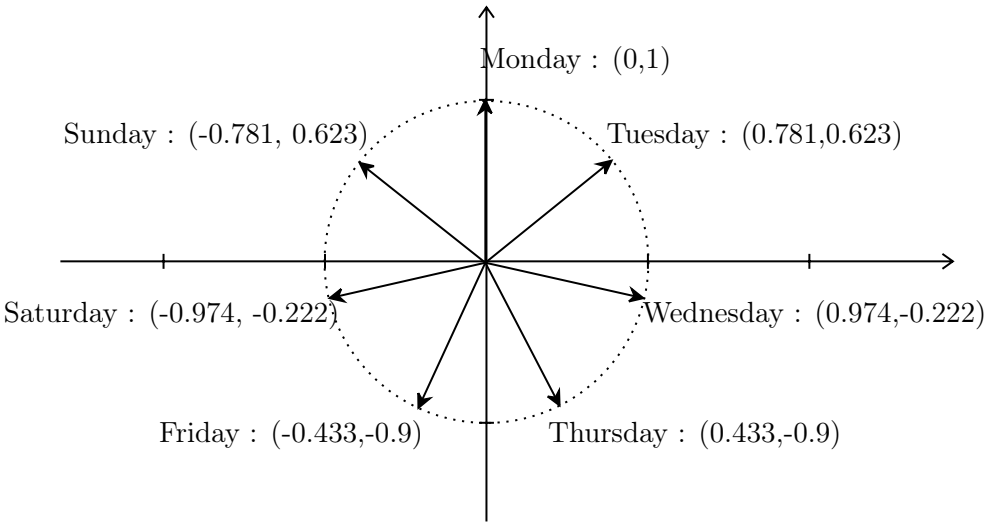


Figure 3.7 – Seven unit vectors representing the weekdays on the plane. The cyclical relationship between them is conserved as opposed if we represent them with scalar values.

year	month	day	hour	date	time	dayofyear	weekofyear	week	weekday	day_name
2015	2	9	14	2015-02-09	14:38:55.400000	40	7	7	0	Monday

Table 3.4 – Generated features after transforming the timestamp 2015-02-09 14:38:55.400.

3.3.2 Timestamps

As we have seen in the exploratory analysis, most of our data is in the form of timestamps. This is a combination of a date and a time with millisecond precision. Although there are algorithms that have the capacity to find complex relationships in the data. In practice, we notice that it may require considerable large datasets. One way to facilitate the learning task is by decomposing the timestamp in several features that capture different periodic characteristics. In Table 3.4 we show the resulting features after decomposing the timestamp 2015-02-09 14:38:55.400.

Another popular way of working with timestamp is by generating relative features like time differences calculated by measuring the distance with respect to a reference point (usually the first timestamp on the list).

3.3.3 Continuous variables

When the feature measured can be associated with a real number, we call it a continuous variable. To represent this real number, we generally use a float

data type. Since the memory and the computing power is always limited, we can only represent real numbers up to a certain precision.

Since there is an extensive literature about how to derive features [56, 96, 35] from the continuous variables and considering that in our problem this kind of data do not play a central role, we will not extend any further into the discussion of them.

3.3.4 Relative features

Sometimes, as to make the raw features useful, we need to transform them from absolute features into relative features. For instance, in our problem, in the general case, different customers will not share the same applications. Even if they do share some, the $AppId$ assigned to a common application will be usually different in each database. One way of dealing with this is, instead of counting the frequency of each $AppId$, we can first map the $AppId$ to the frequency of use. So, instead of having the frequency by $AppId$, we will have the frequency of the most used application, of the second most used application, and so on. After this transformation, we can compute the frequency. By doing so, there will be two advantages. The first one is that now, what we learn about one database can be applied to other databases, with different applications, transferring learning. The second one is that this knowledge can also be used as a prior for other models.

3.4 Entropy of the behavior

If we review the figures we got as a result of the analysis in Section 3.2, we will notice that all of them can be seen as the realization of a random variable. This random variable will have a certain associated probability distribution. For instance, if we normalize the values of Figure 3.5, we get a discrete probability distribution describing the probability of the user logging in at each of the 24 hours of the day (period $T = 24$).

Let us suppose that we observe the behavior of two users for a long interval of time and that user 1 present logins over all hours of the day and that user 2 present logins only at one hour of the day. If someone asks us to predict when these two users will log in next, we would think that it is much easier to guess the hour for user 2 correctly. This "level of predictability" linked to the concentration of the mass of the probability distribution is formally quantified by, what is known as, the *entropy* of the random variable.

Now, we will define some concepts as found in [40], that will help us to formalize our intuitions and quantify them.

3.4.1 Entropy

The *entropy* of a discrete random variable X with alphabet $\mathcal{X}$ and probability mass function $P(x)$ is defined by

$$H(X) = \sum_{x \in \mathcal{X}} [-\log P(x)]\, P(x). \tag{3.1}$$

Since $\lim_{p \to 0} -\log p = 0$, we should consider $P(x)[-\log P(x)] = 0$ when P(x)=0.

To understand why the entropy defined like this will help measure the concentration of the probability mass, let first focus on the $-\log(y)$. In particular, since our distribution is discrete $P(x)$ can only take values in the interval $[0, 1]$, so the range of $-\log(y)$ will be $[0, \infty)$, in particular, it will take the value 0 only if $P(x) = 1$ and a take values closer to ∞ as it gets far away from 1. Finally, we see that we apply a weighted sum over all the possible values of x. So in the extreme case of only one value of x having $P(x) = 1$, the entropy will be 0. On the opposite case, if we distribution is uniform (all values of x with equal probability), we will have the greatest value of entropy. Finally, since we are summing positive numbers, the entropy will always be positive or zero.

The reader may also notice that we do not specify the base of the log. The reason is that in our application, the result does not depend on the base of the log, although we will usually use the natural logarithm. For completion, when the base of the log is 2, the entropy is expressed in bits, and when it is e, the entropy is expressed in nats.

Now we define the *differential entropy*,

$$h(X) = -\int f(x)\, log\, f(x) dx, \tag{3.2}$$

that we will use when dealing with a continuous random variable X with probability density function f.

In this case, we can use the same intuition as for the entropy, although now, since the density can take values greater than 1, the differential entropy can take negative values.

All the following definitions will be expressed for the case of a discrete random variable. Since we only need to replace the sums by integrals to generate the corresponding definition for a continuous random variable.

If we measure the entropy of the conditional probability distribution $P(X|Y)$ of a pair of discrete random variables (X, Y) with joint distribution $P(X, Y)$, we get the *conditional entropy*

$$H(X|Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y)\, \log P(x|y) \tag{3.3}$$

and, by the same logic, we define the joint entropy $H(X, Y)$ as

$$H(X, Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) \, log \, P(x, y). \tag{3.4}$$

Since by the product rule $P(X, Y) = P(X|Y) \, P(Y)$, we can show that

$$H(X, Y) = H(X|Y) \, H(Y). \tag{3.5}$$

If we think of entropy as the information quantity required to describe a random variable, we can interpret Eq. 3.5 as showing that the information we need to describe X and Y is equal to the information to describe X given Y plus the information to describe Y [16].

A last relevant quantity is the *cross-entropy* defined as

$$H(P||Q) = -\sum_{x \in \mathcal{X}} P(x) \log Q(x) \tag{3.6}$$

Kullback-Leibler Divergence (relative entropy)

Let us now assume that we have two distributions P and Q defined over the same domain. We can use the concept of entropy to generate a measure of "distance" between these two distributions. E.g., in our case, we could be interested in generating a notion of similarity between the probability distribution of launching the different applications available to two different users.

The Kullback-Leiber Divergence (KLD) is defined as

$$KLD(P\|Q) = \sum_{x \in \mathcal{X}} P(x) \log P(x) - \sum_{x \in \mathcal{X}} P(x) \log Q(x) \tag{3.7}$$

$$= \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)}, \tag{3.8}$$

The relative entropy $KLD(P||Q)$ is a "measure" of the inefficiency of assuming that the distribution is Q when the true distribution is P.

Jensen-Shannon Divergence

The KLD is not a metric. For instance, it is not symmetric ($KLD(P\|Q) \neq KLD(Q\|P)$). By defining a new quantity known as the Jensen-Shannon Divergence (JSD)

$$JSD(P\|Q) = \frac{1}{2}KLD(P\|M) + \frac{1}{2}KLD(M\|P) \tag{3.9}$$
$$= JSD(Q\|P), \tag{3.10}$$

where $M = \frac{1}{2}(P + Q)$, we can respect the symmetry condition. Regrettably, we will still be missing a last condition (the triangle inequality). To respect this last condition, we can use the squared root of the JSD, as suggested in [45].

3.4.2 Mutual information

Consider two random variables X and Y with a joint probability mass function $P(x, y)$ and marginal probability mass function $P(x)$ and $P(y)$. The mutual information $I(X;Y)$ is the relative entropy between the joint distribution and the product distribution $P(x)P(y)$:

$$I(X;Y) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} P(x,y)\, log\frac{P(x,y)}{P(x)P(y)} = KLD(P(x,y)||P(x)P(y)) \tag{3.11}$$

The relative entropy is the "extra" entropy that we will have because we use Q instead of P (is relative because it is not the total entropy but the extra after the level of entropy of P). So the total entropy or "inefficiency" will be $H(P) + KLD(P||Q)$.

3.4.3 Behavior entropy analysis

We applied these concepts to the analysis of the user behavior by building a tool to estimate the (differential) entropy of the user activity for different intervals of time T. To do so, we use a simple (naïve) technique. In the first stage, we estimate the probability density, and then we estimate the differential entropy from the estimated density.

Figure 3.8 shows the results of these estimations. We see that the entropy is strongly related to the periodicity of the behavior. This is because points will tend to align for the true period of the behavior, generating distributions with lower entropy. Figure 3.9 shows a comparison between an arbitrary value of T and $T \approx 24$, which is the period that produces the minimal differential entropy for this particular user. It is also important to notice that this is the case because we are normalizing to 1 the length of all periods to make them comparable. Without this normalization, as the period increases, the domain of the distribution would also grow.

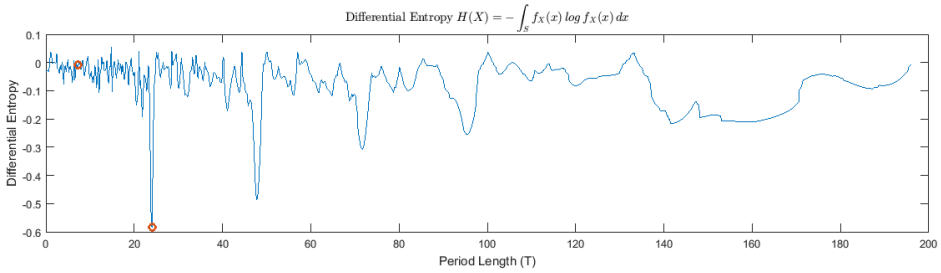


Figure 3.8 – Entropy analysis. Estimated differential entropy of the real user behavior (session logins) for different periods. The orange circles mark the value for $T \approx 7.35$ hs and $T \approx 24$ that which are compared in Figure 3.9

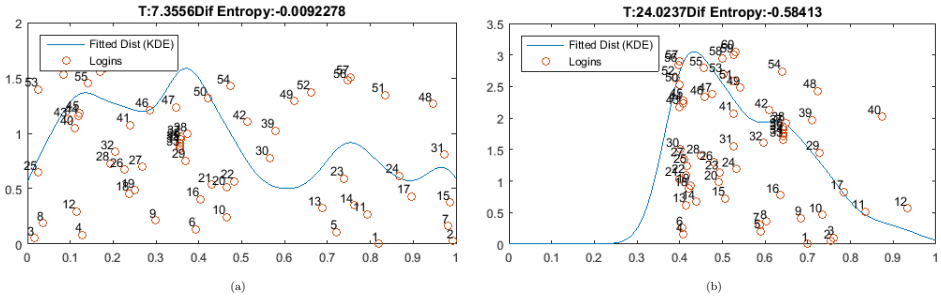


Figure 3.9 – Comparison of the results for (a) $T \approx 7.35$ hs and (b) $T \approx 24$ hs which is the period with minimal differential entropy. The numbered orange circles represent the logins as they appear in the dataset and the blue line represents the estimated density.

AppId	4	8	10	13	14	15	17	18	19	21	23	25	26	27	28
Count	5.0	1.0	1.0	2.0	5.0	1.0	8.0	1.0	9.0	16.0	16.0	17.0	134.0	7.0	71.0

Table 3.5 – Count of launches by application for user 2 from Systancia.

3.5 Learning representations and user profiling

While in the previous section, we focused on hand-crafted transformations, in this section, we will explore techniques that will let the algorithm learn the representations. An in-depth review of different techniques can be found in [13]. We will show how this learned representations will help us with the task of profiling users. This is, to use features of the users (mostly related to their behavior) to identify subgroups of users.

Let us start with a concrete example of the kind of behavioral vectors that we can associate to a user. First, we need a criterion to define which aspect of the behavior we want to focus on. So let us focus on the role of the user in the organization. In the case of a hospital, this would mean to be able to group physicians, administrative staff, and other roles. If we assume that the set of applications a person use are linked with their function in the organization (e.i., physicians would use specific applications for their function or will follow specific patterns when using them) we can define the behavioral vector of the user, for instance, as the quantity of times she use a certain application on a given period of time (like we show in Table 3.5).

Of course, the type and complexity of the features that compose the behavioral vector will allow us to capture different levels of relationship between the user.

3.5.1 Eigenbehaviors

Let us assume we have a dataset $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ composed by column vectors of D components with zero mean (without lack of generality since we can always subtract the mean vector $\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$ to each vector). We can think of the dataset as a cloud of points in a D-dimensional space, where each one of the D components of a data point represent the magnitude of a feature for that point.

In general, it is interesting to know the covariance between the features. So if we want to compute the covariance matrix of the features, we can first construct a matrix $\mathbf{X}$ by "stacking" the transposed N column vectors in a way that the ith row of the matrix contains the ith vector (this matrix is known as the "design matrix") and compute the covariance as

$$\mathbf{S} = \frac{1}{N}\mathbf{X}^T\mathbf{X} \tag{3.12}$$

Although the covariance matrix is interesting in itself, it can also be used to find projections of data in spaces of dimension M (with $M < D$). This is because the directions of the greatest variance will be the ones with the smaller error when projecting the data, and we will be keeping as much of the original structure (in the sense of the covariance) as possible. So now we try to find the direction of the greatest covariance

$$\mathbf{u}_1 = \arg\max_{\|\mathbf{u}\|=1} \mathbf{u}^T \mathbf{S} \mathbf{u}. \tag{3.13}$$

Solving this constrained optimization problem (introducing a Lagrange multiplier λ), we find that $\mathbf{u}_1$ should satisfy

$$\mathbf{S}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1 \tag{3.14}$$

what indicates that $\mathbf{u}_1$ should be an eigenvector of $\mathbf{S}$, and since $\mathbf{u}^T\mathbf{S}\mathbf{u}$ is the covariance of the direction $\mathbf{u}$, to choose the one with the greatest covariance, we will choose the eigenvector with the greatest eigenvalue.

If we proceed and compute all the eigenvectors and eigenvalues, we will be able to decompose S as

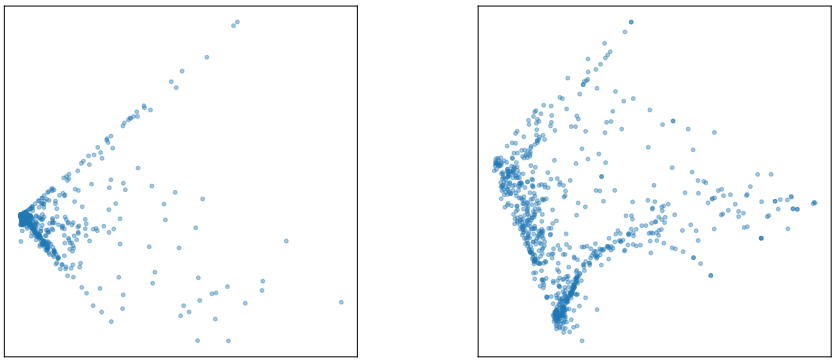
$$\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T \tag{3.15}$$

where $\mathbf{U}$ is the matrix which columns are the eigenvectors, and Λ is a diagonal matrix which values are their corresponding eigenvalues. This decomposition is known as Principal Component Analysis (PCA) [70].

Although this is a general technique for matrix decomposition when the design matrix is constructed by combining vectors that encode some aspect of the user behavior, we obtain a tool for behavior analysis. For instance, a design matrix composed by vectors of 24 components representing the hours of a day (and a binary value at each component representing if the user was present at that location) was used in [44] to obtain the "eigenbehaviors" of the users (principal components of the design matrix). A similar analysis for behavior prediction can be found in [42].

Group behavior

Instead of constructing the design matrix with samples of the behavior of the user over time, another possibility is to consider a mean behavior for each user and sample different users. By doing so, we get the eigenbehavior of the group. By measuring the distance between the behavior of a new user and the eigenbehaviors associated with each group, we can classify them.



(a) (b)

Figure 3.10 – Comparison of PCA results. (a) PCA applied to the matrix of distributions (b) PCA applied to the distance matrix

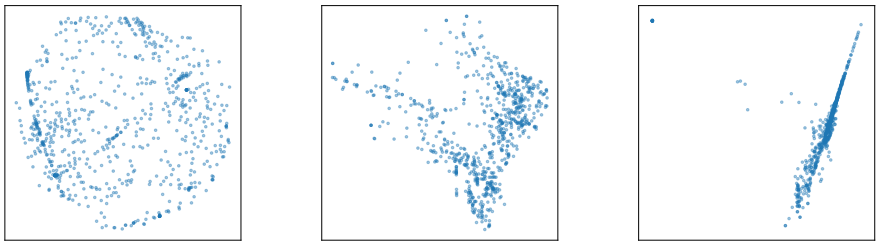
3.5.2 User profiling

We will compare two different embeddings, one embedding generated by normalizing the vector of counts of the application launches, as shown in Table 3.5. This is equivalent to computing the parameters of a categorical distribution over the applications by maximum likelihood. As a result, for each user, we will have one distribution over the set of applications. A second embedding, will result from using the square root of the JSD (described in 3.4.1) as a metric to measure the distance between the distribution associated with each user and generate a distance matrix. A square matrix $\mathbf{D} = (d_{ij})$ where each $d_{ij} = \sqrt{JSD(P_i \| P_j)}$ and size U equal to the quantity of users.

Although we can cluster the users in the original space, we would like to generate visualizations that could be presented to the administrator of the system. In the rest of this section, all the visualizations will be based on the Hospital Center X dataset unless specified otherwise [27].

We applied the PCA technique, as explained in Sec. 3.10a. We noticed that although the result when using the distance matrix (Figure 3.10b) is better than the one obtained by using the distributions (Figure3.10a), neither of them let us clearly group users. Since PCA is a linear technique, we explore non-linear techniques to try to improve our results.

Different manifold learning techniques will give priority to different characteristics of the original structure. Isomaps [125] (see Figure 3.11b) tempts retain as much global structure as possible by using the geodesic distance while Local Linear Embedding (LLE) [111] (see Figure 3.11c) tries to keep the local structure. Finally, when we try Multidimensional Scaling (MDS)



(a) (b) (c)

Figure 3.11 – Comparison of the results of (a) MDS, (b) Isomap and (c) LLE applied to the distance matrix.

[41], we observe a well-known phenomenon of getting a "ball" where all points are uniformly spread in the plane.

A technique proposed by [133], the t-Distributed Stochastic Neighbor Embedding (t-SNE), tries to tackle this problem. We test this approach, and we get to generate clearly defined groups, as shown in Figure 3.12. Although, we see that the results we get for the distance matrix (Figure 3.12b) are superior.

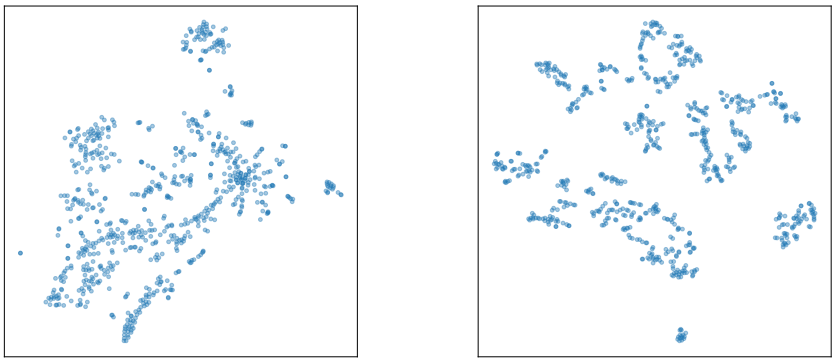
The final result of the t-SNE technique strongly depends on the hyperparameters of the model (perplexity, early exaggeration, and number of iterations). In Figure 3.13, we can see the effect of different values of the perplexity.

3.5.3 Clustering

There are different techniques that we can use to associate each point to a group. We will start by exploring the most common one, the k-means algorithm [27]. We show our result in Figure 3.14a, we see that since k-means work by associating points with respect to the distance to the centroid of the cluster, it can produce results that will not follow our intuition. In particular, we can see that effect for the yellow cluster in Figure 3.14a. To avoid this problem, we can use spectral clustering [98], which construct an adjacency graph. We see in Figure 3.14b, that the problem of the yellow cluster is solved. But we noticed that the final result strongly depends on the initialization.

In both of these techniques, we need to specify a priori the number of clusters. A possible solution to avoid the need of administrator defining this parameter consist of using a nonparametric approach like a Dirichlet process [97].

In Figure 3.14, we use the methods discussed above for the Systancia database, which have less than one hundred users. Now we can clearly display



(a) (b)

Figure 3.12 – Comparison of t-SNE results. (a) t-SNE applied to the matrix of distributions (b) t-SNE applied to the distance matrix.

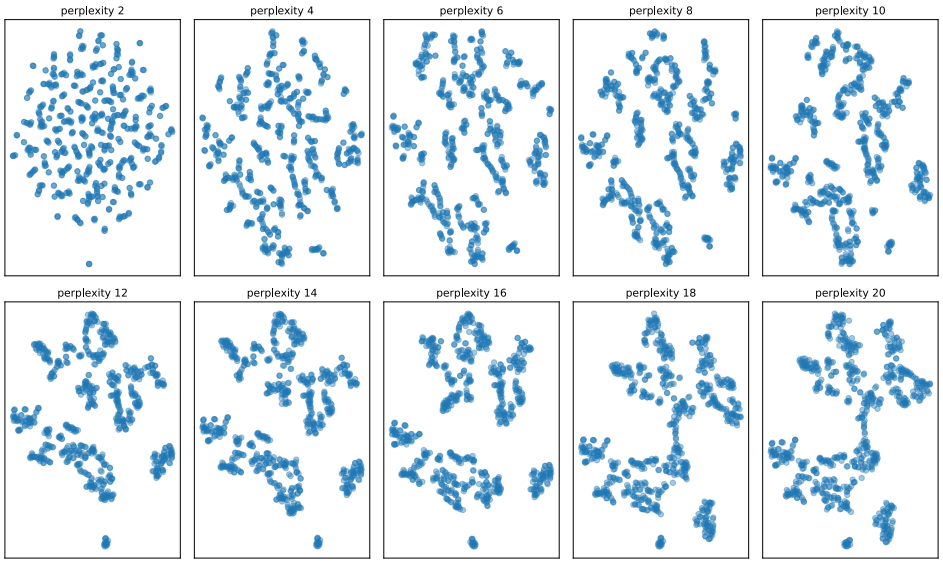
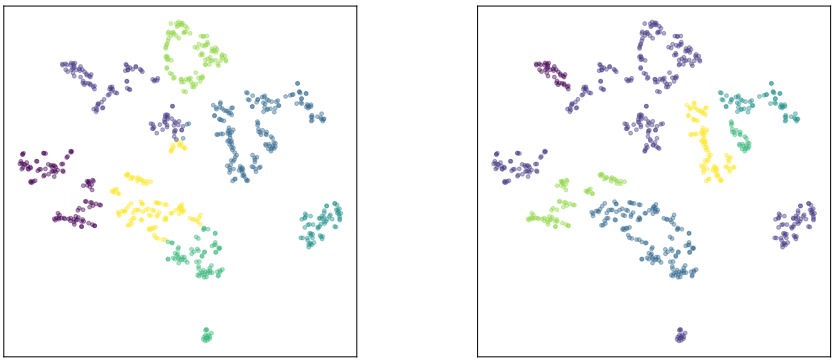


Figure 3.13 – Effect of the perplexity for t-SNE applied on the distance matrix.



(a) (b)

Figure 3.14 – Comparison of the clusters obtained by (a) k-means and (b) spectral clustering.

User	55	3	112	79
55	0	1.446	0.499	1.411
3	1.446	0	1.207	0.493
112	0.499	1.207	0	1.473
79	1.411	0.493	1.473	0

Table 3.6 – Distances between selected users of Systancia

the IDs for each user. To explore the difference between points depending on their location we will consider two arbitrary distant pairs of close points in the plane, users 55 and 112 which are close in the upper part of the Figure 3.14 and users 3 and 79 which are close in the left bottom corner.

In Figure 3.16, we can compare the distributions for each of the users. We see that the closeness between user 55 and 112 is mostly due to the application 28, which is the most launched application for both of them. While, users 3 and 79 share in common the use of applications $6, 7, 21$ and 23. In the reduced distance matrix values (Table 3.6) we can see how the fact that users 3 and 79 do not share any common application with user 55 and that user 3 and 112 do share in common application 16 is capture by assigning to the first two relationships similar values of about 1.4 while for the last one it assign a smaller value of 1.2.

Finally, to get a better understanding of the distance between them, we show in Table 3.6 the resulting distance matrix.

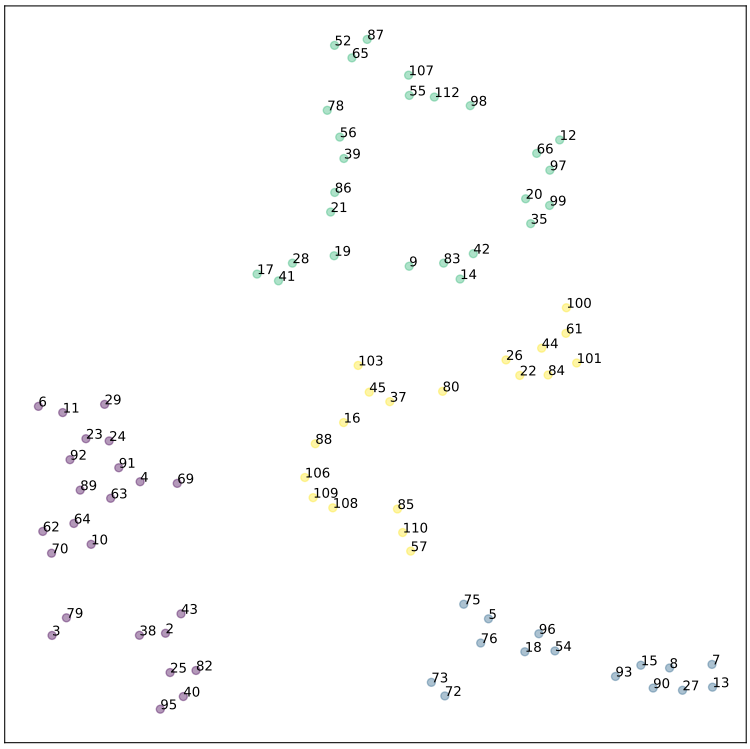


Figure 3.15 – Resulting clusters after applying t-SNE and k-means on Systancia's matrix of distributions.

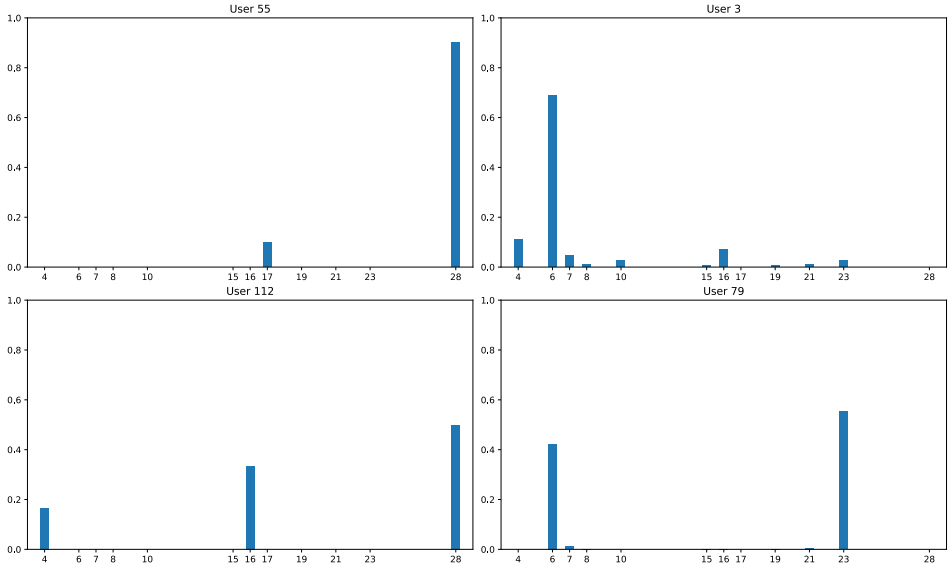


Figure 3.16 – Comparison between the distributions of different users

3.6 Discussion

It is known that the results of the k-means algorithm depend on the initial conditions. The problem with this approach is that it is sensitive to outliers. The final partition is not always optimal.

To evaluate the clustering, an objective function that measures its quality is required. Most of them try to optimize the inter and intra-cluster distances, for instance, by computing some metric like the quantization error, Davies–Bouldin index, Silhouette index, or other clustering indices [148, 9].

3.7 Conclusion

Any software product developed along several years by different people is susceptible to presenting data quality issues or inefficiencies. The detailed scrutiny of the databases and the data itself in a data science project generate an opportunity for improvement. After noticing that our problem shows a particularly intensive use of categorical data, we developed different methods for exploiting it. Although some of them seem to be extensively applied in the industry, they do not seem to have been study in depth in academia, hence the relevance of this analysis. Finally, in section 3.5, we have shown how to combine general ML methods to solve the problem of generating profiles of users based on behavioral characteristics.

Chapter 4

User modeling and behavior prediction in cloud computing architectures

4.1 Introduction

Opening sessions and remote applications implies to load a non-negligible amount of data, parameters, and services. Therefore, desktop and application virtualization suffers from delay. The launching time of the applications can be reduced by predicting the future activity of the users and loading the required resources in advance.

In this chapter, the modeling of the user behavior will be framed as a statistical decision theory [108] problem which main component is the density estimation problem. Under this framework, the distribution of the session and application launches are estimated from recorded data. The estimated distribution is then used to decide where to establish the intervals in which we will execute an action (launching sessions or applications).

Implemented in a cloud computing product (AppliDis), this solution has been deployed to production for customers (thousands of users, hundreds of applications, tens of servers) around the world and tested over an interval of time bigger than one year.

This chapter is organized as follows. Section 4.2 describes the essential elements of statistical decision framework that support our methods. Section 4.3 shows how the user model can be used for predicting user activity, present different use modes of the proposed solution, and proposes a method to find an interval of time that captures most of the structure of the recurrent patterns of the behavior. Section 4.4 presents different approaches to modeling the PDF that describe the user behavior and propose a method to improve the modeling results when only small datasets are available. In Section 4.5, the performance of the proposed system is evaluated and a way of comparing

different behavior prediction algorithms is proposed. Section 4.6 describes the implementation of the system and the necessary GUI before being deployed in production. Finally, Section 4.7 describes some preliminary results of applying deep learning techniques to our problem.

4.2 User modeling and statistical decision theory

In the context of session opening and remote application launching, we define *user behavior* as a combination of regular activity patterns over time that will generally present a certain stochasticity. For instance, for a user who works from Monday to Friday from 8 a.m. to 5 p.m., we expect to see some opening of the sessions on Monday around 8 a.m. with a certain variability related to different events that affect her punctuality. A probability model can be used to capture this random phenomenon, and the activity data will allow us to estimate the parameters generating the *user model*.

UM can be framed as a statistical decision problem where the objective is to infer the distribution of the activity from data. This is known as inference.

We can differentiate two approaches based on their interpretation of the probabilities, the parameters, and the data.

In a frequentist approach, the data $\mathcal{D} = \{X_1, \ldots, X_N\}$ is a random sample drawn from a distribution indexed by an unknown fixed parameter θ. We infer θ by looking for the value that maximizes the plausibility of the data.

$$\hat{\theta} = \arg\max_{\theta} P(\mathcal{D}|\theta). \tag{4.1}$$

While in a Bayesian approach, the unknown parameter θ is a random variable, which distribution (called prior) reflects a subjective degree of belief of the parameter having a certain value. Utilizing Bayes' rule, this subjective distribution is updated given the data (producing a posterior distribution).

$$P(\theta|\mathcal{D}) = \frac{P(\mathcal{D}|\theta)P(\theta)}{P(\mathcal{D})}. \tag{4.2}$$

4.2.1 Statistical experiment

A statistical experiment is a family of probability distributions $\mathcal{P} = \{P_\theta : \theta \in \Theta\}$ parameterized by θ that belongs to the set of possible parameters Θ. The selection of this family of distributions is based on assumptions. We say that the family is flexible if it includes a great number of distribution, and we assume that it will be big enough to contain the true distribution (or at least a good enough approximation). As we will see later, there are also drawbacks to the selection of a family of distributions that is too big.

4.2.2 Objective

The objective of a statistical decision theory problem is what we are trying to infer. In our case,

$$T : \Theta \to \mathcal{Y}, \theta \mapsto T(\theta) \tag{4.3}$$

will be the identity function, because we are interested in the value of θ. However, we could be interested in estimating any other function of θ, e.g., some norm of it.

4.2.3 Decision rule (estimator)

The decision rule or *estimator*

$$\delta : \mathcal{X} \to \hat{\mathcal{Y}}, X \mapsto \delta(X) \tag{4.4}$$

is a function of the sample. The range of this function does not need to be equal to the range of our objective function.

4.2.4 Loss function

The loss function

$$L : \mathcal{Y} \times \hat{\mathcal{Y}} \to \mathbb{R}, (T(\theta), \delta(X)) \mapsto L(T(\theta), \delta(X)) \tag{4.5}$$

will encode a notion of cost associated to the estimation made by the decision rule.

4.2.5 Risk function

Finally the risk function

$$R\{T(\theta), \delta(X)\} = \mathbb{E}_\theta \left[L\{T(\theta), \delta(X)\}\right] \tag{4.6}$$

indicate the global cost we pay for choosing a decision rule δ.

4.2.6 Evaluating the estimator

Consistency

One of the first intuitive requirements for an estimator would be that the estimate improves as the sample size increases. This notion is captured by the *consistency*. As with any random variable, we can also define consistency for an estimator. An estimator is *weakly-pointwise consistent* if $\hat{p}(x) \to p(x)$ in probability for every $x \in \mathbb{R}$ and *strogly-pointwise consistent* if $\hat{p}(x) \to p(x)$ almost surely.

Bias of an estimator

If we consider the frequentist approach, the estimator is a random variable that depends on the realizations of the data, and the uncertainty is embedded in the distribution of this random variable. The first thing we could ask ourselves is which is the accuracy of this estimator, this is, if we repeat the process of applying the estimator to a sample, what would be the difference between the mean value of the estimator and the parameter we want to estimate. So, the *bias* of a point estimator δ of a parameter θ is defined as

$$Bias_\theta \delta = \mathbb{E}_\theta(\delta) - \theta \tag{4.7}$$

So in the case of the estimator returning a distribution, we can use this same definition in a point-wise fashion and we say that $\hat{p}(\boldsymbol{x})$ is an *unbiassed* estimator of $p(\boldsymbol{x})$ if for all $\boldsymbol{x} \in \mathbb{R}^N$, $\mathbb{E}_p \hat{p}(\boldsymbol{x}) = p(\boldsymbol{x})$.

Variance of an estimator

The variance,

$$Var(\delta) = \mathbb{E}_\theta\{(\mathbb{E}_\theta \delta - \delta)^2\}, \tag{4.8}$$

will give the precision of the estimator.

Bias-variance trade-off

As we have seen before, we can use a loss function to establish a notion of "distance" between the estimate (given by the estimator) and the true probability density (which is unknown).

First we define in which space our densities will live. We will work with the space

$$L^2(a,b) := \left\{ f :]a,b[\to \mathbb{R} | \int_a^b |f(x)|^2 dx < \infty \right\}. \tag{4.9}$$

This means that our densities will be square-integrable functions. When equipped with the inner product

$$\langle f, g \rangle = \int_a^b f(x)g(x)dx,\ f, g \in L^2(a,b), \tag{4.10}$$

$L^2(a,b)$ is a Hilbert space with the associated $L^2 - norm$ defined as:

$$\|f\|_{L^2(a,b)} = \sqrt{\int_a^b |f(x)|^2 dx},\ f \in L^2(a,b). \tag{4.11}$$

Now we can use this norm to measure the distance between the estimate and the true density. The square root of the $L^2 - norm$ is the Integrated Squared Error (ISE):

$$ISE(\hat{p}, p) = \int_a^b \left[\hat{p}(x) - p(x)\right]^2 dx. \tag{4.12}$$

But the ISE will be a random variable that will depend on the sample we use to compute the estimate $\hat{p}$, so in order to eliminate that dependence we can compute the expected value of the ISE an get the Mean Integrated Squared Error ($MISE$) defined as:

$$MISE(\hat{p}, p) = \mathbb{E}\left[\int_a^b \left[\hat{p}(x) - p(x)\right]^2 dx\right] \tag{4.13}$$

Another possibility is to measure the expected squared error at each point, which leads to the Mean Squared Error (MSE):

$$MSE(\hat{p}, p, x) = \mathbb{E}\left[\hat{p}(x) - p(x)\right]^2 = var\left[\hat{p}(x)\right] + \left\{bias[\hat{p}(x)]\right\}^2 \tag{4.14}$$

$$var\{\hat{p}\} = \mathbb{E}_p\left[\hat{p}(x) - \mathbb{E}_p\left\{\hat{p}(x)\right\}\right]^2 \tag{4.15}$$

$$bias\{\hat{p}(x)\} = \mathbb{E}\left[\hat{p}(x)\right] - p(x) \tag{4.16}$$

If $MSE(x) \to 0$ for all $x \in \mathbb{R}$ as $n \to \infty$ then $\hat{p}$ is said to be a point-wise consistent estimator of p in quadratic mean.

If we integrate the MSE, we get

$$IMSE(\hat{p}, p) = \int_a^b \mathbb{E}\left[\hat{p}(x) - p(x)\right]^2 dx \tag{4.17}$$

where thanks to Fubini's theorem [127] we can show that $MISE = IMSE$.

4.3 Behavior prediction

Behavior prediction, in this context, means to be able to use the UM (we obtain thanks to the process defined in Section 4.2) to indicate a time interval (or several intervals) in which the user will be active with a certain probability. In Figure 4.1, we show that to generate the prediction, we need to combine the model with some criteria that will allow to define the starting and ending points of such intervals. In this section, we will define these criteria.

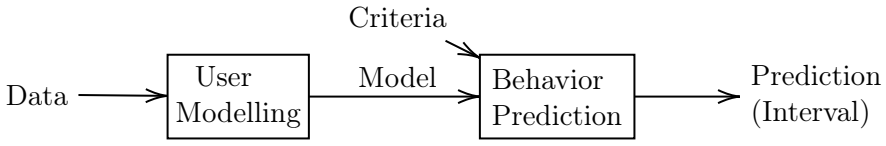


Figure 4.1 – Schema of the use of the UM for Prediction. User data is processed to find a model of the activity which is then combined with some criteria to produce a prediction.

Table 4.1 – Applications data.

x	$P(x)$
0	0.05
1	0.05
2	0.3
3	0.2
4	0.4

4.3.1 Prediction: Linking probabilities with intervals

The UMs that we will generate will be represented either by a discrete or a continuous probability distribution. One possible criterion, to choose the interval that will cover a certain amount of probability mass m, could be to choose a central interval which is defined as the interval that left two tails of mass $m/2$ at each side of the distribution. The problem with central intervals is that for certain distribution, it is possible to find smaller intervals that cover the same amount of probability mass m. Since in our case, the size of the intervals is associated to the use of resources (a session or an application in idle state waiting for the user to connect) we want to define a criterion that will guarantee that this interval is the smallest for the given probability mass m.

Intervals for the discrete model

For the case of a discrete distribution, we will explain the mechanism that will let us find the smallest interval that covers at least a probability mass m with the following example. Let us assume that we have the discrete distribution as defined in Table 4.1 over a finite domain.

We can generate a mapping as shown in Table 4.2 between probabilities and intervals by first ordering Table 4.1 in descending order by $P(x)$ and by x and then progressively adding the different values of x to an empty set (note, that since we also order in descending order by x we first added 1 to the set and then 0).

We see that in the discrete case, this method only offers a finite amount of intervals that cover specific amounts of probability mass m (in this case,

Table 4.2

$P(\mathcal{I})$	$\mathcal{I}$
0	$\{\}$
0.4	$\{4\}$
0.7	$\{4, 2\}$
0.9	$\{4, 2, 3\}$
0.95	$\{4, 2, 3, 1\}$
1	$\{4, 2, 3, 1, 0\}$

$0, 0.4, 0.7, 0.9, 0.95$ and 1). To allow our model to choose between the continuous of values in $(0, 1)$ we need to consider what happens when we select intervals over time. This aspect of the problem will be covered later in this chapter, but we introduce the notion of the solution also with an example. To solve this problem, we need to introduce the idea of randomization. Let us assume that we want to fix $m = 0.8$. In that case, a solution is to launch 50% of the time the interval that corresponds with a probability mass of 0.7 ($\{4, 2\}$) and 50% of the time the interval that correspond with a probability mass of 0.9 ($\{4, 2, 3\}$).

Intervals for continuous model

In the case of a continuous distribution, applying the same criterion of choosing the smallest interval that cover the required amount of probability mass m becomes, some how, easier to state. Starting by the user model in the form of a PDF (represented by the yellow curve in Figure 4.2), we can use a threshold $0 \leq T_3 < \infty$ to define a *prediction interval*,

$$\mathcal{I} = \{\theta : \hat{p}_\sigma(\theta) \geq T_3\}, \tag{4.18}$$

where we see that T_3 let us control the *interval size* (s):

$$s = S(T_3) = \int_{\mathcal{I}} \frac{1}{T}\, d\theta \tag{4.19}$$

and the *probability mass* (m) associated with the prediction interval:

$$m = M(T_3) = \int_{\mathcal{I}} \hat{p}_\sigma(\theta)\, d\theta. \tag{4.20}$$

With this two functions we can define a parametric curve

$$C\colon \begin{cases} s = S(T_3) \\ m = M(T_3) \end{cases} ; 0 \leq T_3 \leq \max(\hat{p}_\sigma), \tag{4.21}$$

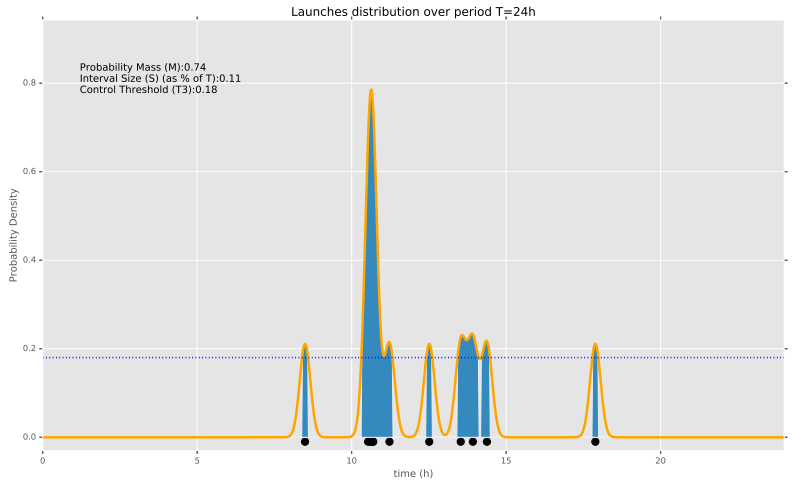


Figure 4.2 – The resulting PDF (orange curve) for an office user for a periodicity $T = 24$. This was inferred with the first 12 application launches. The threshold T_3 (blue dashed line), the probability mass m (integral of the blue region), the effect of the bandwidth h, the interval $\mathcal{I}$ and its size s.

where each point will be associated with an interval $\mathcal{I}$ of size s that will cover a probability mass m.

Considering that m is the probability of the user being active inside the interval (according to the user model), the first thing to notice is that by defining $\mathcal{I}$ as in Eq. 4.18, for a certain probability mass m we are minimizing the interval size s. This means that $\mathcal{I}$ will be the smallest interval that will let us cover m percentage of the user activity, and its size will be s.

Noticing that the range of S and M is $[0, 1]$ we can consider two extreme cases: if $T_3 = 0 \Rightarrow s = 1, m = 1$ and the prediction interval $\mathcal{I} = [0, T]$. On the other hand, if $T_3 = \max(\hat{p}_\sigma) \Rightarrow s = 0, m = 0$ and the prediction interval $\mathcal{I} = \emptyset$.

Since the intervals are associated with the cost of the utilization of resources (like CPU and RAM) to preload a session or an application and covering a certain amount of the user activity is related with the savings of the time gain. We see that there is a trade-off between resource utilization and time saving.

Also, the reader who is familiar with the Bayesian approach to statistics will recognize that our criterion is equivalent to the method for the selection of the Highest Posterior Density (HPD) region [97].

4.3.2 Prediction: Use modes

The pipeline we have described applies to individual users. When we consider the whole set of users of an AppliDis' system installed at a given customer we can think about three different use modes [28]:

- A first use mode, where the administrator of the system wants to set the expected value of the covered probability of the whole system to a certain value, in this case, we can minimize the sum of all the interval sizes.
- A second one, where the administrator wants to use a certain amount of the idle computing power of the whole system.
- A third one, based on finding the optimal balance between the last two modes. To choose T_3 we can solve an optimization problem with cost function:

$$J(T_3) = \alpha_1 S(T_3) - \alpha_2 M(T_3), \tag{4.22}$$

  where α_1 and α_2 are positive constants that quantify the willingness of trading size of the interval for probability of covering the launch of a session or application. In particular, α_1 can be related to the computational cost (CPU and RAM) of keeping a session or application idle, and α_2 can be related to the cost associated to the waiting time of the user.

Minimizing eq. 4.22 analytically,

$$r = \frac{\alpha_1}{\alpha_2} = \frac{M'(T_3)}{S'(T_3)} \tag{4.23}$$

shows that the minimum of the cost function will be at the point where the ratio between α_1 and α_2 equals the ratio between the first derivatives of M and S with respect to T_3.

On a more general framework, we can define the cost function J to be a non-linear function of s and m.

4.3.3 Applying the proposed pipeline: Controlling the average application start-up time

Although the parameters m and s allow to control the system in a very flexible way and for different use modes (as shown in Section 4.3.2), they are not intuitive for the administrator of the system. Much more intuitive quantities are the resource consumption (like CPU and RAM consumption)

and the time gain. This is why, in this section, we present the link between these quantities first considering just an individual user, and then we discuss an approach to solve the problem for a group of users.

The application start-up time is the time between the moment the user opens the application until it is ready to be used. This time depends on the application, the server resources (available at the time the application is launch), and other factors (as the network load). All these random factors will make the start-up time non-deterministic. Hence, we will consider it as a random variable. For the same reasons, each time the application is correctly predicted, and we reduce the start-up time, the resulting time gain will also be random. To eliminate this randomness from the estimate of the start-up time, we will compute the expected value of this quantity for each user.

Expected start-up time for a user

The expected start-up time of the applications for a given user will be given by

$$t_i = c_i\, BT_i + (1 - c_i)\, NBT_i = NBT_i - c_i\, (NBT_i - BT_i), \tag{4.24}$$

where

- c_i, is the average percentage of correct predictions (this is the percentage of the launched sessions or applications that have been correctly predicted and therefore boosted).
- NBT_i, is the average of non-boosted applications' start-up time.
- BT_i, is the average of boosted applications' launch time and
- t_i is the expected start-up time of the applications.

All the previously defined quantities are for user i. E.g., let us assume that our algorithm is able to correctly predict 80% of all the launches ($c_i = 0.8$) of the user i and that the expected value of the start-up time of an application without any acceleration is 15 seconds ($NBT_i = 15$) and the expected value of the start-up time of an application that has been correctly predicted and therefore boosted is 3 seconds ($BT_i = 3$). Then the expected start-up time is

$$t_i = (0.8)\, 3 + (1 - 0.8)\, 15 = 15 - 0.8\, (15 - 3) = 5.4s. \tag{4.25}$$

Eq. 4.24 can also be used to compute the required average of correct predictions to achieve a certain wished expected start-up time of the applications (t_i) by solving for c_i. For a given start-up time t_i,

$$c_i = \frac{(NBT_i - t_i)}{(NBT_i - BT_i)}. \tag{4.26}$$

In that case, we only need to plug the value of the wished expected start-up time of the applications (t_i) in Eq. 4.26 and recover NBT_i and BT_i from the historical data. Then we set the target of correct prediction of our algorithm to the value c_i.

As we can see, these results would depend on the assumption that our user model is correct, this is that $c \approx m$. Later in this chapter, we will show how we can asses this assumption.

Average start-up time for a group

Usually, the administrators of the systems are not focused on the start-up time of individual users but on the start-up time of a group of users. We now propose some approaches to tackle the problem of optimizing the start-up time for a group of users [29].

Let G be the set of users $\{u_i\}_{i=1}^{N}$. The *expected application start-up time for a group of users* can be defined as:

$$GT = \frac{1}{|G|} \sum_{G} T_i(c_i|BT_i, NBT_i) \tag{4.27}$$

where $|G|$ is the cardinality of the set G (quantity of users in the group), and T_i is a function parameterized by BT_i and NBT_i that returns the time associated to the average percentage of correct prediction (c_i).

If we want to make GT equal to the group average wished start-up time (TT) we can either,

- set all T_i to the same value or
- we can set an optimization problem (a relaxed nonlinear continuous knapsack problem):

$$\text{minimize} \qquad \sum_{i=1}^{N} R_i(c_i) \tag{4.28}$$

$$\text{subject to} \qquad \sum_{i=1}^{N} T_i(c_i|BT_i, NBT_i) = TT. \tag{4.29}$$

$$0 \leq c_i \leq 1\,,\; i = 1, \ldots, N \tag{4.30}$$

Where, R_i is a non-linear function that returns the resource cost associated with the level of correct prediction c_i. To find an approximate solution, we can use a heuristic, e.g., a greedy algorithm [23].

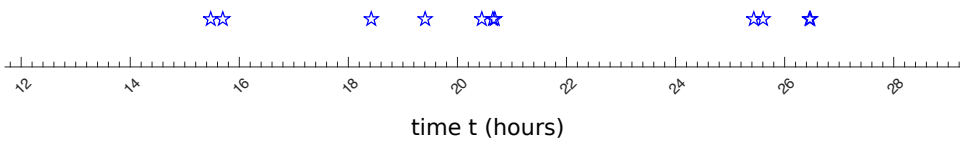


Figure 4.3 – The user activity can be plotted over a time starting from an arbitrary reference at a given point in time. In this figure, each star represents an activity event (for instance, an application launched by the user) between 12 and 28 hours since an arbitrary reference $t = 0$.

4.3.4 The behavior periodicity

Section 4.3.1 shows how, assuming knowledge of the true probability distribution of the behavior (session or application launches) over an interval of time T, we can tell which is the probability of the session or application launches to fall inside any region of the domain. Moreover, it shows that we can identify the *smallest* region that covers any arbitrary probability and that the size of this region will be proportional to the probability covered. Now, in this section, we will focus on how to define a suitable interval of time T that will serve as the domain of the probability distribution that defines the user model.

If we observe the activity of the user over a period T, as represented in Fig. 4.3, we will notice that the application launches tend to be clustered. Usually, when the user starts a session, she will use a set of applications to perform some tasks, and then a period of inactivity will follow. This means that the activity events (opening sessions or launching applications) are not independent of each other. In the case of models that are based on independence assumptions (like,for instance, the ones we use) we have (at least) two different options: the first option is just to relax the independence assumption and see if the results remain useful even though this condition is violated and second one is to condition on a subset of the events for which the independence assumption still holds (for instance, in most cases, the first application launches of a particular weekday will usually be mutually independent being normally distributed around the time the user is supposed to initiate her activity).

Another relevant point to notice is that, in general, the distribution of events over an interval of time, will not be unimodal. For instance, if we consider the intervals of $T = 1$ hour in Fig. 4.3, we will see that the activity events will tend to accumulate at the beginning and the end of the hour, generating two modes.

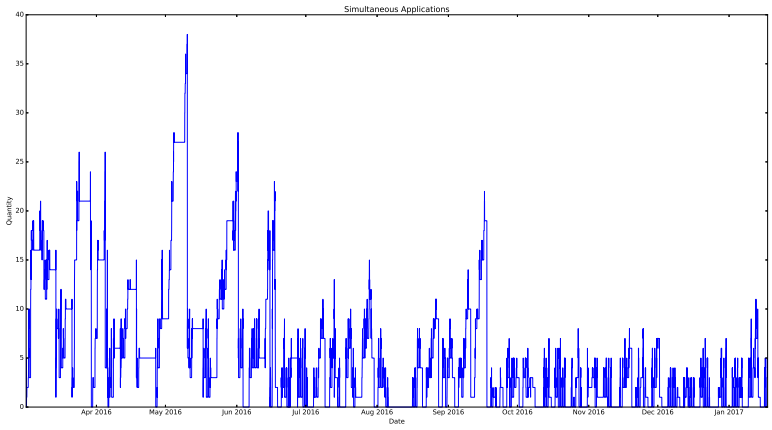


Figure 4.4 – Quantity of simultaneously open applications for a user over a period of 10 months.

4.3.5 Analysis of periodicity (defining T)

In this section, we will study the periodicity of the behavior and develop a method to detect the best value for T automatically.

Starting with a signal that represents the number of simultaneous applications open at a given moment, as shown in Fig. 4.4, we will try to identify the period that accounts for the biggest amount of behavioral patterns and use this value to set the domain of the distribution.

Since the data has a precision of milliseconds, we cannot recover variations over periods of less than a millisecond. However, since we measure human activity, we can assume that there will not be any significant use of an application with a total duration of less than a minute. For this reason, we choose a minute granularity so that the signal will count the amount of simultaneous open applications for any given minute for a time interval defined by the date of the earliest application launch (*AppStartDate*) and the latest application launch (*AppEndDate*) of a particular user.

Following the same approach as in [136], this signal will be transformed utilizing the normalized Discrete Fourier Transform (DFT).

Given a sequence $x(n)$ with $n = 0, \ldots, N-1$ the DFT of the sequence $x(n)$ will be a vector $X(k) \in \mathbb{C}^N$:

$$X(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) e^{-2\pi i k n / N}, k = 0, \ldots, N-1 \tag{4.31}$$

then, utilizing the periodogram, which is defined as

$$P(k) = \|X(k)\|^2, k = 0, \ldots, \lceil \frac{N-1}{2} \rceil \tag{4.32}$$

we can estimate the Power Spectral Density (PSD), which gives the power at each frequency.

The interesting frequencies will be the ones with high power in the PSD and to select them, we can apply different methods to generate thresholds:

1. Permutations.
2. A white noise process.
3. Standard deviation.

The first method is based on permutations and estimates the maximum power in the periodogram that corresponds to our sequence after elimination of the periodic structure by permutation.

$$p_{max} = \arg\max_f \|X_{per}(f)\|^2 \tag{4.33}$$

The second approach relies on a white noise process and computes the maximum power in the periodogram of a process where X_t is normally distributed with mean μ and variance σ equal to the estimated values of mean and variance of the sequence $x(n)$.

Finally, the third approach consists in measuring the standard deviation σ from a moving mean and choose the values that deviate more than $d = c\,\sigma$ (being c a fixed constant) from the moving mean. The benefit of this approach is that, in this case, the threshold is not a single value and adapts to the variations of the signal like we see in Fig. 4.5.

It is also important to notice that there are two points (related to the two extremes of the frequency spectra) to consider when analyzing which patterns we will be able to detect. As to being able to detect patterns that repeat within a short interval of time (high-frequency patterns), there will be a minimum sampling rate to ensure that the reconstruction is good (this means to get the same result as if the signal would be continuous). E.g., if we are sampling each minute how many applications are open, and the user open and close an application in a fraction of a second (between two sampled points) we will miss that open and close. This means that we will not have information about this behavior that has a very high frequency (period very short). On the other hand, to detect patterns that repeat over a long interval of time (low-frequency patterns), there will be a minimum time we have to record launches to detect this periodic pattern [26]. E.g., if we want to detect a pattern with a period of one year, we need at least to record more than one year of data (it is not possible to detect a periodic pattern from just one sample).

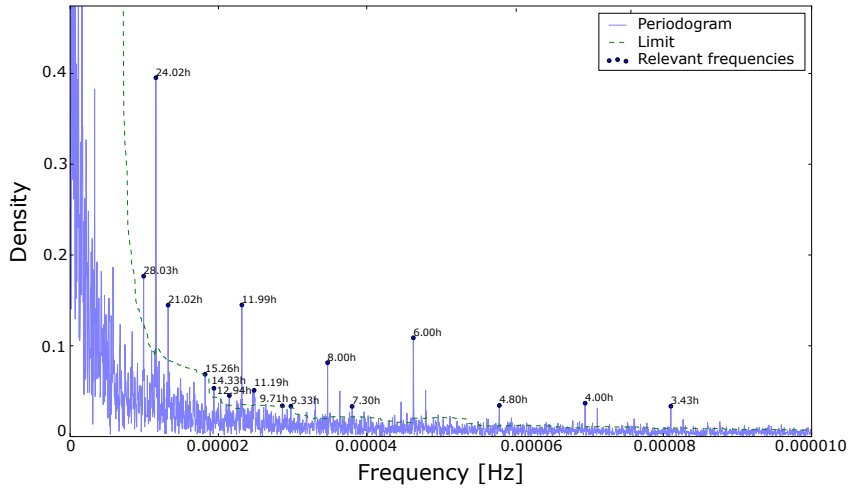


Figure 4.5 – Periodogram, limit and selected frequencies for an office user.

4.4 Modeling the application launches

We choose to start by generating a user model of the distribution of the first session login over a period $T = 24\,h$ based on the observation that:

- human behavior tend to naturally present daily patterns,
- the distribution of the first session logins tends to be unimodal for a given weekday,
- the samples seem to be independent of each other and
- the fact that first session logins belong to the group of activities that report the greater time gain when recovered.

In the dataset, we have for each session login the user U, the weekday WD, and the hour H (extracted from the timestamps). In particular, we will keep just the data points that correspond to the first session login of each day.

These three random variables will have a joint probability distribution $P(U, WD, H)$, and we can represent one of the possible factorizations with a simple Bayesian network, as shown in Fig. 4.6.

In particular, the graphical model in Fig. 4.6 implies the following factorization:

$$P(U, WD, H) = P(H|WD, U)P(WD, U) \tag{4.34}$$
$$= P(H|WD, U)P(WD|U)P(U). \tag{4.35}$$

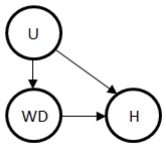


Figure 4.6 – Simple graphical model of the considered variables.

To approach our problem, in the following sections, we will define different statistical models of $P(H|\,WD, U)$, which is the conditional probability of having the first session login at a certain hour given the weekday.

4.4.1 Modeling with a discrete distribution: Categorical distribution

Let us assume that we have a sequence of discrete random variables $X_1, \ldots, X_N$. The sample space $\Omega = \{0, \ldots, 23\}$ represents the 24 hours of the day. This means that the variables can take on one of $K = 24$ mutually exclusive states. A realization of X can be represented with a one-hot encoded vector (e.g., $\mathbf{x} = (0, 0, 1, \ldots, 0, 0)^T$), this is, a vector of size K that takes the value 1 in the position that corresponds to the outcome while all other values are 0 and satisfy $\sum_k x_k = 1$.

We can now generate a parameter vector $\boldsymbol{\theta}$ where each component θ_k will denote the probability of $x_k = 1$ (imposing $\theta_k \geq 0$ and $\sum_k \theta_k = 1$) and define the distribution of $\mathbf{x}$ given $\boldsymbol{\theta}$ as:

$$P(\mathbf{x}|\boldsymbol{\theta}) = \prod_k \theta_k^{x_k} \tag{4.36}$$

The fact that $\sum_x P(\mathbf{x}|\boldsymbol{\theta}) = 1$ is easy to show, since $\sum_x P(\mathbf{x}|\boldsymbol{\theta}) = \sum_k \theta_k$ and we imposed $\sum_k \theta_k = 1$. This distribution $P(\mathbf{x}|\boldsymbol{\theta})$ is known as the categorical distribution [16].

Let us assume we have a sequence of observations $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$. The likelihood function of the data given the parameter vector is given by

$$P(\mathcal{D}|\boldsymbol{\theta}) = P(\mathbf{x}_1|\boldsymbol{\theta})P(\mathbf{x}_2|\boldsymbol{\theta}, \mathbf{x}_2) \prod_{n=3}^{N} P(\mathbf{x}_n|\boldsymbol{\theta}, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}) \tag{4.37}$$

where by assuming that the observations are independent, we can avoid considering the dependence on the previous observations, and we get:

$$P(\mathcal{D}|\boldsymbol{\theta}) = \prod_{n=1}^{N} P(\mathbf{x}_n|\boldsymbol{\theta}) \tag{4.38}$$

$$= \prod_{n} \prod_{k} \theta_k^{x_{nk}} \tag{4.39}$$

$$= \prod_{k} \theta_k^{\sum_n x_{nk}} \tag{4.40}$$

The log-likelihood will be then:

$$\log P(\mathcal{D}|\boldsymbol{\theta}) = \sum_{k} N_k \log \theta_k \tag{4.41}$$

where $N_k = \sum_n x_{nk}$.

Frequentist approach to the parameter estimation: Maximum Likelihood Estimator (MLE)

To derive the maximum likelihood estimator of the parameter vector $\boldsymbol{\theta}$, we need to look for the value of $\boldsymbol{\theta}$ that maximizes the likelihood (or the log-likelihood).

$$\boldsymbol{\theta}^{MLE} = \arg\max_{\boldsymbol{\theta}} P(\mathcal{D}|\boldsymbol{\theta}) \tag{4.42}$$

$$s.t. \sum_{k=1}^{K} \theta_k = 1 \tag{4.43}$$

Since this is a constrained optimization problem, we can use a Lagrange multiplier λ to find the maximum of this function with respect to θ_k. This is, to find the maximum of

$$\sum_{k=1}^{K} N_k \ln \theta_k + \lambda \left(\sum_{k=1}^{K} \theta_k - 1 \right), \tag{4.44}$$

we set the derive of 4.44 with respect to θ_k to zero

$$\frac{N_k}{\theta_1} - \lambda = 0 \tag{4.45}$$

solving for θ_k and replacing in 4.43 we get

$$\theta_k^{MLE} = \frac{N_k}{N}. \tag{4.46}$$

This means that the maximum likelihood estimator of the vector parameter $\boldsymbol{\theta}$ is given by the relative frequency of the hours of first session logins. Since all the information about the vector of parameters $\boldsymbol{\theta}$ is summarized in the vector of sums of the number of logins for each hour (making it a sufficient statistic), we do not need to keep any other data.

The MLE is a consistent estimator. This means that as the size of the dataset increase, the estimator tends in probability to the true value of the parameter $\boldsymbol{\theta}$.

Bayesian approach to the parameter estimation: Maximum a posteriori (MAP)

As explained in Section 4.2, another way of learning from the data is by applying a Bayesian framework. In this case, we will start by defining a prior over the set of the 24 hours of the day.

A valid candidate for the prior $p(\boldsymbol{\theta})$ is the Dirichlet distribution

$$Dir(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})} \prod_k \theta_k^{\alpha_k - 1} \tag{4.47}$$

where

$$\sum_k \theta_k = 1 \text{ and } \theta_i \geq 0 \text{ for all } k \in [1, K], \tag{4.48}$$

which also provides the benefit of being conjugate of the multinomial.

By multiplying the prior and the likelihood, defined in Equation 4.4.1,

$$P(\mathcal{D}|\boldsymbol{\theta}) = \prod_k \theta_k^{\sum_n x_{nk}} = \prod_k \theta_k^{N_k} \tag{4.49}$$

we get the unnormalized posterior, which is proportional to the posterior

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) \tag{4.50}$$

$$= Dir(\boldsymbol{\theta}|\alpha_1 + N_1, \ldots, \alpha_K + N_K) \tag{4.51}$$

This model is known as the Dirichlet-multinomial model [97].

To obtain the MAP estimator we now look for the parameter vector $\boldsymbol{\theta}$ that maximize the posterior distribution, this is

$$\boldsymbol{\theta}^{MAP} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{D}) \tag{4.52}$$

$$s.t. \sum_{k=1}^{K} \theta_k = 1. \tag{4.53}$$

To solve for $\boldsymbol{\theta}$ we can, once again, use a Lagrange multiplier yielding

$$\theta_k^{MAP} = \frac{N_k + \alpha_1 - 1}{N + \alpha_0 - K} \tag{4.54}$$

where

$$\alpha_0 = \sum_k \alpha_k. \tag{4.55}$$

The main benefits of this approach are that:

- it allows to use prior knowledge (e.g., for an office we do not expect to see activity during the weekends) or the relative frequency observed on other customers as a prior distribution,
- it is naturally suitable for online learning and
- it provides a measure of confidence over the parameter of the user model.

4.4.2 Modeling applications with a continuous distribution: Kernel Density Estimation (KDE)

Although in Section 4.4.1 the study focused on modeling the user behavior over a period of 24 hours by using a discrete distribution, we now show an approach based on a continuous distribution that will allow us to construct a user model that

- can identify the adequate main period for the user model,
- produces several intervals inside that period if needed,
- can produce intervals of any size,
- with any granularity (not just hourly),
- and to provide an isomorphic mapping between the intervals and the probabilities.

To estimate the PDF of the user model representing the user activity at time t, we propose to use a Kernel Density Estimator (KDE) [97]. A KDE is a non-parametric density model with a particular kind of kernel function named smoothing kernel k_h that satisfy:

$$\int k_h(t)dt = 1, \tag{4.56}$$

$$\int tk_h(t)dt = 0, \tag{4.57}$$

$$\int t^2 k_h(t)dt > 0. \tag{4.58}$$

where $h > 0$ is a smoothing parameter called the bandwidth that let us control the smoothness of the PDF by modifying the width of the kernel [31]. Given a dataset of N samples, the PDF can be estimated by centering a kernel function on each data point (t_i):

$$\hat{p}(t) = \frac{1}{N}\sum\nolimits_{i=1}^{N} k_h(t - t_i). \tag{4.59}$$

To take into account the periodicity of the data, we can choose as basis function a circular distribution. In our work, we choose the normal distribution wrapped around a circle of diameter T, known as a wrapped Gaussian and defined as

$$W\mathcal{N}(\theta; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \sum_{k=-\infty}^{\infty} \exp\left[\frac{-(\theta - \mu + Tk)^2}{2\sigma^2}\right], \tag{4.60}$$

where μ is the mean and σ is the standard deviation of the normal distribution (and will play de role of the bandwidth).

After transforming each t_i via the mapping $t_i \mapsto \theta_i$, $\theta_i = t_i \, mod \, T$. We estimate the probability distribution of the data,

$$\hat{p}_\sigma(\theta) = \frac{1}{n}\sum_{i=1}^{n} W\mathcal{N}(\theta; \theta_i, \sigma), \tag{4.61}$$

by centering over each data point θ_i a wrapped Gaussian.

4.4.3 Model selection - defining h

Rule-of-thumb method

For a Gaussian kernel and assuming p to be normal, we can show that minimizing the approximate mean integrated squared error yields

$$h_{opt} = \left(\frac{4}{3}\right)^{\frac{1}{5}} \sigma n^{-\frac{1}{5}} \approx 1.06\sigma n^{-\frac{1}{5}}. \tag{4.62}$$

Where σ can be replaced with an estimate of the standard deviation $\hat{\sigma}$. [115] show that this method will tend to over smooth. Specifically, he showed for a bimodal distribution how as the distance between the modes gets larger, the over smoothing effect augments.

When visually inspecting the dataset and choosing $T = 24$, we see a bimodal distribution with modes centered at the beginning of the morning shift and the afternoon after launch. For this particular choice of T, with the wrapped Gaussian kernel, even if we notice a tendency to over smooth, we get acceptable results.

Cross-validation method

Another popular method to find the bandwidth h is based on minimizing the ISE that can be expanded as,

$$ISE = \int \left[\hat{p}(x)\right]^2 dx + 2\int \hat{p}(x)p(x)dx + \int \left[p(x)\right]^2 dx, \tag{4.63}$$

where the last term does not depend on h. Additionally we can estimate the second term using leave-one-out cross-validation [81].

From our dataset $X_1, \ldots, X_n$ we remove X_i and use the remaining samples to compute the density estimate at the point X_i

$$\hat{p}_{-i}(X_i) = \frac{1}{(n-1)h} \sum_{j \neq i} k\left(\frac{X_i - X_j}{h}\right). \tag{4.64}$$

Which is used to compute the unbiased cross-validation score (UCV)

$$UCV(h) = \int \hat{p}^2(x)dx - \frac{2}{n}\sum_{i=1}^{n} \hat{p}_{-i}(X_i), \tag{4.65}$$

for different values of h and select

$$h^* = \arg\min_h UCV(h). \tag{4.66}$$

4.4.4 Incorporating a PID controller to improve results

Even though we can re-estimate the value of the kernel bandwidth as new observations of the application launches are gathered, it is normal to expect the estimation of the PDF to present some deviation from the real underlying PDF. This means that a certain interval $\mathcal{I}$ intended to cover a probability mass m (based on our estimated PDF) will be, in general, finally covering a probability mass m' different from the expected m. This effect will be particularly strong when just a very small dataset is available, for instance, when we start collecting data for a new user.

To diminish this effect, we introduced a PID controller to the pipeline that after each application launch, will measure the deviation of the current correct prediction moving average from the target and adjust the threshold T_3. The controller output is equal to the sum of three terms, each one parameterized by a gain (named proportional gain k_p, integral gain k_i and derivative gain k_d). The proportional term relates to the time the system will need to reach the target and the error in the steady state, the integral term relates to the bias (the residual steady-state error), and the derivative term relates to the oscillation in the transient state.

The diagram in Fig. 4.7 shows the estimation process that starts when the PID is initialized when the *target m* is fed to the UpdatePID() function. This value could be either a wished target (e.g., during the initialization) or be the result of the computation of the current correct predictions moving average. The PID will then compute and output the necessary modification to the previous value of *control m* that we call Δ *control m*. The new value of *control m* is computed by adding the proposed modification Δ *control m* and clipped to ensure that the final new *control m* value belongs to the interval of admissible values (between 0 and 1). The new *control m* allows the Validation() function to compute a prediction interval for the next application launch, test if it falls inside it and output a Boolean value (*correct*?) indicating if it has been correctly predicted. Finally, the UpdateCurrentm() function takes this last result and updates the new correct prediction moving average, generating a new value of *current m*, and closing the feedback loop.

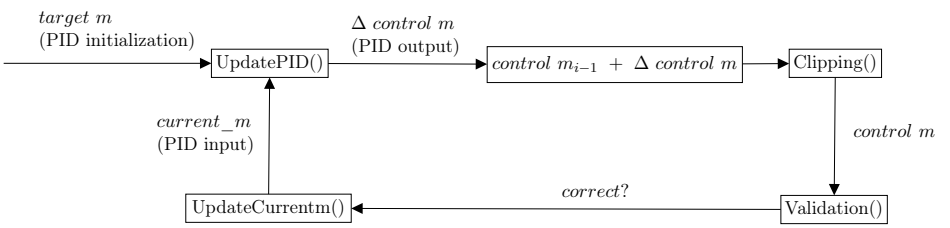


Figure 4.7 – Diagram showing the role of the PID controller and the feedback loop.

4.5 Results

To test the performance of our proposed pipeline, we will now use the two application launches datasets described in Section 3.2.2 to run what we call a simulation since it allows us to verify the results that we would obtain if the pipeline would be active at the customer. One by one, the application launches will be provided to the pipeline for training and generation of a prediction interval that will be tested with the next application launch (effectively dividing our dataset in train and test subsets at each iteration).

One part of each dataset will be used for hyperparameter tuning, while another part will be held out to get the final results once the hyperparameter of the system have been selected.

For each simulation, we need to define the following inputs:

- the database,
- the set of considered users,
- the period T over which we will generate the user model,
- the three gains of the PID (k_p, k_i, k_d), and
- a list of targets for the average of correct predictions.

The targets that we expect our algorithm to attain and hold as the user/s launch their applications will usually be set of values from 0 to 1 with increments of 0.1 to be able to evaluate the performance properly.

As a result, the simulation returns a dictionary of sequences that can be used to generate a plot similar to the one we see in Figure 4.8 containing:

- the target for the average correct predictions (dashed black line)
- moving average of the percentage of correct predictions (solid green line)
- moving average of the percentage of time that the applications remained idle waiting for the user to launch it (solid red line)
- size of the prediction interval as a fraction of T (dotted red line)
- probability mass m covered by the interval (dotted blue line)

4.5.1 User model evaluation

First, we evaluate the performance of the user models (rule-of-thumb and the cross-validation) alone. This means making the PID controller inactive by setting all the gains to 0. We verify the presence of the issues described in

Section 4.4.4, being that a significant number of launches were needed to stabilize the moving average number of correct predictions, and a non-negligible bias was present. Figure 4.8 shows the result of one of such simulations for one user and a target of average correct predictions of 70% [29]. In this case, we observe a difference of about 10% after the moving average stabilize.

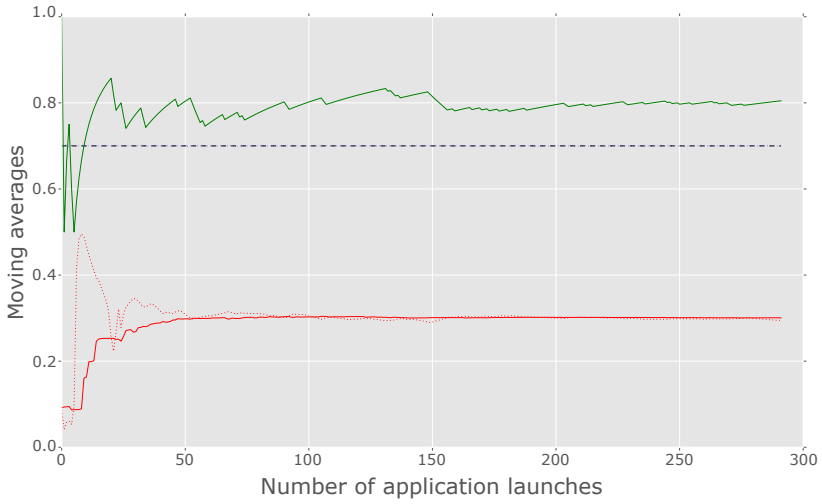
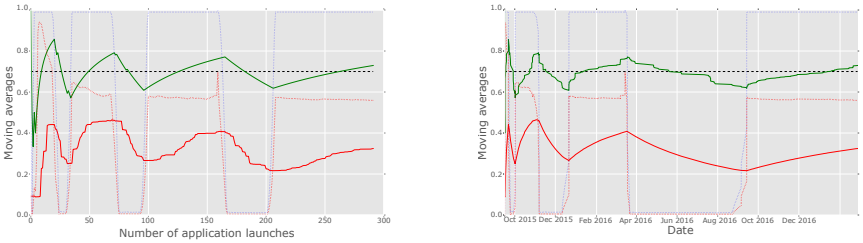


Figure 4.8 – Simulation results setting a target of 70% of correct predictions and without PID showing the target for the average correct predictions (dashed black line), the moving average of the percentage of correct predictions (solid green line), the moving average of the percentage of time that the applications remained idle waiting for the user to launch it (solid red line), the size of the prediction interval as a fraction of T (dotted red line) and the probability mass m covered by the interval (dotted blue line).

Next, we explore the effect of the different components of the PID on the validation dataset and look for a suitable set of values by performing a grid search. The results for a particular set of parameters is shown in Figure 4.9.

Setting the constrain of avoiding big variations around the target yield the selection of the gain values: $k_p = 0.5$, $k_i = 0$, $k_d = 0$.

Since we would like the system to be capable of getting better estimates of the underlying probability density as more application launches are used for training. We verified how fast it converges to a reliable user model and we show the results for the rule-of-thumb (Figure 4.10) and cross-validation method (Figure 4.11). These figures overlay two different set of plots. The green curves show target (x-axis) versus moving average of correct predictions (y-axis), while the red curves show target (x-axis) versus proportion of resources consumed for different dataset sizes (10, 50 and 100 application launches). As a reference, the expected result of an ideal model is indicated



(a) (b)

Figure 4.9 – Results of the simulation after incorporating the PID controller showing the target for the average correct predictions (dashed black line), the moving average of the percentage of correct predictions (solid green line), the moving average of the percentage of time that the applications remained idle waiting for the user to launch it (solid red line), the size of the prediction interval as a fraction of T (dotted red line) and the probability mass m covered by the interval (dotted blue line). The results can be plotted by (a) application launch or (b) over time.

by a dashed black line.

We observe that:

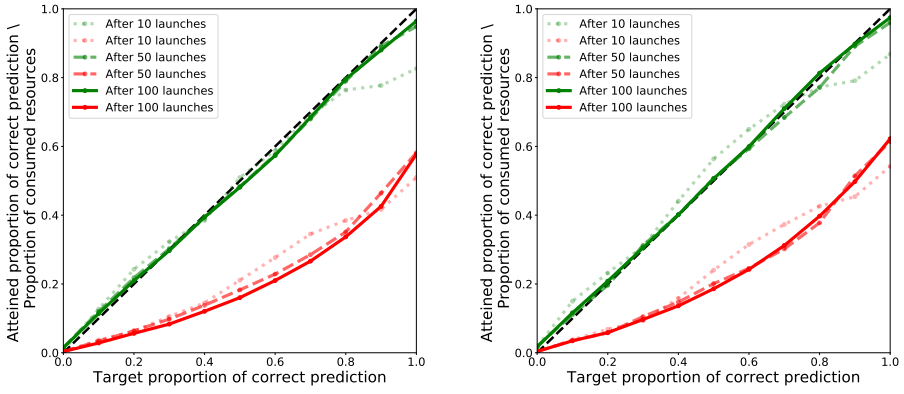
- While both methods (rule-of-thumb and cross-validation) present difficulties to attain high targets (between 0.8 and 1), the cross-validation method is about 150% more efficient (when comparing the moving average proportion of correct prediction and the proportion of resources consumed).

- The rule-of-thumb method seems to converge faster but tends to consume more resources making it suitable for users with little data.

- The cross-validation method with the PID can produce about 80% of correct predictions while keeping the overhead below 40% and can get more than 50% of correct predictions with less than 20% overhead.

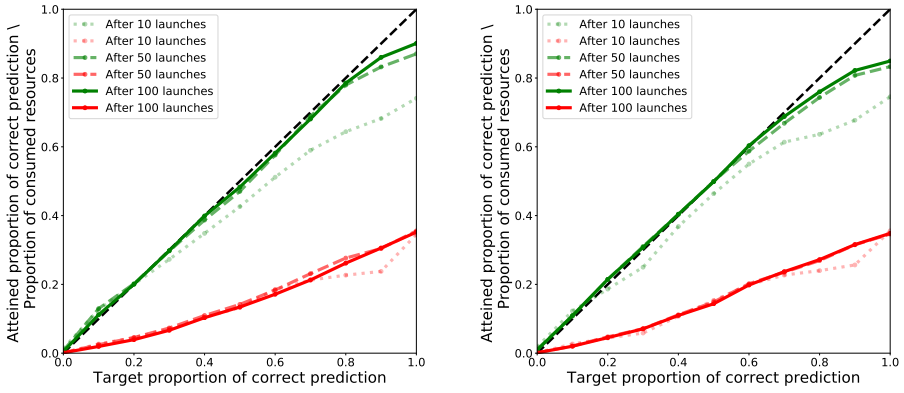
Although the rule-of-thumb is indeed a less computationally expensive method, in general, it is not a problem for the cloud to distribute the extra load associated with training the cross-validation method during an interval of time where the computational demand of the servers is low.

We conclude that the recommended default setting for the pipeline will be to set the values of the PID gains to $k_p = 0.5$, $k_i = 0$, $k_d = 0$, and apply the cross-validation method.



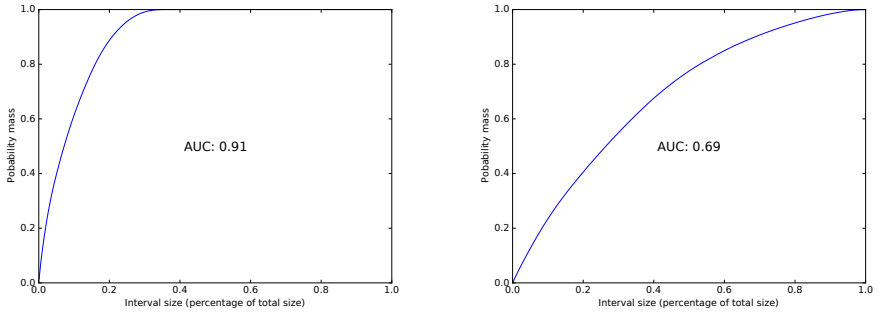
(a) Group 0 (b) Group 1

Figure 4.10 – Results of applying the rule-of-thumb method for two different user groups of Systancia with PID ($k_p = 0.5$, $k_i = 0$, $k_d = 0$) and $T = 24$ hours.



(a) Group 0 (b) Group 1

Figure 4.11 – Results applying the cross-validation method for two different user groups of Systancia with PID ($k_p = 0.5$, $k_i = 0$, $k_d = 0$) and $T = 24$ hours.



(a) Office (b) Hospital

Figure 4.12 – AUC values for two different customers, (a) office and (b) hospital.

4.5.2 Summarizing the user model

In the last sections, we focus on a set of values to evaluate our user models and predictive performance. However, we can simplify the model comparison by reducing that set of numbers to a single scalar value. To accomplish this objective we can consider the curve defined by Equation 4.21 (and depicted in blue in Figure 4.12) that relates the interval size and probability mass associated to that interval (which is equivalent to the amount of application launches covered by that interval) and compute the area below to summarize the trade-off between them.

Then (for user models defined by a continuous distribution), the area under the curve (AUC) will be defined as

$$\text{AUC} = \int_0^{\max(\hat{p}_\sigma)} M(T_3)|S'(T_3)|\, dT_3 \tag{4.67}$$

where, as described in Section 4.3.2 ,T_3 is the threshold that defines the interval size $S(T_3)$ and the probability mass $M(T_3)$.

Figure 4.12, compares the AUC values for two different kinds of customers, office, and hospital. We see that the result agrees with the intuition that predicting behavior in the case of the office will be easier since the behavior of the users usually presents less entropy.

Then an AUC close to 1 will indicate that we can cover a significant number of applications launches with small intervals. This is the same as saying that the probability mass gets concentrated into small regions of our domain, which in that case, will result in small values of differential entropy. The differential entropy of the behavior will always generate an upper bound to the accuracy of our predictions. However, we cannot precisely determine the entropy of the behavior (to do so, we need to know the exact probability

distribution). So in our case, as in other studies [117], we can use estimates of the entropy and mutual information to establish limits to the prediction accuracy. The estimated entropy could also be used as a criterion to decide for which users we do not want to generate a prediction.

4.6 Implementation in production

The ideas and algorithms developed in this chapter were implemented and shipped to the customer as a new module of Booster, named Booster Prediction in AppliDis Fusion 5.

As any failure could seriously damage the trust of the customer, we decided that the best strategy was to start by first implementing the algorithm described in Section 4.4.1, which is conceptually simpler to later add the more complex kernel-based approach of Section 4.4.2. This decision let us:

- detect missing supporting components in the system,
- reduce failure modes and generate a minimal set of unit test, and
- establish a benchmark.

Successfully implementing a prediction algorithm in production requires that all the components of the system that will support and interact with the algorithm work properly. For instance, if the prediction algorithm is based on the principle of launching "ghost" sessions and applications to reduce the waiting time, we need to have complete control of them, this is, being able to launch, identify and close them at any time (what was not actually the case when we started the project, in particular, for closing a session).

A simple model let us uncover the whole general structure of the components that need to be developed and their complexity. By treating each component as a black box, we were able to focus later in improving them individually. The approach of separating each component let us find bugs and analyze failure modes more easily.

By generating a first benchmark, we established a reference for comparison, and speed up the development process by facilitating design decisions.

At the same time, since there is a clear difference between the prediction algorithm objective and how we can use this result to approach the business problem. It was essential for us to decouple these two different problems. Decoupling the results from specific characteristics of the AppliDis problem let us: propose general application algorithms and ignore details of each customer or the implementation aspects of AppliDis that would make impossible the comparison of results between customers between AppliDis versions and over time. E.g., the time gain is linked to the application average launching time, servers static and dynamic characteristic, load balancing parameters,

and way of working. This is why it was essential for us to developed a general algorithm and to report a measure of performance that is independent of all these variables. For instance, if tomorrow something changes about how the balancing loader (e.g., the way we choose the server in which it will launch the next application), we will still be able to compare results between different versions of AppliDis.

4.6.1 C++ implementation

In Figure 4.13 we show the GUI of the first C++ implementation of the algorithm of Section 4.4.1. This GUI is based on the Microsoft Foundation Class (MFC) Library and was divided into six parts that let us input the control parameters and visualize the results.

To compute and display the predicted intervals for the different users, we needed to:

1. Connect to the remote SQL AppliDis database by inputting the database login data in the "Connection" section.
2. Then the amount of training data could be selected by setting the start and end dates of the time interval we wanted to use for training (in the section "Time Interval for Training") and the hyperparameters of the model could be set in the section "Parameters".
3. The algorithm was trained by clicking on the "Training" section.
4. After either inputting the User Id or the User Name and clicking on the "Show User Info"; the last login, the frequency matrix, the sum by weekday, the sum by hour of the day are displayed in the section "User Information".
5. Finally, the predicted intervals, along with the percentage of sessions launched inside that interval was then displayed in the "Prediction Information" section.

After several tests, this implementation was converted to a service application and added to AppliDis.

4.6.2 Controlling the algorithm from AppliDis: Booster GUI

To control the service application, we started the development phase of the Graphical User Interface to give to the AppliDis' administrator control over the algorithm.

In Figure 4.14 we show the Prediction Management tab of the implemented AppliDis Booster Control Panel with the corresponding other available tabs: Activation ("Activation"), Static Management ("Gestion Statique"), Prediction Management ("Gestion prédictive") and Booster Improvement Program ("Amélioration AppliDis Booster").

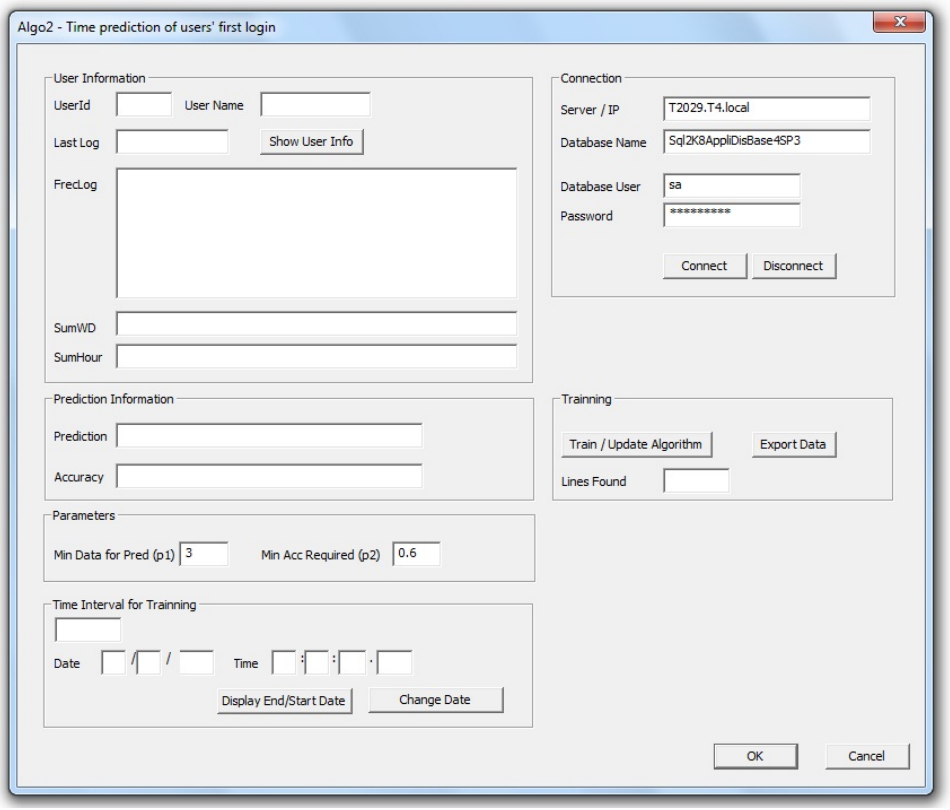


Figure 4.13 – GUI of the first C++ implementation of the algorithm of Section 4.4.1.

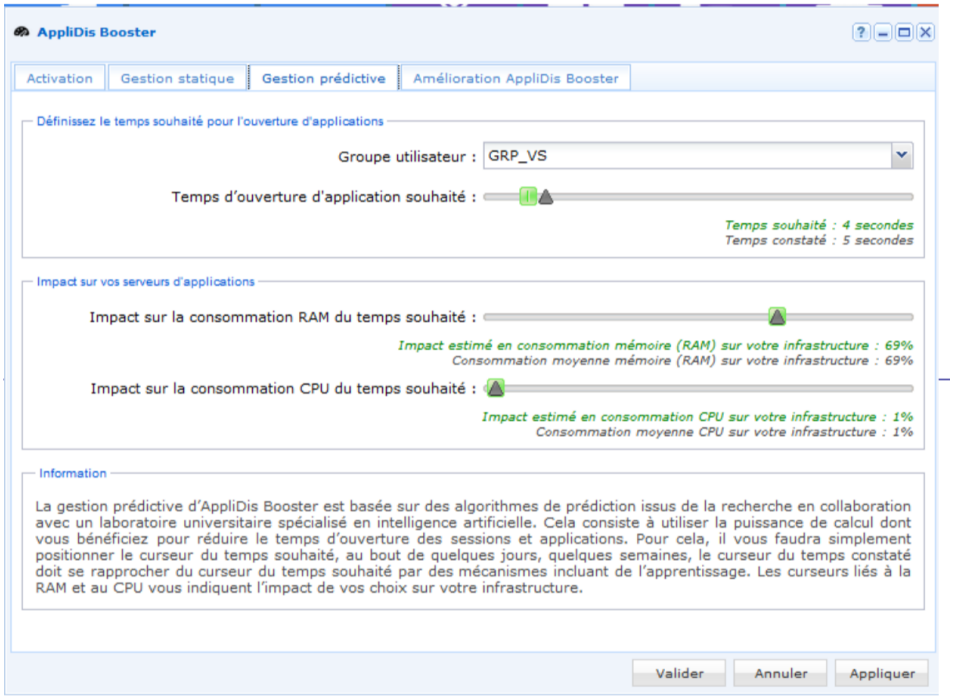


Figure 4.14 – GUI of the implemented control for the Booster prediction component.

Since directly exposing the hyperparameters of the model can be confusing for the administrator, we decided to expose the trade-off between launching time and resources, which is much more relevant and intuitive. Figure 4.14, shows the green control that allows to set the desired launching time for each group of users, while the estimation of the necessary resources are computed in an online fashion and accordingly reflected by updating the positions of the RAM and CPU sliders. Finally, the grey sliders show the observed average launching time for the selected group and the observed CPU and RAM consumption.

Finally, the Booster Improvement Program ("Amélioration AppliDis Booster") lets the administrator of the system send stats of the time gains to Systancia via the tab shown in Figure 4.15. The administrator of the system could also see a list of the data gathered, as shown in Figure 4.16.

4.6.3 C++ simulator implementation

To generate a tool that validates the results of the proposed algorithms and to show to the AppliDis' customer the benefits of activating Booster, we developed a Booster Simulator. The simulator displayed the results based on time gains and overhead generated by the algorithm, which are the key

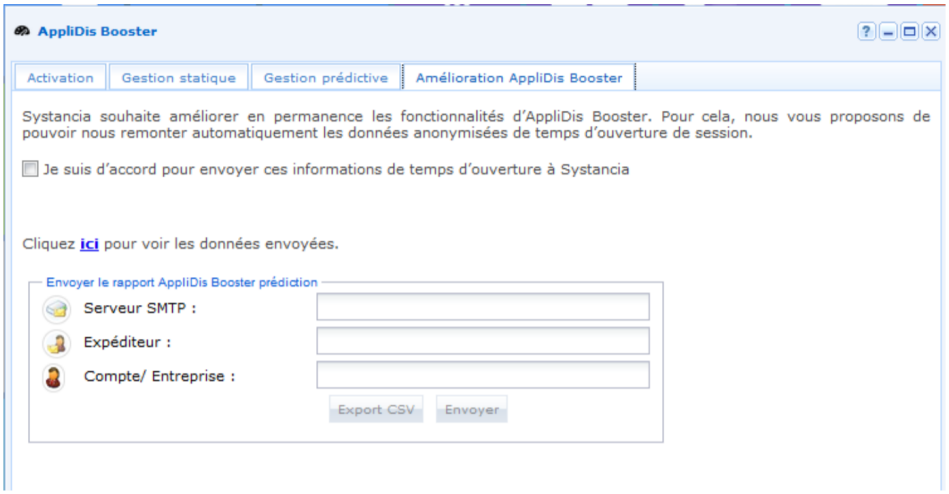


Figure 4.15 – Tab of the improvement program.

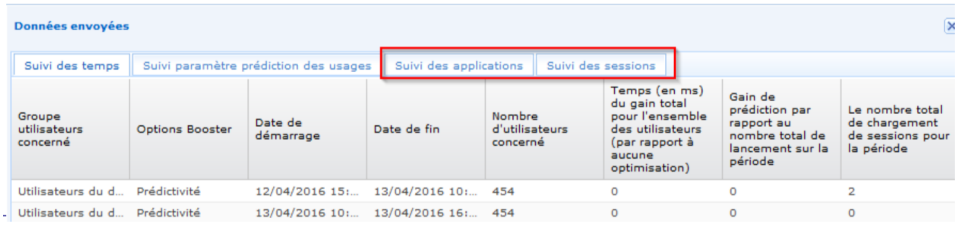


Figure 4.16 – List of data sent by the system administrator as a part of the improvement program.

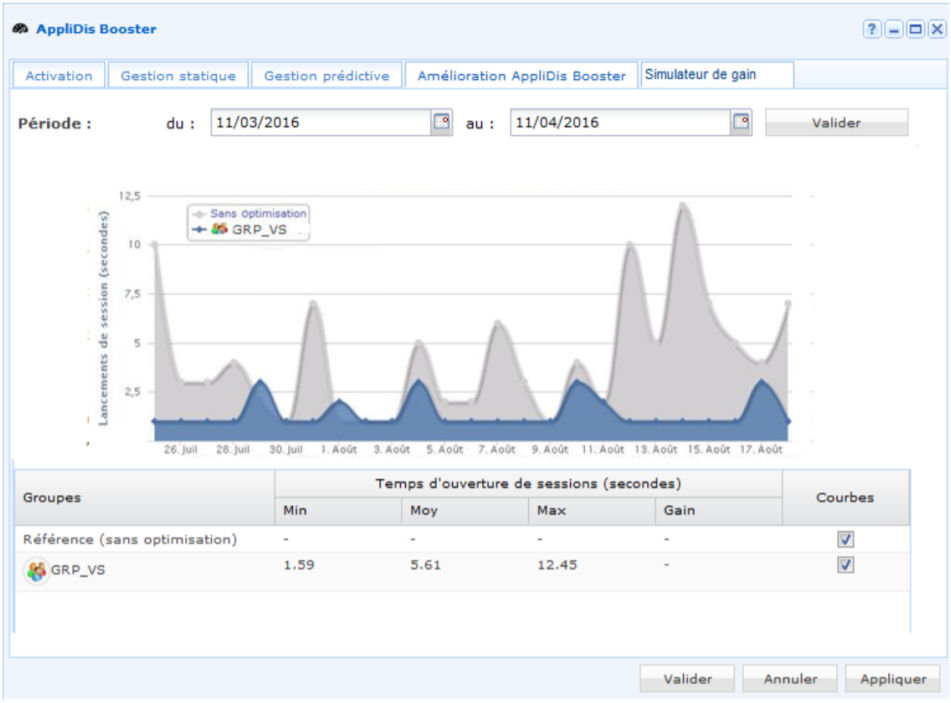


Figure 4.17 – Booster simulator GUI. The launching time (in seconds) of a group (blue curve) is compared with the launching time without Booster (grey curve).

performance indicators for the system administrator. As shown in Figure 4.17, the minimum, average, maximum, and total time gains of a group of users for a given interval of time were displayed and could be visualized as a graph.

Since the simulation relies on historical data, we motivated the system administrators to keep the activity records gathered by AppliDis.

Computational performance improvements

Since the simulations were computational intensive, we look for different methods to reduce the computing time. We decided to avoid the use of special hardware (e.g., GPUs) since we wanted the simulator to be available to any customer. The reduction of computing time was based on two main improvements.

The first improvement was to re-implement the most computational expensive Python function of the simulator in pure C. This change let us run our code 10 times faster. Unlike C, which is a statically typed language, Python is a dynamically typed language and needs to inspect the variables type creating a considerable overhead. To make the re-implementation easy

to maintain, we used Cython [10], a superset of Python.

The application of parallel computing generated the second reduction of the computation time.

4.7 Future work

As discussed in Section 2.3.6, a natural next step would be to explore an artificial neural networks approach, in particular, recurrent neural networks. The results of the first steps in this direction could be found in [14], where the author took sequences that represented the quantity of simultaneously open applications for a user over a period as shown in section 4.3.5 and tried to predict the last part of the sequence. Different architectures were tested, mainly vanilla one layer RNNs and three-layer RNNs. Even though these are valuable first steps, there is still much research and work to do. Mainly, an in-depth analysis of the impact of different initialization, testing other architectures like LSTM or bi-LSTM and analyzing different regularization methods. Since, at the moment of the development of that project, Systancia's GPUs were not available, these models were trained with a CPU. Using the available GPU could substantially improve the results by testing a broader set of models.

Conclusion

As the adoption of cloud computing architectures moves into more and more organizations, virtualization applications and software virtualization becomes a major and decisive issue in terms of performance. Indeed, cloud computing provides applications, storage capabilities, and computational resources easily to the users through a network. However, long start-up times of these services can jeopardize the user's experience. At the same time, there is a growing necessity in collecting data from the users. These heterogeneous and complex data is generally only stored in databases and data centers. In this thesis, we propose to not only store but also to exploit the data in real time with ML algorithms. The loading of an application takes between 15 and 20 seconds. The users do not appreciate to wait so long. Based on ML algorithms that analyze user usage to predict its behavior, key features are extracted, and a prediction module provides near-immediate access to applications while leaving the decision-making power of the administrator to control the consumption of resources. Chapter 1 provides an overview of cloud computing and virtualization architectures with a focus on Systancia's solution: AppliDis.

As discussed in Chapter 2, predicting human behavior is a problem of a great and growing economic value that is relevant to a big amount of different high social impact industries. This diversity of industries translates into a rich set of complex data sources that can be used to generate an enormous spectrum of features. The complexity of these data sources makes the creation of a good data model a fundamental step that will allow not only an efficient design of the physical database schema but a better analysis and knowledge extraction.

Chapter 3 provides a concrete example of how the data understanding phase can help us improve the way we organize the data and its quality to obtain a better input for our models. It also shows how different types of variables (continuous, categorical, and timestamps) in the databases can be transformed into informative features that will make the models easier to train and achieve better performance. In the context of our problem, we proposed an approach for user profiling based on these features that can be used to automatically group users with similar behavioral patterns and simplify the task of manually grouping users to the system administrators.

The grouping criterion is flexible since it can be adapted by selecting features that focus on specific characteristics of the users that may be of interest. This approach can also be used to detect anomalies, for instance, the use of applications that do not match the user profile or excessive time spent in particular applications can indicate the misused of a user account. The resulting visualizations can also be used in a dashboard as an appealing way of display information about the users of the system.

Even though predicting human behavior is an interesting problem by itself, the goal of this thesis is to use the extracted behavioral patterns to improve the user experience. To do so, in Chapter 4, we proposed a pipeline that allows the administrator of cloud computing systems to accelerate the launching time of applications while keeping the resource consumption low. The high value of accelerating the launching time becomes apparent after considering the fact that an important amount of French hospitals use AppliDis as their cloud computing solution, where even a small reduction of the launching time can produce a dramatic difference in time-sensitive activities.

The proposed pipeline is composed of:

- a module that study the size of the periodic patterns in the user behavior and finds an interval of time that captures most of the structure,
- a module that constructs a user model by estimating a PDF,
- a module that control the accuracy of the user model,
- a module that generates a prediction in the form of an interval in which we expect the user to launch an application, and
- a module that optimizes the launching application time for a group of users.

Where each component of the pipeline can be associated with a set of methods and techniques:

- To study the periodicity of the behavior, we apply Spectral Analysis techniques and the theory of Stochastic Processes.
- To construct the user model, we apply Bayesian, and frequentist approaches to density estimation, including both parametric and non-parametric approaches, with Kernel Density Estimation (KDE) belonging to the latter category. In the case of the KDE, to define the appropriate size for the kernel bandwidth, we apply the cross-validation method (which is used in Machine Learning to find the best hyperparameters of a model).
- To control the accuracy of the user model, we apply methods from control theory.

- To optimize the launching application time, we proposed the use of heuristics and techniques from linear and nonlinear programming.

Since, by default, after each application launch, the performance of the model is checked after each, and the model updated the proposed pipeline adapts quickly to changes of the user behavior (concept drift). The predictive system allows for three different use modes which are: achieving a certain average launching time, spending a fixed resource budget, or finding a balance between the last two. As explained before, in the context of time-sensitive activities, it is sometimes desirable to prioritize specific users or groups. Our proposed solution takes this into consideration and provides the system administrator a way of distributing the computational resources and of reducing the launching time of specific users or groups.

As described in the Section 4.5, applying KDE with the cross-validation method and a default initialization of the PID, allowed us to generate accurate models of the user behavior that can correctly predict about 80% of the events with an overhead less than 40%. In cases where the resources are limited, we can still get more than 50% of correct predictions with less than 20% overhead. For a customer with an average launching time of about 13s (like in the case of Systancia), this would imply reducing the launching time to about 2.6s and 6.5s, respectively.

The modular architecture of our solution makes it robust, easy to debug, and allows for progressive improvement of the system since we can focus on particular components with clearly defined functions. The proposed approach is well suited for customers with scarce data (e.g., new customers) since it allows the introduction of prior knowledge or set conditions. At the same time, to comply with the highest standards on data privacy policies, our approach does not require the data to leave the customers to perform well. The estimation approach can also be changed as needed switching between methods that allow batch and online training and scale well as the number of users increases.

Although our solution has been described and applied in the context of a very specific problem, it remains general and can also be applied to the prediction of other types of events (with the time and date of the events being the only requirement for our pipeline to work). This idea motivated the introduction of a performance measure that allows comparing results independently of the application context.

All the features described above proved to play a key role in the success of the solution that has been implemented and deployed to thousands of customers around the world as a component of Systancia's AppliDis. Using machine learning algorithms to analyze user usage and to predict its behavior will offer a wide range of new cloud services to meet the needs of each.

List of acronyms

AC	Autocorellation
ACF	Autocorrelation function
AE	Autoencoder
AR	Autoregressive
ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average
AUC	Area Under Curve
BN	Bayesian Network
BP	Behavior Prediction
CDP	Continuous Data Protection
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRM	Customer Relationship Management
CUDA	Compute Unified Device Architecture
DFT	Discrete Fourier Transform
DNA	Deoxyribonucleic Acid
DT	Decision Trees
EDA	Exploratory Data Analysis
GAN	Generative Adversarial Networks
GARCH	Generalized Autoregressive Conditional Heteroskedasticity
GPS	Global Positioning System
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
GSP	Generalized Sequential Pattern
GUI	Graphical User Interface
HMM	Hidden Markov Model
HPD	Highest Posterior Density
IaaS	Infrastructure as a Service
ISE	Integrated Squared Error
IT	Information Technology
JSD	Jensen Shannon Divergence

KDE	Kernel Density Estimation
KLD	Kullback–Leibler Divergence
LDS	Linear Dynamical System
LLE	Locally Linear Embedding
LSTM	Long Short-Term Memory
MAP	Maximum a posteriori
MDS	Multidimensional Scaling
MISE	Mean Integrated Squared Error
ML	Machine Learning
MLE	Maximum Likelihood Estimator
MLP	Multilayer Perceptron
ODBC	Open Database Connectivity
PaaS	Platform as a Service
PACF	Partial Autocorrelation Function
PC	Personal Computer
PCA	Principal Component Analysis
PDF	Probability Distribution Function
PID	Proportional–Integral–Derivative
RAM	Random-Access Memory
RBM	Restricted Boltzmann Machine
RDP	Remote Desktop Protocol
RDS	Remote Desktop Services
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SaaS	Service as a Service
SB	Server Based
SBC	Server-Based Computing
SID	Security Identifier
SMS	Short Message Service
SPADE	Sequential Pattern Discovery using Equivalence classes
SPAM	Sequential Pattern Discovery
SQL	Structured Query Language
SSM	State Space Model
SVM	Support Vector Machine
t-SNE	t-distributed Stochastic Neighbor Embedding
TPU	Tensor Processing Unit
UCV	Unbiased Cross-Validation
UM	User Model or User Modeling
UML	Unified Modeling Language
USB	Universal Serial Bus
VDI	Virtual Desktop Interface
VM	Virtual Machine
XML	Extensible Markup Language

Bibliography

[1] Ahmad Abdel-Hafez and Yue Xu. A survey of user modelling in social media websites. *Computer and Information Science*, 6(4):59, 2013.

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, June 2005.

[3] Yannis Agiomyrgiannakis and Yannis Stylianou. Wrapped Gaussian Mixture Models for Modeling and High-Rate Quantization of Phase Data of Speech. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(4):775–786, May 2009.

[4] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *International Conference on Management of Data (SIGMOD)*, pages 207–216. ACM, 1993.

[5] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, A Inkeri Verkamo, et al. Fast discovery of association rules. *Advances in knowledge discovery and data mining*, 12(1):307–328, 1996.

[6] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, pages 3–14. IEEE, 1995.

[7] James F. Allen. An interval-based representation of temporal knowledge. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'81, pages 221–226, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[8] James Frederick Allen. *A Plan-based Approach to Speech Act Recognition*. PhD thesis, University of Toronto Department of Computer Science, 1979. AAI0532011.

[9] Olatz Arbelaitz, Ibai Gurrutxaga, Javier Muguerza, Jesús M. Pérez, and Iñigo Perona. An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1):243 – 256, 2013.

[10] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D.S. Seljebotn, and K. Smith. Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31 –39, march-april 2011.

[11] Robert M Bell, Yehuda Koren, and Chris Volinsky. All together now: A perspective on the netflix prize. *Chance*, 23(1):24–29, 2010.

[12] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, March 1994.

[13] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[14] Kenza Benzaoui. Paradigmes d'apprentissage machine pour l'analyse de comportements utilisateurs – applications à la prédiction de lancements d'applications virtualisées. Report, Université de Technologie de Belfort Montbéliard, 2017. Mémoire de Master 2 AII (Automatique et Informatique Industrielle) spécialité SEC : Systèmes embarqués et communicants.

[15] Nachiket Sadashiv Bhosale and Sachin S Pande. A survey on recommendation system for big data applications. *Data Mining and Knowledge Engineering*, 7(1):42–44, 2015.

[16] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

[17] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-based systems*, 46:109–132, 2013.

[18] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307 – 327, 1986.

[19] E. Gohari Boroujerdi, S. Mehri, S. Sadeghi Garmaroudi, M. Pezeshki, F. Rashidi Mehrabadi, S. Malakouti, and S. Khadivi. A study on prediction of user's tendency toward purchases in websites based on behavior models. In *2014 6th Conference on Information and Knowledge Technology (IKT)*, pages 61–66, May 2014.

[20] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

[21] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[22] John S. Breese and David Heckerman. *Decision-Theoretic Troubleshooting: A Framework for Repair and Experiment*, pages 271–287. Springer US, Boston, MA, 1999.

[23] Kurt M Bretthauer and Bala Shetty. The nonlinear knapsack problem – algorithms and applications. *European Journal of Operational Research*, 138(3):459 – 472, 2002.

[24] Sergey Brin, Rajeev Motwani, Jeffrey D Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. *Acm Sigmod Record*, 26(2):255–264, 1997.

[25] Igor Cadez, David Heckerman, Christopher Meek, Padhraic Smyth, and Steven White. Model-based clustering and visualization of navigation patterns on a web site. *Data Mining and Knowledge Discovery*, 7(4):399–424, 2003.

[26] Matias Callara and Patrice Wira. Machine learning pour l'analyse de comportements et la classification d'utilisateurs. In *Congrès National de la Recherche des IUT (CNRIUT'2017)*, Auxerre, France, 2017.

[27] Matias Callara and Patrice Wira. User behavior analysis with machine learning techniques in cloud computing architectures. In *International Conference on Applied Smart Systems (ICASS 2018)*, Medea, Algeria, 2018.

[28] Matias Callara and Patrice Wira. A data-driven approach for user behavior prediction to boost productivity and sustainability of data centers and cloud-supported working environments. In *European International Conference on Transforming Urban Systems (EICTUS 2019)*, Strasbourg, France, 2019.

[29] Matias Callara and Patrice Wira. A probabilistic learning approach for predicting application launches in cloud computing architectures. In *IEEE/SICE International Symposium on System Integration (SII 2019)*, Paris, France, 2019.

[30] Matias Callara and Patrice Wira. Machine learning paradigms for user behavior modeling: An overview. *Journal of Applied Computer Science Methods*, submitted, 2019.

[31] Ricardo Cao, Antonio Cuevas, and Wensceslao González Manteiga. A comparative study of several smoothing methods in density estimation. *Computational Statistics & Data Analysis*, 17(2):153–176, 1994.

[32] Lucas Carlson. *Programming for PaaS*. O'Reilly Media, Inc, Sebastopol, California, first edition edition, 2013. OCLC: ocn818415020.

[33] Miguel A Carreira-Perpinan and Geoffrey E Hinton. On contrastive divergence learning. In *Aistats*, volume 10, pages 33–40. Citeseer, 2005.

[34] George Casella and Roger Berger. *Statistical Inference*. Duxbury Resource Center, June 2001.

[35] Silvia Cateni, Marco Vannucci, Marco Vannocci, and Valentina Colla. Variable selection and feature extraction through artificial intelligence techniques. In *Multivariate Analysis in Management, Engineering and the Sciences*. IntechOpen, 2013.

[36] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. The crisp-dm user guide. In *4th CRISP-DM SIG Workshop in Brussels in March*, volume 1999, 1999.

[37] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[38] Pau-Choo Chung and Chin-De Liu. A daily behavior enabled hidden Markov model for human behavior understanding. *Pattern Recognition*, 41(5):1572–1580, May 2008.

[39] Philip R. Cohen and C. Raymond Perrault. Elements of a plan-based theory of speech acts. *Cognitive Science*, 3(3):177–212, 1979.

[40] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.

[41] Trevor F. Cox and Michael A. A. Cox. *Multidimensional scaling*. Number 88 in Monographs on statistics and applied probability. Chapman & Hall/CRC, Boca Raton, Fla., 2. ed edition, 2001.

[42] Peng Dai, Shen-Shyang Ho, and Frank Rudzicz. Sequential behavior prediction based on hybrid similarity and cross-user activity transfer. *Knowledge-Based Systems*, 9(6):91–126, March 2015.

[43] J. Durbin and S. J. Koopman. *Time series analysis by state space methods*. Number 38 in Oxford statistical science series. Oxford University Press, Oxford, 2nd ed edition, 2012.

[44] Nathan Eagle and Alex Sandy Pentland. Eigenbehaviors: identifying structure in routine. *Behavioral Ecology and Sociobiology*, 63(7):1057–1066, May 2009.

[45] D. M. Endres and J. E. Schindelin. A new metric for probability distributions. *IEEE Trans. Inf. Theor.*, 49(7):1858–1860, September 2006.

[46] E.A. Feigenbaum, P. McCorduck, and P. Nii. *The Rise of the Expert Company: How Visionary Companies are Using Artificial Intelligence to Achieve Higher Productivity and Profits*. Macmillan, 1988.

[47] Edward A Feigenbaum. Expert systems in the 1980s. *State of the art report on machine intelligence. Maidenhead: Pergamon-Infotech*, 1981.

[48] Philippe Fournier-Viger, Antonio Gomariz, Manuel Campos, and Rincy Thomas. Fast vertical mining of sequential patterns using co-occurrence information. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 40–52. Springer, 2014.

[49] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S Yu. A survey of parallel sequential pattern mining. *arXiv preprint arXiv:1805.10515*, 2018.

[50] Jim Gao. Machine learning applications for data center optimization, 2014.

[51] Sharad Goel and Daniel G Goldstein. Predicting individual behavior with social networks. *Marketing Science*, 33(1):82–93, 2013.

[52] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[53] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[54] Ramanathan Guha, Vineet Gupta, Vivek Raghunathan, and Ramakrishnan Srikant. User modeling for a personal assistant. In *Proceedings*

of the Eighth ACM International Conference on Web Search and Data Mining, pages 275–284. ACM, 2015.

[55] Thomas Guyet and René Quiniou. Extracting temporal patterns from interval-based sequences. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1306–1311. AAAI Press, 2011.

[56] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A Zadeh. *Feature extraction: foundations and applications*, volume 207. Springer, 2008.

[57] Mohamed Ramzi Haddad, Hajer Baazaoui, Djemel Ziou, and Henda Ben Ghezala. A predictive model for recurrent consumption behavior: An application on phone calls. *Knowledge-Based Systems*, 64:32–43, July 2014.

[58] S. Haykin and R. Gwynn. *Neural Networks and Learning Machines*. Prentice Hall, 3rd edition, 2009.

[59] Xinran He, Stuart Bowers, Joaquin Quiñonero Candela, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, and Ralf Herbrich. Practical Lessons from Predicting Clicks on Ads at Facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pages 1–9. ACM Press, 2014.

[60] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[61] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

[62] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Master's thesis, Institut fur Informatik, Technische Universitat, Munchen*, 1991.

[63] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997.

[64] Eric Horvitz, Jack Breese, David Heckerman, David Hovel, and Koos Rommelse. The lumière project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 256–265, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[65] Eric J. Horvitz. Machine learning, reasoning, and intelligence in daily life: Directions and challenges (invited talk). In *Proceedings of Artificial Intelligence Techniques for Ambient Intelligence*, January 2007.

[66] Eric J. Horvitz, John S. Breese, and Max Henrion. Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning*, 2(3):247 – 302, 1988.

[67] Kevin L Jackson and Scott Goessling. *Architecting cloud computing solutions: build cloud strategies that align technology and economics while effectively managing risk*. Packt, 2018. OCLC: 1005108907.

[68] Peter Jackson. *Introduction to expert systems*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[69] B. Jensen, Jakob Eg Larsen, Kristian Jensen, Jan Larsen, and Lars Kai Hansen. Estimating human predictability from mobile sensor data. In *Machine Learning for Signal Processing (MLSP), 2010 IEEE International Workshop on. IEEE*, pages 196–201, 2010.

[70] I. T. Jolliffe. *Principal component analysis*. Springer series in statistics. Springer, New York, 2nd ed edition, 2002.

[71] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 1–12, New York, NY, USA, 2017. ACM.

[72] V. M. Katsageorgiou, M. Zanotto, H. Huang, V. Ferretti, F. Papaleo, D. Sona, and V. Murino. Unsupervised mouse behavior analysis: A

data-driven study of mice interactions. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 925–930, Dec 2016.

[73] Michael Kavis. *Architecting the cloud: design decisions for cloud computing service models (SaaS, PaaS, and IaaS)*. The Wiley CIO series. Wiley, Hoboken, New Jersey, 2014.

[74] Faten Khalil, Jiuyong Li, and Hua Wang. Integrating recommendation models for improved web page prediction accuracy. In *Proceedings of the thirty-first Australasian conference on Computer science-Volume 74*, pages 91–100. Australian Computer Society, Inc., 2008.

[75] Wael Khreich, Eric Granger, Ali Miri, and Robert Sabourin. A survey of techniques for incremental learning of HMM parameters. *Information Sciences*, 197:105–130, August 2012.

[76] Alfred Kobsa. Generic user modeling systems. *User Modeling and User-Adapted Interaction*, 11(1-2):49–63, March 2001.

[77] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[78] Farshad Kooti, Kristina Lerman, Luca Maria Aiello, Mihajlo Grbovic, Nemanja Djuric, and Vladan Radosavljevic. Portrait of an online shopper: Understanding and predicting consumer behavior. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 205–214. ACM, 2016.

[79] Miklós Kurucz, András A Benczúr, Károly Csalogány, and László Lukács. Spectral clustering in social networks. In *Advances in Web Mining and Web Usage Analysis*, pages 1–20. Springer, 2009.

[80] Dan Kusnetzky. *Virtualization: a manager's guide*. O'Reilly, Beijing ; Sebastopol, first edition edition, 2011.

[81] Neil D. Lawrence, Antony I. T. Rowstron, Christopher M. Bishop, and Michael J. Taylor. Optimising synchronisation times for mobile devices. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 1401–1408, Cambridge, MA, USA, 2001. MIT Press.

[82] Srivatsan Laxman and P. Shanti Sastry. A survey of temporal data mining. *Sadhana*, 31(2):173–198, 2006.

[83] Bac Le, Hai Duong, Tin Truong, and Philippe Fournier-Viger. Fclosm, fgensm: two efficient algorithms for mining frequent closed and generator sequences using the local pruning strategy. *Knowledge and Information Systems*, 53(1):71–107, 2017.

[84] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[85] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W Keller. Energy management for commercial servers. *Computer*, 36(12):39–48, 2003.

[86] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining Social-Network Graphs*, page 325–383. Cambridge University Press, 2 edition, 2014.

[87] Henry Lin and Max Tegmark. Critical Behavior from Deep Dynamics: A Hidden Dimension in Natural Language. *arXiv preprint arXiv:1606.06737*, 2016.

[88] Miao Lin, Wen-Jing Hsu, and Zhuo Qi Lee. Predictability of individuals' mobility with high-resolution positioning data. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 381–390, New York, NY, USA, 2012. ACM.

[89] Bing Liu. *Association Rules and Sequential Patterns*, pages 17–62. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[90] Yan Liu and Sanjay Chawla. Social media anomaly detection: Challenges and solutions. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 817–818. ACM, 2017.

[91] Caroline Lo, Dan Frankowski, and Jure Leskovec. Understanding behaviors that lead to purchasing: A case study of pinterest. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 531–540. ACM, 2016.

[92] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[93] Eren Manavoglu, Dmitry Pavlov, and C. Lee Giles. Probabilistic user behavior models. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 203–210. IEEE, 2003.

[94] Mary McGlohon, Leman Akoglu, and Christos Faloutsos. Statistical properties of social networks. In *Social network data analytics*, pages 17–42. Springer, 2011.

[95] Peter M. Mell and Timothy Grance. Sp 800-145. the nist definition of cloud computing. Technical report, National Institute of Standards & Technology, Gaithersburg, MD, United States, 2011.

[96] Hiroshi Motoda and Huan Liu. Feature selection, extraction and construction. *Communication of IICM (Institute of Information and Computing Machinery, Taiwan) Vol*, 5(67-72):2, 2002.

[97] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[98] Andrew Y Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[99] Loc Nguyen and Phung Do. Combination of Bayesian network and overlay model in user modeling. In *International Conference on Computational Science*, pages 5–14. Springer, 2009.

[100] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, March 2008.

[101] Peter von Oven and Barry Coombs. *Mastering VMware Horizon 7: learn advanced desktop virtualization techniques and strategies and dive deeper into VMware Horizon 7, take responsibility for optimizing your end user experience*. Packt, Birmingham Mumbai, second edition edition, 2016. OCLC: 1059536849.

[102] Abhinav Parate, Matthias Böhmer, David Chu, Deepak Ganesan, and Benjamin M. Marlin. Practical prediction and prefetch for faster access to applications on mobile phones. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, page 275. ACM Press, 2013.

[103] J. Pearl. *Bayesian Networks: A Model of Self-activated Memory for Evidential Reasoning*. Report. UCLA, Computer Science Department, 1985.

[104] Cathryn Peoples, Gerard Parr, Sally McClean, Bryan Scotney, Philip Morrow, SK Chaudhari, and Ravi Theja. An energy aware network management approach using server profiling in 'green' clouds. In *2012 Second Symposium on Network Cloud Computing and Applications (NCCA)*, pages 17–24. IEEE, 2012.

[105] C. Raymond Perrault, James F. Allen, and Philip R. Cohen. Speech acts as a basis for understanding dialogue coherence. In *Proceedings*

of the 1978 Workshop on Theoretical Issues in Natural Language Processing, TINLAP '78, pages 125–132, Stroudsburg, PA, USA, 1978. Association for Computational Linguistics.

[106] Sankaran Prithviraj. *Architecting Cloud SaaS software - solutions or products engineering multi-tenanted distributed architecture software*. Pearson, Chennai, 2016. OCLC: 1078361255.

[107] Kira Radinsky, Krysta Svore, Susan Dumais, Jaime Teevan, Alex Bocharov, and Eric Horvitz. Modeling and predicting behavioral dynamics on the web. In *Proceedings of the 21st international conference on World Wide Web*, pages 599–608. ACM, 2012.

[108] Howard Raiffa and Robert Schlaifer. *Applied statistical decision theory*. Wiley classics library. Wiley, New York, wiley classics library ed edition, 2000.

[109] K. Thirupathi Rao, P. Sai Kiran, and L. Siva Shanker Reddy. Energy efficiency in datacenters through virtualization: A case study. *Global Journal of Computer Science and Technology*, 2010.

[110] Elaine Rich. User modeling via stereotypes. *Cognitive Science*, 3(4):329 – 354, 1979.

[111] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[112] Salvatore Scellato, Mirco Musolesi, Cecilia Mascolo, Vito Latora, and Andrew T. Campbell. NextPlace: a spatio-temporal prediction framework for pervasive systems. In *International Conference on Pervasive Computing*, pages 152–169. Springer, 2011.

[113] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[114] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[115] Bernard W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, London, 1986.

[116] Brent Smith and Greg Linden. Two decades of recommender systems at amazon. com. *Ieee internet computing*, 21(3):12–18, 2017.

[117] C. Song, Z. Qu, N. Blumm, and A.-L. Barabasi. Limits of Predictability in Human Mobility. *Science*, 327(5968):1018–1021, February 2010.

[118] Chaoming Song, Tal Koren, Pu Wang, and Albert-László Barabási. Modelling the scaling properties of human mobility. *Nature Physics*, 6(10):818–823, October 2010.

[119] Libo Song, David Kotz, Ravi Jain, and Xiaoning He. Evaluating location predictors with extensive Wi-Fi mobility data. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1414–1424. IEEE, 2004.

[120] Julian Straub, Jason Chang, Oren Freifeld, and John W. Fisher III. A Dirichlet Process Mixture Model for Spherical Data. In *AISTATS*, 2015.

[121] Ilya Sutskever and Geoffrey E. Hinton. Learning Multilevel Distributed Representations for High-Dimensional Sequences. In *AISTATS*, volume 2, pages 548–555, 2007.

[122] Ilya Sutskever, Geoffrey E. Hinton, and Graham W. Taylor. The Recurrent Temporal Restricted Boltzmann Machine. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1601–1608. Curran Associates, Inc., 2009.

[123] R.S. Sutton, A.G. Barto, and F. Bach. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, 2018.

[124] Ling Tang, Anying Wang, Zhenjing Xu, and Jian Li. Online-purchasing behavior forecasting with a firefly algorithm-based svm model considering shopping cart use. *Eurasia Journal of Mathematics, Science and Technology Education*, 13(12):7967–7983, 2017.

[125] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[126] Gerald Tesauro, Nicholas K. Jong, Rajarshi Das, and Mohamed N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *2006 IEEE International Conference on Autonomic Computing*, pages 65–73. IEEE, 2006.

[127] George B. Thomas, Ross L. Finney, and Maurice D. Weir. *Calculus and analytic geometry*, page 919. Addison-Wesley, Reading, Mass, 9th ed edition, 1996.

[128] B.S. Todd. *An Introduction to Expert Systems.* Number no. 95 in An introduction to expert systems. Oxford University Computing Laboratory, Programming Research Group, 1992.

[129] John Wilder Tukey. *Exploratory data analysis.* Addison-Wesley series in behavioral science. Addison-Wesley Pub. Co, Reading, Mass, 1977.

[130] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The anatomy of the facebook social graph. *CoRR*, abs/1111.4503, 2011.

[131] Paul E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2):161–186, Nov 1989.

[132] Gregory Valiant and Paul Valiant. Estimating the unseen: improved estimators for entropy and other properties. In *Advances in Neural Information Processing Systems*, pages 2157–2165, 2013.

[133] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[134] Vladimir Naumovich Vapnik. *Statistical learning theory.* Adaptive and learning systems for signal processing, communications, and control. Wiley, New York, 1998.

[135] Emre Velipasaoglu. Machine-learned model quality monitoring in fast data and streaming applications. Strata Data Conference, 2018.

[136] Michail Vlachos, Christopher Meek, Zografoula Vagena, and Dimitrios Gunopulos. Identifying similarities, periodicities and bursts for online search queries. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 131–142. ACM, 2004.

[137] Michail Vlachos, S. Yu Philip, and Vittorio Castelli. On Periodicity Detection and Structural Periodic Similarity. In *SDM*, volume 5, pages 449–460. SIAM, 2005.

[138] Milan Vojnovic. On mobile user behaviour patterns. In *2008 IEEE International Zurich Seminar on Communications*, pages 26–29. IEEE, 2008.

[139] Laurent Voneau. Architecture AppliDis (AD00020). Technical report, Systancia, July 2004.

[140] Lizhe Wang. *Cloud computing: methodology, system, and applications.* CRC Press, 2017. OCLC: 1017818602.

[141] Yichuan Wang, Xin Liu, David Chu, and Yunxin Liu. EarlyBird: Mobile Prefetching of Social Network Feeds via Content Preference Mining and Usage Pattern Analysis. In *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 67–76. ACM Press, 2015.

[142] Ghaith Warkozek, Vincent Debusschere, and Seddik Bacha. Automated parameters retrieval for energetic model identification of servers in datacenters. In *PowerTech (POWERTECH), 2013 IEEE Grenoble*, pages 1–6. IEEE, 2013.

[143] Ghaith Warkozek, Elisabeth Drayer, Vincent Debusschere, and Seddik Bacha. A new approach to model energy consumption of servers in data centers. In *2012 IEEE International Conference on Industrial Technology (ICIT)*, pages 211–216. IEEE, 2012.

[144] Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[145] Geoffrey I. Webb, Michael J. Pazzani, and Daniel Billsus. Machine learning for user modeling. *User modeling and user-adapted interaction*, 11(1-2):19–29, 2001.

[146] Tingxin Yan, David Chu, Deepak Ganesan, Aman Kansal, and Jie Liu. Fast app launching for mobile devices using predictive user context. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 113–126. ACM, 2012.

[147] Jaewon Yang and Jure Leskovec. Patterns of temporal variation in online media. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 177–186. ACM, 2011.

[148] Faraz Zaidi and Guy Melançon. Evaluating the Quality of Clustering Algorithms using Cluster Path Lengths. In *10th Industrial Conference, ICDM*, volume 6171/2010 of *Advances in data mining. Applications and theoretical aspects*, pages 42–56, Berlin, Germany, 2010. Springer.

[149] Shuai Zhang, Lina Yao, and Aixin Sun. Deep learning based recommender system: A survey and new perspectives. *arXiv preprint arXiv:1707.07435*, 2017.

[150] Ingrid Zukerman, David W. Albrecht, and Ann E. Nicholson. *Predicting users' requests on the WWW*. Springer, 1999.