Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2019

Diogo Filipe Peixoto Ferreira

Sistema de Sugestões Sensível ao Contexto

Conversational Aware Suggestion System

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2019

Diogo Filipe Peixoto Ferreira

Sistema de Sugestões Sensível ao Contexto

Conversational Aware Suggestion System

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2019

Diogo Filipe Peixoto Ferreira

Sistema de Sugestões Sensível ao Contexto

Conversational Aware Suggestion System

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Diogo Nuno Pereira Gomes, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e Doutor Mário Luís Pinto Antunes, Professor Adjunto Convidado do Instituto Superior de Contabilidade e Administração da Universidade de Aveiro.

o júri / the jury

presidente / president	**Professor Doutor Arnaldo Silva Rodrigues de Oliveira** Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro
vogais / examiners committee	**Professor Doutor Joel Perdiz Arrais** Professor Auxiliar do Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra
	Professor Doutor Diogo Nuno Pereira Gomes Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Um enorme obrigado ao Professor Doutor Diogo Gomes e Doutor Mário Antunes, não só por toda a disponibilidade e ajuda prestada ao longo de todo este projeto, mas principalmente pelas oportunidades que me deram ao longo deste e outros trabalhos, que fizeram de mim um melhor profissional e, acima de tudo, uma melhor pessoa. Obrigado ao IT UID/EEA/50008/2019, por proporcionar todas as condições necessárias à realização desta dissertação. Obrigado também à Fundação Portuguesa para a Ciência e Tecnologia pelo suporte financeiro, nomeadamente através do projeto UID/EEA/50008/2019 e Smart EnterCom (POCI-01-0247-FEDER-021949).

De seguida, quero agradecer aos meus pais, pela oportunidade dada de seguir os meus sonhos e por sempre me suportarem e apoiarem incondicionalmente em todo o meu percurso pessoal e educacional.

Quero agradecer também à minha namorada, por toda a paciência e por sempre ter acreditado em mim e nas minhas capacidades, independentemente do tempo isolado que passei a desenvolvê-las.

Obrigado também a todos os meus amigos e colegas que tive oportunidade de conhecer ao longo destes anos, principalmente aqueles que deixaram marca, como é o caso das QL©. Seguimos juntos, independentemente de onde estejamos!

Um especial obrigado ao meu amigo Pedro, por ter dividido comigo sofrimentos e conquistas ao longo destes cinco anos, bem como pela sua influência numa procura pela excelência deste trabalho. Um abraço a ti, irmão.

Por fim, um muito obrigado a Deus, por me ter dado forças para enfrentar os desafios que foram surgindo e por me ter dado capacidade de os conseguir sempre superar.

Palavras Chave

Computação pervasiva, Consciência de Contexto, Sistema de recuperação de informação, Sistema de sugestões, Mineração de texto, Aprendizagem por reforço

Resumo

Ao longo dos últimos anos, os sistemas pervasivos têm sido fonte de um grande desenvolvimento. Contudo, as interações humano-humano também podem tirar vantagem deste tipo de sistemas recorrendo à sua capacidade para entender o ambiente que o rodeia.

Nesta dissertação, foi desenvolvido um sistema pervasivo – chamado Sistema de Sugestões Sensível ao Contexto (CLASSY) – que está consciente dos vários contextos conversacionais e que sugere documentos considerados potencialmente úteis para os utilizadores ou que, de alguma forma, poupam tempo na execução de uma tarefa específica. Foram também propostas duas aproximações diferentes – a de vizinhança, que usa similaridade semântica, baseando-se em proximidades de forma a classificar relações entre palavras; e a de Aprendizagem por Reforço, que usa *feedback* implícito dos utilizadores associado a cada sugestão, como fonte de conhecimento que pode ser utilizado para melhorar a performance do sistema ao longo do tempo.

Os testes realizados mostraram que as aproximações acima referidas melhoraram não só o comportamento pervasivo do sistema, mas também a sua performance global.

Foi, ainda, analisado um caso de estudo referente à importância de *feedback* em ambientes com contexto limitado, onde os resultados mostraram que o mesmo continua a ser uma importante fonte de conhecimento, independentemente das características do ambiente conversacional.

Keywords Pervasive computing, Context-Awareness, Information Retrieval, Suggestions system, Text mining, Reinforcement Learning

Abstract

Over the last few years, pervasive systems have experienced some interesting development. Nevertheless, human-human interaction can also take advantage of those systems by using their ability to perceive the surrounding environment.

In this dissertation, we have developed a pervasive system – named ConversationaL Aware Suggestion SYstem (CLASSY) – which is aware of the conversational context and suggests the users potentially useful documents or that, somehow, save time executing a specific task. We have also proposed two different approaches – the Neighborhood one, that uses semantic similarity, based on proximity data in order to classify the relationship between tokens; and the Reinforcement Learning one, that uses implicit feedback associated with each suggestion as a source of knowledge that can be used to improve the system's performance over time.

The conducted tests showed that these two approaches not only enhanced the pervasive behavior of the system, but also increased its global performance.

A case study regarding the importance of feedback on context-limited environments was also carried out, whose results showed that it is still a useful source of knowledge regardless the conversational environment's characteristics.

Contents

List of Figures **v**

List of Tables **vii**

Acronyms **ix**

1 Introduction **1**

1.1 Motivation 1

1.2 Objectives 2

1.3 Outline 3

2 Background and State of The Art **5**

2.1 Pervasive computing 5

2.1.1 Driving areas 6

2.1.2 Main characteristics 7

2.1.3 Computational model 8

2.2 Context-awareness 9

2.2.1 Context 9

2.2.2 Main features 10

2.3 Text mining 11

2.3.1 Main pipeline 11

2.4 Information Retrieval 13

2.4.1 Base components 14

2.4.2 Main stages 14

2.5 Recommendation Systems 15

2.5.1 Types of recommendation systems 15

2.5.2 Challenges and solutions 17

2.6 Learning to Rank 18

2.6.1 Learning process 18

2.6.2 Learning approaches 19

2.6.3 Evaluation methods . . . 19
2.6.4 Evaluation metrics . . . 20
2.6.5 Online Learning to Rank . . . 21
2.7 Reinforcement Learning . . . 22
2.7.1 Reinforcement Learning definition . . . 22
2.7.2 Reinforcement Learning algorithms . . . 23
2.8 Related work . . . 24
2.9 Conclusion . . . 27

3 CLASSY prototype 29
3.1 Context processing . . . 30
3.2 Knowledge database . . . 30
3.3 Suggestions filter and cache . . . 31
3.4 Implementation . . . 32
3.4.1 Search engine based system . . . 32
3.4.2 Context processing module implementation . . . 32
3.4.3 Knowledge database implementation . . . 34
3.4.4 Suggestions filter and cache module implementation . . . 36
3.4.5 Interaction interface . . . 36
3.4.6 Additional features . . . 37
3.5 Results . . . 38
3.5.1 Datasets exploration . . . 38
3.5.2 Testing environment . . . 39
3.5.3 Base approach results . . . 41
3.6 Conclusion . . . 42

4 CLASSY evolution 45
4.1 Neighborhood approach . . . 45
4.1.1 Neighborhood score . . . 45
4.1.2 Implementation . . . 47
4.1.3 Results . . . 50
4.2 Reinforcement Learning approach . . . 52
4.2.1 User feedback . . . 52
4.2.2 Reinforcement Learning problem modeling . . . 52
4.2.3 Replacement strategies . . . 55
4.2.4 Implementation . . . 56
4.2.5 Results . . . 57
4.3 Conclusion . . . 61

5 Case study: The importance of feedback on context limited environments 63

5.1 Documents 63
5.2 Conversational environment 64
5.3 Feedback 64
5.4 Setup 64
5.4.1 Songs 64
5.4.2 Reddit content 64
5.4.3 Implementation 66
5.5 Results 67
5.6 Conclusions 70

6 Conclusion and Future Work **71**
6.1 Conclusions 71
6.2 Contributions 73
6.2.1 *SmartEnterCom* project 73
6.2.2 Lisbon Machine Learning School 2019 73
6.2.3 Ubiquitous Computing and Ambient Intelligence 2019 74
6.2.4 Data Science Portugal meetup #61 74
6.2.5 Data Science Portugal Day 2019 74
6.2.6 IEEE International Conference on Human-Machine Systems 2020 . . . 75
6.3 Future Work 75

References **82**

A Appendix A - LxMLS2019 CLASSY poster **83**

B Appendix B - ICHMS2020 CLASSY paper **85**

List of Figures

1.1 CLASSY environment interactions. 2

2.1 Taxonomy of computer systems research problems in pervasive computing [1]. 6
2.2 Pervasive computational model [2]. 8
2.3 Text mining main pipeline [3]. 12
2.4 The process of retrieving information [4]. 15
2.5 Collaborative filter process. 16
2.6 Content based filter process. 16
2.7 Hybrid filter process. 17
2.8 The RL feedback loop. 22
2.9 ACLD architecture [5]. 25
2.10 The steps of the proposed keyword extraction method [6]. 26

3.1 CLASSY architecture. 30
3.2 Implicit query formation. 31
3.3 Context processing module interactions. 33
3.4 Suggestions filter and cache module interactions. 36
3.5 CLASSY Discord interaction. 37

4.1 Neighborhood vector creation. 46
4.2 Solr plugin class diagram. 47
4.3 Neighborhood score computation steps. 49
4.4 Singular context state computation. 53
4.5 Example of a context distribution in a two-dimensional space. 55
4.6 Agent database model. 56
4.7 Impact of α parameter in the performance 58
4.8 Impact of *depth* parameter in the performance 59
4.9 Hybrid versus RL approaches with minimum document score of 40 59
4.10 Hybrid versus RL approaches with minimum document score of 140 60
4.11 Hybrid versus RL approaches with minimum document score of 220 60

4.12 CLASSY final architecture 62

5.1 Reddit case study setup implementation. 66
5.2 Examples of CLASSY interactions on Reddit. 68
5.3 `/r/popheads` channel simulation. 69
5.4 `/r/music` channel simulation. 69
5.5 `/r/ifyoulikeblank` channel simulation. 70

List of Tables

3.1 Base approach - minimum document score impact. 41
3.2 Base approach with Noise files system - number of noise files impact. 41
3.3 Base approach with Noise files system - number of messages between suggestions impact. 42

4.1 Neighborhood memory example. 48
4.2 Neighborhood computation performance. 50
4.3 Neighborhood approach - window impact. 50
4.4 Neighborhood approach - minimum document score impact. 51
4.5 Hybrid approach - minimum document score impact. 51
4.6 Replacement techniques performance. 61

Acronyms

ACLD AMIDA Automatic Content Linking Device

AI Artificial Intelligence

AP Average Precision

API Application Programming Interface

ASR Automatic Speech Recognition

CLASSY ConversationaL Aware Suggestion SYstem

CSV Comma-separated values

DBGD Dueling Bandit Gradient Descent

DBMS Database Management System

DCG Discounted Cumulative Gain

DQN Deep Q Network

DR Data Retrieval

DSPT Data Science Portugal

FIFO First In First Out

GPS Global Positioning System

ICHMS International Conference on Human-Machine Systems

IDF Inverse Document Frequency

IM Instant Messaging

IPv4 Internet Protocol Version 4

IPv6 Internet Protocol Version 6

IR Information Retrieval

JIT-DR Just-In-Time Document Retrieval

JSON JavaScript Object Notation

LRU Least Recently Used

LTR Learning to Rank

LxMLS Lisbon Machine Learning School

MAP Mean Average Precision

MART Multiple Additive Regression Trees

MDP Markov Decision Process

MGD Multileave Gradient Descent

ML Machine Learning

MRR Mean Reciprocal Rank

NDCG Normalized Discounted Cumulative Gain

NLP Natural Language Processing

NN Neural Network

OCR Optical Character Recognition

OLTR Online Learning to Rank

ORM Object Relational Mapper

P2P Peer-to-peer

PDF Portable Document Format

PoC Proof of Concept

POS Part-of-speech

QR Quick Response

R&D Research and Development

R&I Research and Innovation

RL Reinforcement Learning

RR Reciprocal Rank

SARSA State-Action-Reward-State-Action

SMEs Small and Medium-sized Enterprises

SVM Support Vector Machine

TF Term Trequency

TF-IDF Term Frequency-Inverse Document Frequency

UCAmI Ubiquitous Computing and Ambient Intelligence

UI User Interface

URL Uniform Resource Locator

UX User Experience

XML Extensible Markup Language

1

Introduction

In this chapter, we will describe the motivation and objectives of this work, focusing on making communications more intelligent, especially when it comes to conversational environments (commonly known as chats). As they became part of our daily life, making them more useful will benefit our personal and professional life in general.

Furthermore, we will also mention the research project under the development of this work, as well as a general outline of the document.

1.1 Motivation

Over the last few years, the use of conversational environments became part of our daily life, whether for personal or professional purposes. The amount of exchanged messages, together with the number of users interacting with these types of environments, make them a good source of conversational data that can be grouped into some specific sets of topics.

With the advent of chatbots, independent programs to conduct a conversation or mimic the human behavior in conversational environments, the interaction with the latter became much more simple as well as user-friendly. Using their interfaces and including some type of intelligence to these systems (mainly domain-specific one), it is possible to enhance the conversational environments, by conducting interactions that somehow enrich or augment the context of the conversation.

Moreover, this work also integrates the scope of the project "Comunicações Empresariais Inteligentes (*SmartEnterCom*)", that involves a core of Research and Development (R&D) whose consortium includes two Small and Medium-sized Enterprises (SMEs) – the Finesource Lda (current GOContact [1]) (lead promoter) and Wavecom SA [2] (co-promoter) – and a non-corporate entity of the Research and Innovation (R&I) system – Instituto de Telecomunicações

[1] `https://www.gocontact.pt/`
[2] `https://www.wavecom.pt/`

/ Aveiro [3] – the latter which is the place and entity that supports this dissertation. These three entities agreed on the collaborative development of products and features for a period of three years, with the goal of creating a Proof of Concept (PoC) and their technological facets to potentiate the organization of internal and external enterprise communications, as well as the access to structured communication information [7]. The work of the R&D core will be mainly the creation of software prototypes that should be further refined in order to be placed in a commercial environment, with the requirements outlined by the enterprise entities. This work is developed as part of Activity 2 – Context-aware applications – with the goal of developing applications that gather, store and use context information, as pervasive as possible, in a human-computer interaction environment. These interactions must contemplate intelligent behaviors, such as contact ordering, environment integration (audio, image, and operation mode) and regarding suggestions of working groups.

1.2 Objectives

The need to create tools that facilitate the communication processes, mainly those regarding a conversational environment, led to the definition of some requirements that later showed to be the strategic pillars of this work – the demand to be pervasive, context-aware, and a communication facilitator. Some of these requirements are met by some other projects, or especially, chatbots. However, they usually lack in being context-aware, since, when the system needs to know some description of the problem/context, they require the user to explain the situation, instead of trying to understand it by itself, which sometimes is not possible (or at least not trivial).

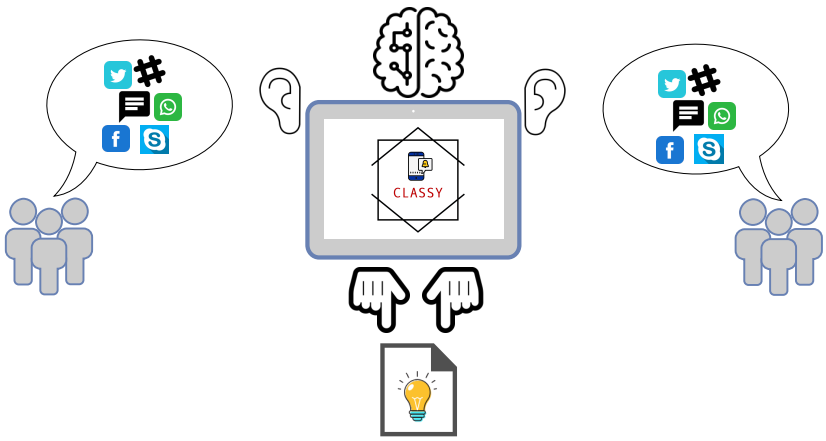


Figure 1.1: CLASSY environment interactions.

Taking this into account, the main goal of this work is to develop a system – named ConversationaL Aware Suggestion SYstem (CLASSY) – that is capable of being aware of the

[3] https://www.it.pt/ITSites/Index/3

conversational interactions and contexts, and compare them with a given knowledge database. The latter will have information considered useful to the conversational topics, thus, providing documents potentially relevant at a specific time, in a specific context. The concept of context has wide use in distinct fields and it is extremely important when it comes to perceiving the current situation and environment. It is also useful in many business use cases, like technical support, where an accurate definition of the client context can help solve the problem faster. This concept will be described in more detail in Section 2.2.

A simple example of the behavior of the system should be: considering a messaging platform like Slack [4], where the participants are a team of developers responsible for creating a specific product, and considering that the system has access to the code repository, namely commits information. When one of the developers talks about one specific commit, the system should suggest it, in order to facilitate the inspection and contextualization to the other developers. This way, they do not need to search for that specific commit over all the repository. This task could be easy when the developers directly mention the commit, by using, for instance, its identifier, but it can be very difficult if the developers decide to only talk about the commit, never specifically mentioning it. This forces the system to infer the context, and find, through the words used by the developers, that they are talking about a specific commit that it has access to, and so it can be suggested. These interactions are also illustrated in Figure 1.1.

Despite the behavior described above, the system should be pervasive, meaning that it should not be always invading the users' conversational environment. For that, the system should discard suggestions that are considered to have a low degree of certainty, saving these interactions to when it is certain the suggestions will be useful.

1.3 Outline

This document will be structured as it follows:

- **Chapter 2 - Background and State of the art**, where some theoretical and practical topics, such as Pervasive computing and Text-mining, will be discussed, which supported the content and development of this work. We will also review some projects considered to be related work, analyzing their strengths and weaknesses in order to develop a more robust system;

- **Chapter 3 - CLASSY prototype**, where we will describe the base modules that constitute the base architecture of the proposed system and their goals, as well as the implementation decisions and performance results;

[4] `https://slack.com/`

- **Chapter 4 - CLASSY evolution**, where we will describe two main approaches developed to enhance the system performance as well as its pervasive behavior, along with all their implementation details and performance tests;
- **Chapter 5 - Case study: The importance of feedback on context limited environments**, where the effective importance of feedback will be evaluated, considering conversational environments with much more limited context (amount or meaning);
- **Chapter 6 - Conclusion and Future work**, where the final conclusions of this work will be drawn, along with the description of the main contributions and also some suggestions for future improvements.

2

Background and State of The Art

In this chapter, we will present and discuss some theoretical and practical topics, which supported the content and development of this work. This review will also take into account related research projects, focusing on their strengths and weaknesses, in order to develop a more robust system.

2.1 Pervasive computing

Over the last few years, discussions regarding pervasive computing have become more and more common, seeing some great development [1, 2, 8, 9], due to the overall innovations and possibilities inherent to this concept, and mainly by the fact that our surrounding environment is becoming even more monitored every day [10, 11, 12].

As said by Saha and Mukherjee [2], this concept can be defined as the "growing trend of embedding computational capability into everyday objects to make them effectively communicate and perform useful tasks in a way that minimizes the end user's need to interact with computers as computers". Thus, in a pervasive system, a "device can be a portal into an application-data space, not a repository of custom software that a user must manage" [2], which means that, unlike traditional computational systems, such a system needs constant information about the surrounding environment. This information can be gathered through various sensors, like location ones, that provide information regarding the position of a specific user or object, so that a specific location-based task can be performed.

However, pervasive computing can be sometimes confused with ubiquitous computing. Even if these two concepts are complementary, they are relatively distinct. The latter defines that the computing power should be omnipresent, therefore devices should be scattered everywhere [13], while pervasive computing focuses in noninvasive devices, which do not necessarily need a user explicit interaction in order to perform a specific task. This is the reason

why pervasive computing will be one of the main pillars of this work and, in order to better understand this concept, it is important to know its driving areas as well as its main characteristics.

2.1.1 Driving areas

Pervasive computing exists mainly due to two very distinct areas, in both its genesis and purpose – Distributed computing and Mobile computing. As shown in Figure 2.1, there are some research problems in pervasive computing related to those in mobile computing and distributed systems, mainly due to the emergence of new and more complex scenarios, when moving from left to right in the figure.

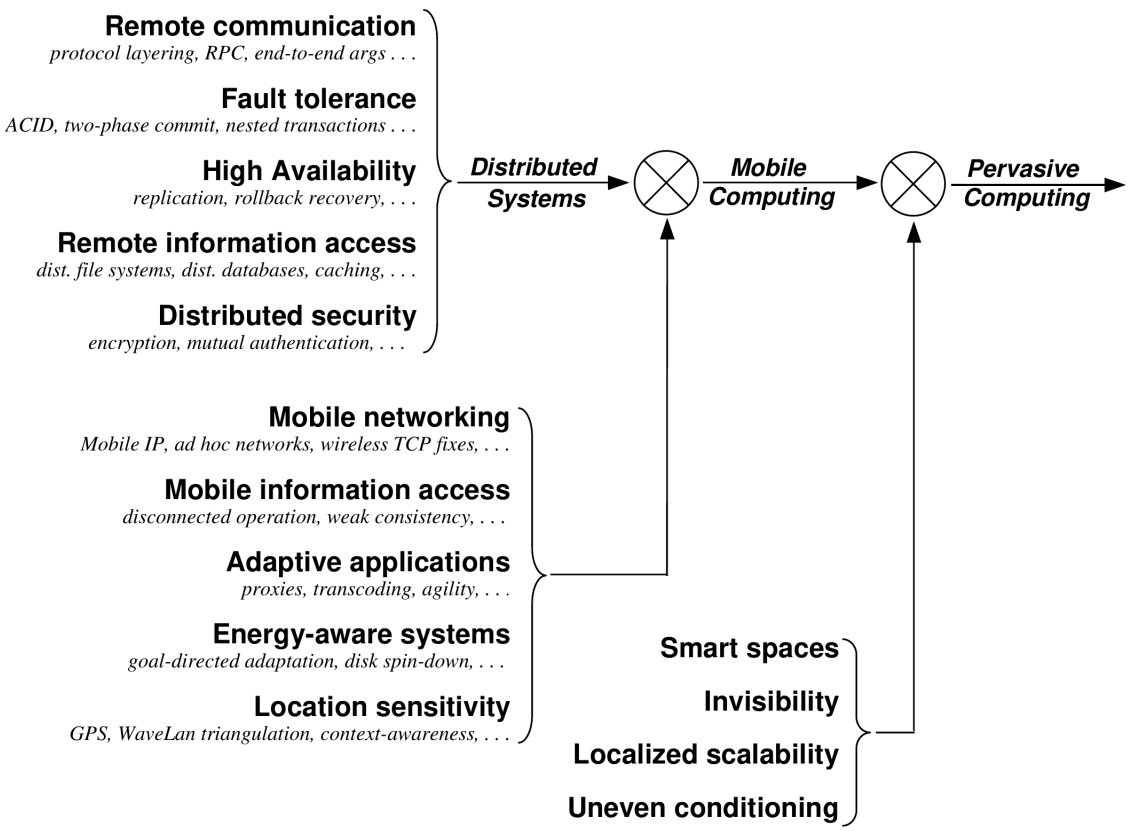


Figure 2.1: Taxonomy of computer systems research problems in pervasive computing [1].

2.1.1.1 Distributed computing

The distributed computing, together with the emergence of the communication networks, made it possible to share computational resources between personal computers, as well as the remote and transparent access to the information. All these mentioned features are supported by fault-tolerant, high availability, and security mechanisms [1]. As such, Saha

and Mukherjee [2] also mentioned that the distributed systems enabled that "the computer they use to access these information has become largely irrelevant ".

2.1.1.2 Mobile computing

Mobile computing emerged from the integration of cellular technology on the web, enabling the user to access the same information, regardless of the location. Being available anytime, anywhere, powers pervasive systems, as they are a superset of mobile computing, according to its motto "*all the time, everywhere*" [2].

2.1.2 Main characteristics

The two driving areas described above played an important role in the emergence of pervasive computing. Thus, its main characteristics are, once more, an extension of the characteristics of the latter, but considering the environment where pervasive computing operates. Therefore, the pervasive computing main characteristics are the following [1]:

- The use of "smart" spaces, where space can be defined as any limited area, such as a room, where the placement of smart sensors and devices allow the provision of computational capacity to an environment that is not necessarily intelligent. This can be achieved through monitoring and control mechanisms that provide information to a smart system that perform specific tasks as needed, like automatically adjusting the temperature of a house division;

- Invisibility, which means that a pervasive system should cause as little distraction as possible, always acting in a logical way so that the user does not encounter unusual situations that cause him to have to leave his perception at the subconscious level;

- Localized scalability, that is needed due to a large number of interactions between the users and the system, which consumes bandwidth and energy. Instead of relying on distributed systems, where the physical location of a user is not relevant, in a pervasive system it is pertinent since interactions triggered in a location further from the node (usually considered less relevant) can not overload users that are interacting with a system in a physically nearer place, so it is up to the system to reduce distant or less relevant interactions;

- Masking of uneven conditioning, since the penetration of the pervasive system in the environments is very dependent on external factors to technology, like the organizational structure or economic environment. As such, a place like a conference room, endowed with a considerable amount of technological devices will be a much smarter environment than any other. With this in mind, and considering it is necessary to reduce the asymmetries experienced by the user in terms of the "smartness" of an environment, it

is a good idea to have a personal space considered smart, since it is where the user will spend most of his time.

2.1.3 Computational model

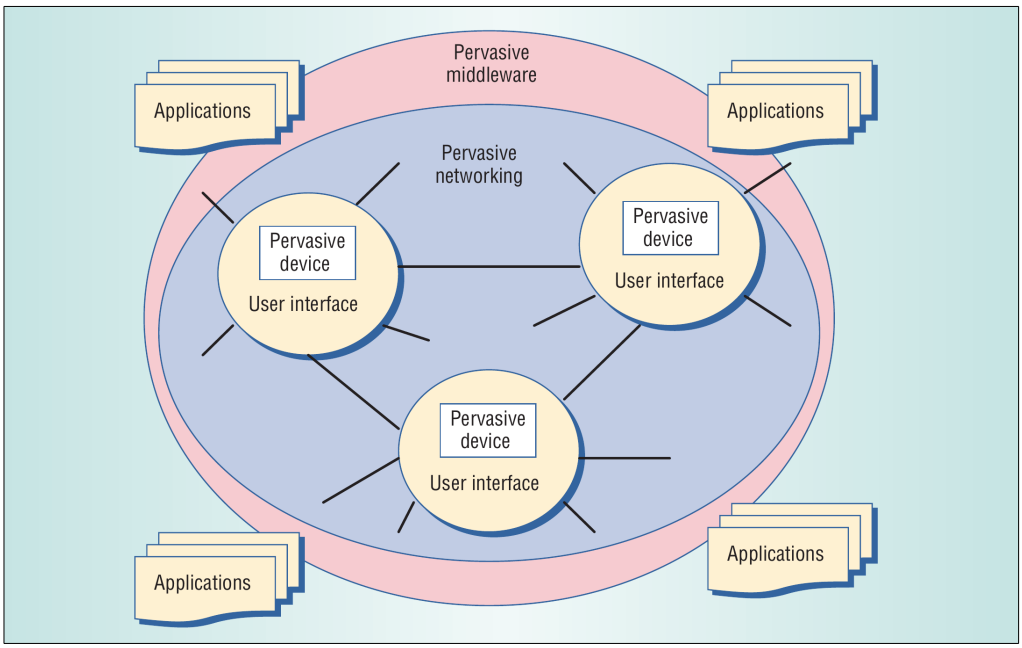


Figure 2.2: Pervasive computational model [2].

Considering the characteristics mentioned in Section 2.1.2, when it comes to building a pervasive system, there are some components that should be taken into account (Figure 2.2) [2]:

- Devices of any kind, from screens, keyboards and cellphones, to cameras or temperature sensors, that allow gathering data from objects in order to identify its characteristics and behaviors, regarding a pervasive system;

- Pervasive networking, that should accommodate in a scalable and fault-tolerant way a large number of devices that will be constantly exchanging data;

- Pervasive middleware, that provides an interface between the network and the applications developed to the pervasive system. This interaction field aims to reduce the complexity associated with the system, masking the existing heterogeneities and making the system as invisible as possible. This middleware is typically constituted by firmware and software with a client-server or a Peer-to-peer (P2P) architecture, but may also be in the form of a User Interface (UI), which enables the use of interactive tools;

- Pervasive applications, which should also handle the middleware and the pervasive network effectively, contrarily to the conventional ones, dealing with potential problems that may arise on a large scale.

2.2 Context-awareness

Context-awareness is a research area that is intrinsically related to pervasive computing, since the first refers to the "application's ability to adapt to changing circumstances and respond based on the context of use" [14].

As a pillar of the pervasive systems, context-awareness is a framework used in many different research areas. In health, Bricon-Souf and Newman [15] developed a project of a bed with a built-in screen, used as an entertainment device by the patient or as a medical record consultation by the doctors and nurses. The bed would have information regarding the type of user interacting with the device and would adapt the presented information to the necessities of the person. Context-awareness is also present in the networking and telecommunications, with an example of a project aiming to define a network selection logic using users' context information, as the location, coverage degree or connection speed [16]. Another relevant area, which is tightly related to this work, is text and speech, where context information regarding the source can be gathered, using features like keywords and textual metrics to perform, for instance, sentiment analysis [17]. This information can also be used as part of a pervasive system in order to counteract bad mood executing some tasks that are known to make the user happier.

2.2.1 Context

To better understand the context-awareness concept, it is necessary to define what is considered to be a context. According to the dictionary, a context is "the set of circumstances or facts that surround a particular event or situation" [18]. Despite being correct, this definition is too vague to be used as a tool in a human-computer interaction environment. As such, a new definition has emerged, which defines context as "any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object" [19]. Even if this definition still seems broad, it is because the context-awareness framework deals with a diverse and high amount of inputs, bot implicit and explicit, enabling almost every application that reacts to some type of input to be considered context-aware.

2.2.1.1 Context information

As proposed by Dey, Abowd, and Salber [20], the context information can be divided into four categories:

- Identity, where an entity is cataloged through an identifier that should be unique in the application scenario;
- Location, which should describe as best as possible the spatial information, through geographical coordinates, elevation, as well as other information that can be used to establish spatial relations between entities;
- State, defined by the inner entity's characteristics that can be monitored. For instance, when talking about a physical local, it can be defined as the current temperature, humidity, or noise level. However, when referring to people, it can be the vital signals, state of mind, or activity that is currently performing.
- Time, that allows the observation of continuous differences or even the order of events, by adding historical information to the context.

These context phenomena are usually observed by asynchronous computational systems, given that they are composed of heterogeneous devices that communicate through a wireless network, prone to high delays, loads and relays.

2.2.2 Main features

Context-aware applications should support a given set of features, as proposed by Pascoe (1998) [21]:

- Contextual sensing, where a device should handle the different states of the environment (the context) and present them to the user in an intuitive manner, augmenting his sensing system. An example of this feature, presented by the author, is the Global Positioning System (GPS), which offers the user a simple way to know, with high precision, the current location.
- Contextual adaptation, where a system uses context information to efficiently and transparently adapt its behavior. This can be exemplified through the decrease of the screen luminosity when in a darker environment;
- Contextual resource discovery, that enables the system to locate resources that share total or part of the context, in a way that is possible to explore both when in the same scope. A good example is a device with a small screen which will transmit information to a nearby screen of larger dimensions when it exists;
- Contextual augmentation, where the system augments the environment where it is present, completing the current context with additional digital information. The digital tourist guides are a good example of this feature, by enabling the users to check for additional information when they are near a tourist attraction.

2.3 Text mining

Since this work will focus on handling and extracting knowledge from text, it is important to do some research in the text mining area. Text mining refers to the "process of extracting interesting and non-trivial patterns or knowledge from text documents", and can be seen as an extension of data mining, that focuses on discovering and extracting knowledge from structured databases [22].

It is estimated that more than 80% of the companies data is stored in a textual way [22], which makes the text a great study object and it is used in the most different fields. For instance, in linguistics, it was used in text summarization, by building a graph using document keywords [23]. In health, a totally different field, Gong *et al.* [24] developed an application that uses text from the medical reports in order to build a database with searchable and useful information, that aims to simplify the patient analysis based on his history and on the history of other patients.

Natural Language Processing (NLP) is a complementary area of text mining that is gaining a lot of popularity over time. As described by Chowdhury [25], it is an "area of research and application that explores how computers can be used to understand and manipulate natural language text or speech to do useful things". Thus, it focuses more on syntax handling as a way to understand and produce text following the correct writing rules. However, in the context of this work, the main goal is to somehow understand and process context (and text) through semantic analysis, which is the main reason why NLP topics will not be further detailed, even though they are worth mentioning.

2.3.1 Main pipeline

The text mining pipeline involves, generally, three main stages, as shown in Figure 2.3 [3]:

2.3.1.1 Pre-processing

As the first step in the application of text mining techniques, the pre-processing can be divided into some tasks:

1. Data collection - before starting the text-mining pipeline, it is necessary to select a data source. Since text mining works with digital information, data must be transcribed digitally, by using, for instance, detection mechanisms like the Optical Character Recognition (OCR) [26]. Privacy and security related issues should be taken into account when performing this task, using, for instance, data usage consents;

2. Data cleaning - the cleaner the data source the better its quality, since the knowledge extraction will be more effective. This cleaning procedure can be performed using innumerable techniques, like stopword removal, where words that have no informative value are discarded, as prepositions or the most commonly used words according to a

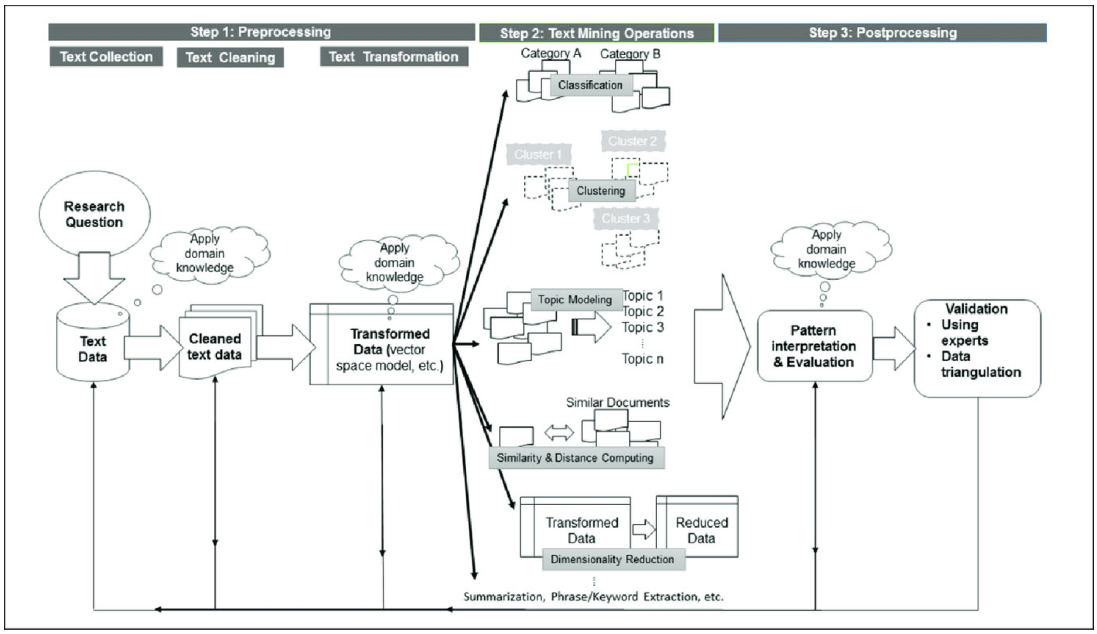


Figure 2.3: Text mining main pipeline [3].

specific language; lemmatization, that tries to turn a word into its lemma (canonical form) or stemming, that reduces a word to its morphological root. By applying all these techniques, some words are shortened or even discarded, reducing the size of the main text, which enables a faster analysis;

3. Data transformation - in this step, the text will be converted into mathematics structures, like matrices, where columns are usually the features and rows the study object, like a line, paragraph, or a document. These features are highly individualized and related to the study object, making them a key to success, as a good feature engineering step will potentiate the final results. Features like the Term Trequency (TF) and Term Frequency-Inverse Document Frequency (TF-IDF) [27] are two examples used in most text mining problems.

2.3.1.2 Text mining operations

After the pre-processing step, and having the data represented logically from the algorithmic side, it is time to apply some analytic methods in order to extract knowledge from the study object. Some innumerable approaches and models allow the achievement of the goals proposed, and, thanks to the research effort that has been made lately in the area of Machine Learning (ML), there are more and more every day. The most common are :

- Classification [28] that aims to assign a specific category to a document;

- Topic modeling [29], that seeks the automatic extraction of relevant topics from a document;
- Clustering [30], which tries to interpret relationships between documents and understand which relates the most;
- Similarity/distance computation [31], that tries to quantify the similarity between two documents through a specific measure;
- Dimensionality reduction [32], used, for instance, to perform text summarization.

2.3.1.3 Post-processing

The post-processing (or analysis of results) is intrinsically related to the application of the models mentioned in Section 2.3.1.2 since its goal is to analyze their output and verify if the results are as expected. If this is not the case, changes should be made in this step in order to adjust the results and bring them closer to the expected. This stage is generally performed by experienced people in the domain as they find it easier to analyze and interpret the results in order to decide what changes can improve the results.

2.4 Information Retrieval

Nowadays, we are regularly confronted with more or less complex information from different sources. As such, two areas emerged whose main goal was to manipulate this type of data, namely the one present in documents – Data Retrieval (DR) [33] and Information Retrieval (IR) [34]. Even if both focuses on finding documents in a collection that contains the keywords searched by the user, DR targets a strictly satisfaction of specific conditions, defined by regular or mathematical expressions, since in a system of this kind certainty is an important measure, whereas presenting an object that does not fit into the defined search is an undesirable result. On the other hand, the IR is more flexible in terms of results, since its study object is generally natural language that, by itself, is a source of ambiguity. As such, the extraction of information and knowledge has to deal with complex subjects like syntax and semantics, as well as to decide which information or type of information can be considered relevant to the current search context. Thus, a more formal definition of IR can be the "representation, storage, organization and access to information items" [4], being this information able to take the most varied forms, like documents, web pages or multimedia objects.

This area will be one more pillar of this work given that there will be necessary mechanisms and algorithms to handle and gather knowledge from a vast set of documents.

2.4.1 Base components

The main goal of the IR is intrinsically dependent on two base components [4]:

- The user, responsible for the translation of the necessary information into a language noticeable by the system (generally a query). By having such a fundamental role, a user can compromise right from the beginning the usefulness of the results returned by the system if the query was not correctly built. The role of the user is based on two main tasks – retrieval and browsing. The introduction of a query that more or less accurately describes the search terms is an example of a retrieval task, while browsing occurs when a user, based on a specific repository, iterates over the documents in order to find relevant information. In this case, unlike the first one, there is no explicit information about the search terms, being the user, by interacting with the system, the one to filter the information;

- The documents, usually represented by keywords or indexing terms, that simplify the storage and search tasks. However, with the development of the computational power and storage, the documents started to be stored in an almost integral way, applying just some pre-processing techniques (mentioned in Section 2.3.1.1), with the intention of saving some space and processing resources, by discarding data that does not add any informational value. Beyond its content, also the meta-data, such as the internal structure (e.g, chapters, sub-chapters), authors and creation date are relevant to the IR system, as they enable a better understanding of the document as a whole.

2.4.2 Main stages

Since IR mostly processes digital information, there are three main stages around that study object generally performed, which are listed in more detail in Figure 2.4:

- Information representation, where the main goal is to find effective ways to represent a set of informative data (or documents, as it is usually called) so that it can be easily identified to post-process;

- Search, offering more or less formal methods, through queries that should be perfectly interpreted and understood by the system;

- Information gathering, that focuses on the search mechanism that will enable the retrieval of relevant documents regarding the performed query.

It is important to notice that, even though interactions with an IR system are made through queries, similarly to a Database Management System (DBMS), they operate differently, as the first performs results ranking, which means that, even if a document does not exactly match the query, it can still appear in the results list, even if with less relevance.

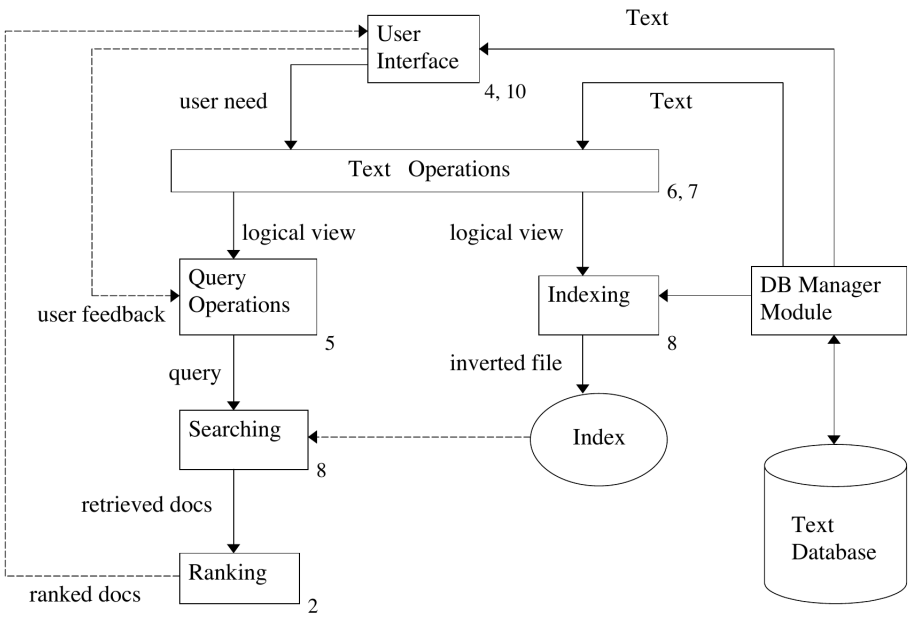


Figure 2.4: The process of retrieving information [4].

2.5 Recommendation Systems

Recommendation systems are a subarea of IR and Artificial Intelligence (AI), that are described as being "powerful tools and techniques to analyze huge volumes of data, especially product information and user information, and then provide relevant suggestions based on data-mining approaches" [35].

2.5.1 Types of recommendation systems

The advent of online stores and targeted advertising has led to the continued evolution and prominence of recommendation systems, which are subdivided into different types:

- Collaborative filtering (Figure 2.5), which is the most simple and usual form of recommendation systems in which the system filters a large number of alternatives, using user preferences. This type of system is used in movies and shows streaming products: if a user A shares the same movies and shows preferences with a user B, when one of them sees new content, it will also be recommended to the other, since it has a high probability to suit the other user's taste. The collaborative filtering can also be considered as user-based or item-based. In the first, the recommendations are made through users neighborhood, where this neighborhood should be defined initially, analyzing all the users and their preferences. The item-based is study object-oriented since the relationships are computed based on similarities between the items, using their characteristics;

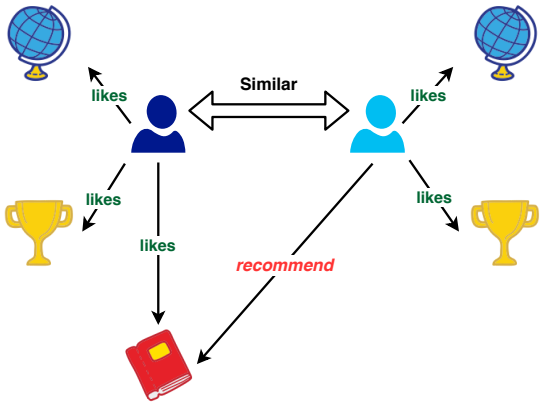


Figure 2.5: Collaborative filter process.

- Content-based (Figure 2.6), that uses a little different approach from the last one, since it uses user and item properties and not only their characteristics. In order to do this, it is necessary to create a user and item profile, analyzing the descriptive content of both and gathering useful information. A practical example of this type of filter is the recommendation based on the movie genre, where, if a user *A* watched *Deadpool*, the system is very likely to recommend *Ant Man*, as they are both *Marvel* movies.

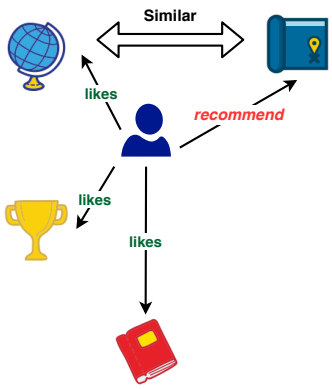


Figure 2.6: Content based filter process.

- Hybrid (Figure 2.7), that merges the last two filters aiming to cancel or decrease their disadvantages and potentiate their advantages. This type of filter can be much more accurate, since it uses more information and offers more useful data to the recommendation task, based not only in their neighborhood preferences but also in their content;

- Context-aware, that focus on working with the current context, since preferences by themselves may not be clear enough, as they vary greatly with the context (e.g, the hour of the day or state of mind).

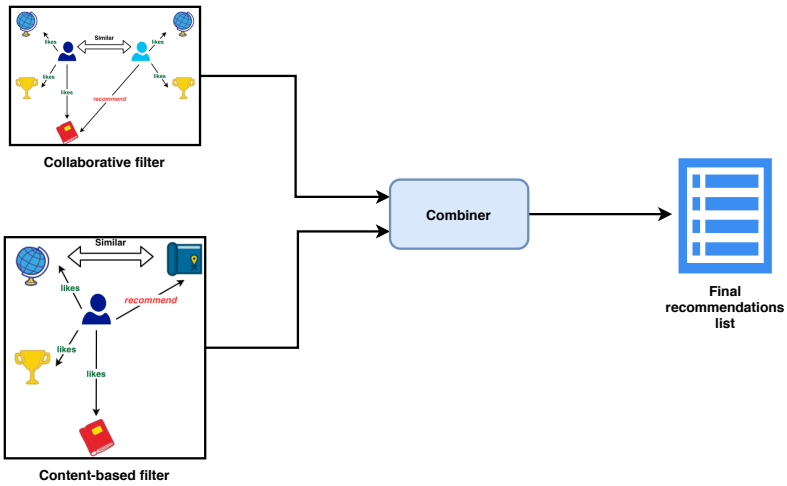


Figure 2.7: Hybrid filter process.

2.5.2 Challenges and solutions

Recommendation systems face a wider range of challenges related to data and preferences, namely [36]:

- Cold start, a very common problem in AI and ML, that occurs due to the lack of data in an initial system state. An example is the creation of a movie streaming service, where it would be impossible to make suggestions when there is no evaluation information about the movies or when there is no historical data about the users' views. A possible solution to the cold start problem is gathering user data through other sources, like social networks, in order to have an initial knowledge base to support recommendations;

- Scalability, a frequent issue in the computational systems, and that result in the need to accommodate a high number of users. To achieve this, the computational resources must adapt transparently, not endangering possible erroneous recommendations;

- Sparsity, common in a system with a high number of users and items, and that occurs when a user relates to a few items or even did not rate any. This issue is sometimes described as a lack of knowledge. In this case, the use of a hybrid system may solve this problem, since it combines both content and neighborhood information, creating more knowledge about the current user condition;

- Privacy, since recommendations are based on user information (e.g, name, age, email) as well as personal preferences, characteristics that can be misused in an information leak event;

- Over-Specialization, a common issue on content-based systems, since this is limited to only analyzing the content, and lacks on recommendation diversity, forcing the user to

focus on a very specific item niche, and making it impossible for him to discover new and different items. This problem can be minimized by using neighborhood information as it allows to suggest items outside the usual recommendation area;

- Predictability, a common aspect in these types of systems, since, even if the recommendations are made diversely, they can be considered familiar to the user, which lacks the surprise factor these systems should offer.

2.6 Learning to Rank

With a strong connection with IR, Learning to Rank (LTR) is "in the intersection of ML, IR, and NLP", and its main objective is to "perform ranking using machine learning techniques" [37].

The ranking concept can be defined as "an act or instance of indicating relative standing" [38]. As its definition, its use is also general, being applied in many areas, namely sports and competitions. Also in the IR area, it plays a key role, allowing for the most concise and context-aware presentation of search results.

2.6.1 Learning process

Since LTR has the use of ML as its main basis techniques, it also shares its main process steps, based on the supervised learning methodology [37]:

1. Dataset building, composed usually by queries, documents, and their relationship, by associating a query content to a document identifier (or vice versa). The relevance information, when it exists, it is also included in the dataset, since it provides useful insights for further analysis;

2. Feature engineering, where several features will be chosen as being relevant to the problem, based on the dataset thorough analysis. In LTR area, are used features like the BM25 [39], a probabilistic model that, given a document, represents its relevance regarding a query, and the TF and TF-IDF already mentioned in Section 2.3.1.1;

3. Evaluation, performed by comparing the ranking list with the expected result and then using specific metrics that will be presented later.

Even if this process is similar to the one used on classical ML techniques, the ranking list building steps present some differences. In LTR initial phase, the score to each one of the documents is computed, and, then, based on that value, or sometimes adding additional information, the output label will be assigned. This label will serve as the base to sort out the results.

2.6.2 Learning approaches

As suggested by Joachims, Li, Liu, and C. Zhai [40], LTR algorithms can be categorized as:

- Pointwise, with the input space as the features of each document, the output space as the relevance value of a document given a query, and the hypothesis space containing the functions that are capable of predicting the relevance of a document against a query taking into account a pair (query, doc). This approach resembles a regression or classification problems. *Staged logistic regression* [41] and *Prank* [42] are two examples of algorithms that follow this approach;

- Pairwise, where the input space is composed by the features of a document pair, the output space is their optimal order, and the hypothesis space is the functions capable of predicting the order of two documents against a query, in order to minimize the number of necessary inversions in the result list. Algorithms that fall under this category are *Ranking-SVM* [43], that resort to Support Vector Machines (SVMs) and click data as additional information, *RankNet* [44], that uses a strategy based on Neural Networks (NNs), and *LambdaRank* [45], that was an improvement to the latter in terms of speed and precision, based on the modification of the cost function;

- Listwise, with the input space representing the features of all the documents associated with a query, while the output and hypothesis space can be one of the two approaches mentioned above, optimizing the results based on the average of all queries. In this approach fit algorithms like *AdaRank* [46], a boosting algorithm applied to the IR area and *LambdaMART* [47] that joins the above mentioned *LambdaRank* with Multiple Additive Regression Trees (MART).

2.6.3 Evaluation methods

Regarding the evaluation of the results performed by LTR algorithms, two approaches can be taken regarding its measurability:

1. Numerical measures, such as the indexing times (documents per minute, for instance), query times, and some binary attributes, as if it is a free service or if the UI is appealing;

2. User-centered measures, that can be subjective and dependent on the setup and the target. In the case of a web engine, it can evaluate whether the user has completed his search task, working as mean towards an end, rather than the end itself, as the goal is to get the expected results.

2.6.4 Evaluation metrics

Just like the usual classification model in the ML context, are also used numerical metrics to evaluate the results in LTR, like the Accuracy, given by:

$$Accuracy = \frac{true\ positives + true\ negatives}{true\ positives + true\ negatives + false\ positives + false\ negatives} \tag{2.1}$$

Even if it is a useful metric and often used in the classification problems, it can not provide much information by itself when there is a large discrepancy between the number of samples in each class. To tackle this, there are usually two other metrics used – precision and recall:

$$Precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{2.2}$$

$$Recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{2.3}$$

Using them individually will not allow drawing very good conclusions either. For instance, when returning all indexed documents regarding a specific query, the recall will be high but the precision will be very low. As such, a new measure emerged, which results from the harmonic mean between the last two – F-score:

$$F_\beta = (\beta^2 + 1)\frac{recall \cdot precision}{recall + \beta^2 \cdot precision} \tag{2.4}$$

, where β defines the recall importance relative to the precision (a higher β means that the precision will have more weight when computing the F-score, and a lower β will result in a higher recall weight).

The above-mentioned metrics are efficient when it comes to evaluating the performance of an IR system as a whole, since they compute a single value regarding a list of results, regardless of the query. This approximation may not be desired when the goal is to analyze each query individually and its result list. In these cases, there are metrics like:

- Mean Average Precision (MAP) [48], that computes the Average Precision (AP) mean, where this represents the average of the values resulting from the precision depending on the recall, for each result in the list:

$$AveragePrecision = \frac{\sum_{i=1}^{n}(P(i) \cdot relevance(i))}{\#relevant\ documents} \tag{2.5}$$

$$MAP = \frac{\sum_{i=1}^{\#queries} AveragePrecision(i)}{\#queries} \tag{2.6}$$

- Normalized Discounted Cumulative Gain (NDCG) [49], a normalized version of the Discounted Cumulative Gain (DCG), where the latter uses the result relevance as a gain measure (often called usefulness). Starting from the top of the results list, the gain is decreased whenever a document with lower rank is found. Using the DCG value regarding a specific position as well as its ideal value (the max DCG one), it is possible to compute the normalized value to that position (the NDCG):

$$DCG_p = relevance_1 + \sum_{i=2}^{p} \frac{relevance_i}{\log_2 i} \tag{2.7}$$

$$idealDCG_p = \sum_{i=1}^{ordlist} \frac{2^{relevance_i} - 1}{\log_2(i+1)} \tag{2.8}$$

$$NDCG_p = \frac{DCG_p}{idealDCG_p} \tag{2.9}$$

  , where *ordlist* represents the result list, ordered by its relevance;

- Mean Reciprocal Rank (MRR) [50], a mean of the Reciprocal Rank (RR) of each query, where the latter concerns the multiplicative inverse of the position of the first hit:

$$MRR = \frac{1}{\#queries} \sum_{i=1}^{\#queries} \frac{1}{rank_i} \tag{2.10}$$

2.6.5 Online Learning to Rank

Following the same fundamentals of (offline) LTR, described in the previous sub-sections, Online Learning to Rank (OLTR) emerged as a different approach regarding the source of data. Rather than using datasets built by domain experienced users, OLTR has only access to data regarding the interaction between the user and the system. This data can take many forms, from mouse movements to query reformulations or clicks, the one often used since it usually represents the user preferences in a simple way. This approach is considered to be a merge between LTR and Reinforcement Learning (RL) fields.

Unlike offline LTR, OLTR is constantly struggling with the so-called exploration-exploitation dilemma, that opposes two behaviors: the system only returns documents considered to be relevant, not being able to acquire feedback about other potential good results, or the system returns documents that can be a good source of feedback which makes it possible to improve the system quality over the time, despite having a high risk of returning results that may not be relevant to the user [51].

Some OLTR algorithms are Dueling Bandit Gradient Descent (DBGD) [52], whose learning process is done through successive interleaved comparisons (interleaved tests) between the current ranker and the new exploratory one, and, more recently, Multileave Gradient

Descent (MGD) [53] that, instead of only compare a pair of rankers, it does interleaved tests between a wider set of them.

2.7 Reinforcement Learning

RL is a subarea of ML, and it describes the "the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment" [54]. Instead of giving the "right" answer, as it occurs on supervised learning approaches, the agent receives a reward value that indicates whether the agent is doing well or poorly. The RL approach will be important in the context of this work as it will be necessary to learn how to make a suggestion based on the environment changes (Figure 2.8).

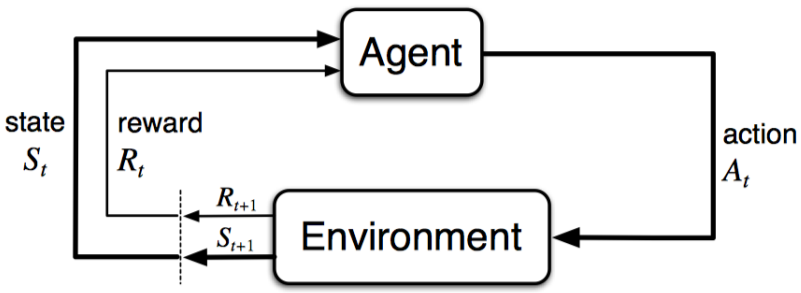


Figure 2.8: The RL feedback loop.

2.7.1 Reinforcement Learning definition

The standard RL models are usually defined through the Markov Decision Process (MDP) formalism [55], defined as the (S,A,P,R,γ) tuple, where:

1. **S** is the set of states, like the position, orientation and the surrounding of the agent, in a game map;
2. **A** is the set of actions, like moving right, left, up or down in a game map;
3. **P(s, a, s')** are the transition probabilities of being in the state s and taking action a to go to the new state s';
4. **R(s, a, s')** is the reward function that computes the reward of moving from state s to s' taking action a;
5. $\gamma \in [0, 1]$ that is the discount factor which is a measure of how far ahead in time the algorithm looks.

Taking into account the model, which defines the dynamics of the environment, RL algorithms can take two distinct forms – *model-based* and *model-free*. Model-based approaches

assume that the model learns the transition probability through the definition described above in the tuple rationale, where the agent will know how to behave when it finds a state previously-learned. However, these algorithms scale up badly when the state and action space grows considerably. Contrarily, model-free algorithms rely on trial-and-error to learn its policies.

Algorithms can also be divided accord to their policy use, where an agent can be *on-policy* or *off-policy*. In the first one, the agent learns based on the past state-actions decisions, while the latter uses an action obtained from other policy (that is not necessarily the last one).

Even if the model definition presented above is still one of the most used ones in standard RL problems, sometimes it cannot solve the issue straightforwardly, as it is not possible to define or enumerate some elements of the tuple. In these cases, some other approaches must be designed respecting the RL goal of learning with trial-and-error described above.

The goal of the whole process is to maximize the expected value of the total reward, by finding the optimal policy (the action for which the value function is optimal).

2.7.2 Reinforcement Learning algorithms

Although the scientific community regularly proposes new RL, some deserve to be mentioned due to the contributions to the state of the art, namely:

- *Q-learning* [56], an off-policy, model-free algorithm based on the Bellman equation [57], whose goal is to optimize the Q-value, that refers to the long-term return of the current state, taking an action under a policy. The *Value iteration* [55], an algorithm that tries to find the optimal value function, and the *Policy iteration* [58], that manipulates the policy directly, rather than finding it indirectly via the optimal value function, are Q-learning based algorithms that try to solve this problem;

- *State-Action-Reward-State-Action (SARSA)* [59] is very similar to Q-learning, however, it is an on-policy algorithm, where the Q-value is learned based on the action performed by the current policy instead of the greedy one;

- *Deep Q Network (DQN)* [60] leverages the above-mentioned Q-learning by using NNs to estimate the Q-value function, using the current situation as inputs of the network, that outputs the Q-value for each of the possible actions. It also applies two important techniques to the training process – Experience replay and Separate Target Network. The former tries to solve the problem of the high correlation between samples by randomly selecting the stored samples to update the learning process, while the latter is the reset of the target network at every C steps, which resulted in more stable trainings with fewer fluctuations.

2.8 Related work

In this section, we will describe some projects and research proposals, which are related to the work developed in this dissertation.

Some products regarding the unified organizational communication system are already being commercialized, namely:

- Fuze, a cloud-based communication platform that offers voice calls, video-conferences as well as Instant Messaging (IM) with presence management, file sharing, and also collaborative agendas. Furthermore, with a focus on productivity, the platform allows to add third-party services and features;

- Cisco Webex, a unified communication platform that focuses on corporation collaborative processes. Besides IM, presence management (showing the best way to contact a friend - the available device), video, and audio calls, the platform also allows creating virtualized conference rooms where it is possible to edit a collaborative whiteboard. The communication ciphering and the connection to the cloud are also important features. There is also an Application Programming Interface (API) available in the *Cisco Spark for Developers* that allows the development of client-specific features;

- Unify, a range of unified communication systems: *OpenScape Enterprise*, *OpenScape Fusion* e *Circuit*. The first two focus on unified communication technology, while the latter also concerns User Experience (UX) aspects, supporting voice, video, screen, document sharing (even in external applications), IM, and it can be integrated with external agendas to schedule meetings. It also has features like contextual search, making it possible to search for documents, contacts or conversations, which makes it the most relevant product in the context of this work.

Even if the above-presented services focus on the unified communications and in simplifying some organizational processes, all of them lack in pervasive and intelligent ways to manage its contents and communications, delivering this task to external developers that use its API to create plugins that fulfill those needs.

Besides commercially-available products, the scientific community has also proved to develop projects relevant to this work. As such, all the projects mentioned below collect conversational data, applying some techniques that were already explained in the previous sections, in order to build implicit queries (which means that they are not explicitly introduced by the user) that are posteriorly sent to an IR-based system.

Popescu-Belis *et al.*, in a project named AMIDA Automatic Content Linking Device (ACLD) [5], created a Just-In-Time Document Retrieval (JIT-DR) system, whose use case was a meeting environment, where the system performed document suggestions that might be relevant to the conversation topic. The data was collected through an Automatic Speech

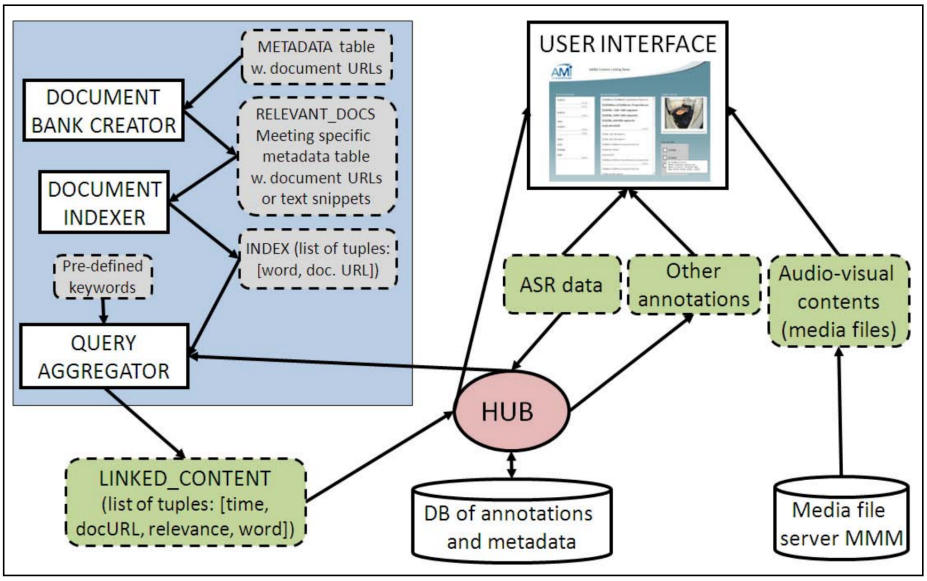


Figure 2.9: ACLD architecture [5].

Recognition (ASR) at fixed time intervals to ultimately perform a pre-built database search. The collected data were pre-processed in order to only retain the keywords, with the application of some techniques mentioned in Section 2.3.1.1, such as stopword removal and stemming. The keywords were then grouped to build an implicit query and sent to the IR system, whose task was to return relevant documents, ordered by relevance. The overall system architecture, depicted in Figure 2.9, was based in four main components:

- Document database, which should gather all the documents considered to be relevant to the current meeting;
- Indexer, whose task is to create an index over the database in order to make the search task more efficient and effective;
- Query aggregator, that performs searches at fixed time intervals, using the keywords extracted from the meeting, and products a documents list considered to be relevant at the time;
- UI, responsible for showing the results gathered from the IR in a user-friendly manner, as well as the meta-data and summary of the last meetings.

Given that the proposed techniques and architecture are simple and easily extrapolated, they can be considered a good starting point when it comes to building a suggestion system. Thus, they will be used as the baseline for our system. However, the simplicity associated with their work raises some questions when it comes to the ability to deal with complex and noisy input. Unfortunately, nor this paper, neither subsequent publications, provided any results, and so the effectiveness of this architecture is unclear.

Habibi and Popescu-Belis (one of the authors of the previous work) studied the problem related to the extraction of keywords in a conversational environment [6], based on the ACLD system described above. They focused their efforts on solving how to extract multiple topics out of a conversational context, rather than a single one, which introduces a lot of uncertainty when analyzing the main conversational topic and making a suggestion based on it. Compared to the previous work, and taking into account the latter proposed ACLD system, the main difference is an intermediary step before the implicit query formation (shown in Figure 2.10), which firstly tries to extract the main topics, based on the selected keywords, and only then builds topic independent implicit queries to submit to the IR system. The results of all these implicit queries are then merged, ranked based on the topic similarity, and only the top document of the results list gets to be suggested to the user. The system was compared to other existing keyword extraction methods through judgments of human raters, recruited via Amazon Mechanical Turk, and showed that their system "provides on average the most representative keyword sets, with the highest α-NDCG value, and leading-through multiple topically-separated implicit queries to the most relevant lists of recommended documents" [6]. It is important to mention that neither the ACLD project, nor this last work explored any forms of implicit or explicit feedback that allow AI agents to adjust the system as a whole, and not only focusing on query reformulation, as a way to enhance the usefulness of the suggestions.

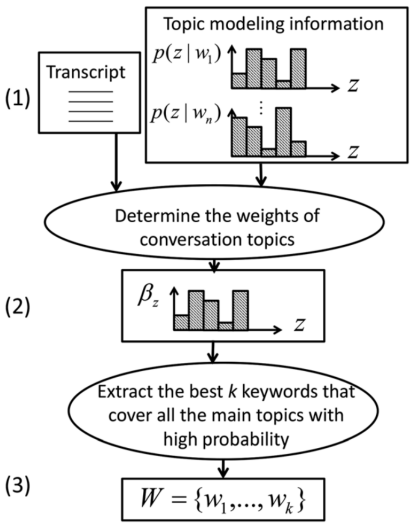


Figure 2.10: The steps of the proposed keyword extraction method [6].

Another similar work, with a common goal, was proposed by Tate and R.Nandwalkar [61], where a JIT-DR system was developed to make suggestions in an enterprise meeting environment. Even though the logic associated to this system was really similar to the ACLD one, in this case, the authors used a subsequent step following the extraction of keywords,

which consisted on using the Metaphone [62], an algorithm published by Lawrence Philips in 1990, composed of 19 rules that enhance the matching of words whose phonetics are similar. This algorithm was not used to perform word matching, which was its original purpose, but instead, to use its internal processes as a means to reduce word length and speed up the analysis step. The authors compared this new keyword extraction approach to a simpler one similar to the one proposed by Popescu-Belis *et al.* in the ACLD, and concluded that their new approach extracts more keywords for different inputs, leading to higher final accuracy.

Recently, Benítez-Guijarro, Callejas, Noguera, and Benghazi [63] proposed a different approach, whose goal was to tackle the lack of flexibility regarding keyword extracting techniques. They intended to "improve the current state of the art related to the interaction between nutritional coaching software systems and their users" by introducing a syntactic and semantic analysis of sentences instead of keyword spotting. To attain that, they analyzed and decomposed the sentences to find a common structure between the users' messages that are based on some general entities like actions, quantifiers, and ingredients. These entities and their dependencies allowed them to use a rule-based approach to interpret the meaning of the sentence, so that, by looking at the information associated with each valuable syntagma, through a knowledge database, the nutritional information associated with the sentence could be computed. Even though this approach works well in a specific domain, such as the demonstrated nutritional context, where the authors achieved a 96% accuracy compared to the 44% of the keyword extraction, in a general context it is impractical since it would require the definition of a set of semantic rules to interpret the meaning of the input, and to be able to interpret every imaginable situation, which is not feasible.

2.9 Conclusion

This chapter showcased some areas considered relevant as theoretical and practical background, namely Pervasive Computing, Context-awareness, Text mining, Information Retrieval, Recommendation Systems, LTR, and RL. In each of these topics, some interesting projects were also presented which will serve as an inspiration to this dissertation.

We also analyzed some research projects considered as related work on organizational communication and pervasive recommendation systems. In general, all of them collected and processed conversational data to build implicit queries that were posteriorly sent to a knowledge database, such as an IR system. Even taking into account the proposal of techniques to globally improve the system performance, all of the considered projects lack in some aspects. Features such as the ability to handle noisy environments, and the use of intelligent approaches to enhance the system performance are examples of weaknesses of the evaluated works.

3

CLASSY prototype

The main goal of this work is to develop a system that can suggest information in a conversational environment. Information can be of the most various formats, from Portable Document Format (PDF) to simple text files. However, they must be text-based, since the techniques described in the following sections were developed to handle text. Therefore, from now on, we shall call documents to any piece of data or information. Project issues, reports, and articles are also examples of information that can be considered a document.

On the other hand, the conversational environment can also take many forms, ranging from tweets, usually with a limited number of characters, to chat rooms, where messages can reach thousands of characters. Here, there is an important requirement, which is the existence of a continuous context along with the conversation. In other words, a message sent at a given time should be somehow related to the context of the previous messages to allow the analysis and extraction of a general context.

To perform the processing and storage of the documents, and constantly analyze the conversational environment, it is necessary to implement a reliable and efficient architecture, compliant with the real-time nature of these types of environments. The developed system based on this architecture was named **ConversationaL Aware Suggestion SYstem (CLASSY)**. Its main goal is to always be sensing and analyzing the conversational environment to find contexts that are somehow related to the stored relevant documents, and to suggest them accordingly.

The holistic view of the overall architecture, illustrated in Figure 3.1, is essentially divided into three main modules – Context processing, Knowledge database, and Suggestions filter and cache – that will be described in detail below.

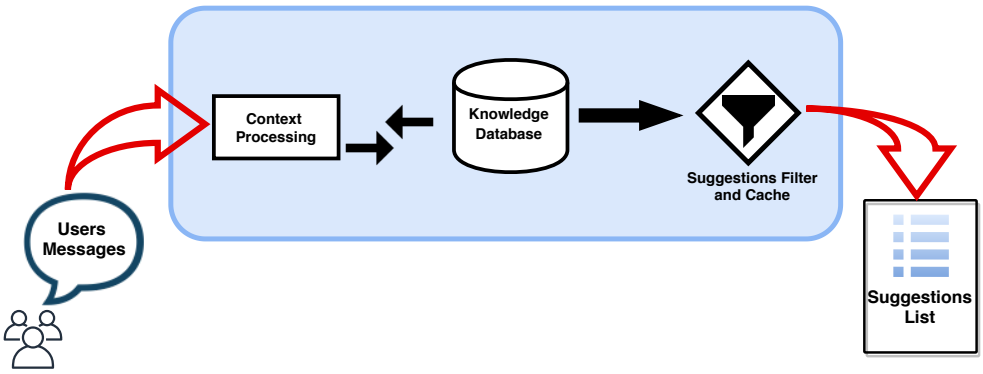


Figure 3.1: CLASSY architecture.

3.1 Context processing

The context processing module is where users' messages are analyzed and processed to find the most relevant tokens (also known as keywords), which will be used as a source of context. This module works by executing the following steps:

1. Relevant token extraction, which, taking the full conversational text segment as the data source, reduces its size by maximizing the amount of remaining pertinent information, or, in other words, gathers only the tokens considered to be relevant in the segment;

2. Context update, a stage where the context is updated with the new tokens obtained in the previous step. This context has a specific size and it is defined by the relevant tokens of only the last n queries (CLASSY parameter) to maintain its freshness and enable the transition between divergent conversational topics;

3. Implicit query formation, where a query is built with the disjunction of the context entries, boosted by its recency (the position in the context). An example of this stage is shown in Figure 3.2, where $Q*$ are the queries, $RTQ*$ are the relevant tokens from each query, and IQ the resulting implicit query:

3.2 Knowledge database

The knowledge database is one of the most important pieces of the whole architecture. It is responsible for robustly and efficiently pre-processing and storing the content of the documents considered to be relevant to the topics discussed along with the conversational segments so that it can be queried in real-time. It will use the implicit queries built in the module above (Section 3.1) as data source, and then retrieve the documents considered suitable to the received query.

```
Q1: Have you seen the last report?
Q2: The technical report?
Q3: Yes, the third one.

RTQ1: [last, report]
RTQ2: [technical, report]
RTQ3: [third]

IQ: (last report)^1 OR (technical report)^2 OR (third)^3
```

Figure 3.2: Implicit query formation.

As the documents can take many forms, this module should be capable of adjusting its global schema in an automatic way, by applying algorithms to discover the internal structure of each document, or programmatically, by enabling developers to define the structure of the documents, so that it can receive any type of document of any type of content topic, giving a general value to the system.

Due to the unknown document and content size, it should be capable of storing it in a non-direct way, by analyzing and extracting only query-relevant information and saving it in efficient structures, so it can be queried and retrieved as fast as possible.

3.3 Suggestions filter and cache

The Suggestions filter and cache module are responsible for post-processing the retrieved results from the knowledge database (Section 3.2) to minimize the missugestions and, consequently, to avoid irrelevant interactions with the users.

This post-processing step has to analyze each element of the retrieved results list and apply some filters or thresholds complying with its attributes (such as the number of matched tokens or the score), with the goal of decreasing the number of bad suggestions performed and, then, increase the certainty level associated with each suggestion.

Additionally, to increase the value of the suggestions and minimize the conversational intrusion of the system, a suggestion cache should be created, to avoid making the same suggestion in consecutive time-stamped messages, as it does not add any value to the user. This cache should also establish maximum time constraints in which a document is considered to be valuable again; thus, it can be discarded from the suggestion cache when it steps behind that limit since the topic flow is randomly distributed, and so a topic can be reused at any time.

3.4 Implementation

Having defined the overall CLASSY architecture, it is now time to define its implementation as well as the tools used.

The first big decision, that showed to have a big impact in all the system design, came up with the definition of the knowledge database. Although there are many ways to design and implement it, it is necessary to have an efficient system, easily extended with new features, simple to set up and, most importantly, that does not take a lot of time to implement and test since this was not the focus of this work. Additionally, another important aspect is to have a module capable of processing text documents, saving them in an efficient structure, and then retrieving results based on some type of query. The list of these requirements proved to be the definition of a specific type of system – an IR – which has the goal of "representing, storing, organizing and accessing information items", as discussed in section 2.4.

3.4.1 Search engine based system

Due to some past experience with a specific Search Engine – Apache Solr [1] – the range of options was reduced. Solr is a reliable, scalable, and fault-tolerant open-source search platform, written in Java, from the Apache Lucene project [2]. Its main relevant features in the context of this work are the "full-text search, faceted search, real-time indexing, database integration and rich document (e.g., Word, PDF) handling", which simplifies some work when it comes to extracting and storing information from the documents.

3.4.2 Context processing module implementation

The context processing module, described in section 3.1, was solely implemented as Python [3] methods. The high-level interactions, that will be described below, are shown in Figure 3.3.

The first stage of this component – the relevant token extraction – was also carried out using an analyzer defined in the Solr engine. It applied a set of processing techniques to the considered text, and it was used both at indexing and query time, to force the query tokens to follow the same rules and have a similar syntax to the indexed tokens of the documents. The analyzer pipeline is:

1. **UAX29 URL Email Tokenizer**, which splits the text field into tokens, using white spaces and punctuation as delimiters. Delimiter characters are discarded, except dots that are not followed by a white space, words with hyphens that contain numbers, internet domain names containing top-level domains validated against the white list in

[1] `https://lucene.apache.org/solr/`
[2] `https://lucene.apache.org/`
[3] `https://www.python.org/`

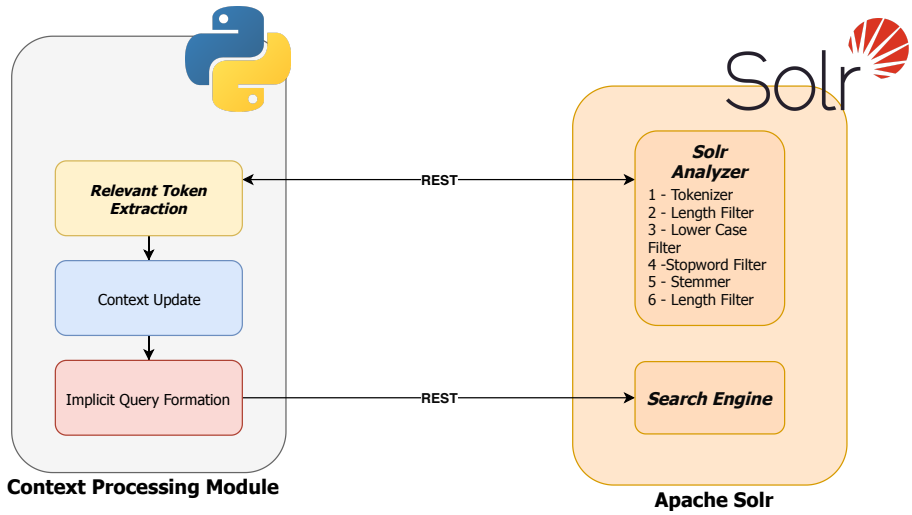


Figure 3.3: Context processing module interactions.

the IANA Root Zone Database, Uniform Resource Locators (URLs) and emails, Internet Protocol Version 4 (IPv4), and Internet Protocol Version 6 (IPv6) addresses;

2. **Length filter**, which removes words whose length is smaller than 2 characters or exceeds 12. These values are a fair compromise considering the average word lengths of multiple languages [4], so we could filter out small irrelevant words and bigger ones, which are atypical;

3. **Lower case filter**, that transforms any uppercase letters tokens to the equivalent lowercase ones;

4. **Stop word filter**, that discards or stops the analysis of tokens that are on a certain provided stop words list;

5. **Snowball Porter Stemmer**, which is a language-specific stemmer generated by Snowball [5], a software package that runs pattern-based word stemmers;

6. **Length filter**, which removes words whose length is smaller than 3 characters or exceeds 12. This filter follows the same logic as the previous one but now considering smaller words resulting from the stemmer process.

Furthermore, for every token set of each analyzed query, we compute its Inverse Document Frequency (IDF), discarding tokens whose value sited under a certain pre-defined threshold (a CLASSY parameter), corresponding to those that do not show sufficient informative interest.

[4] `http://www.ravi.io/language-word-lengths`
[5] `https://snowballstem.org/`

3.4.3 Knowledge database implementation

As already mentioned, we have implemented the knowledge database using the search engine system Apache Solr.

Apache Solr has two main ways of indexing the documents:

- The `POST tool`, which is able to parse and index a set of document types (e.g, Comma-separated values (CSV), Extensible Markup Language (XML), JavaScript Object Notation (JSON), PDF, Microsoft Word) by inspecting its internal structure and creating a schema that mirrors the latter;
- The programmatic one, by customly parsing the documents, and using the client API to add the content to the Solr index.

The latter shows to be more flexible, and we will use it here and after. However, when indexing custom documents programmatically, there is an important task that must be performed – the schema definition. The Solr schema is a file where the details about the document fields and their types are stored. An example of a simple schema regarding a movie is showed in Listing 3.1. There are some self explainable fields, but others are worth describing. The `indexed` enables the use of the value of the field in queries to retrieve matching documents if set to true; the `stored`, if true, enables the retrieval of the current field value at query time, and the `multiValued` indicates whether a single document contains multiple values for this type of field.

Additionally, we have created a new field – named `doc_tokens` – to store all the keywords that describe the defined entity. This field has a custom type that applies the pipeline defined in the relevant token extraction stage (described in Section 3.4.2) to the text source. This text source is also defined by the developer in the Solr schema file, by copying the content of the fields considered relevant to the `doc_tokens` one. An example of the definition of this field is shown in Listing 3.2, where the pipeline text source is the merge of the title, summary and genre field content. The `termVectors`, `termPositions` and `termOffsets` are flags to enable or disable APIs that allow the inspection of information and statistics of the terms of a field, such as the number of documents and term frequency, which are necessary to compute the IDF score.

After the index schema has been set, the engine can be booted up, with all documents indexed, and so it is able to receive the implicit query, built in the context processing module (as described in Section 3.1), and then set up a list of relevant documents according to those queries.

All the communications between CLASSY sub-modules and the Solr search engine are made using the `PySolr` [6], an open-source lightweight Python client for Apache Solr that provides interfaces to query the server and to get the respective results. As an open-source

[6] `https://github.com/django-haystack/pysolr`

```
<field name="imdb_id" type="string" indexed="true" stored="true"
      required="true"
      multiValued="false"/>

<field name="title" type="string" indexed="true" stored="true"
      required="true"
      multiValued="false"/>

<field name="genre" type="string" indexed="true" stored="true"
      required="true"
      multiValued="false"/>

<field name="published" type="tdate" indexed="true" stored="true"
      multiValued="false"/>

<field name="summary" type="string" indexed="true" stored="true"
      multiValued="false"/>

<field name="actors" type="string" indexed="true" stored="true"
      multiValued="true"/>
```

Listing 3.1: A Solr schema regarding a movie.

```
<field name="doc_tokens" type="nameText" indexed="true" stored="true"
    multiValued="true"
    termVectors="true"
    termPositions="true"
    termOffsets="true"/>

<!-- Text source definition -->
<copyField source="title" dest="doc_tokens"/>
<copyField source="summary" dest="doc_tokens"/>
<copyField source="genre" dest="doc_tokens"/>
```

Listing 3.2: `doc_tokens` field definition.

project, with the code publicly available, it can be easily expandable with new global or custom features.

3.4.4 Suggestions filter and cache module implementation

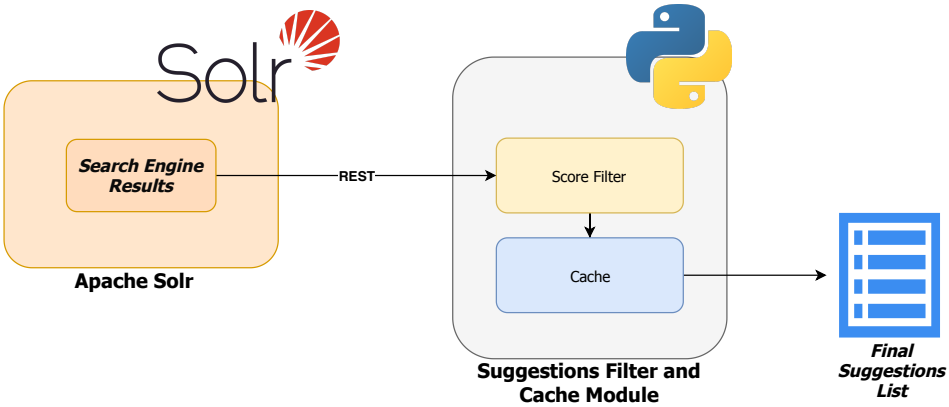


Figure 3.4: Suggestions filter and cache module interactions.

The Suggestions filter and cache module, described in Section 3.3, was also implemented as a set of Python methods. Its interactions, that will be explained subsequently, are illustrated in Figure 3.4.

This component will receive from the Solr engine the list of relevant documents, according to the sent implicit query, and then apply a simple filter that will ignore documents whose score computed by the Solr engine is below a threshold (a CLASSY parameter). Therefore, it will only retain the documents considered to have enough value to be suggested to the user.

After that, the resulting list is re-analyzed based on the cache that stores the last made suggestions and their timestamps so that those same suggestions are not repeated in consecutive or nearly consecutive time windows. The cache has an expiration timer (a CLASSY parameter) in which a document is considered being valuable again; thus, it can be discarded from the suggestion cache when it steps behind that limit.

3.4.5 Interaction interface

Since CLASSY was built to directly interact with the users' conversational environment, it is useful to develop an interface that is user-friendly at the same time that it enables the system to be always aware of the messages and conversational contexts.

Bearing this in mind, we have adapted the CLASSY interface in order to be used by a Chatbot in any chat environment that accepts these types of agents.

As a way to save time, and to work as a proof of concept only, we have developed a Discord [7] chatbot using the open-source `discord.py` [8] library, that simplifies the use of the Discord API. This will enable CLASSY to receive all messages sent to the Discord channel, execute its pipeline, and then send the suggestions (if any) to the same channel, through a friendly interface. An example of this interface is showed in Figure 3.5, where the CLASSY agent suggested a Wikipedia page based on the message sent by the user Hazhor. This setup was used to perform a live demo at the Lisbon Machine Learning School (LxMLS) 2019 (check Section 6.2.2 for further details).

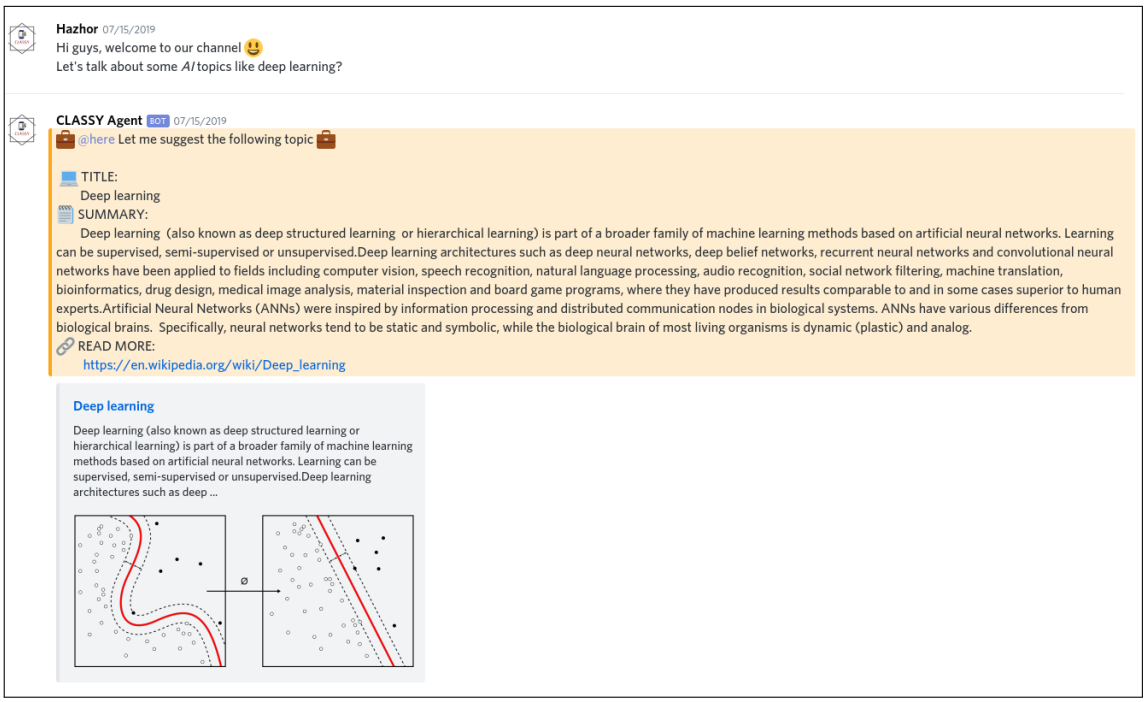


Figure 3.5: CLASSY Discord interaction.

3.4.6 Additional features

The use of a large set of topics and contexts in a conversational scenario will result in an increase of noise perceived by the system. Moreover, it is necessary to create a negative class document that would contain a rather large historical registry of exchanged messages. Implicit queries including the negative class documents in their results list are seen as having more noise associated with its context, therefore not adding value as a suggestion source. This negative class is put into practice in the form of a noise file system that is also stored in the knowledge database. This system has a maximum size (a CLASSY parameter), that defines the number of noise files that can be created and used as the negative class. As mentioned,

[7] `https://discordapp.com/`
[8] `https://github.com/Rapptz/discord.py`

these files, containing the historical registry of exchanged messages, are populated via a round-robin approach, so that, after the last possibly created file exceeds a specific size, the first one is emptied and filled with the most recent messages.

Additionally, as a user, it is not pleasant to be constantly flooded with suggestions after every single message, especially if some of them are not as useful as one would like. In that regard, the suggestion system should only suggest new documents with a distance of n messages (a CLASSY parameter). This behavior not only reduces the invasive presence of the suggestion system but also refines its suggestions since every suggestion can take into account more contextual information provided by the message window.

However, there are times where, instead of talking about a specific topic, people explicitly mention some related keys. In this field, these keys can take a lot of forms, such as the title of a document, the author of a tweet, the key to an issue and so on. These types of interactions do not provide context but specific topic matches; thus, it is useful to perform a direct search for them. With that in mind, we build a simple query using all tokens of the received message and try to find a direct match between the query and the document, based on the field that we expect to find (e.g, title, author, key). If a hit occurs, then the result of this stage is favored over the context results.

3.5 Results

In this section, we will describe the performance and behavioral results concerning this initial approach, as well as the datasets and testing environment used to perform the tests.

3.5.1 Datasets exploration

Since this work is developed in the context of the SmartEnterCom project (described in Section 1.1), one of the companies associated with the project provided us two datasets with real data to develop and test CLASSY system.

The conversational dataset was extracted from a Hangouts [9] team chat, where five developers exchanged 2325 unique messages about the development of a specific company product related to real-time communications. This dataset will serve as the conversational environment within our tests. Each message contains the following fields:

- User, which corresponds to the account name of the message author;
- Date, with the timestamp of the message;
- Message, containing the text content of the message.

[9] `https://hangouts.google.com`

The issues dataset contains almost 100 development issues from the Jira [10] issue tracking platform used in the project, where the conversational dataset is inserted. Among others, each issue is composed of the following information:

- Unique Id of the issue;
- Key of the issue, that contains a unique identification of the issue inside the project context (like `ProjectName-ISSUEID`);
- Timestamp of creation and update;
- Summary of the issue content;
- Description of the issue;
- Author related information (like email, and display name);

Each issue will be parsed and indexed as a document in the Solr engine, working as the initial system knowledge. The Solr schema regarding the structure of the issues is shown in Listing 3.3.

Both the conversational and issues dataset are composed of Portuguese messages (with some technical concepts in English) and so, the Solr analyzer, namely the stopword filter and the stemmer, was configured to parse Portuguese words.

3.5.2 Testing environment

Like most datasets that contain real data, the ones described above lack one important feature that is extremely helpful when it comes to evaluation – annotations. This means that there is no previous information about matches between the issues and the conversational segments. The only way to check the usefulness of a suggestion is to actually **manually** analyze each suggestion and the context in which it was made.

With this in mind, we have created a testing and simulation environment, that will run separately from the developed interaction interface described in Section 3.4.5. This simulation environment will make all inspection and evaluation processes easier so that they can make their judgment in an agile way. We have developed this environment only for evaluation and testing purposes, during CLASSY development, and therefore, is not part of the system. The main features of this environment are:

- Evaluation interface - each time the system performs a suggestion, the testing environment will show the evaluator the last n messages along with all the information regarding the displayed suggestion. This way, the evaluator will have all the necessary tools to judge the usefulness of the suggestion;

[10] `https://www.atlassian.com/software/jira`

```
<!-- Data fields -->
<field name="issue_id" type="string" indexed="true" stored="true"
    required="true"
    multiValued="false"/>
<field name="key" type="string" indexed="true" stored="true"
    required="true"
    multiValued="false"/>
<field name="created" type="tdate" indexed="true" stored="true"
    required="true"
    multiValued="false"/>
<field name="updated" type="tdate" indexed="true" stored="true"
    required="true"
    multiValued="false"/>
<field name="summary" type="string" indexed="true" stored="true"
    multiValued="false"/>
<field name="description" type="string" indexed="true" stored="true"
    multiValued="false"/>
<field name="displayName" type="string" indexed="true" stored="true"
    multiValued="false"/>
<field name="emailAddress" type="string" indexed="true" stored="true"
    multiValued="false"/>

<!-- Token extraction -->
<field name="doc_tokens" type="nameText" indexed="true" stored="true"
    multiValued="true" termVectors="true"
    termPositions="true" termOffsets="true"/>

<copyField source="description" dest="doc_tokens"/>
<copyField source="summary" dest="doc_tokens"/>
```

Listing 3.3: The Solr schema regarding an issue.

- Global clock - since there are some time-related system constraints (like the maximum time in cache described in Section 3.4.4 and others that will be described in the following chapter), it is necessary to create a global clock to synchronize all of them, so that the suggestions are handled correctly;
- Simulation data - to facilitate the post-analysis, all the information regarding CLASSY performance throughout the time of a specific simulation, namely the total and useful suggestions counters, is saved, so that it can be used later for, for instance, plot the system accuracy over the simulation time.

3.5.3 Base approach results

In order to create a comparative benchmark to evaluate CLASSY and all its evolution phases, we have first implemented the base approach. This uses the blocks described in this chapter without making any enhancements to the IR system neither its score method, whose goal was to add value to the system by developing a valuable pre-processing step of the messages and post-processing of the result list.

Minimum document score	Total suggestions	Useful suggestions	Accuracy
10	476	89	18.70%
20	232	64	**27.59%**
30	171	22	12.87%

Table 3.1: Base approach - minimum document score impact.

Regarding this base approach, the first thing to analyze is the minimum document score and its impact on the results. This study is shown in Table 3.1. As one might observe, there is a big difference between the total number of performed suggestions and those considered useful. This confirms what was mentioned about the amount of noise derived from the messages.

Number of noise files	Minimum document score	Total suggestions	Useful suggestions	Accuracy
1	20	89	39	43.82%
2	20	102	47	46.08%
3	20	119	67	**56.30%**
4	20	131	69	52.67%

Table 3.2: Base approach with Noise files system - number of noise files impact.

Additionally, we have implemented the noise files system, taking into account the proposed solution to overcome this issue. The next step is to study the impact of the number of noise files in the final result. These results are shown in Table 3.2. On those tests, the minimum

score was maintained at 20, since it was the one showing the best results, with the variations following the same behavior of the ones shown in Table 3.1. As one can observe, using the noise files system not only reduced the total number of suggestions, but it also increased the proportion of useful ones. Here, the setup with three noise files also showed to be the best compromise in terms of accuracy.

Messages btwn. suggestions	**Number of noise files**	**Minimum document score**	**Total suggestions**	**Useful suggestions**	**Accuracy**
1	3	20	119	67	**56.30%**
2	3	20	78	29	37.18%
3	3	20	55	29	47.27%
4	3	20	46	21	45.65%
5	3	20	32	18	56.25%
6	3	20	24	7	29.17%

Table 3.3: Base approach with Noise files system - number of messages between suggestions impact.

Using the above approach as the baseline, it is interesting to tune the system in order to make it more pervasive. With that in mind, we investigated the impact of the number of messages between consecutive suggestions in the results, as showed in Table 3.3. These results do not show a systematic evolution since they are very dependent on the position of useful messages along with the conversation. However, for the analyzed dataset, one and five messages between suggestions showed to be the best compromise in terms of accuracy, with the first one being more intrusive, as it is logical.

3.6 Conclusion

In this chapter, we have presented the architecture and implementation of the CLASSY prototype. Its purpose is to constantly gather and process information from real-time conversation environments to find the current context and make suggestions accordingly.

The presented overall architecture should meet not only functional requirements, such as high performance, to gather information, analyze it and find suggestions in a real-time environment, but also non-functional requirements such as being non-invasive and following all the pervasive rules.

Additionally, this architecture was built to be as general as possible, so it can work with any type of document, any type of conversational topic, in any language. For that, it uses techniques that can analyze and process general text, as well as a schema-oriented knowledge database that was built to accommodate any type of document, with any type of structure, with no or minimum developer interaction.

This general behavior not only enables this architecture to be vertically and horizontally scalable, but it also minimizes the work involved in deploying it in different contexts with different topics.

The decision of using the Solr search engine as the knowledge database empowered the functionality and efficiency of the whole system, since it enabled the use of some specific features (like the analyzer) in other modules, simplifying and enhancing the performance and code structure.

Furthermore, as the implementation of the modules was made through Python methods merged in a single class, it enables the use and share of the code as a unique library that can be used with a few simple lines of code. This, allied to the use of the knowledge database as a Solr search engine, facilitate the deployment of the whole system in a container environment (like Docker [11]), regardless the hardware or machine. As a matter of fact, this was also the strategy followed along with CLASSY development and testing, creating an independent environment where the Solr engine is in an autonomous container as well as the remaining CLASSY core, and the two containers are connected by a specific network.

The interaction interface, along with the additional features, also tries to add a more user-friendly and pervasive behavior as well as a better performance to the system, by introducing some mechanisms to reduce the associated noise and its invasive presence and increasing its usefulness.

Moreover, the use of the `PySolr` as a Python Solr client and its open-source nature facilitated the implementation of all CLASSY modules. This enables the developer to add more custom features, something that was necessary, namely to apply the analyzer pipeline through the received text, as well as to compute the IDF score of the tokens. Despite being features that were not implemented, there were some high-level methods (mainly related to the communication between the client and Solr engine) that smooth the development of new interactions with other less used Solr APIs.

All the code regarding the CLASSY prototype, as well as other approaches and setups that will be described in the following sections, can be found at `https://gitlab.atnog.org/diogoferreira/classy` [12].

Finally, the base approach, with the addition of the developed features, such as the noise file system, achieved an accuracy score of **almost 57%**. Considering that no enhancements to the Search Engine system neither its score method were made, but only a valuable pre-processing step of the messages and post-processing of the result list, this is an encouraging result, mainly because the other proposed approaches will be based on this one.

[11] `https://www.docker.com/`
[12] Contact for access permissions.

4

CLASSY evolution

The system architecture and implementation described in the previous chapter compose the so-called `basic approach`, in which a set of text-based techniques where used, allied to some frameworks like Apache Solr, to build a general and clean system.

However, in order to include a more pervasive behavior and also enhance the ConversationaL Aware Suggestion SYstem (CLASSY) performance, some iterative improvements and approaches were taken – the Neighborhood and RL – which will be described in this chapter.

4.1 Neighborhood approach

The Neighborhood approach was the first improvement made on the base CLASSY. Its goal was to use semantic similarity associated with each token, as a way of getting similar contexts and, based on them, find more informed relationships to make accurate suggestions.

4.1.1 Neighborhood score

Semantic distance/similarity is a property of lexical units, typically between words but this notion can be generalized to larger units, such as phrases or sentences. Two words are considered semantically close if there is a lexical-semantic relation between them. There are two types of lexical relations: classical relations (such as synonyms, antonyms, and hypernyms) and ad-hoc non-classical relations (such as cause-and-effect). If the closeness in meaning results from a certain classical relation, then the terms are said to be semantically similar. On the other hand, semantic relatedness is the term used to describe the more general form of semantical closeness caused by any semantic relation. For instance, the nouns *liquid* and *water* are both semantically similar and related, whereas the nouns *water* and *boat* are semantically related but not similar.

There are a lot of types of semantic measures, but we are interested only in one – corpus-based – that focuses only on co-occurrence statistics from large corpora. They rely on the hypothesis that tokens with similar contexts tend to be semantically close [64]. The set of contexts of each target token u is represented by its distributional profile, described by the set of tokens that tend to co-occur with u within a certain distance, along with numeric scores signifying this co-occurrence tendency with u. This profile, based on the one introduced by M. Antunes, D. Gomes, and R. Aguiar [65], is defined as:

$$P(u) = [\{w_1, o(u, w_1)\}; ...; \{w_i, o(u, w_i)\}] \tag{4.1}$$

where u is the target token, and $o(u, w_*)$ the number of times that w_* occurs near to u.

The building process of those vectors is described in Figure 4.1, where $D*$ are the contents of the documents, $RTD*$ the relevant tokens from each document, and the neighborhood representation is given by the vector with each neighbor and its magnitude.

```
D1: This is the third report.
D2: This report describes the built automatic system.
D3: This cost report gives respect to the costs related with the
   last semester.

RTD1: [third, report]
RTD2: [report, automatic, system]
RTD3: [cost, report, cost, semester]


Report neighborhood(window = 2):
[third: 1, automatic: 1, system: 1, cost: 2, semester: 1]

Report neighborhood(window = 1):
[third: 1, automatic: 1, cost: 2]
```

Figure 4.1: Neighborhood vector creation.

Having built the neighborhood of all relevant tokens of a document, it is possible to obtain a **neighborhood score** between any two tokens, by computing the inner product of the neighborhood vectors:

$$NS(u, v) = |u| \times |v| \times cosine(u, v) \tag{4.2}$$

$$cosine(u, v) = \frac{\sum_{i=1}^{n} o(u, w_i) \times o(v, w_i)}{\sqrt{\sum_{i=1}^{n} o(u, w_i)^2} \times \sqrt{\sum_{i=1}^{n} o(v, w_i)^2}} \tag{4.3}$$

This score is a good indicator of how two tokens are related and inherently how can they refer to the same context.

4.1.2 Implementation

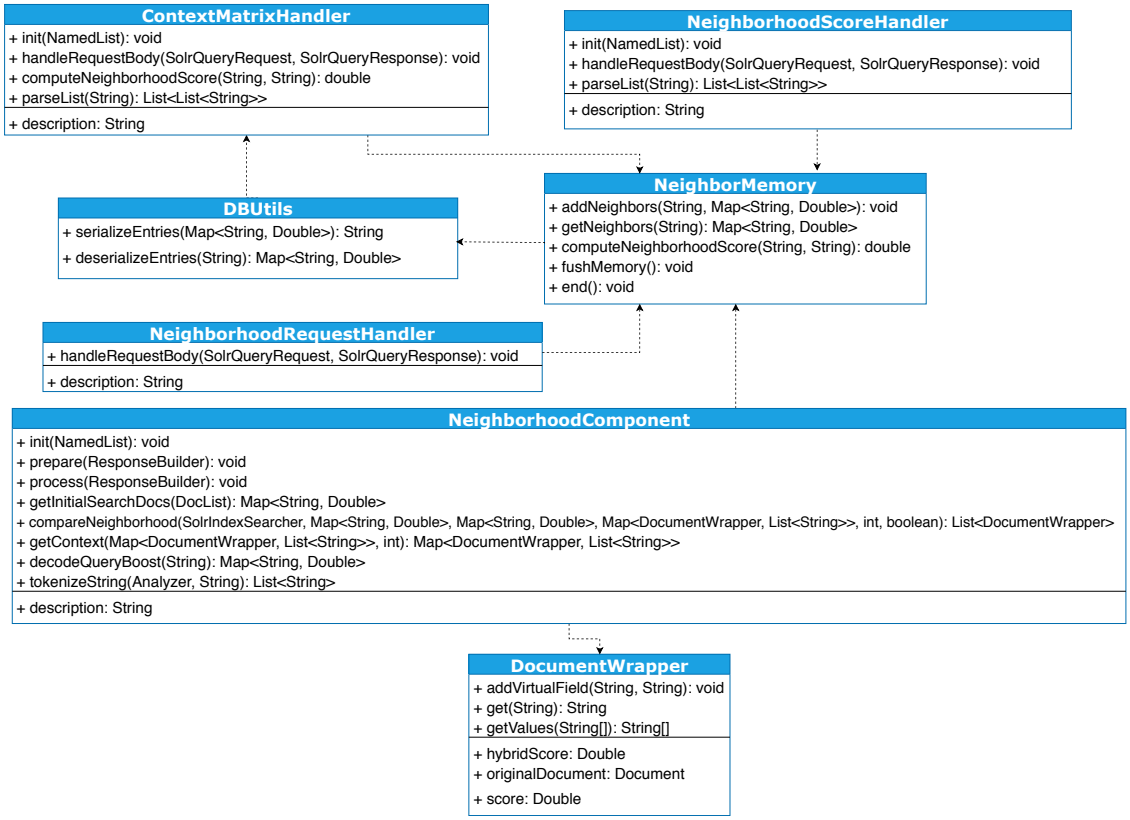


Figure 4.2: Solr plugin class diagram.

We implemented the process of building neighborhoods and computing its scores as a plugin inside the Solr engine, resorting to all the potential and efficiency of the Lucene's internal structures that enabled the handling of individual token analysis. The plugin structure is shown in Figure 4.2, where each class goal will be explained later.

The pseudo-code shown in Algorithm 1 reflects the steps needed to build the neighborhood for all tokens. The *window* is a CLASSY parameter which defines the maximum distance between two tokens that makes them neighbors.

Algorithm 1: Neighborhood creation

```
1 window := getWindowSize(params)
2 memory := getNeighborhood()
3 uniqueTokens := getUniqueTokens()
4 for token ∈ uniqueTokens do
5     neighbors := []
6     for doc ∈ getDocsWhereTokenAppears(token) do
7         postingsList := getPostingsList(doc)
8         neighbors := neighbors + getNeighbors(token, postingList, window)
9     saveNeighbors(memory, neighbours)
```

This algorithm was implemented as a `Lucene request handler` that can be executed through a simple call to the `/buildNeighbourhood` endpoint. These endpoints are defined in the `solrconfig.xml` file.

Since it is necessary to store the neighborhood information to consult later, we chose to use the `Redis` [1] database as a way to save these data. `Redis` is an open-source, in-memory data structure store used as a key-value database that supports a bunch of data structures and provides high availability and automatic partitioning. Its key-value structure showed to be adequate to our problem, by defining the tokens as the key and the neighborhood list as the value.

In order to facilitate the handling of neighborhood information, we also developed a middleware module between the Solr engine and the Redis database. Among others, this middleware serializes and deserializes data, as, for efficiency purposes, the neighborhood list internal structure is saved in the database as a string. An example of the neighborhood memory and its internal structure is shown in Table 4.1.

Key	**Value**
report	third:1;%;automatic:1;%;cost:2
computer	language:2;%;brand:1
chat	window:1;%;system:2;%;history:2

Table 4.1: Neighborhood memory example.

With all the neighborhood information available, it is now possible to compute the neighborhood score at query-time and use it to re-rank the top n documents retrieved by the Lucene's engine. This process will improve the contextual analysis of each document. The steps towards computing the neighborhood score, illustrated in Figure 4.3, are:

[1] `https://redis.io/`

1. Neighborhood pair computing - the neighborhood vector of each token of the document will be compared with the neighborhood vector of each token of the query. The score associated with each individual document token results of averaging all comparisons;

2. Intermediate sub-query value computation - the score computed between each document and each sub-query will be the average of the neighborhood scores of all sub-query tokens;

3. Recency boost - each sub-query has a boost, that is associated with its recency. The score of each sub-query results from the multiplication between the recency value and the intermediate sub-query value;

4. Final score - The final document score will be the sum of all neighborhood values computed between each document and each sub-query.

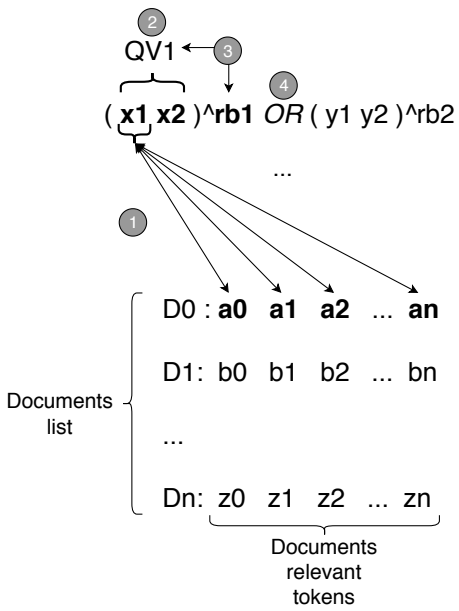


Figure 4.3: Neighborhood score computation steps.

These steps are once again implemented as a Solr plugin, but this time as a `Search component`, represented in Figure 4.2 as `NeighborhoodComponent`. A `Search component` defines additional logic used by the main `Search Handler` when performing queries, which enables the computation of the neighborhood score at query-time. This handler is running in the `/neighbourhood` endpoint, and it is where CLASSY will send its implicit queries to.

Also, we have created an endpoint (`/neighbourhoodScore`) that, given a source set of tokens and a list of a set of tokens (with length >= 1), computes its neighborhood scores, with the same rationale as the neighborhood score computation steps, but ignoring the boosts. This `Search Handler` is represented in Figure 4.2 as `NeighbourhoodScoreHandler`.

4.1.3 Results

Considering the definition and implementation details described in the previous sections, we then performed some tests in order to evaluate the neighborhood approach performance, comparing to the base one, described in Chapter 3.

The datasets, as well as the testing environment used to perform the evaluation below, were the same as the ones used in the base approach, described in Sections 3.5.1 and 3.5.2.

Collection	**Language**	**Number of unique tokens**	**Time spent**	**Tokens per second**
CollectionM150	English	2522	1.7 seconds	1483.5
CollectionM200	English	3723	2.3 seconds	1618.7
CollectionI	Portuguese	500	400 milliseconds	1250
CollectionE	Portuguese	19479	5.3 minutes	61.3

Table 4.2: Neighborhood computation performance.

Regarding this neighborhood approach, the first thing to investigate is the performance of the neighborhood computation process, by checking the number of analyzed tokens' neighborhood per second. To perform these tests, we used four different types of collections of documents, in different languages and with different structures: `CollectionM150` and `CollectionM200`, which are a selection of 150 and 200 songs information, respectively, regarding their artists, genre and lyrics; `CollectionI` which is composed by 97 JIRA issues' information; and `CollectionE` which is composed by the programs of the 16 most relevant Portuguese political parties regarding the legislative elections of October 2019. The execution times and throughput are shown in Table 4.2, where it is possible to notice that this process is very efficient when it comes to small or medium-length documents. However, when dealing with larger documents, the performance drops significantly.

Window size	**Number of noise files**	**Minimum document score**	**Total suggestions**	**Useful suggestions**	**Accuracy**
1	3	17	39	27	**69.23%**
2	3	17	70	27	38.57%
3	3	17	140	39	27.86%

Table 4.3: Neighborhood approach - window impact.

Also, it is important to verify the neighborhood window impact, which defines the maximum distance between two tokens that makes them neighbors. The results in Table 4.3 showed that using a window size bigger than one, adds a considerable amount of noise, thus, resulting in much more invasive and useless suggestions.

Minimum document score	Number of noise files	Window size	Total suggestions	Useful suggestions	Accuracy
5	3	1	162	34	20.99%
8	3	1	99	31	31.31%
11	3	1	74	29	39.19%
14	3	1	51	27	52.94%
17	3	1	39	27	69.23%
20	3	1	35	26	74.29%
23	3	1	31	25	**80.65%**

Table 4.4: Neighborhood approach - minimum document score impact.

The results of studying the impact of the minimum document score in this approach are shown in Table 4.4. As we increase the minimum document score, the accuracy also increases, with a notorious value of 80.65% for a score of 23. Although these results seem optimistic, the reason for this sudden accuracy increase is the decrease of contextual suggestions concerning the direct matches performed by the system, which we want to avoid.

Minimum document score	Number of noise files	Window size	Total suggestions	Useful suggestions	Accuracy
40	3	1	142	40	28.17%
60	3	1	97	38	39.18%
80	3	1	71	35	49.30%
100	3	1	61	31	50.82%
120	3	1	47	29	61.70%
140	3	1	46	29	63.04%
160	3	1	41	28	68.29%
180	3	1	38	27	71.05%
200	3	1	36	27	75.00%
220	3	1	35	27	**77.14%**

Table 4.5: Hybrid approach - minimum document score impact.

In order to mitigate this behavior, but still use its inherent advantages, we have proposed a new score approach, that merges the scoring method of both the base and neighborhood approaches, as shown:

$$HybridScore = LuceneScore \times NeighborhoodScore \tag{4.4}$$

This new score aims to mitigate extreme differences and make the general scoring method more robust. The results in Table 4.5 show that the hybrid approach is a good compromise between being pervasive and performing useful suggestions, by balancing the contextual and direct ones. However, when the minimum document score exceeds 160, the behavior of the

single neighborhood approach rises again, and the number of contextual suggestions starts to decrease.

4.2 Reinforcement Learning approach

CLASSY is a system built to serve and enhance users' communications. This means that every suggestion performed will be observed or judged in terms of usefulness by, at least, one user. This allows us to enhance the system's performance by adding the ability to learn based on user feedback.

4.2.1 User feedback

As showcased in Section 2.6, there are two types of user feedback: explicit, where a user explicitly rates the action through a measure of quality (for instance a score between 0 and 10); and implicit, where the information regarding the value of the response is tied to the user action. In this work, due to time constraints, we will be focusing only on one specific explicit feedback approach – the user click. Although click information can sometimes have some disadvantages, such as the associated noise and the so-called "misclick" rate, it will give the system a general approximated measure of performance.

4.2.2 Reinforcement Learning problem modeling

After the definition of the evaluation factor, that will serve as a performance review measure, it is necessary to model the problem in a way as to allow the system to actually learn by experience.

The majority of the RL problems can be modeled following the MDP formalism, as already mentioned in Section 2.7, which is composed by:

- A set of states that, in the conditions of this work, will somehow describe the current context;
- A set of actions available in each state;
- The transitions probabilities between states regarding a specific action;
- The transitions reward.

However, the problem we are trying to solve is continuous and randomly distributed, as the only information the system has when it first starts is relative to previously indexed documents, not knowing the contexts, neither its cardinality or relationships. Also, the indexed documents list can be updated and changed along with the agent execution. With this in mind, we propose an approximation to a MDP, which tries to tackle the continuity

and random distribution problem. This approximation is defined by the states, actions, and policy, as detailed in the following sub-sections.

4.2.2.1 States

A state describes the environment perceived by the agent at a specific time. Also, if a suggestion is performed in a given instant, according to a context x, and posteriorly penalized, should not be done when the context x appears again. This means that the state definition should be broad enough to be replicated (and so guarantees convergence), but should also describe the current situation with detail.

Using the proposed Neighborhood score definition (described in Section 4.1.1), a singular context state is defined as the top n tokens (CLASSY parameter) that characterize the strongest neighborhood connection between a query (q_i) and a document (d_i). This process is illustrated in Figure 4.4, where the strongest connection, with a Neighborhood score of 6, is defined by the token vector $[k1, k2, k3]$, sorted by magnitude.

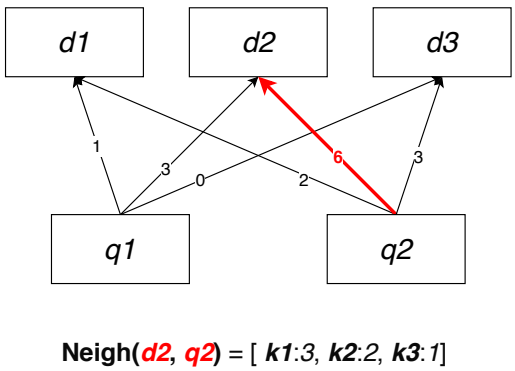


Figure 4.4: Singular context state computation.

Moreover, each suggestion will have a defined context regarding the implicit query, that will define the current state. This means that each implicit query can be described as a set of probabilities of belonging to each context state:

$$Q = \sum_{i=1}^{n} P(Q|ctx(x_i)) \tag{4.5}$$

So, for each implicit query, we use the singular contexts from the top *depth* results (CLASSY parameter), sorted by the score, as the global context state, in order to find the action that maximizes the reward.

4.2.2.2 Actions

Having collected the states that define the environment perceived by the agent, the possible action for each of these states is the suggestion of one of the documents that composed the

states vector, namely the one with the highest probability. So, when considering a *depth* value of 3, the top 3 documents from the result list will be considered by the agent (or the maximum possible, when the list is smaller). Moreover, conceiving that the singular contexts of those 3 documents are defined by $ctx1, ctx2, ctx3$, respectively, the possible actions are suggesting document with $ctx1$, document with $ctx2$, or document with $ctx3$.

4.2.2.3 Policy

Initially, all actions are equiprobable, giving the agent the responsibility of adjusting those probabilities to reflect user feedback and find the suggestions that better fit the contexts throughout the time.

However, the usage of a IR system helps to reduce the cold-start problem, by introducing some initial knowledge about the sample. Furthermore, it will also enable faster initial changes when the top document is not the most useful, since the other documents will prevail over this right on the next step.

The sum of the probabilities of the actions regarding a state should always be 1.0, so their adjustment is defined as:

$$P(s_i, a_j) = P(s_i, a_j) + R(s_i, a_j) \tag{4.6}$$

The reward value (r) was defined as 1.0, for suggestions that were clicked (and so considered as useful), or documents that were not suggested but whose related suggestion performed was not clicked. On the other hand, for suggestions that were not clicked, or for results that were not suggested but whose related suggestion performed was clicked, the reward is -1.0. This reward function is further conditioned by the learning rate α, which determines how quickly newly acquired information overrides the old one.

$$R(s_i, a_j) = \begin{cases} r\alpha & \text{if } suggested \\ \frac{-r\alpha}{|ctx(x_s)|-1} & \text{if } not\, suggested \end{cases} \tag{4.7}$$

The equations above consider an optimistic scenario. The fact that a document d_1 resulting from the implicit query q_1 can arise a context $ctx(x_1)$, but a document d_2 resulting from the implicit query q_2 can also arise the same $ctx(x_1)$, makes the problem more hard to solve. However, this is an expected behavior, due to the context generalization needed, but it requires an intermediate step to correctly compute the transition probabilities.

This intermediate step adjusts the probabilities mentioned above at every sample as if the agent was seeing the specific context set for the first time, by normalizing the values associated with each state, so that the adjustments needed at a time always follow the rule of the sum of the probabilities being always equal to 1.0. Also, when considering new context entries, the normalization of the remaining ones should consider that this new entry will be equiprobable, with probability $\frac{1.0}{|ctx|}$.

4.2.3 Replacement strategies

Giving the continuous nature associated with our problem it is possible to understand that the number of documents and contexts handled by the agent will grow rapidly. So, in order to save space and improve performance, it is necessary to implement a replacement strategy capable of discarding states which are not useful anymore (or at least in a current time window). This strategy will have an impact on the performance of the system, since discarding a state means that the associated probability computed before will also be discarded and so when it appears again, there will be no previous information about it.

With this in mind, the first two considered strategies were the Least Recently Used (LRU), which discards the least recently used items, maintaining new ones, and also First In First Out (FIFO), that discards the oldest entries.

Also, a replacement strategy regarding the similarity of the context vectors can be very useful. Broad contexts (states with a high value of global similarity) should be discarded since they are, by definition, closer to several other contexts than narrow ones. The Figure 4.5 showcases an example of a context distribution regarding a two-dimensional space and, as it is possible to observe, the context $c7$ is considered to have more similarities with the other ones, and so it is treated as a general one, making it less useful, and so it can be discarded. However, this strategy can be too radical if the distribution in the space is not so perfect and there is a considerable amount of general states. With that in mind, it is necessary to discard only the ones considered to have more global similarities and, therefore, that add less value. This can be achieved by applying the Pareto principle [66], which states that "for many events, roughly 80% of the effects come from 20% of the causes". Following the preceding logic, when the list of contexts reaches a specific size (a CLASSY parameter), the general ones that affect 80% of the effects will be discarded, and the contexts considered to be more specific and so more useful will be maintained.

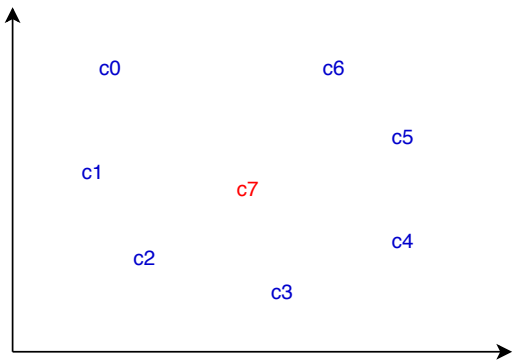


Figure 4.5: Example of a context distribution in a two-dimensional space.

4.2.4 Implementation

When it comes to getting user feedback, it is necessary to somehow "force" the user to click on the suggestion when it shows some value. This can be accomplished by creating a proxy system that will be responsible for generating an URL for each suggestion/document and handle their clicks. When a user clicks on one of these URLs, the agent will handle all the click information and will redirect the user to the original URL of the document.

This proxy was implemented as a web server using the `Flask framework` [2]. It also uses `SQLAlchemy` [3] as an Object Relational Mapper (ORM) to mediate database interactions, since click-related data must be saved and queried every time.

Since the agent itself has a strong relationship with this click proxy system, we opted to merge the latter into the agent, resulting in an independent and self-sufficient RL click-based system.

In terms of the agent data modeling, it was divided into three main classes, as showed in Figure 4.6.

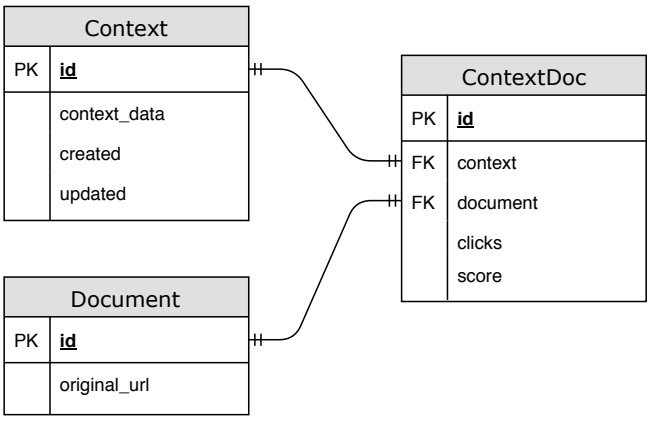


Figure 4.6: Agent database model.

Regarding these model classes, there are three main agent endpoints:

- `/classy`, which will handle the sample entries (through a `POST` request). It will receive the context and the original URL from the `LENGTH` top documents and create the document and state entities, if needed, as well as the unique relationship between them. The latter will be adjusted by the RL agent regarding the rules described in Section 4.2.2. Then, by using all the information available at that point, it will retrieve the suggestions with the highest probabilities, by creating a unique URL (named the `classy_url`) that will handle the click and posterior redirection to the original document;

- `/classy/<classy_url>`, that, as stated above, will handle the click information and run the probability adjustment algorithm regarding the list of possible suggestions for

[2] `https://palletsprojects.com/p/flask/`
[3] `https://www.sqlalchemy.org/`

that state. It will also redirect the user to the original document (through the URL) so that user can check its content;

- `/classy/<classy_url>/score`, that will retrieve the current probability associated to a specific `DocContext` entry.

Click information by itself is a good measure for giving positive reinforcement to a specific entry. However, it is not possible to evaluate bad suggestions, as, if the user does not perform any click action, the agent will not have any type of feedback information. Therefore, it was necessary to implement a suggestion memory, where all the suggestions made will be stored, as well as the other `LENGTH-1` candidates that will be affected by the usefulness of the latter, followed by the timestamp of the suggestion action. Then, whenever the agent needs to perform an action, it will evaluate the suggestion memory to find expired suggestions, comparing the elapsed time with a maximum time allowed to receive feedback (a CLASSY parameter). When it finds an expired suggestion, it will give negative reinforcement to the suggestion performed and positive reinforcement to the `LENGTH-1` associated candidates, using the rationale described in Section 4.2.2. That way, the agent guarantees not only that it gets as much information as possible, but also faster convergence.

Regarding the context state, for each document, it is computed inside the `Search Handler` described in Section 4.1.2, adding additional logic to the neighborhood score computation algorithm, in order to store, for each document, the strongest connection, and then retrieve its associated context vector.

In terms of replacement strategies, we have implemented LRU and FIFO directly in the agent, by using the creation and update time information. We also have implemented the strategy regarding the similarity of the contexts vectors, with the support of a new Solr `Request Handler` (endpoint `/contextMatrix`), represented in Figure 4.2 as `ContextMatrixHandler`, which, giving a set of contexts, computes its neighborhood similarities, and returns it as a score matrix with diagonal 0. To make the implementation of this strategy easier, it also accepts a parameter that makes the handler return, for each context, the sum of all similarities with all the other contexts. This similarity matrix is then cached in another Redis database (handled by the middleware described in Section 4.1.2), in order to achieve maximum performance and faster responses.

4.2.5 Results

Having tuned the base implementation of CLASSY, and then tested it merged with the Neighborhood approach, it is interesting to join this new RL approach, and observe the changes in terms of performance.

The tests that will be shown below take into consideration the CLASSY neighborhood setup with the hybrid score, described in Section 4.1.3, as it showed a better compromise

between being pervasive and being useful. Again, will be used the same datasets and testing environment described in Sections 3.5.1 and 3.5.2.

Regarding the RL approach, two main parameters should be analyzed in order to investigate its impact on the final system performance – the learning rate (α) and *depth*, explained in Section 4.2.2. The optimal testing scenario will independently consider both parameters. However, as they are two-way dependent, it is not possible to perform the tests this way and so, one of the parameters is fixed as the other one changes.

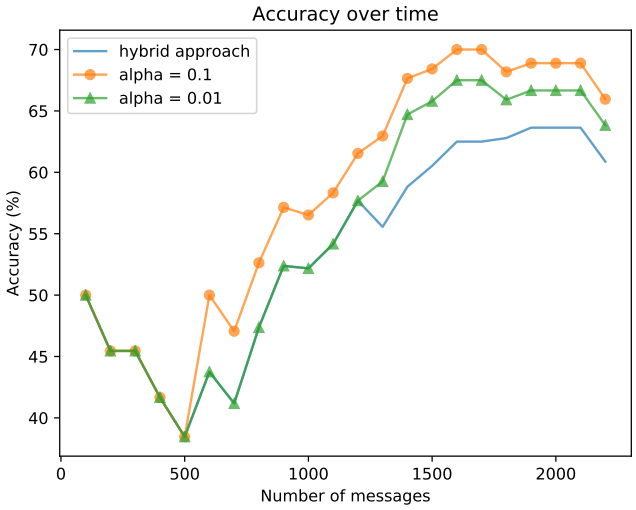


Figure 4.7: Impact of α parameter in the performance ($Min.Doc.Score = 140$; $depth = 3$).

In Figure 4.7, one can observe the impact of the α parameter over time, in the system's accuracy. The sudden decrease at the 500 messages mark is justified by the random distribution of the conversational topics over the dataset, which also leads to a random distribution of the suggestions made. Thus, in that period, a set of bad suggestions was performed, leading to an accuracy decrease. The performance is almost the same until the middle of the simulation, mainly for α `= 0.01` and the hybrid approach. However, after that, both RL setups performed better, with a significant final difference of more than 5% between the hybrid approach and the α `= 0.1`. This delay in achieving better performance is something considered normal, since RL takes some time to gather information concerning the best options, leading to a later convergence. It is also interesting to observe that, during the simulation, the RL setup with α `= 0.1` was able to reach 70% of accuracy, which is a good indicator that this approach can produce better results.

In terms of the *depth* parameter, its impact on system's performance is showed in Figure 4.8. The distribution behaves the same as described above, but here the improvements appear earlier in the simulation, where all RL setups showed a better performance. At the end,

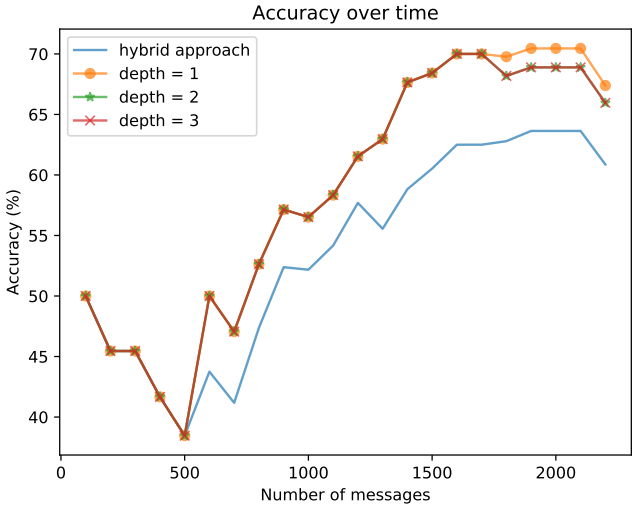


Figure 4.8: Impact of *depth* parameter in the performance ($Min.Doc.Score = 140$; $alpha = 0.1$).

depth = 2 and *depth* = 3 showed to have equal performance, while *depth* = 1 produced a slightly better accuracy.

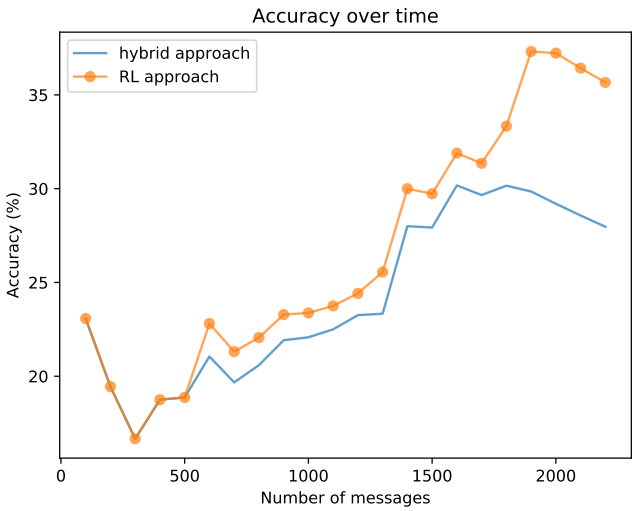


Figure 4.9: Hybrid versus RL approaches ($Min.Doc.Score = 40$).

It is also interesting to see how this new RL approach performs in terms of extreme setups (the ones used in the hybrid approach tests – min score $\in \{40, 140, 220\}$), comparing to the hybrid one. The results of these tests are shown in Figures 4.9, 4.10, and 4.11. As it is logical, when considering the setup with *Min.Doc.Score* of 40 the RL based system will have more opportunities to learn, by the fact that more suggestions were performed over time,

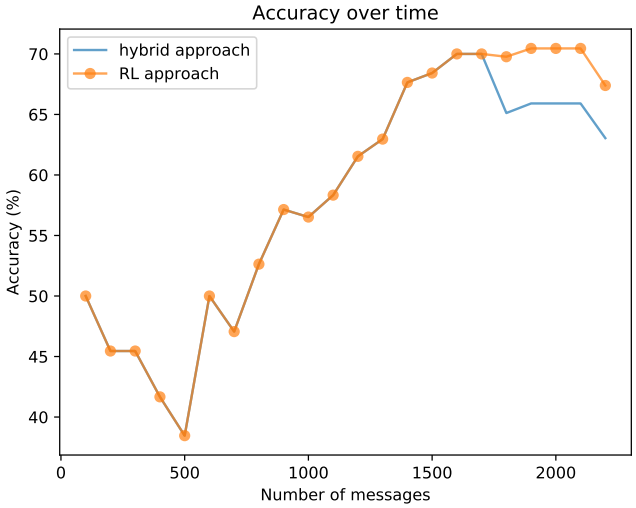


Figure 4.10: Hybrid versus RL approaches ($Min.Doc.Score = 140$).

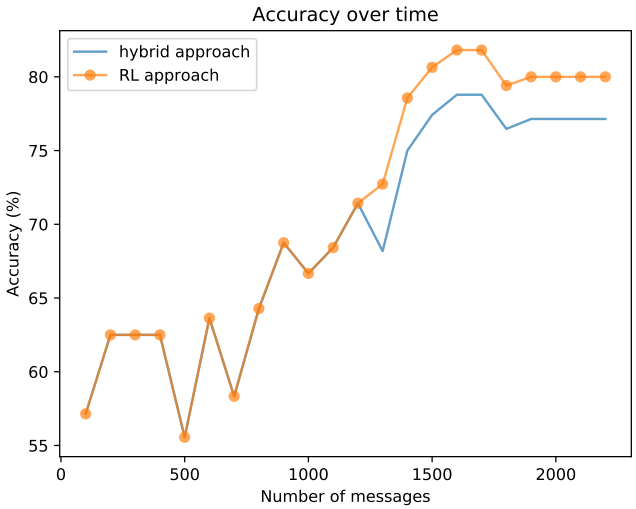


Figure 4.11: Hybrid versus RL approaches ($Min.Doc.Score = 220$).

leading to faster convergence, and showing best results earlier. The setup with *Min.Doc.Score* of 140 was the one that showed fewer differences regarding the hybrid approach, but stills reaching better performance after some time, with a difference of 5% when reaching the 2000 messages. At last but not least, the RL setup with *Min.Doc.Score* of 220 also showed good results, overtaking the 80% of accuracy when reaching the 1750 messages. The decrease of suggestions over time also showed to have an impact on the distribution, mainly in an earlier phase.

Strategy	Number of contexts	Total suggestions	Useful suggestions	Accuracy
LRU	10 15 20	35	27	77.14%
FIFO	10 15 20	35	27	77.14%
Similarity	10 15 20	33	28	84.85%

Table 4.6: Replacement techniques performance.

Finally, we should evaluate the impact of the newly proposed replacement technique, when comparing to the better-known ones (LRU and FIFO). The results in Table 4.6 showed that not only the number of contexts does not impact the overall performance, but also that the similarity-based replacement strategy proposed performs better compared to the other existing ones. These results were executed considering the setup which has achieved the best results in the tests made above and so, it is expected that the replacement strategy proposed performs even better in a scenario with more suggestions.

4.3 Conclusion

In this chapter, we described two CLASSY approaches, whose goal was to make the system more pervasive and improve its performance – the Neighborhood and the RL approach.

The Neighborhood approach uses the semantic similarity related to each word by inspecting its sentence neighbors, as a mean to get similar contexts and, based on them, find more informed relationships to make more accurate suggestions. The proposed neighborhood score, based on the concept above, gives another tool to evaluate context similarities, and then be more precise when it comes to comparing the conversational environment with the indexed documents. This neighborhood score, by itself, showed amazing results, with **increases of almost 24%** (using only this score) comparing to the base approach. However, in order to maintain a better compromise between being pervasive and making useful suggestions (both contextual and direct), we developed a hybrid approach, that merges the scoring method of both the base and neighborhood approaches, and with this performing an **accuracy of 77%**.

On the other hand, the RL approach was introduced as a way to learn from experience, by inspecting users' interactions and gathering implicit feedback through the clicks in the suggestions. These interactions make it possible for the agent to positively or negatively reinforce a suggestion comparing to a set of candidates, enabling, over time, that the suggestions

performed regarding a specific state are increasingly accurate. A system like that has a problem regarding the time to converge since it needs to gather a lot of information to learn what is good and bad until it understands this definition by itself. However, even considering the small datasets used to perform these tests, this RL approach enhanced the CLASSY global performance, having passed the 80% accuracy mark in some of the tests performed, which constitutes an **increase of almost 4% of accuracy**.

Both approaches were developed through the use of Solr plugins as a way to get token related information efficiently, due to the internal Solr and Lucene structures and statistics. The development of these plugins was not an easy task since the logic needed information from low-level modules, which were not necessary to most of the developers, leading them to be terribly documented.

The use of databases for information storing in the agent was mediated through the use of an ORM framework, which makes it possible to change from a database to another with almost no changes, making CLASSY scalable and robust as the data amount increases.

Also, the introduction of those two approaches led to changes in the global architecture, which can be observed in Figure 4.12. The addition of the RL agent took into account the same rules considered in Section 3.4, by developing independent modules that can be deployed in autonomous container environments, which was once again done.

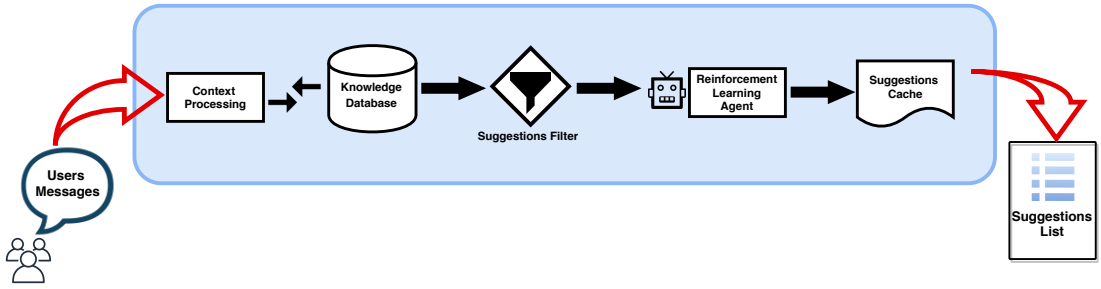


Figure 4.12: CLASSY final architecture (regarding Neighborhood and RL approaches).

It is also important to notice that all the tests in Sections 3.5.3, 4.1.3 and 4.2.5 were performed considering two real datasets, that are far from being perfect. The fact that both were small, limited the performance of some tests, mainly the ones related to the RL approach, as it needs a lot of initial information in order to converge successfully. Moreover, they were not annotated, which forced, to each of the tests showcased in this chapter, the manual inspection, and judgment of the usefulness of each suggestion. These issues meant that a lot of time had to be spent testing and analyzing each CLASSY change in terms of performance, even considering that we developed a testing environment to facilitate this process.s

5

Case study: The importance of feedback on context limited environments

The introduction of an agent that uses feedback relative to each suggestion performed has shown to be a good approach, comparing to those that only handle the conversational content, in context-based environments. However, it is also interesting to evaluate if these types of systems still work in an environment with fewer context, such as social networks. In these environments, a message can lead to many threads, with them being limited in terms of the context.

With that in mind, we have implemented a specific setup, aiming to gather information from a social network and suggest useful content to the users. This setup will allow us to validate two points – the usefulness of a feedback-based agent on these types of environments, and also ConversationaL Aware Suggestion SYstem (CLASSY) performance in a different use case.

5.1 Documents

The goal of this case study is to interact directly with users, which means that it is important to somehow cover a topic capable of generating controversy and motivate users to interact with the system. For that reason, we chose the *music* topic. The documents will be the information regarding the songs considered hot/most searched at the time (title, artists, genre, and lyrics).

5.2 Conversational environment

Social networks are a good source of text-based content, that covers a broad range of topics. The first attempt was to use Twitter posts as a data source to CLASSY. However, since the goal of this case study was to focus on a specific niche, it was not trivial to filter users nor posts by topics. Thus, considering the requirements, we chose Reddit [1] as a conversational source. Reddit structure is very rich and helpful to find specific niches since it is divided into channels, each of them regarding a specific topic. Considering the chosen topic – music – it was easy to select some channels where CLASSY will be interacting.

5.3 Feedback

The chosen conversation environment – Reddit – also implements a good measure of the usefulness of a post, using a points system. As a user, one can upvote or downvote a specific post or comment to reflect its judgment. Thus, this score can be a good way to evaluate if a suggestion was really helpful to the users interacting with a specific post.

5.4 Setup

Having defined the conversational environment, as well as the topics of the document and the feedback metrics of this case study, we had to design and implement a system setup capable of handling not only CLASSY interactions but also evaluating its performance over time.

5.4.1 Songs

As already mentioned, the knowledge database will contain information regarding the songs currently considered *hot*. To gather these data, we have used the `Musixmatch developer API` [2], that offers a simple way to get the trending songs and their information. These data are then indexed by Apache Solr and made available to be queried. The Solr schema definition considering this specific use case is denoted in Listing 5.1.

5.4.2 Reddit content

We chose three specific music-related channels from the Reddit community to conduct this case study, namely the `/r/music`, "The musical community of Reddit", `/r/popheads`, "the latest and greatest in pop music, all in one subreddit", and `/r/ifyoulikeblank`, "a subreddit for people who are in search of new, relevant media - whether it be music, TV shows, movies,

[1] `https://reddit.com/`
[2] `https://developer.musixmatch.com/`

```
<field name="id" type="string" indexed="true" stored="true"
      required="true"
      multiValued="false"/>
<field name="title" type="string" indexed="true" stored="true"
      required="true"
      multiValued="false"/>
<field name="artist" type="string" indexed="true" stored="true"
      required="true"
      multiValued="false"/>
<field name="genre" type="string" indexed="true" stored="true"
      multiValued="true"/>
<field name="lyrics" type="text" indexed="true" stored="true"
      multiValued="false"/>
<field name="created" type="tdate" indexed="true" stored="true"
      required="true"
      multiValued="false"/>
<field name="updated" type="tdate" indexed="true" stored="true"
      required="true"
      multiValued="false"/>


    <!-- Processing fields -->
<field name="doc_tokens" type="nameText" indexed="true" stored="true"
      multiValued="true" termVectors="true"
      termPositions="true" termOffsets="true"/>


    <!-- Token extraction -->
<copyField source="title" dest="doc_tokens"/>
<copyField source="artist" dest="doc_tokens"/>
<copyField source="genre" dest="doc_tokens"/>
<copyField source="lyrics" dest="doc_tokens"/>
```

Listing 5.1: A Solr schema regarding a music.

or anything else". We have selected these channels as they presented the largest number of interactions (post and comments).

Fortunately, there are a lot of tools to handle Reddit's content, due to its huge community. The `PRAW: The Python Reddit API Wrapper`, which, as the name says, is a Python wrapper to the Reddit API, will enable the inspection of older and more recent posts inside the channels, as well as the creation of comments with the suggestions, and analysis of their scores.

5.4.3 Implementation

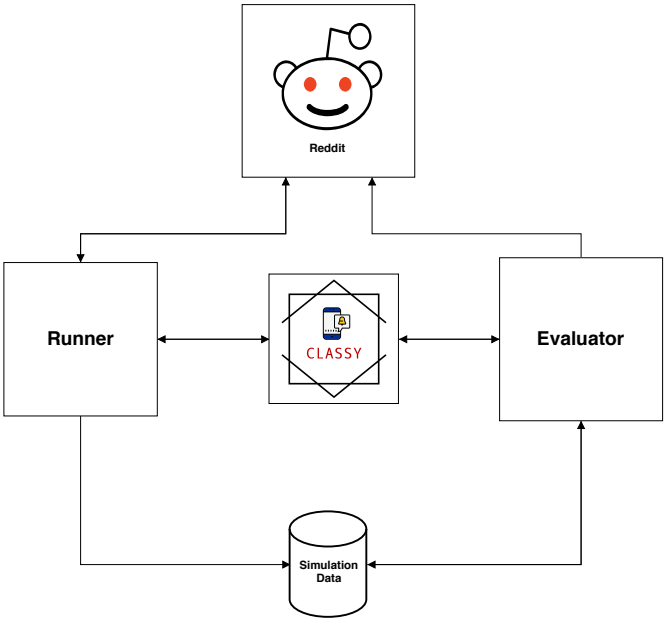


Figure 5.1: Reddit case study setup implementation.

This setup implementation is divided into five main modules, as illustrated in Figure 5.1:

- Reddit platform, where all the interactions will occur;
- CLASSY system, which will be the suggestions agent, whose design and implementation were described in detail in Chapters 3 and 4. The CLASSY parameters were tunned considering the best results achieved in the tests shown in Section 4.2.5, aiming to be the most accurate and least noisy possible. Additionally, we have performed an adjustment in the minimum document score, where a value of 500 proved to be the best compromise for this use case;
- Runner module, that will gather the channels content and then send each message to CLASSY, in order to simulate a conversation segment. Its execution has two main focuses, aiming to perform a global analysis and consider all types of contexts – the

hot posts and the stream posts. In the first, the runner will only parse the title of the post and then the comment threads individually, since they offer a context segment; the second aims to feed CLASSY only with the post content (title and description) in order to find suggestions that are directly useful to the post and not only to conversations related to it. The system will first analyze the top 500 *hot* posts in each channel, and then analyze all the posts published in the stream in real-time;

- Simulation data(base), where all the information regarding the suggestions is stored in order to make real-time and post-mortem evaluations. Each entry will contain the history of messages fed into CLASSY before the suggestion was triggered, the identifier of the Reddit comment containing the suggestion, as well as some other useful information, such as if it was a direct match;

- Evaluator module, that will query all the entries in the simulations database to find its score at the time, in order to extract CLASSY global performance. This module will run every 30 minutes, producing an evaluation summary.

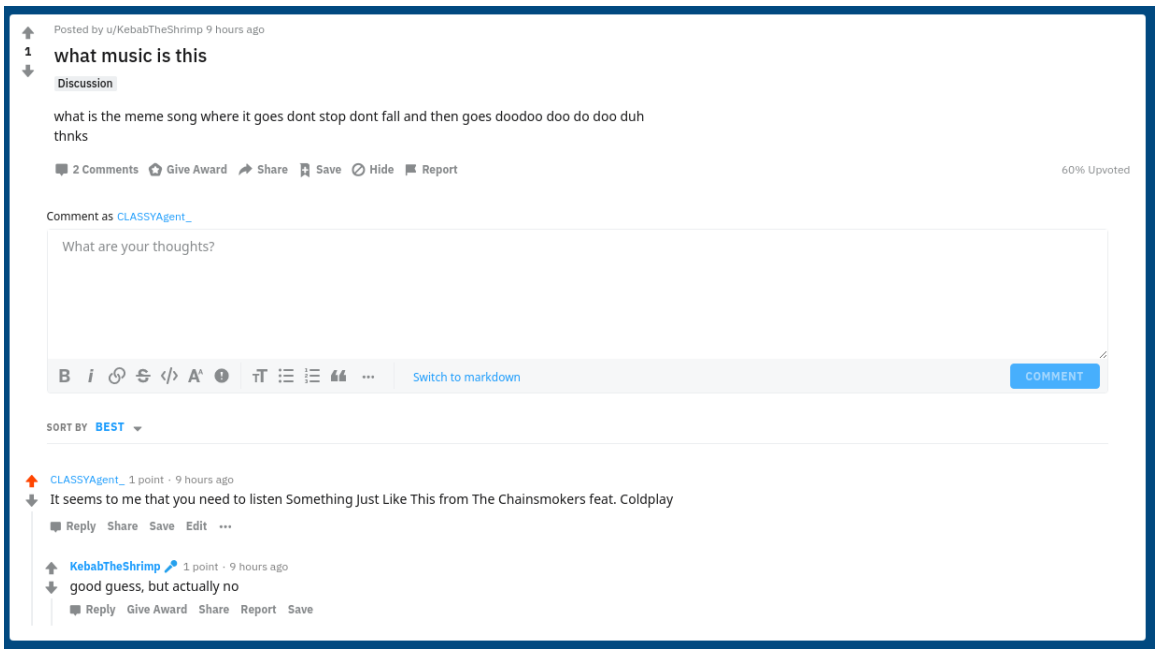
5.5 Results

The live simulation was performed for almost 20 days sequentially divided by the chosen channels, and it produced some funny and interesting interactions. Some of them, exemplified in Figure 5.2, showed to pass the Turing Test [67], since the users were not capable of realizing that they were interacting with a virtual agent, instead of a human. This shows that being aware of the context is halfway to get better pervasive behavior in these scenarios.

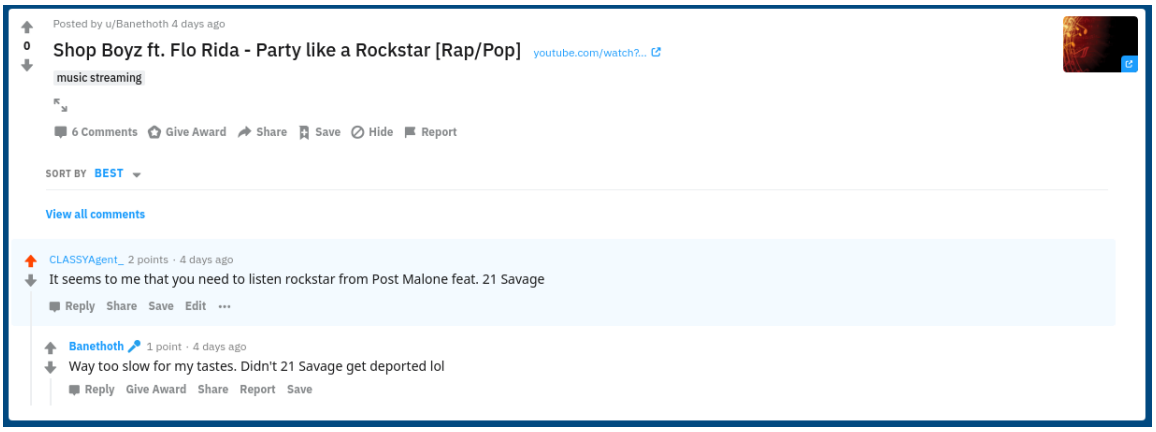
However, in order to properly evaluate the system's performance in this case study, it is also important to perform the same simulation using the CLASSY setup without users' feedback. This can be accomplished by ignoring the choices made by the RL-based agent, using the setup considering only the neighborhood approach (described in Section 4.1). This evaluation, unlike the one used in the feedback-based simulation, will instead use the final judgment from the users who interacted with each post. This means that, contrarily to the judgment in the live running, that can change over time, in this simulation, that will not happen. Also, when the simulation without feedback triggers a suggestion that was not made in the live scenario, the judgment will be made manually by an external judge.

The Figures 5.3, 5.4, and 5.5 show the results achieved by CLASSY with and without feedback, with a sampling of 5 hours for visualization purposes. Also, the sizes of the bubbles represent the number of suggestions made at a specific time, growing as a power of 2.

A feedback-based approach, like the RL one, performed better in all channels when compared with the one that disregards it. The final differences were 21% in the `/r/popheads` channel, 17% in `/r/music`, and 7% in `/r/ifyoulikeblank`. The lower differences in the lat-



(a)



(b)

Figure 5.2: Examples of CLASSY interactions on Reddit.

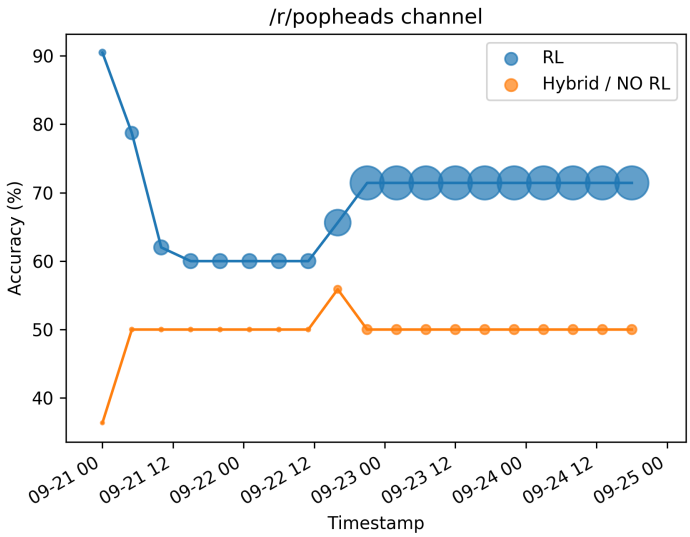


Figure 5.3: `/r/popheads` channel simulation.

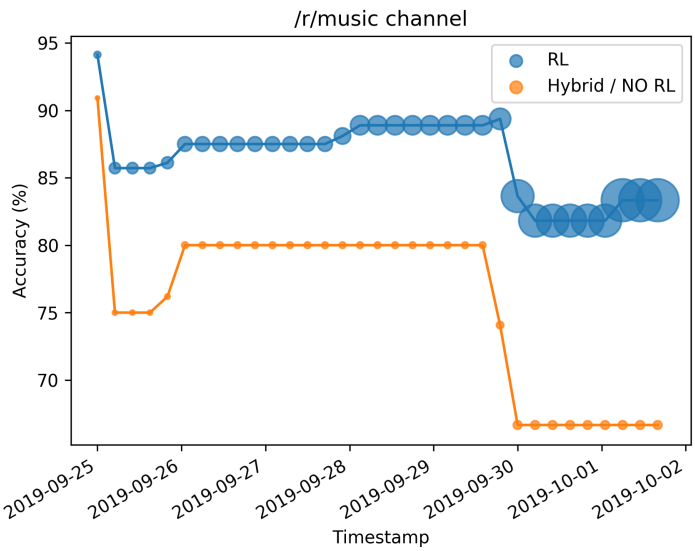


Figure 5.4: `/r/music` channel simulation.

ter happened since the channel does not focus exclusively on music. The channel's users also have more indie tastes, which does not match so well the current top music.

Also, it is possible to check that the RL approach also makes more suggestions – 7 to 4 in `/r/popheads` channel; 12 to 6 in `/r/music`; and 11 to 6 in `/r/ifyoulikeblank`. This is an expected behavior, since the agent has more information considering what is right and wrong, and so it can have more confidence to perform the suggestion.

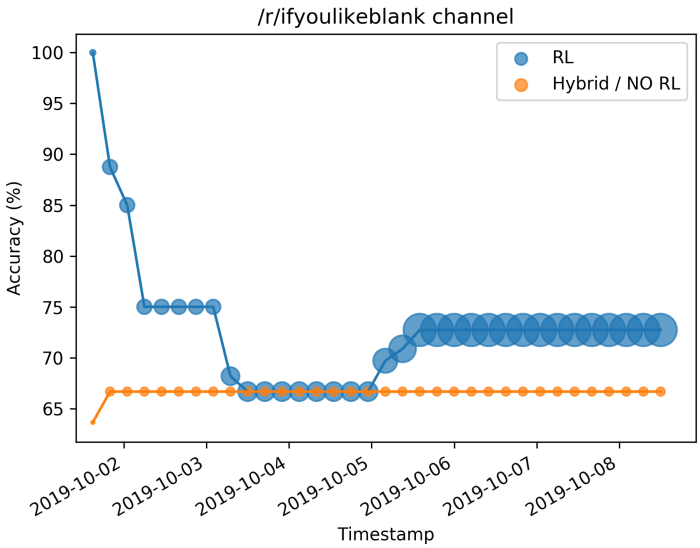


Figure 5.5: `/r/ifyoulikeblank` channel simulation.

Considering the results, an interesting feature from the RL-based approach is that it is able to recover its performance over time, due to the amount of extra knowledge obtained, unlike the non-feedback-based.

5.6 Conclusions

In this chapter, we have designed and implemented a setup to assert the usefulness of feedback to increase a suggestion system performance, like CLASSY, in context limited conversational environments, such as social networks.

We chose to use three music-related Reddit channels to perform the case study, namely `/r/popheads`, `/r/music`, and `/r/ifyoulikeblank`. Considering this music topic, we also chose the top 200 current hits and indexed their information in the knowledge database.

A full setup was also designed, aiming to gather all the information from the Reddit channels, send it to CLASSY, post the triggered suggestions, and then follow these posts in order to evaluate the system performance, creating a performance track, to post-mortem analysis. We resorted once again to open-source tools, like `python-musixmatch` [3] and `PRAW`, both Python wrappers to handle the data required to create this setup.

The results showed that even considering context limited environments, the users' feedback is an important feature for a suggestion system like CLASSY. Not only do they increase its performance globally, but they also feed the system with more knowledge and make it perform more accurate suggestions over time.

[3] `https://github.com/yakupadakli/python-musixmatch`

6

Conclusion and Future Work

6.1 Conclusions

With the increased use of conversational environments for personal or professional purposes, all the exchanged information can be used to somehow facilitate some tasks. Bearing this in mind, this work aimed to design and develop a system, entitled ConversationaL Aware Suggestion SYstem (CLASSY), which is able of being always aware of the real-time conversational environment and, by analyzing the messages contexts, perform suggestions that are considered useful by the participants.

The proposed global architecture (described in detail in Chapter 3), was designed to meet functional requirements, such as high performance while gathering information, analyzing it and finding suggestions in a real-time environment, and efficient storage, to accommodate a large number of documents in the knowledge database. Moreover, non-functional requirements must be taken into account, such as being non-invasive and to follow the assumptions of a pervasive system. Additionally, this architecture was built to be as general as possible, so it is compatible with any type of document, any type of conversational topic, in any language. To achieve that, it uses techniques that can analyze and process general text, as well as a schema-oriented knowledge database that was built to accommodate any type of document with no specific structure and none or minimal developer interaction. This general behavior not only enables this architecture to be vertically and horizontally scalable, but it also assures minimal work to deploy it.

The prototype, apart from the defined module implementation, also considers an interaction interface and other additional features, which try to add a more user-friendly and pervasive behavior, as well as a better performance to the system, by introducing some mechanisms to reduce the noise associated. Therefore it reduces its invasive presence and increases its usefulness.

The base approach, with the addition of the developed features, such as the noise file system, could achieve an accuracy of **about 57%**. Considering that we have made no enhancements to the Search Engine system, neither to its score method, but only a valuable pre-processing step of the messages and post-processing of the result list, this is an encouraging result, mainly because this works as the base approach for the remaining ones.

However, aiming to achieve better performance and pervasive behavior, we developed two other approaches – Neighborhood and the RL.

The Neighborhood approach uses semantic similarity related to each word by inspecting its sentence neighbors, as a means to get similar contexts and, based on them, find more informed relationships to make more accurate suggestions. The proposed neighborhood score, based on the above concept, gives another tool to evaluate context similarities, and then be more precise when it comes to comparing the conversational environment with the indexed documents. This neighborhood score, by itself, showed amazing results, with **increases of almost 24%** (using only this score), comparing to the base approach. However, in order to maintain a better compromise between being pervasive and making useful suggestions (contextual and direct), a hybrid approach was developed, which merges the scoring method of both the base and neighborhood approaches, performing an **accuracy of 77%**.

On the other hand, the RL approach was introduced as a way to learn from experience, by inspecting users' interactions and gathering implicit feedback through the clicks in the suggestion. These interactions make it possible for the agent to positively or negatively reinforce a suggestion, comparing to a set of candidates, enabling, over time, that the suggestions performed regarding a specific state are increasingly accurate. Such a system has a problem regarding the time to converge since it needs to gather a lot of information to learn what is good and bad until it understands this definition by itself. However, even considering the small datasets used to perform these tests, it could enhance the CLASSY global performance, having passed the 80% accuracy mark in some of them, which constitutes an **increase of almost 4% of accuracy**.

With the goal of evaluating the usefulness of feedback to increase a suggestions system performance, such as CLASSY, in context limited conversational environments, we have conducted a case study regarding some Reddit channels related to the topic "music". In this case study, we have suggested songs currently considered top hits and used users' interactions (upvotes and downvotes) as a feedback signal.

The results showed that even considering context limited environments, the users' feedback is an important feature for a suggestion system like CLASSY. Not only do they increase its performance globally, but they also feed the system with more knowledge and make it perform more accurate suggestions over time.

Besides all this, we have implemented all approaches and setups designed throughout this work through open-source tools, namely Apache Solr as the knowledge database, which facilitated the implementation of all CLASSY modules, as it implements most of the re-

quirements and enables the developer to add custom features. Furthermore, the implementation of CLASSY modules using Python methods merged in one simple class, facilitate the use and share of the code as a unique library that can be used with a few simple lines of code. This also simplifies the deployment of the whole system as a container environment, regardless of the hardware or machine. The code and environment setups regarding all approaches and the case study, described in this work, can also be found at `https://gitlab.atnog.org/diogoferreira/classy` for further examination.

It is also important to notice that all the performed tests considered two real datasets that were far from being perfect. The fact that both were small, limited the performance of some tests, mainly the ones related to the RL approach, as it needs a lot of initial information in order to converge successfully. Moreover, they were not annotated, which forced, to each of the tests showcased in this chapter, the manual inspection and judgment of the usefulness of each suggestion. These issues meant that a lot of time had to be spent testing and analyzing each CLASSY change in terms of performance, even considering that we have developed a testing environment to facilitate this process.

6.2 Contributions

The work developed throughout this dissertation led to a set of contributions in many fields, as we will enumerate.

6.2.1 *SmartEnterCom* project

In the context of the Activity 2 – Context-aware applications, a context-aware framework was designed and developed, using users' context information interpretation for behavior adaptation, in the context of collaborative applications in pervasive communities.

6.2.2 Lisbon Machine Learning School 2019

The CLASSY system, including both base and neighborhood approaches (the ones that were developed and tested at the time), was showcased in the LxMLS 2019 Demo Day through a poster, containing the inner functioning of the system (Appendix A), and a live demo created specifically for this event. The live demo was composed of two use cases:

- Simulation of the issues dataset (explained in Section 3.5.1) through the developed interaction interface (mentioned in Section 3.4.5) as a way to show the overall system operation;
- Creation of a live channel with a dedicated agent that had in its knowledge database the content of the main AI related topics [1]. The visitors would scan a Quick Response (QR)

[1] `https://en.wikipedia.org/wiki/Glossary_of_artificial_intelligence`

code that would redirect them to the channel, where they can talk and discuss AI related topics and see the CLASSY agent interacting with them by suggesting some Wikipedia articles useful to the conversational context.

This participation was relevant for showcasing the work developed and to gather feedback from the visitors of the event, in order to enhance the system usability, interaction, and performance. It was also interesting to validate the global component in terms of applications of this system since many visitors suggested personal use cases where they thought it would be interesting to implement this system.

6.2.3 Ubiquitous Computing and Ambient Intelligence 2019

A conference paper [68] regarding the development of a pervasive system, as well as with the implementation of the base and neighborhood approaches and their impacts on the system performance, was submitted to the 13th International Conference on Ubiquitous Computing and Ambient Intelligence (UCAmI). The paper was accepted by the Conference Committee and presented in early December 2019 in Toledo - Spain as part of the Smart Environments track.

6.2.4 Data Science Portugal meetup #61

"Data Science Portugal (DSPT) is an informal community of data science enthusiasts created with the purpose of sharing knowledge and experience in the fields of data science, machine learning, artificial intelligence, big data, and all related fields. Since September 2016, the Data Science Portugal community is building synergies among data passionate people and bridging many cross-domain fields and addressing several data science problems" [2].

In the context of the #61 DSPT meetup entitled "The art of recommending or suggesting", I was invited to showcase the whole CLASSY system, regarding the three approaches (base, neighborhood, and RL). It was a pertinent participation, not only for personal development purposes but also to verify, as it happened in the LxMLS 2019, that this system can be implemented in a bunch of other uses cases with almost no additional changes.

6.2.5 Data Science Portugal Day 2019

In the context of the DSPT day 2019 [3], one of the biggest Data Science events in Portugal, organized by DSPT, I was one of the 14 chosen students that had the opportunity to showcase their work in the main stage. The CLASSY system design and implementation regarding the three approaches (base, neighborhood, and RL) was once more presented.

[2] `https://www.datascienceportugal.com/`
[3] `https://dsptday.com/`

6.2.6 IEEE International Conference on Human-Machine Systems 2020

A conference paper focusing on the RL agent design and implementation, along with the study of the importance of feedback in context limited conversational environments, was submitted to the 1st IEEE International Conference on Human-Machine Systems (ICHMS) 2020. The notification of acceptance will be made in mid-February. The integral content of the paper submitted can be consulted in Appendix B.

6.3 Future Work

Since this work was about designing and implementing a system as a proof-of-concept, some elements were not studied nor implemented for time-saving purposes, and so they should be considered future work, namely:

- Implementation of a summarization tool in the CLASSY system since, as shown in the tests performed in Section 4.1.3, the neighborhood computation process is slow when considering documents with a lot of content, so, summarizing it will enhance the system performance;
- Test the complete system in a larger dataset (more documents and more meaningful messages) since the available datasets were small in content, and so the results were somehow limited in terms of performance, namely the ones considering the RL approach, as these types of system learn with trial-and-error and more trials means more learning;
- Development of an initial auto-tune system, that will find the best initial thresholds (e.g, minimum document score, TF-IDF), by showing some recommendations to the developer that is deploying the system and adjusting the values based on his feedback;
- Richer pre-processing step, with more complex and powerful techniques, using, for instance, some NLP techniques such as Part-of-speech (POS)-tagging in order to include lexical knowledge to the system;
- Inclusion of LTR techniques and algorithms (explored in Section 2.6) in order to add some value and learning ability to the knowledge database/search engine to enable the results list precision to enhance.
- Addition of other means of feedback perceived by the RL agent, since click data is not always the best way to perceive human judgment regarding the suggestions. In this field, it is possible to use metrics like the dwell time [69], which is the actual length of time that a visitor spends on a document before returning to his initial context. Usually, the longer the dwell time the better, as it indicates that the user inspected the suggestion carefully. Using this or other metrics, it is also possible to avoid the impact of misclicks or malicious users in the system learning process.

References

[1] M. Satyanarayanan, "Pervasive computing: vision and challenges," *IEEE Personal Communications*, vol. 8, no. 4, pp. 10–17, 2001.

[2] D. Saha and A. Mukherjee, "Pervasive computing: a paradigm for the 21st century," *Computer*, vol. 36, no. 3, pp. 25–31, mar 2003.

[3] V. B. Kobayashi, S. T. Mol, H. A. Berkers, G. Kismihók, and D. N. D. Hartog, "Text mining in organizational research," *Organizational Research Methods*, vol. 21, no. 3, pp. 733–765, aug 2017.

[4] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval.* Addison Wesley, 1999.

[5] A. Popescu-Belis, E. Boertjes, J. Kilgour, P. Poller, S. Castronovo, T. Wilson, A. Jaimes, and J. Carletta, "The AMIDA automatic content linking device: Just-in-time document retrieval in meetings," in *Machine Learning for Multimodal Interaction.* Springer Berlin Heidelberg, 2008, pp. 272–283.

[6] M. Habibi and A. Popescu-Belis, "Keyword extraction and clustering for document recommendation in conversations," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 4, pp. 746–759, apr 2015.

[7] "Smart EnterCom - Plano de Atividades," Tech. Rep., 2017.

[8] F.-L. Wong and F. Stajano, "Location privacy in bluetooth," in *Security and Privacy in Ad-hoc and Sensor Networks.* Springer Berlin Heidelberg, 2005, pp. 176–188.

[9] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste, "Project aura: toward distraction-free pervasive computing," *IEEE Pervasive Computing*, vol. 1, no. 2, pp. 22–31, apr 2002.

[10] P. Jiang, H. Xia, Z. He, and Z. Wang, "Design of a water environment monitoring system based on wireless sensor networks," *Sensors*, vol. 9, no. 8, pp. 6411–6434, aug 2009.

[11] C. Farrar and G. James, "System identification from ambient vibration measurements on a bridge," *Journal of Sound and Vibration*, vol. 205, no. 1, pp. 1–18, aug 1997.

[12] J. Chen, A. H. Kam, J. Zhang, N. Liu, and L. Shue, "Bathroom activity monitoring based on sound," in *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2005, pp. 47–61.

[13] M. Weiser, "Hot topics-ubiquitous computing," *Computer*, vol. 26, no. 10, pp. 71–72, 1993.

[14] E. A. Martinez, R. Cruz, and J. Favela, "Estimating user location in a WLAN using backpropagation neural networks," in *Advances in Artificial Intelligence - IBERAMIA*. Springer Berlin Heidelberg, 2004, pp. 737–746.

[15] N. Bricon-Souf and C. R. Newman, "Context awareness in health care: A review," *International Journal of Medical Informatics*, vol. 76, no. 1, pp. 2–12, jan 2007.

[16] L. Lijuan, L. Tingjie, and C. Xia, "Network selection based on context-awareness services," *China Communications*, vol. 10, no. 12, pp. 50–56, dec 2013.

[17] B. H. Kasthuriarachchy, K. de Zoysa, and H. Premarathne, "Context-aware sentiment classification," in *2015 Fifteenth International Conference on Advances in ICT for Emerging Regions (ICTer)*. IEEE, aug 2015.

[18] Dictionary.com. (2019, Mar.) Definition of context at dictionary.com. [Online]. Available: https://www.dictionary.com/browse/context

[19] A. Dey, D. Salber, G. Abowd, and M. Futakawa, "The conference assistant: combining context-awareness with wearable computing," in *Digest of Papers. Third International Symposium on Wearable Computers*. IEEE Comput. Soc, 1999.

[20] A. K. Dey, G. D. Abowd, and D. Salber, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications," *Human-Computer Interaction*, vol. 16, no. 2-4, pp. 97–166, dec 2001.

[21] J. Pascoe, "Adding generic contextual capabilities to wearable computers," in *Digest of Papers. Second International Symposium on Wearable Computers*. IEEE Comput. Soc, 1998.

[22] A. hwee Tan, "Text mining: The state of the art and the challenges," in *Proceedings of the PAKDD 1999 Workshop on Knowledge Disocovery from Advanced Databases*, 1999, pp. 65–70.

[23] E. Reategui, M. Klemann, and M. D. Finco, "Using a text mining tool to support text summarization," in *2012 IEEE 12th International Conference on Advanced Learning Technologies*. IEEE, jul 2012.

[24] T. Gong, C. L. Tan, T. Y. Leong, C. K. Lee, B. C. Pang, C. C. T. Lim, Q. Tian, S. Tang, and Z. Zhang, "Text mining in radiology reports," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, dec 2008.

[25] G. G. Chowdhury, "Natural language processing," *Annual Review of Information Science and Technology*, vol. 37, no. 1, pp. 51–89, jan 2005.

[26] A. Singh and S. Desai, "Optical character recognition using template matching and back propagation algorithm," in *2016 International Conference on Inventive Computation Technologies (ICICT)*. IEEE, aug 2016.

[27] A. Aizawa, "An information-theoretic perspective of tf-df measures," *Information Processing & Management*, vol. 39, no. 1, pp. 45–65, jan 2003.

[28] X. Hu, J. Stephen, D. Andreas, and F. Ehmann, "Lyric text mining in music mood classification," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2009.

[29] I. Titov and R. McDonald, "Modeling online reviews with multi-grain topic models," in *Proceeding of the 17th international conference on World Wide Web - WWW*. ACM Press, 2008.

[30] B. Larsen and C. Aone, "Fast and effective text mining using linear-time document clustering," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD*. ACM Press, 1999.

[31] G. Kondrak, "N-gram similarity and distance," in *String Processing and Information Retrieval*, M. Consens and G. Navarro, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 115–126.

[32] S. Huang, M. Ward, and E. Rundensteiner, "Exploration of dimensionality reduction for text visualization," in *Coordinated and Multiple Views in Exploratory Visualization (CMV)*. IEEE, 2005.

[33] A. V. Aho and J. D. Ullman, "Universality of data retrieval languages," in *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages - POPL'79*. ACM Press, 1979.

[34] N. DeClaris, D. Harman, C. Faloutsos, S. Dumais, and D. Oard, "Information filtering and retrieval: overview, issues and directions," in *Proceedings of 16th Annual Interna-*

tional Conference of the IEEE Engineering in Medicine and Biology Society. IEEE, 1994.

[35] S. K. Gorakala, *Building Recommendation Engines.* PACKT PUB, 2016.

[36] S. Jain, A. Grover, P. S. Thakur, and S. K. Choudhary, "Trends, problems and solutions of recommender system," in *International Conference on Computing, Communication & Automation.* IEEE, may 2015.

[37] T.-Y. Liu, "Learning to rank for information retrieval," *Foundations and Trends in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2007.

[38] Dictionary.com. (2019, Mar.) Definition of ranking at dictionary.com. [Online]. Available: https://www.dictionary.com/browse/ranking

[39] S. Robertson, "The probabilistic relevance framework: BM25 and beyond," *Foundations and Trends in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2010.

[40] T. Joachims, H. Li, T.-Y. Liu, and C. Zhai, "Learning to rank for information retrieval (LR4ir 2007)," *ACM SIGIR Forum*, vol. 41, no. 2, p. 58, dec 2007.

[41] W. S. Cooper, F. C. Gey, and D. P. Dabney, "Probabilistic retrieval based on staged logistic regression," in *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR.* ACM Press, 1992.

[42] K. Crammer and Y. Singer, "Pranking with ranking," in *Advances in Neural Information Processing Systems 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. MIT Press, 2002, pp. 641–647.

[43] T. Joachims, "Optimizing search engines using clickthrough data," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD.* ACM Press, 2002.

[44] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22nd international conference on Machine learning - ICML.* ACM Press, 2005.

[45] C. J. C. Burges, R. Ragno, and Q. V. Le, "Learning to rank with nonsmooth cost functions," in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, ser. NIPS'06. Cambridge, MA, USA: MIT Press, 2006, pp. 193–200.

[46] J. Xu and H. Li, "AdaRank: a boosting algorithm for information retrieval," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR.* ACM Press, 2007.

[47] Q. Wu, C. J. C. Burges, K. M. Svore, and J. Gao, "Adapting boosting for information retrieval measures," *Information Retrieval*, vol. 13, no. 3, pp. 254–270, sep 2009.

[48] J. Thom and F. Scholer, "A comparison of evaluation measures given how users perform on search tasks," in *Proceedings of the Twelfth Australasian Document Computing Symposium*, 01 2007.

[49] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of IR techniques," *ACM Transactions on Information Systems*, vol. 20, no. 4, pp. 422–446, oct 2002.

[50] B. McFee and G. R. G. Lanckriet, "Metric learning to rank," in *ICML*, 2010.

[51] O. Berger-Tal, J. Nathan, E. Meron, and D. Saltz, "The exploration-exploitation dilemma: A multidisciplinary framework," *PLoS ONE*, vol. 9, no. 4, p. e95693, apr 2014.

[52] Y. Yue and T. Joachims, "Interactively optimizing information retrieval systems as a dueling bandits problem," in *Proceedings of the 26th Annual International Conference on Machine Learning - ICML*. ACM Press, 2009.

[53] A. Schuth, H. Oosterhuis, S. Whiteson, and M. de Rijke, "Multileave gradient descent for fast online learning to rank," in *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining - WSDM*. ACM Press, 2016.

[54] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Int. Res.*, vol. 4, no. 1, pp. 237–285, May 1996.

[55] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[56] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, may 1992.

[57] R. Bellman, "The theory of dynamic programming," *Bull. Amer. Math. Soc.*, vol. 60, no. 6, pp. 503–515, 11 1954.

[58] R. Howard, *Dynamic Programming and Markov Processes*. Published jointly by the Technology Press of the Massachusetts Institute of Technology and, 1960.

[59] G. A. Rummery and M. Niranjan, "On-line Q-learning using connectionist systems," Cambridge University Engineering Department, Cambridge, England, Tech. Rep. TR 166, 1994.

[60] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[61] K. V. Tate and B. R. Nandwalkar, "Document recommendation using keyword extraction for meeting analysis," *International Journal of Computer Science and Information Technologies*, 2016.

[62] L. Philips, "Hanging on the metaphone," *Computer Language Magazine*, vol. 7, no. 12, pp. 39–44, December 1990.

[63] A. Benítez-Guijarro, Z. Callejas, M. Noguera, and K. Benghazi, "Introducing computational semantics for natural language understanding in conversational nutrition coaches for healthy eating," *Proceedings*, vol. 2, no. 19, p. 506, oct 2018.

[64] Z. Harris, *Mathematical Structures of Language.* John Wiley and Son, 1968.

[65] M. Antunes, D. Gomes, and R. L. Aguiar, "Towards IoT data classification through semantic features," *Future Generation Computer Systems*, vol. 86, pp. 792–798, sep 2018.

[66] G. E. Box and R. D. Meyer, "An analysis for unreplicated fractional factorials," *Technometrics*, vol. 28, no. 1, pp. 11–18, feb 1986.

[67] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. LIX, no. 236, pp. 433–460, oct 1950.

[68] D. Ferreira, M. Antunes, D. Gomes, and R. L. Aguiar, "Classy: A conversational aware suggestion system," *Proceedings*, vol. 31, no. 1, 2019.

[69] J. Duncan, R. Ward, and K. Shapiro, "Direct measurement of attentional dwell time in human vision," *Nature*, vol. 369, no. 6478, pp. 313–315, may 1994.

A

Appendix A - LxMLS2019 CLASSY poster

Poster showcased in the Lisbon Machine Learning School (LxMLS) 2019 Demo Day, containing the inner functioning of the CLASSY at the time.

CLASSY:
A Conversational Aware Suggestion System

Diogo Ferreira[1], Mário Antunes[2], Diogo Gomes[2], Rui Aguiar[2]

DETI, Universidade de Aveiro, Aveiro, Portugal [1]

Instituto de Telecomunicações, Universidade de Aveiro, Aveiro, Portugal [2]

Introduction — Over the last few years, pervasive systems have seen some interesting development. Nevertheless, human-human interaction can also take advantage of those systems by using their ability to perceive the surrounding environment. In this work, we have developed a pervasive system – named **CLASSY** - that is aware of the conversational context and suggests documents potentially useful to the users, based on an Information Retrieval system, and proposed a new scoring approach that uses semantic and distance, based on proximity data in order to classify the relationship between tokens.

Architecture

The proposed **CLASSY** system follows the rules of a pervasive system by:

- being based on data that is collected without human interaction
- being robust and useful, by making suggestions when it has a certain degree of certainty

Prevents the user from constantly being interrupted by system interactions.

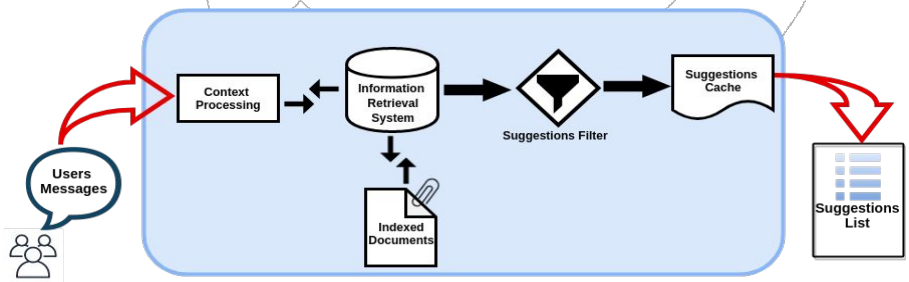


Figure 1 - CLASSY architecture.

Its overall architecture, illustrated in Figure 1 is essentially divided into three main blocks:

- **Context processing**, where user messages are processed to find the most relevant tokens;
- **Information retrieval (IR) system**, that will already have indexed the documents considered relevant;
- **Suggestions filter and cache**, responsible for analyzing the document list retrieved by the IR system, on which some filters are applied and ultimately cache the suggestions.

Neighbourhood approach

To include a more pervasive behavior, a different approach was taken, which tries to use semantic similarity related to each word as a mean to get similar contexts. Each context is defined by a distributional profile. The building process of those profile vectors is described in Figure 2.

```
D1: This is the third report.
D2: This report describes the built automatic system.
D3: This cost report gives respect to the costs related with the last semester.

RTD1: [third, report]
RTD2: [report, automatic, system]
RTD3: [cost, report, cost, semester]


Report neighbourhood(window = 2):
[third: 1, automatic: 1, system: 1, cost: 2, semester: 1]

Report neighbourhood(window = 1):
[third: 1, automatic: 1, cost: 2]
```

Figure 2 - Neighbourhood vector creation

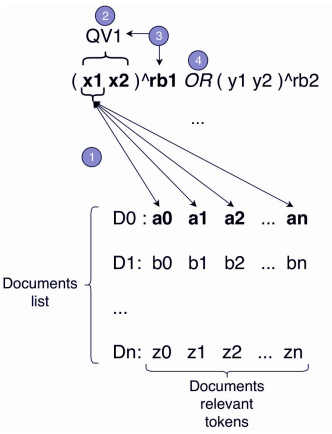


Figure 3 - Neighbourhood score computation steps.

Having built the neighbourhood of all relevant tokens of a document, it is possible to obtain a **neighborhood score** between any two relevant tokens, by computing its inner product. This score is a good indicator of how related two tokens are and inherently how can they refer to the same context.

The steps towards computing the neighbourhood score, illustrated in Figure 3, are:

1. Neighbourhood pair computing;
2. Intermediate sub-query value computation;
3. Recency boost;
4. Final score.

Results

The impact of the **minimum document score** on the base approach:

Minimum document score	Total suggestions	Useful suggestions	Accuracy
10	476	89	18.70%
20	232	64	**27.59%**
30	171	22	12.87%

- There is a big difference between the total number of performed suggestions and those which are considered useful.

Then, the noise files system was implemented. The impact of the number of noise files is shown in the following table:

Number of noise files	Minimum document score	Total suggestions	Useful suggestions	Accuracy
1	20	89	39	43.82%
2	20	102	47	46.08%
3	20	119	67	**56.30%**
4	20	131	69	52.67%

- The use of the noise files system not only reduced the total suggestions but also increased the proportion of useful ones.

Regarding the **neighbourhood approach**, the impact of the minimum document score in this approach:

Minimum document score	Number of noise files	Window size	Total suggestions	Useful suggestions	Accuracy
5	3	1	162	34	20.99%
8	3	1	99	31	31.31%
11	3	1	74	29	39.19%
14	3	1	51	27	52.94%
17	3	1	39	27	69.23%
20	3	1	35	26	74.29%
23	3	1	31	25	**80.65%**

- As we increase the minimum document score, the accuracy also increases, with a notorious value of 80.65% of accuracy. The reason for this is the decrease of contextual suggestions in relation to the direct matches.

In order to mitigate this behavior, a new **hybrid approach** has been proposed, that merges the scoring method of both the base and neighborhood approaches, by multiplying them, to make the general scoring method more robust:

Minimum document score	Number of noise files	Window size	Total suggestions	Useful suggestions	Accuracy
40	3	1	142	40	28.17%
60	3	1	97	38	39.18%
80	3	1	71	35	49.30%
100	3	1	61	31	50.82%
120	3	1	47	29	61.70%
140	3	1	46	29	63.04%
160	3	1	41	28	68.29%
180	3	1	38	27	71.05%
200	3	1	36	27	75.00%
220	3	1	35	27	**77.14%**

- The hybrid approach performance is a good compromise between being pervasive and make useful suggestions (contextual and direct). However, when the minimum document score exceeds 160, the behavior the above approach showed starts to rise again.

This work is funded by FCT/MCTES through national funds and when applicable co-funded by FEDER – PT2020 partnership agreement under the project UID/EEA/50008/2019

B

Appendix B - ICHMS2020 CLASSY paper

Conference paper focusing on the RL agent design and implementation, along with and with the study of the importance of feedback in context limited conversational environments, submitted to the 1st IEEE International Conference on Human-Machine Systems (ICHMS) 2020.

Applying Reinforcement Learning in Context Limited Environments

Diogo Ferreira[1,2], Mário Antunes[1,2] Diogo Gomes[1,2] and Rui L. Aguiar[1,2]
[1] *DETI, Universidade de Aveiro, Aveiro, Portugal*
[2] *Instituto de Telecomunicações, Universidade de Aveiro, Aveiro, Portugal*

***Abstract*—Reinforcement Learning has seen some interesting development over the last years, which made it very attractive to use on recommendation scenarios. In this work, we have extended the previously developed pervasive system, which is aware of the conversational context to suggest documents potentially useful to the users, with the ability to use users' click data as a way to perform better suggestions over time, through a Reinforcement Learning approach. Furthermore, to assure the real significance of these types of approaches in conversational environments, we also conducted a case study regarding the accuracy of feedback on context limited conversational systems.**

***Index Terms*—Pervasive Systems; Context-Awareness; Suggestion Systems; Reinforcement Learning**

I. Introduction

Over the last years, the use of Reinforcement Learning approaches has become more common in many fields [1], [2], [3], due to the flexibility inherent to these types of systems, namely "learn as you go".

The Conversational Aware Suggestion System (CLASSY) was our previous proposal to create a pervasive system aiming to make useful suggestions in scenarios where a specific conversational environment exists. The initial prototype considered only text-based techniques allied to a search engine based architecture and also to a newly proposed approach, named Neighbourhood Approach. The latter used the neighbourhood relationship (applying a distributional profile [4]) as a way to compute a score to re-rank the final results list. With all the developed features, the system was capable of reaching a final accuracy value of 77,14% [5].

Thus, CLASSY is a system built to enhance users' communications. This introduces a metric that can be used as a way to enhance its performance – the ability to learn based on user feedback, achieved by inspecting users' interactions and gather implicit feedback through the clicks in the suggestion. These interactions make it possible for the agent to positively or negatively reinforce a suggestion comparing to a set of candidates, enabling, over time, that the suggestions made regarding a specific state are increasingly accurate.

II. Related Work

Users' feedback has been increasingly valued as a metric by Reinforcement Learning (RL) systems to help them perform specific tasks [6], [7], [8], mainly the ones regarding human-computer interaction.

Y.Seo and B.Zhang [9] proposed a web-based filtering system, that relies on RL methods, capable of filtering results by observing users' reactions and interactions with the documents. Instead of using the term frequency-inverse document frequency (TF-IDF) value for result ranking, it computes a retrieval value, that merges the term frequency (TF) of a term with its weight on the user profile. This state of users' profile is handled by the RL agent, using an $\epsilon - greedy$ [10] algorithm to select the terms that should be considered in the profile. As feedback, it uses metrics like the reading time, bookmarking, scrolling, and thumb up or down. Comparing to other feedback methods, this performed better in terms of relevance as well as adaptation speed. However, as referred by the authors, these results were also enhanced by their testing environment, which was less ambiguous than the global ones.

C.Isbell *et. al* [11] proposed an RL-based agent that resides in the LambdaMOO, a text-based virtual reality online community. The system's goal is to "proactively take actions in this complex social environment, and adapt his behaviour from multiple sources of human reward". Apart from some other features, such as gathering of social statistics, this agent should perform several actions: role calls, which are a common wordplay routine in LambdaMOO, social commentaries, where the agent should make a comment describing the current social state of the Living Room, and social introductions, where the agent introduces two users who have not yet interacted with one another. In terms of rewards, the agent uses implicit and explicit feedback as a way to learn, using the invocation of specific reward and punish verbs (implicit) and by parsing several standard LambdaMOO verbs that are commonly used to express sentiments as explicit feedback. The agent also learns an individual value function for each user, that are combined to determine the next action. Thus, the system is able to maintain user identity so that users who interact more with the agent do not dominate its behaviour. The performed system tests came up with some interesting findings, namely the fact that "humans are fickle, and their tastes and preferences may drift over time" and also that "once the system seems to be behaving as they wish, users feel no need to continually provide reward".

D.Glowacka *et. al* [12] proposed an RL system that

uses users' interactions to guide them in a search task. In the specific context of their work, the goal was to "assist scientists in finding and exploring the relevant literature on a given research topic quickly and effectively". For that, they studied the disadvantages of query-based search techniques, as well as the need for relevance feedback to accomplish their goal. Their proposed system uses feedback on higher-level representations (namely the keywords) as a way to learn with the time. Users directed their search tasks by iteratively clicking on keywords that are suggested through RL methods. Their system also uses an Information Retrieval and Ranking system, together with two modules that aim to build a user-model and also to diverse the document suggests. The suggestion of new keywords is based on a LinRel algorithm [13] that uses the user's feedback obtained in previous iterations. Also, they used the Dirichlet Sampling Algorithm [14] to sample the list of documents to display to the user, and used users' feedback to handle the weights associated with each document in the list. The proposed system outperformed the baseline approach, which was query-based, with almost 10% difference for the F-measure, in the relevance category.

All the above-presented works followed RL approaches, by taking advantage of the users' feedback as a way to enhance the system performance over time. As they focus on use cases different from the one we want to explore, their strengths and weaknesses will be used as a good influence on our proposed solution.

III. Problem Statement

The problem addressed by this paper is the suggestion of information in a conversational environment. The information can be of the most various formats, from Portable Document Format (PDF) to simple text files. However, they must be text-based, since the proposed techniques were developed to handle text. The conversational environment can also take many forms, ranging from tweets, usually with a limited number of characters, to chat rooms, where messages can reach thousands of characters

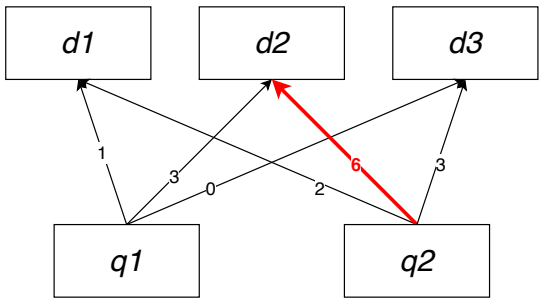
IV. Reinforcement Learning Approach

CLASSY is a system built to enhance users' communications. This means that the user's feedback is an important metric to identify relevant documents. There are usually two types of user feedback – explicit, where a user explicitly rates the system through a measure of quality (for instance a score between 0 and 10), and implicit, where the information regarding the value of the system is tied to the user action. One of the requirements of the project is to be the least intrusive as possible, as such we favour implicit feedback. Though click information can sometimes have some disadvantages like the noise and the so-called "misclick" rate, it will give the system a general approximated measure of performance.

The majority of the RL problems can be modelled following the Markov Decision Process (MDP) [15]. However, the problem we are trying to solve is continuous and randomly distributed. The only information the system has when it first starts is that of previously indexed documents, not knowing the contexts, neither its cardinality or relationships. Also, even the indexed documents can be updated during the agent's execution. With this in mind, we developed an approximation to an MDP, which tries to tackle the continuity and random distribution problem.

A. States

A state describes the environment perceived by the agent at a specific time. Using the proposed Neighbourhood score, a **singular** context state is defined as the top n tokens that characterise the strongest neighbourhood connection between a query (q_i) and a document (d_i). This process is illustrated in Figure 1, where the strongest connection, with a Neighbourhood score of 6, is defined by the token vector $[k1, k2, k3]$, sorted by magnitude.



Neigh(*d2*, *q2*) = [*k1*:*3*, *k2*:*2*, *k3*:*1*]

Fig. 1. Example of a singular state.

Moreover, each suggestion will have a defined context regarding the implicit query, that will define the singular current state. This means that each implicit query can be described as a set of probabilities of belonging to each context state:

$$Q = \sum_{i=1}^{n} P(Q|ctx(x_i)) \tag{1}$$

So, for each implicit query, we use the singular contexts from the top *depth* results, sorted by the score, as the **global** context state, in order to find the action that maximises the reward.

B. Actions

Having collected the states that define the environment perceived by the agent, the possible action for each of these states is the suggestion of one of the documents that composed the states vector, namely the one with the highest probability. Considering that *depth* is defined with the value 3, the top 3 documents from the result list will be considered by the agent (or the maximum possible, when the list is smaller). Moreover, conceiving

that the singular contexts of those 3 documents are defined by $ctx1, ctx2, ctx3$, respectively, the possible actions are suggesting document with $ctx1$, document with $ctx2$, or document with $ctx3$.

C. Policy

Initially, all actions are equiprobable, meaning that, the first time a state is viewed by the agent, it will suggest the top document from the result list (the one with a higher score). However, the usage of an IRS helps to reduce the cold-start problem, by introducing some initial knowledge about the sample. Furthermore, it will also enable faster initial changes when the top document is not the most useful since the other documents will prevail over this right on the next step.

The sum of the probabilities of the actions regarding a state should always be 1.0, so the adjustment of them is defined as:

$$P(s_i, a_j) = P(s_i, a_j) + R(s_i, a_j) \tag{2}$$

The reward value (r) was defined as 1.0, for suggestions that were clicked (and so considered as useful), or documents that were not suggested but whose related suggestion performed was not clicked. On the other hand, for suggestions that were not clicked, or for results that were not suggested but whose related suggestion performed was clicked, the reward is -1.0. This reward function is further conditioned by the learning rate α, which determines how quickly newly acquired information overrides the old one.

$$R(s_i, a_j) = \begin{cases} r\alpha & \text{if } suggested \\ \frac{-r\alpha}{|ctx(x_s)|-1} & \text{if } not\, suggested \end{cases} \tag{3}$$

The equations above consider an optimistic scenario. A document d_1 resulting from the implicit query q_1 can arise a context $ctx(x_1)$, but a document d_2 resulting from the implicit query q_2 can also arise the same $ctx(x_1)$. This is an expected behaviour, due to the context generalisation needed, but it requires an intermediate step to correctly compute the transition probabilities.

This intermediate step adjusts the probabilities mentioned above at every sample as if the agent was seeing the specific context set for the first time, by normalising the values associated with each state, so that the adjustments needed at a time always follow the rule of the sum of the probabilities being always equal to 1.0. Also, when considering new context entries, the normalisation of the remaining ones should consider that this new entry will be equiprobable, with probability $\frac{1.0}{|ctx|}$.

D. Replacement strategies

Giving the continuous nature associated with our problem it is possible to understand that the number of documents and contexts handled by the agent will grow rapidly. So, in order to save space and improve performance, it is necessary to implement a replacement strategy capable of discarding states which are not useful anymore (or at least in a current time window). This strategy will have an impact on the performance of the system, since discarding a state means that the associated probability computed before will also be discarded and so when it appears again, there will be no previous information about it.

With this in mind, the first two considered strategies were the Least Recently Used (LRU), which discards the least recently used items, maintaining the new ones, and also the First In, First Out (FIFO) one, that discards the oldest entries.

We also considered a replacement strategy that takes into account the similarity between context vectors. Broad contexts (states with a high value of global similarity) should be discarded since they are, by definition, closer to several other contexts than narrow ones. Our discard strategy relies on the Pareto principle [16] to this strategy, which states that "for many events, roughly 80% of the effects come from 20% of the causes". Following the logic, when the list of contexts reaches a specific size, the general ones that affect 80% of the effects will be discarded, and the contexts considered to be more specific and so more useful will be maintained.

V. Results

In order to conduct performance tests, we have used two different datasets. The first one contained approximately 2330 messages, originally from an internal chat used by a team of developers, while the second consisted of the content of 100 JIRA [1] issues created by the same team. The objective is to, given the conversational context, suggest the issue they might be talking about. This allows a direct comparison between our current solution and our previous one (named hybrid approach since it combines an IRS with semantic similarity).

Furthermore, it is necessary to evaluate the impact of two parameters: the learning rate α and the *depth*. The optimal testing scenario will independently consider both parameters. However, as they are two-way dependent, it is not possible to perform the tests this way and so, one of the parameters is fixed as the other one changes.

In Figure 2 one can observe the impact of the α parameter over time, in the system's accuracy. The sudden decrease at the 500 messages mark is justified by the random distribution of the conversational topics over the dataset, which also leads to a random distribution of the suggestions made. Thus, on that period a set of bad suggestions was performed, leading to an accuracy decrease. The performance is almost the same until the middle of the simulation, mainly for α `= 0.01` and the hybrid approach. However, after that, both RL setups performed better, with a significant difference of more than 5% between the hybrid approach and the α `= 0.1`.

In terms of the *depth* parameter, its impact on system performance is showed in Figure 3. The distribution

[1] https://www.atlassian.com/software/jira

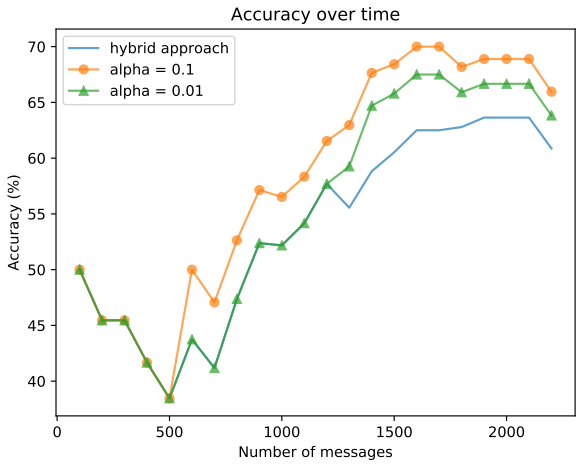


Fig. 2. Impact of α parameter in the performance ($Min.Doc.Score = 140$; $depth = 3$).

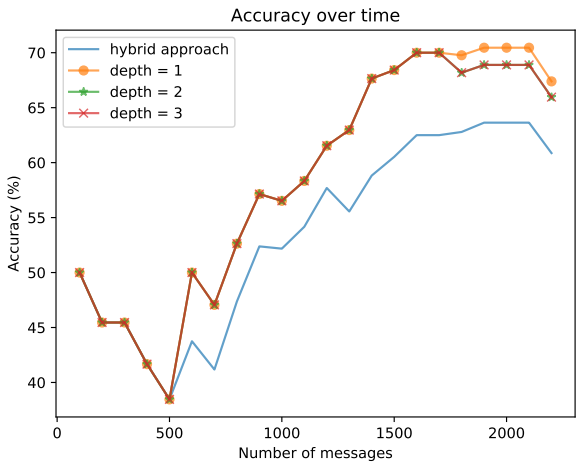


Fig. 3. Impact of *depth* parameter in the performance ($Min.Doc.Score = 140$; $alpha = 0.1$).

behaves the same as described above, but here the improvements appear earlier in the simulation, where all RL setups showed a better performance. At the end, $depth = 2$ and $depth = 3$ showed to have equal performance, while $depth = 1$ produced a slightly better accuracy.

Finally, we should evaluate the impact of the newly proposed replacement technique, when comparing to other existing ones. The results in Table I showed that not only the number of contexts does not impact the overall performance, but also that the similarity-based replacement strategy proposed performs better compared to the other existing ones. These results were executed considering the setup which has achieved the best results in the tests made above and so it is expected that the replacement strategy proposed performs even better in a scenario with more suggestions.

Strategy	Number of contexts	Total sugg.	Useful sugg.	Accuracy
LRU	10 15 20	35	27	77.14%
FIFO	10 15 20	35	27	77.14%
Similarity	10 15 20	33	28	84.85%

TABLE I
REPLACEMENT TECHNIQUES PERFORMANCE.

VI. CASE STUDY: THE IMPORTANCE OF FEEDBACK ON CONTEXT LIMITED ENVIRONMENTS

The introduction of an agent that uses feedback relative to each suggestion performed has shown to be a good approach, comparing to those that only handle the conversational content, in context-based environments. With that in mind, we will also evaluate if this system is able to work in an environment with fewer context, such as social networks. In these environments, a message can lead to a large number of threads, but all of them are limited in terms of the description of the context.

With that in mind, we have implemented a specific setup, aiming to gather information from a social network and suggest useful content to the users. This setup will allow us to validate two points – the usefulness of a feedback-based agent on these types of environments, and also CLASSY performance as a whole, in a different use case. The setup definition is the following:

- Documents - should consider a topic capable of generating controversy and motivate users to interact with the system. For that reason, we chose the *music* topic. The documents will be the information regarding the songs considered hot/most searched at the time (title, artists, genre, and lyrics);
- Conversational environment - we opted to chose Reddit [2], as its structure is very rich and helpful in terms of finding specific niches. Considering the chosen topic – music – it was easy to select some channels where CLASSY will be interacting.
- Feedback - Reddit also implements a good measure of the usefulness of a post, using a points system. As a user, one can up-vote or down-vote a specific post or comment to reflect its judgement. Thus, this score can be a good way to evaluate if a suggestion was really helpful to the users interacting with the specific post.

The setup also considers two execution modules:

- Runner, that will gather the channels content and then send each message at a time to CLASSY, in order to simulate a conversation segment. It will analyse both hot posts made and the stream posts. In the

[2]https://reddit.com/

first, the runner will only parse the title of the post and then the comment threads individually; the latter aims to feed CLASSY only with the post content (title and description) in order to find suggestions that are useful directly. All the information regarding the made suggestions is then stored in a simulations database to make not real-time and post-mortem evaluations.

- Evaluator, that will query all the entries in the simulations database to find its score at the time, in order to extract CLASSY global performance. This module will run every 30 minutes, producing an evaluation summary.

A. Results

The live test was performed for almost 20 days sequentially divided by the chosen channels. It also produced some interesting interactions, with some of them "fooling" the users, since they were not capable of realising that they were interacting with a virtual agent. This shows that being aware of the context is halfway to get better pervasive behaviour in these scenarios.

However, in order to properly evaluate the system's performance in this case study, it is also important to perform the same simulation using the CLASSY setup without users' feedback. This can be accomplished by ignoring the choices made by the RL based agent, using the setup considering only the neighbourhood approach. This evaluation, unlike the one used in the feedback-based simulation, will instead use the final judgement from the users who interacted with each post. This means that, contrarily to the judgement in the live running, that can change over time, in this simulation, that will not happen. Also, when the simulation without feedback triggers a suggestion that was not made in the live scenario, the judgement will be made manually by an external judge.

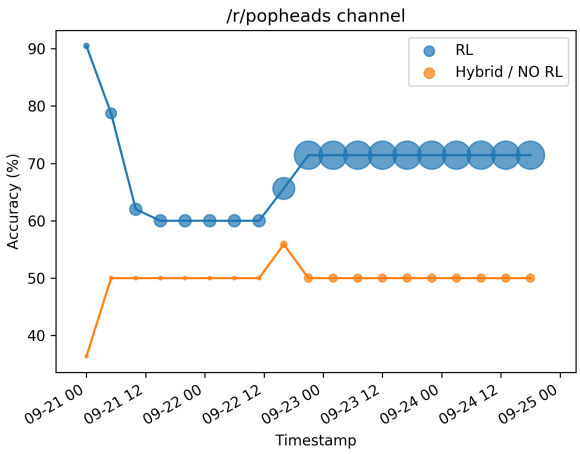


Fig. 4. /r/popheads channel simulation.

The Figures 4, 5 and 6 show the results achieved by CLASSY with and without feedback, with a sampling of

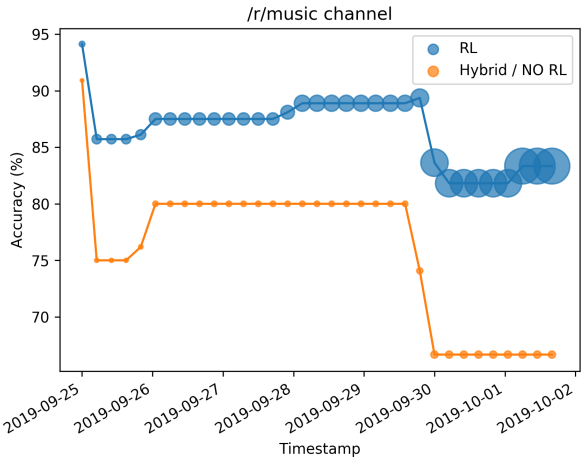


Fig. 5. /r/music channel simulation.

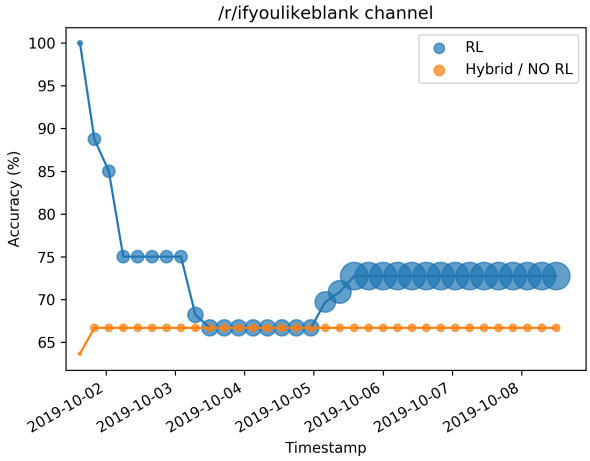


Fig. 6. /r/ifyoulikeblank channel simulation.

5 hours for visualisation purposes. Also, the sizes of the bubbles represent the number of suggestions made at a specific time, growing as a power of 2).

As it may be observed, a feedback-based approach, like the RL one, performed better in all channels, when comparing with the one that disregards it. The final differences were 21% in the `/r/popheads` channel, 17% in `/r/music`, and 7% in `/r/ifyoulikeblank`. The lower differences in the latter happened since the channel does not focus exclusively on music. The channel's users also have more indie tastes, which does not match so well the current top music.

Also, it is possible to check that the RL approach also makes more suggestions – 7 to 4 in `/r/popheads` channel; 12 to 6 in `/r/music`; and 11 to 6 in `/r/ifyoulikeblank`. This is an expected behaviour, since the agent has more information considering what is right and wrong, and so it can have more confidence to perform the suggestion.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have designed and implemented a new CLASSY approach, which has the goal of making the system more pervasive and improve its performance – the Reinforcement Learning one.

This approach was introduced as a way to learn from experience, by inspecting users' interactions and gathering implicit feedback through the clicks in the suggestions. These interactions make it possible for the agent to positively or negatively reinforce a suggestion comparing to a set of candidates, enabling, over time, that the suggestions performed regarding a specific state are increasingly accurate. Thus, even considering the small datasets used to perform these tests, this new approach enhanced CLASSY global performance, having passed the 80% accuracy mark in some of the tests performed, which constitutes an **increase of almost 4% of accuracy**.

Also, we have created a setup to assert the usefulness of feedback to increase a suggestion system performance, like CLASSY, in context limited conversational environments. We chose three Reddit music-related channels as a room to perform the case study. Considering this music topic, we also chose the top 200 current hits and indexed their information in the knowledge database.

The results showed that even considering context limited environments, the users' feedback is an important feature for a suggestion system like CLASSY. Not only do they increase its performance globally, but they also feed the system with more knowledge and make it perform more accurate suggestions over time.

The next step is to add other means of feedback perceived by the RL agent since click data is not always the best way to mirror human judgement regarding the suggestions. In this field, it is possible to use metrics like the dwell time [17], which also avoids the impact of misclicks or malicious users in the system learning process.

VIII. ACKNOWLEDGEMENTS

This work is supported by FCT/MCTES through national funds and when applicable co-funded by FEDER – PT2020 partnership agreement under the project UID/EEA/50008/2019

References

[1] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg, 2006, pp. 363–372.

[2] P. Kormushev, S. Calinon, and D. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," *Robotics*, vol. 2, no. 3, pp. 122–148, jul 2013.

[3] H. Abdelgawad, B. Abdulhai, S. El-Tantawy, A. Hadayeghi, and B. Zvaniga, "Assessment of self-learning adaptive traffic signal control on congested urban areas: independent versus coordinated perspectives," *Canadian Journal of Civil Engineering*, vol. 42, no. 6, pp. 353–366, jun 2015.

[4] M. Antunes, D. Gomes, and R. Aguiar, "Towards iot data classification through semantic features," *Future Generation Computer Systems*, 2017.

[5] D. Ferreira, M. Antunes, D. Gomes, and R. L. Aguiar, "Classy: A conversational aware suggestion system," *Proceedings*, vol. 31, no. 1, 2019. [Online]. Available: https://www.mdpi.com/2504-3900/31/1/40

[6] E. Bhasker, S. Brown, and W. Griswold, "Employing user feedback for fast, accurate, low-maintenance geolocationing," in *Second IEEE Annual Conference on Pervasive Computing and Communications, 2004. Proceedings of the*. IEEE, 2004.

[7] D. Pagano and W. Maalej, "User feedback in the appstore: An empirical study," in *2013 21st IEEE International Requirements Engineering Conference (RE)*. IEEE, jul 2013.

[8] T. Mei, B. Yang, X.-S. Hua, and S. Li, "Contextual video recommendation by multimodal relevance and user feedback," *ACM Transactions on Information Systems*, vol. 29, no. 2, pp. 1–24, apr 2011.

[9] Y.-W. Seo and B.-T. Zhang, "Learning user's preferences by analyzing web-browsing behaviors," in *Proceedings of the fourth international conference on Autonomous agents - AGENTS '00*. ACM Press, 2000.

[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[11] C. Isbell, C. R. Shelton, M. Kearns, S. Singh, and P. Stone, "A social reinforcement learning agent," in *Proceedings of the fifth international conference on Autonomous agents - AGENTS '01*. ACM Press, 2001.

[12] D. Glowacka, T. Ruotsalo, K. Konuyshkova, kumaripaba Athukorala, S. Kaski, and G. Jacucci, "Directing exploratory search," in *Proceedings of the 2013 international conference on Intelligent user interfaces - IUI '13*. ACM Press, 2013.

[13] P. Auer, "Using confidence bounds for exploitation-exploration trade-offs," *J. Mach. Learn. Res.*, vol. 3, pp. 397–422, Mar. 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=944919.944941

[14] D. Glowacka and J. Shawe-Taylor, "Content-based image retrieval with multinomial relevance feedback," in *Proceedings of 2nd Asian Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Sugiyama and Q. Yang, Eds., vol. 13. Tokyo, Japan: PMLR, 08–10 Nov 2010, pp. 111–125. [Online]. Available: http://proceedings.mlr.press/v13/glowacka10a.html

[15] R. Bellman, "A markovian decision process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[16] G. E. Box and R. D. Meyer, "An analysis for unreplicated fractional factorials," *Technometrics*, vol. 28, no. 1, pp. 11–18, feb 1986.

[17] J. Duncan, R. Ward, and K. Shapiro, "Direct measurement of attentional dwell time in human vision," *Nature*, vol. 369, no. 6478, pp. 313–315, may 1994.