proceedings

Proceedings

CLASSY: A Conversational Aware Suggestion System †

Diogo Ferreira 1,2,*, Mário Antunes 1,2, Diogo Gomes 1,2 and Rui L. Aguiar 1,2

1 DETI, Universidade de Aveiro, 3810-193 Aveiro, Portugal; mario.antunes@av.it.pt (M.A.); dgomes@av.it.pt (D.G.); ruilaa@av.it.pt (R.L.A.)
2 Instituto de Telecomunicações, Universidade de Aveiro, 3810-193 Aveiro, Portugal
* Correspondence: pdiogoferreira@ua.pt; Tel.: +351-917-639-227

† Presented at the 13th International Conference on Ubiquitous Computing and Ambient Intelligence UCAmI 2019, Toledo, Spain, 2–5 December 2019.

Published: 20 November 2019

Abstract: 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 semantics and distance based on proximity data in order to classify the relationship between tokens.

Keywords: pervasive systems; context aware; suggestion systems; information retrieval; natural language processing

1. Introduction

Over the last few years, discussions regarding pervasive computing have become more common, and it has seen some great development [1–4], due to the overall innovations and possibilities inherent to this concept and mainly to the fact that our surrounding environment is becoming even more monitored every day [5–7].

Unlike traditional computing systems, a pervasive system requires a feed of constant data about its surrounding environment, acquired by sensors of the most various types, ranging from hardware sensors, like temperature and location, to software, such as agents that parse tweets or messages on a chat room. These sensors capture and provide data in a non-invasive way, enabling the "ability to adapt to changing circumstances and respond based on the context of use" [8].

This sense of context can be used in many fields, such as a conversational one, which will be addressed in this paper. Here, context is somewhat related to the topic on which the whole conversation is unfolding. By perceiving this context, it allows the proposed application to enhance the communications through useful suggestions, as the conversational system takes some actions towards helping users regarding that specific context. This is useful in many situations, such as developer forums or technical support services, since it introduces an intermediate layer between the sources (the user and the developer, respectively) and the target (the technician and the issues/code, respectively) that enhances and facilitates their relationships by using the suggestions to the contexts the source is perceiving.

Along with the direct interaction with the users, the problem of conversational intrusion comes up. A system that faces constant human interaction should have conversational intelligence, enabling it "to actively participate in the conversation and to demonstrate awareness of the topic discussed, the evolving conversational context, and the dialogue flow". This ability is powered by following some social rules like proactivity and conscientiousness in order to make the system more useful and less intrusive [9].

2. Related Work

Over time, a lot of effort was made to build systems capable of facilitating not only the human-computer interaction but also the human-human interaction [10,11]. In the context of this paper, the latter will be the focus, where recommendation systems are included, which, based on the processing of historical data, make suggestions that may be relevant to the end-user.

Popescu-Belis, Boertjes, Kilgour, Poller, Castronovo, Wilson, Jaimes, and Carletta [12], in a project named Automatic Content Linking Device (ACLD), 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 Automatic Speech Recognition (ASR) at fixed time intervals to ultimately perform a pre-built database search. The collected data was pre-processed in order to only retain the keywords by applying some techniques such as stop-words removal and stemming. The keywords were then grouped to build an implicit query and sent to the Information Retrieval (IR) system, whose job was to return the documents that it found relevant. Finally, the returned document list is ordered by relevance. 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, as no results were provided in this paper nor in subsequent publications, the effective use of this architecture was left open.

Sometime after Popescu-Belis et al. published their work in [12], Popescu-Belis (one of the authors of the above work) and Habibi [13] studied the problem related to the extraction of keywords in a conversational environment, based on the ACLD system above described. They focused their efforts on solving the issue regarding the fact that a conversational context is not always bound to one specific topic but to a lot of them, 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 settled in an intermediary step before the implicit query formation, which first 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 with 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" [13].

Another similar work with a common goal was developed by Tate and R.Nandwalkar [14], 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, here the authors used a subsequent step following the extraction of keywords that consisted in using the Metaphone [15], an algorithm published by Lawrence Philips in 1990, composed by 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 with a simpler one similar to the proposed by Popescu-Belis et al. in the ACLD, and concluded that their new approach extracts more keywords for different input, leading to higher final accuracy.

Recently, Benítez-Guijarro, Callejas, Noguera, and Benghazi [16] have 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, ingredients, etc. 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 that are able to interpret every imaginable situation, which is infeasible.

3. Document Recommendation Problem and the CLASSY Solution

The specific problem addressed by this paper is the suggestion of information in a conversational environment. The suggested documents 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 that will be described in the next sections 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.

The proposed CLASSY system follows the rules of a pervasive system, by being based on data that is collected without human interaction and by being robust and useful so that it only makes suggestions when it has a certain degree of certainty regarding its recommendation, therefore, preventing the user from constantly being interrupted by system interactions. Its overall architecture, illustrated in Figure 1, is essentially divided into three blocks – context processing, IR system, and suggestions filter and cache.

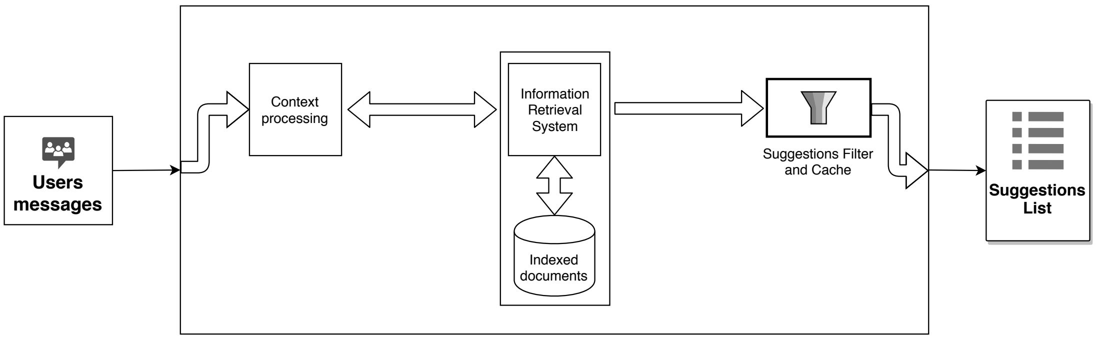

Figure 1. CLASSY architecture, that is essentially divided into three blocks – **context processing**, responsible to find the most relevant tokens and manage the context; **IR system**, which has previously indexed the documents that are considered relevant to the users or conversational topics; and **suggestions filter and cache**, are responsible for analyzing the document list retrieved by the IR system, applying some filters to it, and cacheing the suggestions.

3.1. Context Processing

The context processing block is where user messages are processed in order to find the most relevant tokens – the keywords. This block is composed of the following steps:

- Relevant token extraction, which uses an analyzer defined in the IR engine to apply a set of processing techniques to the considered text. This analyzer is used both in the indexing phase (the documents list) and in the query phase, to force the query tokens to follow the same rules and have a similar syntax to the documents indexed tokens. The analyzer pipeline is the following:

1. UAX29 URL Email Tokenizer, which splits the text field into tokens, using white spaces and punctuation as delimiters. Delimiter characters are discarded, with the exception of dots that are not followed by white space, words with hyphens that contain numbers, internet domain names containing top-level domains validated against the white list in the IANA Root Zone Database, URLs and email, IPv4 and IPv6 addresses;
2. Length filter, that removes words whose length is smaller than 2 and exceeds 12. These values are a fair compromise considering the average word lengths of the various existing languages (http://www.ravi.io/language-word-lengths) to filter small irrelevant words and also the bigger ones, which are atypical;
3. Lower case filter, that transforms any uppercase letters tokens to the equivalent lowercase one;
4. Stop word filter, that discards or stops the analysis of tokens that are on a given provided stop words list;
5. Snowball Porter Stemmer, which is a language-specific stemmer generated by Snowball, a software package that runs pattern-based word stemmers;
6. Length filter, which removes words whose length is smaller than 2 and exceeds 10. This filter follows the same logic as the previous one but now considering smaller words resulting from the stemmer process.

For every token set of each analyzed query, its Inverse Document Score (IDF) is computed, discarding tokens whose value sited under a certain threshold (a system parameter that, for now, should be defined manually by finding the value that attains a good compromise in terms of filtering the most irrelevant tokens), essentially those that do not show sufficient informative interest;

- Context update, a stage where the context is updated with the new tokens resulting from the previous step. This context has a specific size, and it is defined by the relevant tokens of the last *n* queries;
- Implicit query formation, where the query is built with the disjunction of the context entries, boosted with its recency (the position in the context). This implicit query is then sent to the IR system in order to get the result list that will contain documents likely relevant to the user. An example of this stage is shown in Figure 2, where $Q*$ are the queries, $RTQ*$ are the relevant tokens from each query, and IQ is the resulting implicit 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 2. Implicit query formation, based on the disjunction of the extracted context entries (relevant tokens), boosted with its recency.

Additionally, the use of a large set of topics and context in the conversational scenario will result in an increase of noise perceived by the system. For that, it is necessary to create a negative class document that would contain a rather large historical registry of exchanged messages. Implicit queries, including the mentioned negative class in its result list, are seen as having more noise associated with its context, thus not adding value as a suggestion source. This negative class is put into practice in the form of a noise file system. This system has a maximum size, which defines the number of noise files that can be created and used as the negative class. As mentioned, 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.

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

However, sometimes, instead of talking about a specific topic, people explicitly refer to some related keys. In this field, a topic can take a lot of forms, by being the title of a document, the author of a tweet, the key of an issue, and so on. This type of interactions are not context but specific topic matches, thus it is useful to perform a direct search for them. With that in mind, a simple query is built by using all tokens in the received message and trying to find a direct match between the query and the document, based on the field that we expect to find (the title, author, key, etc.) If a hit occurs, then the result of this stage is favored over the context results.

3.2. IR System

The IR system will previously index the documents that are considered relevant to the users or conversational topics. Assuming that, its job is to receive the queries built from the Context Processing block and then set up a list of relevant documents according to those queries.

3.3. Suggestions Filter and Cache

The suggestions filter and cache blocks are responsible for analyzing the document list retrieved by the IR system, on which some filters are applied and ultimately cache the suggestions.

The first one, and the most logical, is filtering the list of documents by its own score, since only documents with a score above a certain threshold are considered to have enough value to be used as a suggestion to the user. This threshold, as the IDF one, is a system parameter that, for now, should be defined manually by finding the value that attains a good compromise in terms of the usefulness of the results.

To increase the value of the suggestions and minimize the conversational intrusion of the system, a suggestion cache must be created, whose goal is to avoid making the same suggestion in consecutive or almost consecutive time-stamped messages, since it does not add any value to the user. This cache also works by establishing a maximum time constraint 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. This maximum time constraint is defined manually since it is a subjective parameter that is domain-dependent.

3.4. Neighbourhood Approach

To include a more pervasive behavior, a different approach – named neighbourhood approach – was taken, which tries to use semantic similarity related to each word as a mean to get similar contexts.

Semantic distance/similarity is a property of lexical units, typically between words, but this notion can be generalized to larger units such as phrases, sentences, etc. 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 is due to 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 ones – corpus-based – that are based only on co-occurrence statistics from large corpora. They rely on the hypothesis that words with similar contexts tend to be semantically close [17]. The set of contexts of each target word u is represented by its distributional profile, the set of words that tend to co-occur with u within a certain distance, along with numeric scores signifying this co-occurrence tendency with u. This profile, introduced by M. Antunes, D. Gomes, and R. Aguiar [18] is defined as:

$$P(u) = [\{w_1, o(u, w_1)\}; ...; \{w_i, o(u, w_i)\}] \tag{1}$$

where u is the target word and $o(u, w)$ the number of times that w occurs near to u.

The building process of those vectors is described in Figure 3, 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 neighbourhood(window = 2):
[third: 1, automatic: 1, system: 1, cost: 2, semester: 1]

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

Figure 3. Neighbourhood vector creation of the *report* token based on three messages, considering a neighborhood window of 2 and 1.

Having built the neighborhood of all relevant tokens of a document, it is possible to obtain a neighborhood score between any two relevant tokens, by computing the inner product, and not only the cosine similarity – since we are interested in the magnitude of the similarity – of its neighborhood vectors:

$$NS(u, v) = |u| \times |v| \times cosine(u, v) \tag{2}$$

$$cosine(u, v) = \frac{\sum_{i=1}^{n} o(u, w_i) \times o(v, w_i)}{\sqrt{\sum_{i=1}^{n} o(u, w_i)^2} \times \sqrt{\sum_{i=1}^{n} o(v, w_i)^2}} \tag{3}$$

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

The process of building neighborhoods and computing their scores was implemented inside the IR engine, resorting to all the potential and efficiency of the Lucene’s internal structures that enabled the handling of individual token analysis. This allowed us to use the neighborhood score, based on the implicit query, as the value used to re-rank the top n documents retrieved by the Lucene engine to improve the contextual analysis of each document. The steps towards computing the neighborhood score, illustrated in Figure 4, are:

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

2. Intermediate sub-query value computation – the score associated between each document and each sub-query will be the average of the neighborhood scores computed as above described;
3. Recency boost – each sub-query has a boost that is associated with its recency. The final score associated with each document and each sub-query will come from the multiplication between the recency value and the intermediate value above presented;
4. Final score – The final document score will be the sum of all neighborhood values computed between each document and each sub-query.

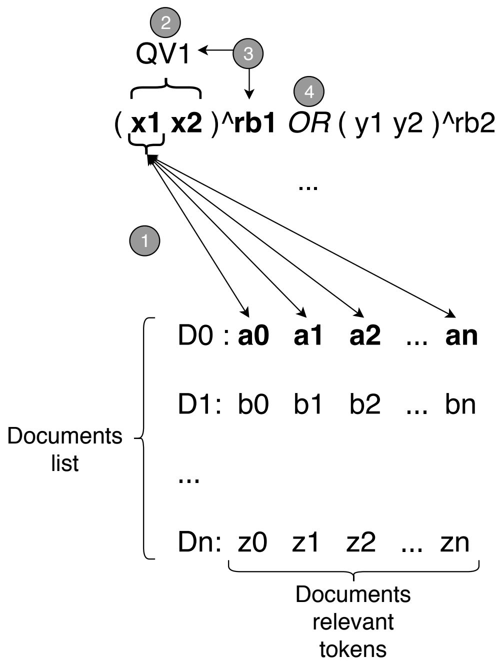

Figure 4. Neighbourhood score computation steps, composed by the neighbourhood pair computing, intermediate sub-query value computation, recency boost, and final score computation.

4. Evaluation

Since the main goal of this work was to build a recommendation system running in a pervasive way, a mature and robust framework was used that serves as the cornerstone of the IR block presented in the last section – Solr (https://lucene.apache.org/solr/). Solr is an open-source search platform from the Apache Lucene project, which features full-text search and real-time indexing and provides distributed search and index replication, whose design was made with scalability and fault tolerance in mind. Furthermore, Solr allows developers to add plugins that can use the indexed data and its structures to modify and enhance the retrieved results.

In order to create a comparative benchmark to evaluate the proposed CLASSY system, we implemented a base approach, that uses the blocks described in the Section 3 without making any enhancements to the IR block nor its score method, whose goal is to add value to the system by developing a valuable pre-processing step of the messages and post-processing of the result list.

Then, we implemented the full CLASSY system, by using the neighborhood approach (described in Section 3.4) as part of our system in order to fully potentiate its pervasive behavior.

To assess the performance of every presented strategy in Section 4, a series of tests were conducted on a dataset containing approximately 2330 messages, originally from an internal chat used by a team of developers of a specific company working with contact center services. The indexed documents were the content of JIRA (https://www.atlassian.com/software/jira) issues created by the team, with the goal of, given the conversational context, suggesting the issue they might be talking about. The objective was to suggest a single issue, rather than a list; therefore, in all conducted tests, the suggestion that was made always respects the top document on the results list.

5. Results

As already mentioned in the above section, besides taking different approaches, other small improvements and features were added.

Regarding the base approach, the first thing to analyze is the minimum document score and its impact on the results. This study is shown in Table 1. As one might observe, there is a big difference between the total number of performed suggestions and those which are considered useful. This contradicts what was mentioned about the amount of noise resulting from the messages.

Table 1. Base approach – minimum document score impact.

Minimum Document Score	Total Suggestions	Useful Suggestions	Accuracy
10	476	89	18.70%
20	232	64	**27.59%**
30	171	22	12.87%

Taking into account the proposed solution to overcome this issue, the noise files system was then implemented. The next step was to study the impact of the number of noise files in the final result. These results are shown in Table 2. On those tests, the minimum score was maintained at 20, since it was the one showing the best results (and the most logical ones). As it is possible to see, the use of the noise files system not only reduced the total suggestions but 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.

Table 2. Base approach + Noise files system – number of noise files impact.

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%

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. As it is possible to observe, these results do not show a systematic evolution, since they are very dependent on the position of the 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.

Table 3. Base approach + Noise files system – number of messages between suggestions impact.

Messages between Recommendations	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%

Finally, the neighborhood approach, which is slightly different from the base one when it comes to scoring the documents, since the latter uses the Lucene score, which is a variant of the TF-IDF scoring method, while the first computes the score based on the neighborhood vector comparison, as shown in Section 3.4.

Regarding this approach, the first thing to investigate is the neighborhood window, which defines the maximum distance between two tokens that makes them neighbors. The results in Table 4 showed that using a window size bigger than one will add a considerable amount of noise, thus resulting in much more invasive and useless suggestions.

Table 4. Neighbourhood approach – window impact.

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%

Having set the neighborhood window size, it is time to analyze again the impact of the minimum document score in this approach. The results obtained with this method, presented in Table 5, are very interesting. As we increase the minimum document score, the accuracy also increases, with a notorious value of 80.65% of accuracy for a score of 23. Although these results seem optimistic, the reason for this sudden increase in accuracy is the decrease of contextual suggestions in relation to the direct matches performed by the system, which essentially is a behavior to avoid.

Table 5. Neighbourhood approach – minimum document score impact.

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%**

In order to mitigate the mentioned behavior but still use its inherent advantages, a new hybrid approach has been proposed that merges the scoring method of both the base and neighborhood approaches, by computing both their values and multiplying them, to make the general scoring method more robust. The results in Table 6 showed that the hybrid approach performance is a good compromise between being pervasive and making useful suggestions (contextual and direct). However, it was observed that when the minimum document score exceeds 160 the behavior the above approach showed starts to rise again, and the contextual suggestions start to decrease.

Table 6. Hybrid approach – minimum document score impact.

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%**

6. Conclusions and Future Work

Users demand systems that are pervasive and not invasive. In this paper, we proposed a system – named CLASSY – that follows those rules so that it is able to be aware of the context where the users are integrated and make suggestions that prove to be useful to them.

Along with the development of CLASSY, we faced several problems related to the acquisition of context as well as dealing with all uncertainty and noise associated with human speech. Those problems led to the introduction of specific features, such as the noise files system, that made the system more robust and useful.

The use of a mature information retrieval engine and all its features potentiated the development and results of a system like that, and the alliance of the IR engine used with the new proposed neighborhood score provided a way to enhance the behavior and the results of this system.

As future work, we plan on adding intelligence to this system, by allowing it to learn from user feedback. For that, we should associate each document-query pair with a specific weight which will reward good suggestions and punish the opposite. The variations in this weight will result from users' implicit feedback, and they will make the system smarter as they increasingly interact with it. To accomplish this, the system will have to be executed in a live conversational environment, which can make it possible to ignore the logical disadvantages of using offline datasets.

Acknowledgments: 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. Saha, D.; Mukherjee, A. Pervasive computing: a paradigm for the 21st century. *Computer* **2003**, *36*, 25–31, doi:10.1109/mc.2003.1185214.
2. Wong, F.L.; Stajano, F. Location Privacy in Bluetooth. In *Security and Privacy in Ad-Hoc and Sensor Networks*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 176–188, doi:10.1007/11601494_15.
3. Satyanarayanan, M. Pervasive computing: vision and challenges. *IEEE Pers. Commun.* **2001**, *8*, 10–17, doi:10.1109/98.943998.
4. Garlan, D.; Siewiorek, D.; Smailagic, A.; Steenkiste, P. Project Aura: toward distraction-free pervasive computing. *IEEE Pervasive Comput.* **2002**, *1*, 22–31, doi:10.1109/mprv.2002.1012334.
5. Jiang, P.; Xia, H.; He, Z.; Wang, Z. Design of a Water Environment Monitoring System Based on Wireless Sensor Networks. *Sensors* **2009**, *9*, 6411–6434, doi:10.3390/s90806411.
6. Farrar, C.; James, G.J., III. System Identification from ambient vibration measurements on a bridge. *J. Sound Vib.* **1997**, *205*, 1–18, doi:10.1006/jsvi.1997.0977.
7. Chen, J.; Kam, A.H.; Zhang, J.; Liu, N.; Shue, L. Bathroom Activity Monitoring Based on Sound. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 47–61, doi:10.1007/11428572_4.
8. Martínez, E.A.; Cruz, R.; Favela, J. Estimating User Location in a WLAN Using Backpropagation Neural Networks. In *Advances in Artificial Intelligence—IBERAMIA 2004*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 737–746, doi:10.1007/978-3-540-30498-2_74.
9. Chaves, A.P.; Gerosa, M.A. How should my chatbot interact? A survey on human-chatbot interaction design. *arXiv* **2019**, arXiv:1904.02743.
10. Bernsen, N.O.; Dybkjær, H.; Dybkjær, L. Cooperativity in human-machine and human-human spoken dialogue. *Discourse Process.* **1996**, *21*, 213–236, doi:10.1080/01638539609544956.
11. Isbister, K.; Nakanishi, H.; Ishida, T.; Nass, C. Helper agent. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems—CHI '00, The Hague, The Netherlands, 1–6 April 2000; ACM Press: New York, NY, USA, 2000, doi:10.1145/332040.332407.
12. Popescu-Belis, A.; Boertjes, E.; Kilgour, J.; Poller, P.; Castronovo, S.; Wilson, T.; Jaimes, A.; Carletta, J. The AMIDA Automatic Content Linking Device: Just-in-Time Document Retrieval in Meetings. In *Machine Learning for Multimodal Interaction*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 272–283, doi:10.1007/978-3-540-85853-9_25.

13. Habibi, M.; Popescu-Belis, A. Keyword Extraction and Clustering for Document Recommendation in Conversations. *IEEE/ACM Trans. Audio Speech Lang. Process.* **2015**, *23*, 746–759, doi:10.1109/taslp.2015.2405482.
14. Tate, K.V.; Nandwalkar, B.R. Document Recommendation Using Keyword Extraction for Meeting Analysis. *Int. J. Comput. Sci. Inf. Technol.* **2016**, *7*, 1763–1766.
15. Philips, L. Hanging on the Metaphone. *Comput. Lang. Mag.* **1990**, *7*, 39–44.
16. Benítez-Guijarro, A.; Callejas, Z.; Noguera, M.; Benghazi, K. Introducing Computational Semantics for Natural Language Understanding in Conversational Nutrition Coaches for Healthy Eating. *Proceedings* **2018**, *2*, 506, doi:10.3390/proceedings2190506.
17. Harris, Z. *Mathematical Structures of Language*; John Wiley and Son: Hoboken, NJ, USA, 1968.
18. Antunes, M.; Gomes, D.; Aguiar, R.L. Towards IoT data classification through semantic features. *Future Gener. Comput. Syst.* **2018**, *86*, 792–798, doi:10.1016/j.future.2017.11.045.