TÉCNICO LISBOA

Using subtitles to deal with Out-of-Domain interactions

Daniel Filipe Nunes Magarreiro

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Prof. Francisco António Chaves Saraiva de Melo

Examination Committee

Chairperson:	Prof. Mário Rui Fonseca dos Santos Gomes
Supervisor:	Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Member of the Committee:	Prof. Carlos António Roque Martinho

November 2014

Dedicated to my grandparents

Resumo

Pessoas que interagem com sistemas de diálogo, muitas vezes colocam questões às quais os sistemas não conseguem responder, denominadas de Out of Domain (OOD).

Recentemente, um corpus construído com interacções de legendas de filmes, chamados Subtle, foi construído e utilizado em Say Something Smart (SSS), um chatbot que lida com interacções OOD. Quando confrontado com uma pergunta do utilizador, SSS consulta a base de conhecimento usando Lucene , um sistema de recuperação de informação, que devolve um conjunto de respostas possíveis. Anteriormente, SSS utilizava duas métricas simples para escolher uma resposta a partir deste conjunto: verificar a semelhança da entrada do usuário com perguntas na base de conhecimento e calculando a frequência da resposta. Embora simples, esta abordagem nem sempre dá bons resultados, especialmente quando se lida com questões menos usuais.

Neste trabalho desenvolvemos duas maneiras alternativas de seleccionar uma resposta de um conjunto de respostas possíveis, com o objectivo de melhorar os resultados globais do SSS: uma é combinar várias medidas de uma forma linear e outra é usar o paradigma Learning to Rank.

Nas nossas avaliações, obtivemos 61,67% de respostas plausíveis usando uma combinação de quatro medidas e também obtivemos resultados promissores com a abordagem Learning to Rank, conseguindo 35 pontos percentuais a mais de respostas adequadas do que com a combinação linear de medidas. Realizámos uma avaliação comparando a versão actual do SSS com a anterior e descobrimos que a actual é capaz de responder a mais perguntas do usuário de maneira adequada, especialmente para Português.

Palavras-chave: Sistemas de Diálogo, Out of domain, Legendas, Learning to Rank.

Abstract

People that interact with dialogue systems often pose questions that cannot be handled by the system, called Out of Domain (OOD).

Recently a corpus built using interactions from movie subtitles, called Subtle, was built and used in Say Something Smart (SSS), a chatbot that deals with OOD interactions. When faced with a user question, SSS consults the knowledge base using Lucene, an information retrieval system, that returns a set of possible answers. Previously, SSS used two simple metrics to choose one answer from this set: checking the similarity of the user input with questions in the knowledge base and the frequency of the answer. Although simple, this approach does not always give good results, especially when dealing with more unusual questions.

In this work we develop two alternative ways of selecting an answer from a list of possible ones, with the goal of improving the overall results of SSS: one is to combine several measures in a linear way and the other to use the learning to rank paradigm.

In our evaluations we were able to obtain 61.67% of plausible answers using a combination of four measures and we also got promising results with the Learning to Rank approach, achieving 35 percentage points more of suitable answers than with the linear combination of measures. We also conducted an evaluation comparing the current version of SSS with the previous and found that the current one is able to answer much more user requests suitably than the previous, especially for Portuguese.

Keywords: Dialogue Systems, Out of domain, Subtitles, Learning to Rank.

Contents

Resumo v
Abstract vii
List of Tables xi
List of Figures xiii
Glossary xv

1 Introduction **1**
1.1 Motivation 1
1.2 Goals 2
1.3 Thesis organization 2

2 Related Work **5**
2.1 Previous work done in INESC-ID 5
2.1.1 Language Understanding Platform 5
2.1.2 Say Something Smart 6
2.1.3 Just.Ask 8
2.2 Dealing with OOD interactions 10
2.2.1 Virtual Agent Max 10
2.2.2 Hans Christian Andersen 10
2.2.3 Sergeant Blackwell 11
2.2.4 IRIS 12
2.3 Corpora 13
2.3.1 Movie-DiC 13
2.3.2 Cornell Movie-Dialogs Corpus 14
2.3.3 Personage Corpora 15
2.3.4 Subtle 16

3 Improving the answers given by SSS **19**
3.1 Building the Subtle corpus: Improvements 19
3.2 Context of the conversation 21
3.3 Adapting Lucene to Portuguese 22
3.4 Normalization techniques 23

3.5 Scoring the T-A pairs . . . 23
3.6 Evaluations . . . 26
3.6.1 Evaluation setup . . . 26
3.6.2 Are subtitles adequate? . . . 26
3.6.3 Evaluating the different measures for scoring . . . 27

4 Learning to rank possible answers **29**
4.1 The learning to rank paradigm . . . 29
4.2 Creation of the training corpus . . . 29
4.3 Extracting new features . . . 30
4.4 Description of the Framework . . . 31
4.5 Evaluating the different algorithms . . . 32
4.5.1 Discussion . . . 33

5 Final evaluation **35**
5.1 Evaluation setup . . . 35
5.2 Evaluation using the English Corpus . . . 36
5.3 Evaluation using the Portuguese Corpus . . . 36
5.4 Discussion . . . 37

6 Conclusions **39**
6.1 Contributions . . . 40
6.2 Future Work . . . 40

Bibliography **46**

7 Questionnaire used to evaluate combination of measures **47**

8 OOD English requests posed to Edgar **54**

9 OOD Portuguese requests posed to Edgar **56**

List of Tables

2.1 Table with the first 5 hits to the query “How are you?” using Lucene, taken from [1] 8
2.2 Example of noise in the corpus 13
2.3 Polarity score with SentiWordNet 16

3.1 Table with the results for each different analyzer 23
3.2 Table with the results for each different setting 28

4.1 Agreement between the two annotators 30
4.2 Precision at One of the different algorithms 32
4.3 Precision at One of the different algorithms 32

5.1 Comparison between the results of the previous version of SSS with the current for the English corpus 36
5.2 Comparison between the results of the previous version of SSS with the current for the Portuguese corpus 37

List of Figures

2.1 Typical architecture of a QA system, taken from [27]. 8
2.2 XML example from [3]. 14

3.1 Example of the new Subtle 20
3.2 Snippet of a subtitle file 20
3.3 Example of the implementation of a conversation 22
3.4 Example of a set of possible results returned by Lucene 24
3.5 Filipe, a chatbot based on SSS. 26
3.6 Example of a question in the questionnaire 28

4.1 Example of the implementation of a conversation 30

5.1 Examples of answers from the Portuguese corpus considered Suitable, taken from [1] . . 35
5.2 Examples of answers from the Portuguese corpus considered Non Suitable, taken from [1] 36

Glossary

LSH	Locally Sensitive Hashing
LUP	Language Understanding Platform
NLU	Natural Language Understanding
OOD	Out of Domain
SSS	Say Something Smart

Chapter 1

Introduction

In this chapter, we present our motivation for this work and our main goals.

1.1 Motivation

Dialogue systems can be divided in two categories: task-oriented and single-domain [8]. In task-oriented systems the dialogue is focused on getting people to achieve a certain task by completing subtasks [21, 34]. Single-domain systems are those able to answer questions about a certain domain in which they specialize. Two examples of single domain systems are Sgt. Blackwell [17] and Edgar [32], an agent developed by L2F[1], under the *FalaComigo* project. Edgar's domain is the Monserrate Palace, the place where he is deployed, and he answers questions posed by visitors of the palace.

People that interact with Edgar often pose questions that cannot be handled by the system. This may happen because, although Edgar's domain is the Monserrate Palace, people that come to interact with him do not know the exact application domain. Another reason is that sometimes people may only want to test the system to see how "smart" it is. These kinds of requests are called Out of Domain (OOD) [16]. In cases like these, Edgar is programmed to simply say something like "I don't know". Although it might be argued that, in light of their assistive nature, such systems should be focused in their domain-specific functions, the fact is that people become more engaged with these applications if OOD requests are addressed [31].

Say Something Smart (SSS) [1] was recently developed in order to try to solve the problem of OOD requests. SSS uses as its knowledge base Subtle, a corpus constituted of interactions. This is an approach already used in other existing systems [35]. Each interaction (adjacent pair) comprises two turns, T-A where A corresponds to an answer to T, the trigger[2]. Subtle is built using interactions extracted from movie subtitles.

SSS is set up to retrieve up to one hundred possible answers based on the user request. To be able to do this, SSS uses Lucene[3], an information retrieval software library, that is used to store all the

[1] `https://www.l2f.inesc-id.pt`

[2] We use the word *trigger*, instead of the usual designation of *question*, since not every turn includes an actual question. Throughout the text, we also use the designations input and request to refer to user turns.

[3] `http://lucene.apache.org/core/`

interactions in Subtle. From the set of possible answers, a single one has to be chosen. The approach followed in the original version of SSS is simply to select the answers in which the corresponding trigger is very similar to the user input, and then returning the most common answer among the interactions left. This is a simple approach to the problem of choosing a single answer, but it does not always give good results. One of the problems is that if none of the triggers meets the similarity threshold then no answer can be chosen, although there might have been some suitable answers among the interactions returned by Lucene. Another issue is that, in many cases, it would be better to check what the most common answer is using all the interactions returned by Lucene, because, many times, after the first filtering there are very few possible answers left.

In this work we investigate alternative ways of selecting an answer from a list of possible ones with the goal of improving the overall results of SSS: one is to combine several measures in a linear way and the other to use the learning to rank paradigm. With the same goal in mind, we also improve the Subtle corpus by adding new rules when pairing triggers with answers. Finally we adapt SSS to work better for Portuguese, seeing that it uses Lucene in the same way for English and Portuguese, which makes the latter language perform much worse. In the end, we conduct an evaluation, illustrating the potential behind the proposed approaches in addressing OOD interactions.

1.2 Goals

In order to address all the different needs previously described, this work's goals are the following:

- Study state of the art of techniques and systems that handle OOD interactions, and how they choose an answer.
- Study state of the art of corpora based on movie interactions that can be used by conversational agents.
- Improve the answers given by the SSS platform by:
  - Improving the way the corpora is built from the subtitles.
  - Developing new ways of choosing an answer from a set of possible ones. To do this, two strategies are envisaged: combine several measures in a linear way and use the learning to rank paradigm.
  - Adapting SSS to work for Portuguese.
- Develop a way of evaluating if a given sentence is appropriate.
- Test and evaluate the different strategies implemented.

1.3 Thesis organization

This thesis is organized as follows. Chapter 2 surveys some related work. The improvements to the SSS engine are described in Chapter 3, and Chapter 4 presents an alternate way of choosing an answer,

by using the learning to rank paradigm. Evaluations comparing the current SSS with the original are presented in Chapter 5. Chapter 6 concludes, pointing directions for future work.

Chapter 2

Related Work

In this section we describe the various works studied. First, we describe some natural language tools developed in INESC-ID, followed by corpora built using movie interactions. After this, we explain how some dialogue systems deal with OOD interactions.

2.1 Previous work done in INESC-ID

In this section we will explain three systems developed in INESC-ID, namely, SSS, the system we are trying to improve, Language Understanding Platform (LUP), and Just.Ask.

2.1.1 Language Understanding Platform

LUP is the Natural Language Understanding (NLU) component of Edgar [30]. LUP uses a classification process to understand literal semantic interpretation and supports three semantic representations:

- **Semantic categories** are assigned to each interaction. A classifier is trained with a corpus mapping utterances to categories and, when new input is given, returns an answer with the category that was assigned to the input;
- **Frames** are attributes meant to be filled with a value. In this case the goal of the NLU process is to determine the frame type and fill its attributes with the corresponding values;
- **Logical forms** are primitive formulas, that is, predicates with one or more arguments. In the example taken from [30], a possible way to represent the question *Who directed Casablanca?* could be `QT_WHO_DIRECTED(casablanca)`. The predicate association can be made using classification and the named entities can be detected using something as simple as a dictionary with named entities (with entries like MOVIE Casablanca).

For the classification process, LUP implemented three different classifiers:

- **Support Vector Machines** - This classifier is based on the concept of linearly separable data. The goal is to constructs a hyperplane to separate the instances according to some criteria (category).

This technique can be extended to classify more than two instances using a one-vs-all approach (using many classifiers). When the SVM receives an input instance, all individual classifiers perform a classification and return the distance of the instance to the corresponding hyperplane. The classifier that returns the biggest distance will attribute a category to the instance.

- **Cross Language Model** - This is the model developed by NPCEditor, and it is explained in Section 2.2.3;

- **String Similarity Measures** - This technique uses string similarity as a classifier. To do this, a distance algorithm chosen by the user goes through training examples and computes the string similarity between the input utterance and each example. Using the fact that the training examples are associated with a semantic category, this technique is able to return the semantic category for the corresponding utterance with the highest similarity score.

2.1.2 Say Something Smart

SSS [1] is a dialogue system developed by Ameixa et al. in the context of his MSc thesis. SSS deals with the problem of out of domain interactions by using movie subtitles as corpora, after some careful parsing of the subtitles.

The original idea of SSS was to use the gathered corpora and integrate it with LUP, but because of the huge number of interactions other methods had to be developed, as LUP cannot handle that amount of data. There were two methods developed in order to deal with this problem: Lucene[1], and Locally Sensitive Hashing (LSH) [13]. Because Lucene had much better results than LSH, we will only describe Lucene.

Lucene is an information retrieval software library, capable of full text indexing and fast searching, even with large amounts of data.

For Lucene to be able to perform a search it is necessary to transform the raw text into indexable Tokens. For this purpose, Lucene uses ”document analyzers” which contain the following tools:

- Tokenizer - Tokenizers are used to break a string down into a stream of terms or tokens. A simple tokenizer might split the string up into tokens wherever it encounters whitespaces or punctuation.

- Stemmer - The Stemmer reduces inflected or derived words to their word stem. For example, a Stemmer might reduce the words “starting” and “started” to their root form “start”.

- Stop-words filter - A filter that contains a list of stop-words, that is words to be ignored. Whenever the filter encounters a token contained in the stop-words list it removes that token.

The Stemmer and the list of stop-words that are fed to the filter are highly language dependent, as each language uses different inflectional and derivational endings as well as different stop-words. Lucene comes with some integrated analyzers, like the one used in SSS, StandardAnalyzer. This analyzer works well for the English language, but not so much for Portuguese, and because of this Lucene

[1] `http://lucene.apache.org/core/`

was only used with the English corpus. As was said before, one of the goals of our work is to make SSS work better for the Portuguese corpus. This goal can be achieved by using an analyzer that works with a Portuguese Stemmer, and a list of Portuguese Stop-words.

When Lucene receives a user query, the analyzer is used to transform the text into tokens. All that is left then is to search the index using these query tokens. In order to retrieve the best results, Lucene compares the user input to the triggers in the corpus, associating a relevance score to each one, and organizes its results in descending order (from the one it considers to be the most relevant to the least).

Although Lucene already orders the hits by relevance, this order may not be ideal to use as a similarity measure because it can give higher scores to triggers with repetitions of words instead of exact matches of the triggers. In order to solve this problem SSS uses the first 100 matches from Lucene and uses its own algorithm to try find the best answer.

First, these matches are filtered to keep only the ones that are most similar to the input, according to a given threshold of similarity (arbitrarily defined by the end user). To compute the similarity, SSS uses a combination of Jaccard (Equation 2.1) and Overlap (Equation 2.2).

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.1}$$

$$Overlap(A, B) = \frac{|A \cap B|}{min(|A|, |B|)} \tag{2.2}$$

In each case, the A and B parameters of the equations are sentences. $|A \cap B|$ represents the number of words equal in both sentences, $|A \cup B|$ represents the total number of distinct words from both sentences, and $min(|A|, |B|)$ the number of words of the smaller sentence. The difference between Jaccard and Overlap is that, in Overlap the common words between the two sentences are divided by the size of the smaller sentence instead of the number of distinct words from both sentences. Equation 2.3 exemplifies the JaccardOverlap, where w is a weight (between 0 and 1).

$$JaccardOverlap(A, B) = w \times Jaccard(A, B) + (1 - w) \times Overlap(A, B) \tag{2.3}$$

After this filtering, if none of the answers remains, the answer “I am sorry but I do not know how to answer your question” is given as a discarding answer. If there is more than one result, the next strategy is simply to choose the answer that was given most times among them, by using, again, JaccardOverlap to compute the similarity among the answers.

In Table 2.1, taken from Ameixa et al. [1], we can see the first 5 hits to the query “How are you?”. Two of the answers will be considered similar by the JaccardOverlap equation, H_4 and H_5, making them the most common answer, and so the final answer will be one of them.

One of the concerns during the evaluation was to determine if this new system could handle previous OOD interactions with Edgar, so the system was tested with 58 of these interactions with each one of the corpora (explained in 2.3.4): Horror, Western, Romance, Sci-fi and All. The different responses were classified as Discarding Answers, when the system was unable to provide an answer and Non

Query - How are you?		
Hit	**Document Fields**	
	Question	**Answer**
H_1	PHILIP: How How are you?	Fine.
H_2	How–how are you doing that?	It's a magic carpet.
H_3	How How can you	I'm good.
H_4	How are you?	I'm great.
H_5	How are you?	Great!

Table 2.1: Table with the first 5 hits to the query "How are you?" using Lucene, taken from [1]

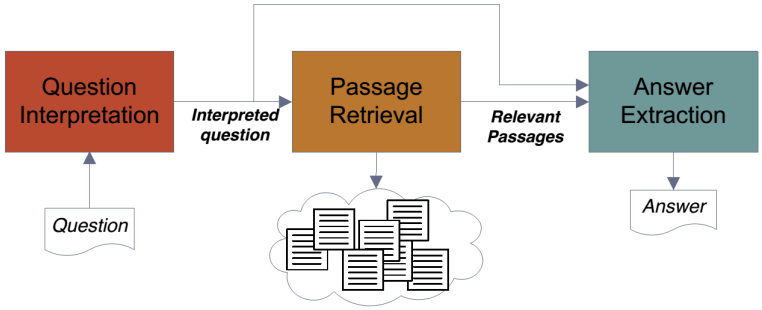

Figure 2.1: Typical architecture of a QA system, taken from [27].

Discarding Answers, when an answer was given. The Non Discarding Answers were also divided into two sub-groups: Suitable Answers, when the response was considered as properly addressing the request, and Non Suitable Answers when the response was not considered appropriate for the request.

The evaluation was made by manually categorizing each of the answers. The best results were obtained using the corpus with all the interactions, and it was able to answer 72% of the requests. From those, about 65% were appropriate for the given input.

2.1.3 Just.Ask

Just.Ask is an open-domain Question Answering (QA) system [27]. Just.Ask uses rule-based and machine learning-based components and also implements several state-of-the-art strategies in question answering.

A typical architecture for QA systems is presented in Figure 2.1, taken from [27], which is the architecture followed by Just.Ask.

- **Question Interpretation**: This component of Just.Ask receives a natural language question as input, then analyses and classifies that question to output an *interpreted question*. It does this by tokenizing the input, extracting the syntactic components and headword, and assigning a category.

  Just.Ask uses the definition of headword from [12]: "One single word specifying the object that the question seeks". The approach followed to determine the headword was to traverse the parse tree of the question, top-down using a set of rules. The parsing of the question was done using the Berkeley Parser [33], trained on the QuestionBank [14].

  The categories considered by the system are extracted from Li and Roth's two-layer question type taxonomy [20]. To assign a category to a question, Just.Ask allows the usage of different classifica-

tion techniques. The classification can be performed by using hand-built rules or machine-learning techniques (e.g., SVMs or Naïve Bayes).

- **Passage Retrieval**: This intermediate component receives the previously interpreted question as input, and by querying various information sources, returns a set of relevant passages for the posed question. Just.Ask uses passages (snippets/paragraphs that search engines associate with each result) instead of the full documents, because it was found that they usually led to a higher accuracy.

  To retrieve relevant passages from unstructured sources, Just.Ask uses Lucene, which already returns the snippets from the relevant results. The semi-structured sources of content, Wikipedia[2] and DBpedia [2], are used by the system to answer non-factoid-like questions, that usually require longer answers.

- **Answer Extraction**: This last component of Just.Ask receives both the previously obtained interpreted question from the Question Interpretation component, as well as the relevant passages returned by the Passage Retrieval component. First it finds a set of the candidate answers, and then it selects the final answer, from this set, to be returned.

  Just.Ask uses different techniques to find the candidate answers from the relevant passages, based on the different question categories, namely, Regular Expressions, Named Entities, WordNet-based recognizers and Gazetteers. The Regular Expressions are used to extract answers from questions related with numbers or abbreviations. In example gave by [27], if an answer needs to be found for something related with temperature, the regular expression $/[0-9]+(K|R|^{o}C|^{o}F)/$ might be used. For questions pertaining particular names (person names, organizations, locations, etc.), Just.Ask uses Stanford's Conditional Random Field-based named entity recognizer [9], a machine learning-based named entity recognizer, that is able to automatically learn a model to extract entities based on a set of annotated examples. Another kind of question that needs a different technique for candidate answer extraction is the *type of* question. These kinds of questions need answers that are composed of subclass of a certain class present in the question (like a type of animal). To do this, Just.Ask uses WordNet's hyponymy tree related to the question's headword to construct a dictionary. It then uses LingPipe's implementation of the Aho-Corasick [11] algorithm to match and extract candidate answers. Finally, for questions related with countries or cities, Just.Ask uses a Gazetteer to accurately extract candidate answers, thus guaranteeing that only candidate answers of the expected type are extracted.

  To return a final answer from the set of candidate answers, Just.Ask can perform four tasks: normalization, aggregation, clustering and filtering. It normalizes date and number values to use a single representation, aggregates lexically equivalent answers, clusters together similar answers and filters these clusters by discarding clusters that contain answers present in the original question. In the aggregation step, the answers aggregated will have a score corresponding to the sum

[2] `http://www.wikipedia.org/`

of the answers aggregated and in the clustering step each cluster has a score calculated by summing each answer. After this, the final answer chosen will be the representative answer of the cluster with the highest score (the most representative answer is the one with the highest score within the cluster).

2.2 Dealing with OOD interactions

In this section we describe how some domain-oriented systems deal with OOD interactions. We start with Virtual Agent Max, followed by Hans Christian Andersen, Sergeant Blackwell and IRIS. We chose these systems because all of them deal with OOD interactions in different ways.

2.2.1 Virtual Agent Max

Virtual Agent Max is an Embodied Conversational Agent (ECA) created by the Artificial Intelligence group at Bielefeld University and it was developed to study how natural conversations of humans can be modeled and made available for Artificial Intelligence systems [38].

To deal with OOD interactions Max uses *Wikipedia* as the knowledge base to find a correct answer [37]. When receiving an answer, Max transforms the user's input into an appropriate query. Then, using Lucene to index the document collection (extracted from *Wikipedia*), it will retrieve the most similar results to the ones in the query.

After this, Max has to choose the most appropriate answer from this set of results and it does this by re-ranking each sentence using three different evidence scores:

1. Normalize the previously computed Lucene retrieval similarity.
2. Use the Jaccard similarity index (Equation 2.1) between the input and each answer candidate.
3. Use heuristics to favor shorter sentences as answers.

After this process, the best scoring answer is chosen.

Max uses Lucene retrieval similarity score to help him choose the most appropriate answer, unlike SSS. At the moment, because SSS does not normalize the sentences when comparing them using its own score, sometimes Lucene's score achieves better results. This is something we address by making the proper normalizations before comparing the questions in SSS.

2.2.2 Hans Christian Andersen

Hans Christian Andersen (HCA) is an ECA based on the famous Danish author and writer [26]. HCA is used in an interactive computer game where a player can interact with him using spoken conversation as well as 2D pen gestures. The goal of HCA is to help players understand and learn about the writer's fairy tales, life and historical period while having fun at the same time.

HCA is able not only to talk about his own domain of expertise but also about everyday topics like movies, games, famous personalities and others. It achieves this by using the web and available QA systems to get appropriate responses.

When the system classifies a user's request as OOD, then it proceeds to retrieve the output from the web. HCA then uses three freely available open-domain QA systems (AnswerBus, Start, and AskJeeves) to find a quick and concise response. These responses then undergo a process of normalization to remove certain stop words and control/graphical characters so that a single one can picked.

This way of dealing with OOD interactions is interesting and it is something that could improve SSS. By using QA systems, HCA is able to answer factoid-like questions which SSS, at the moment, is not prepared answer. This could be a very nice addition to SSS, although, as of now, we postpone this to future work.

2.2.3 Sergeant Blackwell

Sergeant Blackwell (SB) is also an ECA, and was developed at the Institute for Creative Technologies[17]. At its core, SB uses NPCEditor, which is a system used for building and deploying virtual humans with the ability to engage a user in spoken dialogue on a limited domain [18, 19]. NPCEditor supplies a user-friendly editor used for creating virtual humans rapidly. It has been used to create many different virtual humans in several applications.

The NPCEditor system compares the text of the request directly to the texts of the answers and it uses a new text classification approach, called Cross-Language Module (CLM), based on statistical language modeling that, according to the evaluation that was performed, significantly outperforms vector-based approaches.

In the traditional text classification approach, when the system receives a new request, it is compared to the known triggers and the answer corresponding to the best matching group of triggers is returned. The disadvantage of this approach is that it completely ignores the content of an answer. Even if there are appropriate answers in the set of known answers, if the triggers leading to them differ significantly from the user's request, they will not be returned. In order to avoid this problem, NPCEditor compares the user's request to the known answers instead of comparing it to the known triggers.

When it is time to return the most appropriate answer, NPCEditor uses several heuristics to evaluate each answer in the database and returns the answer with the highest value obtained.

SB has a set of 13 responses to deal with OOD interactions. When he classifies a request as OOD he uses one of these responses as a reply. This tactic proved to be better than one that always uses the same answer for OOD interactions [31]. In case the user keeps asking questions that SB is not able to handle, the system tries to direct the conversation to a topic within his domain.

Unlike the original version of SSS, NPCEditor considers the content of the answer important for choosing the best answer. This is something we also do in this work because, many times, the answer contains some of the words of the request.

2.2.4 IRIS

IRIS is a chat-oriented dialogue system made in the Institute for Infocomm Research [4]. It is based on the vector space model framework, and it uses Movie-DiC [3] as its corpus.

IRIS starts by greeting the user, introducing itself, and asking the user's name to use it later. The user's name constitutes first vocabulary term that IRIS learns, and it is stored in the vocabulary learning repository.

Next, the current history vector needs to be initialized. This can be done in one of two ways:

1. If IRIS already talked to the user before, it will load the last stored dialogue history for that user;
2. Otherwise, IRIS will randomly select one dialogue history vector from the dialogue data-base.

IRIS then prompts the user for what the user desires to talk about and, after each input from the user, it performs a set of computations that can lead to a large amount of answer candidates:

- It starts by searching for possible matches between the input and the terms inside the vocabulary learning repository. Every time a term is matched, it is replaced by its corresponding definition stored in the vocabulary learning database;
- Next, IRIS performs a tokenization and vectorization of the user input;
- After the tokenization, the unknown vocabulary terms are identified, and the system can either ask for the meaning of the term or get their definition from an external source of information. Either way, when the definition is obtained and validated, it is stored along with the unknown term in the vocabulary learning repository;
- In case IRIS decides it does not need to ask for the meaning of any term in the request, it is time to compute the vector similarity modules and similarity scores in order to retrieve the best matches from the database. IRIS computes two different similarity scores:
  - Cosine similarity is computed between the input and all the utterances in the database. It is able to retrieve a high number of candidates utterances, usually ranging from 50 to 100, from the dialogue database.
  - To be able to have some context, IRIS also uses the cosine similarity between the current history vector and all the history vectors stored in the dialogue database, and it gives more importance to the most recent interactions. It then uses a log-linear combination scheme to unite the two scores.

After this process, IRIS has retrieved a set of possible answers, based not only on the user's input but also on the history of the entire conversation. All it needs now is to choose an answer from set, and it does this by randomly choosing one of the top ranked utterances.

The final action is taken when the user responds to the answer given by the system. By starting his response with one of the following three symbols the user can give some feedback to the system:

Table 2.2: Example of noise in the corpus

#	Speaker	Turn
1	IRIS	You watching the ballgame?
2	USER	No
3	IRIS	Bianca stares at him for a moment.

- Ban (*): The previous response will be marked as forbidden and it will never show again;
- Reinforce (+): IRIS will increase the probability of giving the same response given a similar user input will;
- Discourage (–): IRIS will decrease the probability of giving the same response given a similar user input will.

This feedback by the user is a nice feature, and, although it is out of the scope of this thesis, it could be useful in SSS, especially in cases where the system is used by the same person for a long time.

2.3 Corpora

In this section we describe three corpus based on movie scripts: Movie-DiC and Cornell Movie-Dialogs Corpus, both composed of a collection of dialogue samples, and the Personage Corpus, which contains lines and information about movie characters. All these corpora, except Movie-DiC, are freely available to download. We end the section by describing Subtle, the movie subtitle corpus used in SSS.

2.3.1 Movie-DiC

Movie-DiC [3] was developed in the Institute for Infocomm Research by Rafael Banchs[3] and is the corpus used in the IRIS system [4]. It stands for Movie Dialogue Corpus and is a corpus built using movie scripts from The Internet Movie Script Database (IMSDb)[4]. Movie-DiC used various techniques to identify dialogs between characters, and for each utterance, also identifies the speaker, the context, and the content of the utterance. Because of differences in the format of movie scripts, not all of them could be parsed and 17% of the scripts had to be discarded, so from the 911 available scripts, only 753 were successfully parsed. The parsed information was then used to create XML documents comprising 132,229 conversations that make a total of 764,146 interactions. The genre of the movie from which the script originated is also annotated, having 16 possible genres. An example XML can be found in Figure 2.2 (taken from [3]).

Although the paper does not mention any kind of evaluation, the system has some noise in the corpus as exemplified in Table 2.2 taken from [4].

This is an interesting corpus, not only because it has information about the speaker, but also because it saves the context of the conversation, when there is one. This kind of information could be interesting for this thesis work, but because Movie-DiC is not freely available we cannot use it.

[3] http://www.i2r.a-star.edu.sg/
[4] http://www.imsdb.com/

```
<dialogue id="47" n_utterances="4">
   <speaker>VALIANT</speaker>
      <context></context>
      <utterance>You shot Roger.</utterance>
   <speaker>JESSICA RABBIT</speaker>
      <context>Jessica moves the box aside and
         tugs on the rabbit ears. The rabbit head pops
         off. Underneath is a Weasle. In his hand is the
         Colt .45 Buntline.</context>
      <utterance>That's not Roger. It's one of
         Doom's men. He killed R.K. Maroon.</utterance>
</dialogue>
```

Figure 2.2: XML example from [3].

2.3.2 Cornell Movie-Dialogs Corpus

The Cornell Movie-Dialogs Corpus [7] was developed by the Department of Computer Science in Cornell University. This dialogue corpus was also built using movie scripts, but various sites were used to get them. After getting the scripts, they were used to search for the movie in The Internet Movie Database (IMDB).

All movies that could not be matched against IMBD or that had less than 5 votes (by the users of IMDB to rate a movie) were discarded. For the remaining 617 titles the following metadata was extracted (using the information in IMDB database):

- Genre
- Release year
- IMDB rating
- Number of IMDB votes
- Cast distribution

After this, the conversations were extracted from the scripts, and were kept only if they had more than 4 interactions. 10,292 conversations were left, making a total of 220,579 interactions. The names of the characters in the script were also matched with the characters in IMDB to try to extract both the gender (using the gender of the actor playing the character to infer the gender of the character) and the end credit position (after how much people they appear in the credits). After this process and some manual annotation, 3,774 character gender were annotated as well as 3,321 character billing positions, out of 9,035 characters.

No evaluation was mentioned in [7].

Although this is the smallest corpus in this section, it has a very interesting feature: saving the entire conversation between two characters. We also implemented this in Subtle, although, because Subtle is built using subtitles, there is no information for us to know if a new conversation has started, so it is not as reliable as the aforementioned corpus.

2.3.3 Personage Corpora

The Personage Corpora [36] is the corpora used by Personage [23, 24] and was developed by researchers in the University of California Santa Cruz in the Computer Science Department. It consists of 862 film scripts from IMSDb, like the corpora of Movie-DiC, taken on May 19, 2010. This corpus has information corresponding to 7,400 characters, with a total of 664,000 lines of dialogue and 9,599,000 tokens.

The characters were grouped into different categories according to genre, director, year, and character gender. This was done using the IMDB ontology. Because most movies belong to multiple genres, the characters can be grouped in more than one category.

As females and males might have different linguistics styles, it may be useful to identify the gender. For this purpose, the Names Corpus, Version 1.3[5], was used to identify the character's gender from the names in the lists (check if the name is in the list of male or female names) and hand-annotated the ones not automatically labeled.

The scripts were parsed in order to extract dialogues for each character in each movie, and these dialogues were saved in files according to the following norm: MOVIE_NAME-CHARACTER.

The next step was to annotate the corpus. The Stanford POS Tagger[6] was used for the POS tagging and the following information was also annotated:

- Basic - Simple features like number of sentences, sentences per turn, number of verbs, number of verbs per sentence
- Polarity - Determines whether a sentence has a positive or negative connotation. This is determined using SentiWordNet 3.0[7]. SentiWordNet assigns to each synset of WordNet three sentiment scores: positivity, negativity, and objectivity. Because there may be different interpretations depending on the context for the same word, SentiWordNet may have more different sentiment scores for the same word. For these cases a simple equation is applied. For example, if there are three different scores for one word, v1, v2, and v3, with v1 being the most common and v2 the second most common, the Formula 2.4 in applied.

$$\frac{v1 \times 1 + v2 \times 1/2 + v3 \times 1/3}{1 + 1/2 + 1/3} \tag{2.4}$$

  The final polarity is assigned according to Table 2.3
- Dialogue Act - To identify the dialogue act type a dialogue act tagger was trained using the NPS Chat Corpus 1.0[8].
- First Dialogue Act - Identifies the Dialogue Act of the first sentence of each turn.

[5] http://nltk.googlecode.com/svn/trunk/nltk-old/data/names.readme
[6] http://nlp.stanford.edu/downloads/tagger.shtml
[7] http://sentiwordnet.isti.cnr.it/
[8] http://faculty.nps.edu/cmartell/NPSChat.htm

Table 2.3: Polarity score with SentiWordNet

Polarity assigned	Range of score (s)
Strong Positive	$s \geq 2/3$
Positive	$1/3 < s < 2/3$
Weak Positive	$0 < s < 1/3$
Neutral	$s = 0$
Weak Negative	$-1/3 < s < 0$
Negative	$-2/3 < s < -1/3$
Strong Negative	$s \leq -2/3$

- Merge Ratio - Finds merging of sentences (merge of subject and verb of two propositions), by using a grammar that looks for verb+noun+conjunction+noun.
- Passive Sentence Ratio - Recognizes passive sentences using scripts from the narumo repository[9], under source/browse/trunk/passive. These scripts implement the rule that if a to-be verb is followed by a non-gerund, the sentence is probably in passive voice.
- Concession Polarity - If a sentence has a concession, it finds its polarity using the previously mentioned Polarity feature set.
- LIWC Word Categories - LIWC calculates the degree to which people (or, in this case, characters) use different categories of words across a wide array of texts.
- Pragmatic Markers - Features to count categories of pragmatic markers and individual word count/ratio.
- Tag Question Ratio - The amount of tag questions detected through the use of regular expressions to parse sentences.
- Average Content Word Length - Finds content words, using WordNet's tag (noun, adjective, adverb, and verb), then average the length of those words.
- Verb Strength - Average sentiment scores of all verbs.

This is a richly annotated corpora that gathers a large amount of information about the way the characters speak. Some of these annotations could be very useful in Subtle, but, at this moment, this is future work.

2.3.4 Subtle

Subtle is the corpus for Say Something Smart, and was collected by Ameixa et al. for his MSc [1]. It was built using subtitles downloaded from the Open Subtitles website[10]. Ameixa et al. collected about 6200 subtitles files (files with subtitles for the whole movie) in English and 4100 in Portuguese from four different genres: Horror, Romance, Western and Sci-fi. Using various techniques to identify dialogue between two characters, several interactions were extracted from these files. Ameixa et al. ended up

[9] http://code.google.com/p/narorumo
[10] http://www.opensubtitles.org

building a corpus of about 3.4 million T-A pairs for the English language and 2 million T-A pairs for the Portuguese language, divided in the four previously genres.

The corpus was evaluated to see if the T-A pairs were being correctly created by manually evaluating a small subset of the corpus. The results of this evaluation were a recall of 97% and a precision of 71%. The high recall value means that 97% of the T-A pairs that were supposed to be created, are in fact being created, and 3% of the T-A pairs are being wrongly discarded. The precision value of 71% means that 71% of the T-A pairs are being correctly paired (and the other 29% are not).

The use of movie subtitles brings two main advantages over scripts and other similar resources. First, the web offers a vast number of repositories with a comprehensive archive of subtitle files. The existence of such collection of subtitle files allows data redundancy, which can be of great help when selecting the adequate reply to a given OOD request. Secondly, subtitles are often available in multiple languages, potentially enabling multilingual interactions.

To build the interactions correctly, Ameixa et al. considered that if a sentence in a subtitle file appears within less of one second after a previous sentence, then these two sentences must form a T-A pair. This value of one second, however, seems arbitrary and rigid, as different movies may have different paces. As will soon become apparent, one of our contributions is the improvement of the Subtle corpus namely by removing the time threshold and, instead, considering the time difference between trigger and answer only when selecting the answer to a user query.

Chapter 3

Improving the answers given by SSS

In this chapter we will explain the different optimizations we did with the goal of improving the answers of SSS. Because SSS depends on a corpus to function properly, we start this chapter by explaining the improvements made to Subtle, the corpus used by SSS. We then move on to explain the various changes made to the SSS engine and we end with an evaluation of these improvements.

3.1 Building the Subtle corpus: Improvements

Before we delve into some of the deeper changes made in the process of building the Subtle corpus we want to refer three small changes:

- Added configuration file to Subtle - Subtle now has a configuration file with the most useful options to build the corpus without having to change and recompile the code, namely:
  - The location of the subtitle files.
  - The maximum time between two subtitles to remove the need of excluding interactions right from the beginning.
  - The option to tag the words in Subtle with a Named Entity Recognizer (NER) and also the location of the NER. This option was previously hard-coded in the Subtle-building program.
- Allowed use of compressed subtitle files to build Subtle - The corpus builder can now read compressed files to build Subtle, namely files in the gzip format[1].
- Solved bugs - Some small bugs that caused errors in the creation of the corpus were found and fixed.

We believe that having a fixed value of time for considering if two consecutive utterances constitute an T-A pair may lead to many useful interactions being discarded. To solve this issue, we introduced a new option: if the user chooses a value of 0 for the time difference in the config file all interactions will be kept. Keeping all the interactions means that we will have all the information present in the subtitles,

[1] http://www.gzip.org/

```
SubId - 100000
DialogId - 1
Diff - 3715
T - What a son!
A - How about my mother?

SubId - 100000
DialogId - 2
Diff - 80
T - How about my mother?
A - Tell me, did my mother fight you?

SubId - 100000
DialogId - 3
Diff - 1678
T - Tell me, did my mother fight you?
A - Did she fight me?
```

Figure 3.1: Example of the new Subtle

```
770
01:01:05,537 --> 01:01:08,905
And makes an offer so ridiculous,

771
01:01:09,082 --> 01:01:11,881
the farmer is forced to say yes.

772
01:01:12,752 --> 01:01:15,494
We gonna offer to buy Candyland?
```

Figure 3.2: Snippet of a subtitle file

but also more incorrect T-A pairs. To treat this issue we added the time difference between every T-A pair to Subtle, so that later we may have an associated probability of a T-A pair actually being correctly built, and use this information to choose the best possible answer. An example of the new Subtle can be seen in Figure 3.1. In the example, SubId is a unique number and refers to a single subtitle file. DialogId is a value used to save the context of the conversation and is explained in detail in Section 3.2. Diff is the aforementioned difference of time between two consecutive utterances in milliseconds, and finally, T is the trigger and A is the answer. Continuing in this example, we can see that in the second T-A pair, the trigger and the answer are very likely to be spoken by the same character. We can also see that the difference of time between the utterances is very small, and so we can later use this knowledge to give a smaller probability to T-A pairs with very low time difference since it is likely they are from the same character. The consideration of the time between utterances as an indicator of whether they are a T-A pair is an improvement over the previous work of Ameixa et al. [1].

We have also found that in some movie subtitles, when there are two consecutive utterances by the same character, the first utterance ends with an hyphen, a comma, a colon or ellipsis and the second starts in lowercase. We added a rule to address this situation, so that, when this happens, the two

utterances are concatenated. To exemplify how this works, consider Figure 3.2, containing a snippet of a subtitle file. Building our corpus from this file would yield the following T-A pair:

```
T - And makes an offer so ridiculous, the farmer is forced to say yes.
A - We gonna offer to buy Candyland?
```

What happened is that the first two utterances were concatenated because the first one ended with a comma and the second started with lowercase.

Ameixa et al. [1] already addressed part of this issue of when to properly concatenate two subtitles for the case where the first utterance ends with ellipsis, but at the start of the second utterance he only considered the lowercase characters representable in ASCII, which is not really a problem when building the English corpus, but can cause errors when building the Portuguese one. We modified this rule to consider the other possible terminations as well as all lowercase characters.

Another issue that we found, has to do with the actual content of the subtitles. Many times there are elements of the subtitle which, for our purposes, are unnecessary. We deal with three of these elements:

- Character identification - In some subtitles the name of the character that is speaking is included at the beginning of the utterance (e.g. `Johnny:Oh hi, Mark.`), specially when a character is not appearing on the screen (which is useful for hearing impaired watchers). We remove these names from the subtitles because we do not want our system to give an answer containing them.

- Sound descriptions - When making subtitles for people with hearing impairment it is very common to describe most of the sounds that are being played (e.g. `[TIRES SCREECHING]`). This results in subtitles which are not actual answers, like we want, so we remove them. We did, however, keep subtitles that contain text outside of the brackets, because these often describe what the person is doing/feeling while saying something (e.g. `[Scoffs] No`).

- Font-changing tags - Subtitles sometimes include tags that video players can interpret to change the normal font of the subtitle to the one specified in the tag (e.g. `<font color‘‘#ffff00" size=14>Sync by honeybunny </font>`). In all the cases where we found subtitles with these tags we found that they did not contain any dialogue, and almost always contained the name of the person that synced the subtitles with the movie. For these reasons we erase these subtitles.

3.2 Context of the conversation

Following the lead of [4], we want to have available the whole conversation between different characters. To do this, we add a new field to Subtle: DialogId. This identification number starts at 1 and gets incremented every time a new T-A pair is considered to be the continuation of the conversation. This happens every time two consecutive utterances occur in a time difference smaller than the maximum time difference defined in the configuration file (if the value is set to 0 than the whole movie will be considered one big conversation). In case the time difference is larger than the value allowed, the DialogId will be reset to 1.

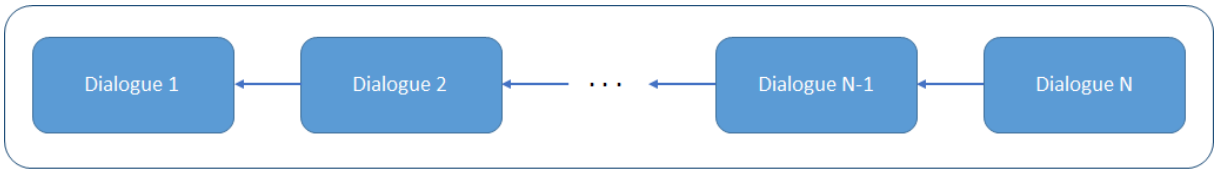

Figure 3.3: Example of the implementation of a conversation

Later, when the corpus is being parsed in SSS, every conversation will be stored. To be able to do this we changed the way SSS works. We want to have each dialogue know what dialogue came before it, much like a reversed singly linked list, where each dialogue knows about the dialogue that preceded it, in the conversation where they appear. Figure 3.3 shows an example of our implementation of a conversation. With this implementation, starting with any dialogue, we are able to go back to the start of the conversation. To implement conversations this way, we changed the data stored with Lucene. Instead of having a Lucene document with the trigger and the answer, we now have a trigger and a unique identifier of a dialogue. This unique identifier is used to retrieve the corresponding dialogue from a database. This way, we make use of Lucene's fast queries and are able to store much more information than before.

3.3 Adapting Lucene to Portuguese

Lucene works by using analyzers, that implement Tokenizers, Stemmers, and Stop-word filters. The key to improve Lucene's answers in Portuguese is to test different analyzers and see which one fares better. In our case, we tested three analyzers:

- Standard Analyzer - The default analyzer for Lucene, built to deal with Latin languages.
- Portuguese Analyzer - The analyzer specifically built to deal with the Portuguese language.
- Snowball Analyzer - This analyzer is more flexible than the other two because it accepts two arguments: the language and a stop-words list. Because of this we tested it using the Portuguese language and three different stop-words lists: an empty one (Setting SNS[2]), one with only the most common stop-words (Setting SMS[3]) and one with all the stop-words (Setting SFS[4]). The stop-words were obtained from `svn.tartarus.org/snowball/trunk/website/algorithms/portuguese/stop.txt`, the official trunk of snowball.

We used a test set comprising 44 questions, extracted from logs created by Ameixa et al. [1], and separated each answer into one of three categories:

- No answer - The system was not able to give an answer.
- Inadequate answer - The system gave an answer that did not make sense.

[2]Snowball Analyzer with no stop-words
[3]Snowball Analyzer with set of the most common stop-words
[4]Snowball Analyzer with full set of stop-words

- Adequate answer - The system gave an adequate answer to the question.

We categorized every answer in the test set. The results can be seen in Table 3.1.

	Standard Analyzer	Portuguese Analyzer	SNS	SMS	SFS
No answer	10	18	11	15	20
Inadequate answer	18	14	15	19	17
Adequate answer	16	12	18	10	7
Precision	36,4%	27,3%	40,9%	22,7%	15,9%

Table 3.1: Table with the results for each different analyzer

The precision was calculated by dividing the number of adequate answers by the total number. The Snowball Analyzer with no stop-words setting achieved the best results, followed closely by the Standard Analyzer. We can see that the more stop-words we used, the less precision we got. This might explain why the Portuguese Analyzer did not achieve the best results, seeing that it uses a list of stop-words. With these results we altered SSS to use the SNS whenever the user chooses to use the Portuguese language.

3.4 Normalization techniques

The original version of SSS used three kinds of normalizations: fixed-case (always use lowercase), removal of punctuation, and removal of diacritical marks. These are very common techniques in natural language processing tasks, but they do not solve all the problems. One of the problems, very frequent in the English language has to do with contractions. For example, the sentence “You are funny!”, after the normalization would become “you are funny”. Nevertheless, the sentence “You’re funny!” would be normalized to “you re funny”. This creates many mismatches between words that should have matched. We tackle this problem by using an English lemmatizer from Stanford CoreNLP [25].

For an easy setup of the normalization techniques in SSS we also added an option to the configuration file to allow any combination of the available normalizations in whatever order the user prefers.

3.5 Scoring the T-A pairs

As we explained before, SSS chooses an answer from the set of T-A pairs that Lucene returns by, first, comparing the input of the user with each trigger and filter those bellow a given threshold; second choosing the most common one in this filtered set. Although this approach often leads to good answers, there are some cases in which it may not work. One of the cases is when the input of the user is not similar to any of the triggers returned by Lucene. This will result in a discarding answer. This way of choosing an answer can also lead to bad answers in cases where few interactions are kept after the first filtering. Consider the example in Figure 3.4 of a set of results returned by Lucene. Choosing a threshold of 0.6 for the Jaccard similarity between the user input and each of these triggers (which means that phrases with similarity bellow this value will be discarded) would result in filtering out the

```
1: T - Hello               A - Hi
2: T - Hello there         A - Hi
3: T - Hello my friend     A - Hello
4: T - Hello               A - No
```

Figure 3.4: Example of a set of possible results returned by Lucene

second and third T-A pair. Then, only the first and fourth pairs would be used for the computation of the most common answer, and, in this case, there would be no common answer and the choice would be random. If, on the other hand, we were to consider all the pairs, then the answer returned would be "Hi".

For these reasons, our improved version of SSS does not include a hard threshold, because every interaction may be useful. On the other hand, it is still very useful to consider the similarity between the user's input and each of the interactions returned by Lucene. Therefore, we decided to score each T-A pair according to some defined measures, the idea being that interactions with higher score have a better chance of containing a correct answer. With this scoring mechanism, there are no discarded pairs and we can guarantee that as long as Lucene is able to return at least one pair, we can always have an answer. To implement this idea, we have different measures, each with a value between 0.0 and 1.0 and an associated weight with values also between 0.0 and 1.0 and the total sum of the weights being 1.0. Each T-A pair gets a score between 0.0 and 1.0 that translates how adequate a T-A pair is for the current user input, weighting appropriately the four selected criteria. The expression used to compute the score is presented in Equation 3.1, where the w's are weights and the M's are the measures. Using this approach, it is very easy to modify the importance of each measure and also to include new measures.

$$TAScore = \sum_{i=1}^{n} w_i \times M_i : w_i \in [0.0; 1.0], M_i \in [0.0; 1.0], \sum_{j=1}^{n} w_j = 1.0 \tag{3.1}$$

Currently we have implemented four different measures to score the T-A pairs that we describe in the continuation. In our description, "similarity" is measured in terms of Jaccard distance.

Trigger similarity with input The first measure, M_1, accounts for the similarity between the user input and the trigger of the interaction. For instance, given the input "*What's your mother's name?*", and the interactions:

```
T9: How nice. What's your
    mother's name?
A9: Vickie.

T10: What was your
     mother's name?
A10: The mother's name
     isn't important.
```

M_1 will assign a larger value to the second interaction, since "*What's your mother's name?*" is more similar to T10 than to T9.

The measure M_1 is particularly important since, as previously discussed, many of the interactions returned by Lucene have triggers that have little in common with the given input. For example, and considering once again the previous input ("*What's your mother's name?*") some of the triggers retrieved by Lucene were:

```
T11: What's your name?

T12: What's the name your mother
   and father gave you?

T13: Your mother? how dare
   you to call my mother's name?.
```

Response frequency The second measure, M_2, targets the response frequency, giving a *higher score to the most frequent answer*. That is to say, we take into consideration the corpus redundancy. We do not force an exact match and we calculate the similarity between each pair of possible answers. Consider, for example, the request "*Where do you live?*" and the interactions:

```
T14: Where do you live?
A14: Right here.

T15: Where are you living?
A15: Right here.

T16: Where do you live?
A16: New York City.

T17: Where do you live?
A17: New Road.
```

M_2 will give a higher value to the answer *Right here*, as it is more frequent than the others. It is important to note, however, that A16 and A17 would get a value higher than zero because they contain words present in other answers (in this case "*New*").

Answer similarity with input We also take into consideration the answer similarity to the user input. Thus, M_3 computes the similarity between the user input and each of the candidate answers (after stop words removal to filter out the less relevant words). If scores are higher than a threshold it is considered that the answer shares too much words with the user input, and a lower score is given to the answer; otherwise, the attained similarity result is used in the score calculus.

Time difference between trigger and answer Finally, we can use in this process the time difference between the trigger and the answer (measure M_4) to compute the likeliness of them actually being part of a dialogue between two characters. A time difference of zero occurs when there is a dialogue in a single subtitle and, in that case, we give it a value of 1.0. As we explained before, in Section 3.1, a small time difference usually means that the same person is talking, so when that happens we give a low score. For time differences that are not too small, we give a high score that decreases with the increase of the time difference, because of the increasingly unlikeliness of the interaction constituting a dialogue.

3.6 Evaluations

3.6.1 Evaluation setup

Filipe (Figure 3.5), a chatbot built with the Ameixa et al. [1] version of SSS, is on-line since November 2013.[5]

Figure 3.5: Filipe, a chatbot based on SSS.

Using it, we have collected 103 requests made to the agent from several anonymous users. From this set, we removed the duplicates and randomly chose 20. These turns, from now on referred as the test set, were used in the experiments described in the following.

3.6.2 Are subtitles adequate?

We started our evaluation with a preliminary inspection of Subtle, in order to understand if adequate answers could be found. Three human annotators evaluated the first 25 answers returned by Lucene to each of the 20 requests from the test set. Their job was to decide, for each request, if *at least one appropriate answer* could be found in these 25 candidate answers.

The first annotator considered that 19 of the user requests could be successfully answered and that one could not (*What country do you live?*). The second annotator agreed with the first annotator in all the cases, except that he considered another unsuccessfully answered request: *Are you a loser?*. The third annotator disagreed with both of them in the *What country do you live?* request, as he considered *It depends.* to be a plausible answer; he also considered that there was no plausible answer to *Where is the capital of japan?* (the other two annotators agreed that *58% don't know.* was a plausible answer to that request). Finally, the first and third annotator agreed that *So what? You want to hit me?*, *Your*

[5] It can be tested in `http://www.l2f.inesc-id.pt/~pfialho/sss/`

thoughtless words have made an incredible mess! or *Shut up.* would be appropriate answers to *Are you a loser?*.

Despite the lack of consensus in these scenarios, the fact is that the three annotators agreed that 17 out of 20 turns had a plausible answer in the set of answers retrieved by Lucene from the Subtle corpus, which is an encouraging result. The next step is then to study the best way to chose a plausible answer from the set of candidate answers retrieved by Lucene. Our framework, presented in Section 3.5, is evaluated in the next section.

3.6.3 Evaluating the different measures for scoring

To find out if the aforementioned measures can really help SSS give good answers we decided to evaluate their impact in the answer selection process.

Experimental setup

We tested five different settings to score each interaction pair:

- S_1 – Only takes into account M_1.
- S_2 – Only takes into account M_2.
- S_3 – Takes into account M_1 and M_2.
- S_4 – Takes into account M_1, M_2 and M_3.
- S_5 – Takes into account all four measures.

For the settings S_{1-4} all measures considered were given the same weight. For S_5, however, the weights were optimized experimentally, yielding:

- 40% weight for M_1.
- 30% weight for M_2.
- 20% weight for M_3.
- 10% weight for M_4.

The aforementioned test set was again used and SSS was tested in each of the five different settings. The best scored answer of each log was returned.

In order to evaluate how plausible the returned answers were, a questionnaire was built (Annex 7). It contained the 20 user request from the test set and the answers given considering each of the settings. We told the evaluators that those were the requests posed by humans to a virtual agent and the possible answers. They should decide, for each answer, if it made sense or not. Figure 3.6 shows an extract of the questionnaire.

In this example we can see that there are only four answers instead of the five expected. This is because, sometimes, even with different settings, the agent gives the same answer. We decided that the questionnaire should only contain unique answers.

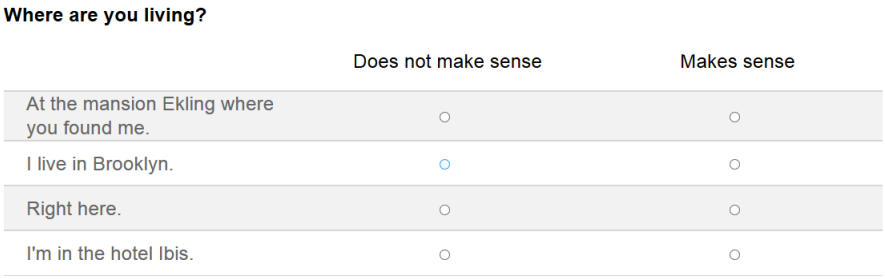

Figure 3.6: Example of a question in the questionnaire

Results

21 persons filled our questionnaire. Using their responses we were able to get, for each answer, the percentage of people that considered that answer to make sense (from now on answer-sense). For each of the five settings, we calculated the average percentage of answer-sense. The results are presented in Table 3.2.

	S_1	S_2	S_3	S_4	S_5
Average answer-sense	39.29%	45.24%	46.90%	61.67%	51.19%

Table 3.2: Table with the results for each different setting

We can see that the S_2 setting achieved better results then S_1, and that S_3 (the combination of measures M_1 and M_2) achieved slightly better results than the previous two. This suggests that the combination of the two strategies may yield better results than any of them alone. Moreover S_4 (which added the third measure M_3) achieved the best results, with a difference of almost 15% compared to the strategy of S_3. We expected that as we added measures, the average answer-sense would rise. The last setting (which added the M_4 measure), however, achieved worse results than S_3. This may be because the weights for each measure were not optimal or simply because M_4 is not a very good measure for choosing the best answer.

Another possible explanation why S4 may not be the best setting is that it is very hard to find the optimal set of values for each measure, which may very well be the reason the results did not match our expectations. To overcome this problem we will try a different approach, explained in the next chapter.

Chapter 4

Learning to rank possible answers

In this chapter we describe a new approach to score the answers retrieved by Lucene: the learning to rank paradigm. We start by explaining what this paradigm does and why it fits our goals. Next, we explain how we created a training corpus, and describe the framework used to run four different learning to rank algorithms. Finally, we end this chapter with an evaluation of the tested algorithms.

4.1 The learning to rank paradigm

The purpose of learning to rank is to construct a ranking model, that is, a model that takes an unordered list of items and sorts it in descending order of relevance. This way, we can use learning to rank to order the list of possible answer retrieved by Lucene and simply return the one with the highest score.

We use this paradigm to solve the problem mentioned in the end of the previous section (Section 3.6.3). It is very hard to find the optimal weight for each of the measures we defined, as there is a huge number of possible combinations of those values. Additionally, it is debatable whether a constant set of weights is adequate for a whole interaction session.

4.2 Creation of the training corpus

To build the corpus for training the ranking model, we gathered a set of eighty triggers, also collected from Filipe's logs (Section 3.6.1), and the first twenty answers retrieved by Lucene to each of these triggers, comprising a total of 1561 answers (should be 1600 but some of the answers Lucene returned were duplicated). The next step was to annotate each of the answers, according to how much sense the answer made, using three categories: yes, no, maybe. To make this annotating easier, we created a small command-line program that uses two files: one with all the non-annotated answers and one with all the annotated answers. The second file will start empty but it will grow as the annotator evaluates the answers. An example of an annotation session can be seen in Figure 4.1.

After one person evaluated all the answers we wanted to know if different annotators evaluated the answers in a similar way. If they did not, we might need multiple annotators per answer. We took a

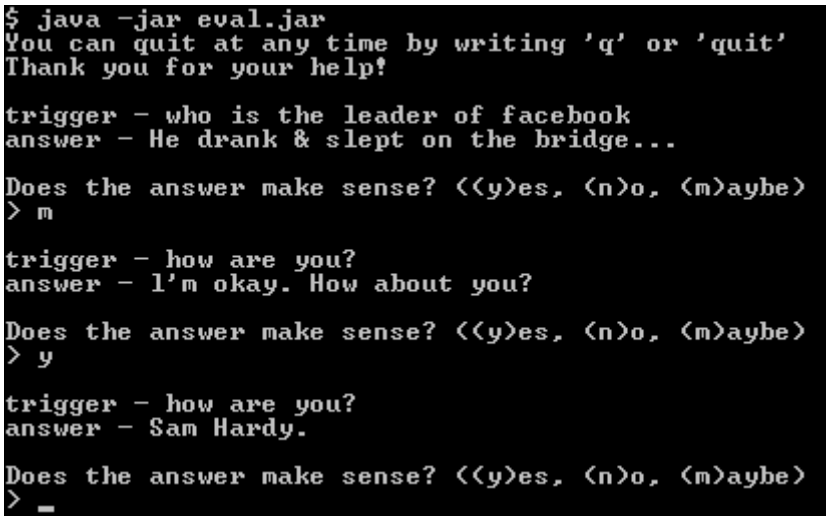

Figure 4.1: Example of the implementation of a conversation

small subset of the corpus, comprising five triggers and their possible corresponding answers, and had another annotator evaluate them. The results are present in Table 4.1. To find out if this was a good agreement, we calculated Cohen's weighted kappa coefficient [6]. We used weighted kappa instead of non-weighted because, in this case a disagreement between "yes" and "maybe" or "no" and "maybe" should be less important than between "yes" and "no". The result was a kappa of 0.602, which according to Landis and Koch [15] represents "substantial agreement". With this result we feel confident that our decision of having only one annotator per answer is a valid one.

	Yes	No	Maybe	Total
Yes	43	9	3	55
No	1	24	1	26
Maybe	3	10	1	14
Total	47	43	5	95

Table 4.1: Agreement between the two annotators

4.3 Extracting new features

After building the corpus we proceeded to the extraction of features from each answer. We selected the measures used to score the T-A pairs explained in Section 3.5, and used them as features. Besides these features we also used the following:

- Length of the answer - Uses the formula described in Equation 4.1, which ranges from 0.0 (only one word in the answer) to 1.0 (Nine or more words in the answer). We calculated the average length (number of words) of 3200 answers returned by Lucene and found it was 5.9, so we think that 9 fits well as a value for a large sentence.

- First word of the trigger - Creates a vector of the size of all unique first words in the triggers of the corpus. In this vector, the position corresponding to the first word in the trigger is 1 and all the

rest are 0. To exemplify this, let's say we have only three triggers in our corpus. The first word of the first trigger is "Why" while the first word of the second trigger is "How" and the first word of the third trigger is, again, "How". These triggers would be encoded as the vectors [1 0], [0 1] and [0 1], respectively.

- First word of the answer - Exactly the same as for the trigger, but using the answer.

$$length(answer) = \begin{cases} \frac{\#words}{8} - \frac{1}{8} & \text{if } 1 <= \#words <= 8 \\ 1 & \text{otherwise} \end{cases} \tag{4.1}$$

4.4 Description of the Framework

To evaluate different learning to rank approaches we used a framework developed by Moreira et al. for her MSc [29]. With this framework we were able to test four different learning to rank algorithms, all of which are thoroughly described in the aforementioned work [29]. We used this framework like a black box, so we will just describe the most important features of each of the algorithms:

- AdaRank [39] - AdaRank is a listwise learning to rank method, which means it receives as input a list of candidates from the training data and predicts their relevance by using scoring functions which try to directly optimize the value of an evaluation metric. AdaRank builds a ranking model using the Boosting approach and attempts to optimize a specific performance measure. It trains one weak ranker at each round of iteration, and combines these rankers to create the final ranking function.

- Coordinate Ascent [28] - Coordinate Ascent is also a listwise method and is an optimization algorithm, used in unconstrained optimization problems. It builds a ranking model by directly maximizing a performance measure.

- RankBoost [10] - RankBoost is a pairwise learning to rank method, which means it receives as input a pair of candidates from the training data and tries to find which of the two is better. RankBoost also builds a ranking model using the Boosting approach and so, it works in a very similar manner as AdaRank.

- RankNet [5] - RankNet is a pairwise method as well and it builds a ranking model through the formalism of Artificial Neural Networks, attempting to minimize the number of misclassified pairs.

This framework uses RankLib[1], a library that has implementations of many learning to rank algorithms. When running one of the algorithms in RankLib we have to choose the number of rounds to train, but when running them with this framework we instead specify the *maximum* number of rounds to train, and it will run the RankLib algorithm with all the values up to that maximum. To exemplify, lets say we ran AdaRank in the framework with the value of the maximum number of rounds being 5. In

[1] `http://sourceforge.net/p/lemur/wiki/RankLib/`

this case, it would run the AdaRank algorithm in RankLib 5 times, first using 1 as the number of rounds, then 2, and so on until the maximum of 5 rounds. In every iteration the framework checks if the final results were better than the previously obtained, according to a specified metric, and in that case it saves these results. In the end we are left with the best possible model up to the maximum number of rounds specified.

4.5 Evaluating the different algorithms

After extracting the features from the corpus, the next step was to run each of the learning to rank algorithms of Moreira et al. framework. We randomly divided the corpus into train and test set in four different ways (folds 1 through 4), each with the features of 10 triggers (and the corresponding answers) in the test set and 70 in the train set. This way we can validate the final model returned by the corresponding algorithm. The evaluation is performed simply by checking if the answer that got the highest score is annotated as being a good answer, from now on called Precision at One, or P@1. We chose this metric because it represents the answer returned by the algorithm, which is what we want to be sure makes sense. We compare these evaluations, with our previous best setting, S_4, described in Section 3.6.3. The results of the evaluation can be seen in Table 4.2.

	Fold1	Fold2	Fold3	Fold4	Average
AdaRank	70%	**80%**	50%	70%	67.5%
Coordinate Ascent	70%	**80%**	**80%**	**100%**	**82.5%**
RankBoost	**100%**	60%	60%	**100%**	80%
RankNet	70%	70%	**80%**	60%	70%
S_4	60%	50%	30%	50%	47.5%

Table 4.2: Precision at One of the different algorithms

These results show that the learning to rank approach got better results than our previous best approach, S_4. It is important to note, however, that, in order to do the same evaluation in both these cases, our S_4 approach only considered 20 possible answers, just like the learning to rank algorithms. This may be the reason why S_4 fared worse in these tests than in the ones described in Section 3.6.3, where this setting could consider 200 possible answers. Between the different learning to rank approaches, Coordinate Ascent achieved the best results followed closely by RankBoost.

To see if the three new features introduced in Section 4.3 were of any help to achieve the aforementioned results, we ran the same tests but now only considering the first 4 measures described in Section 3.5. The results are in Table 4.3.

	Fold1	Fold2	Fold3	Fold4	Average
AdaRank	10%	40%	20%	30%	25%
Coordinate Ascent	10%	20%	**30%**	40%	25%
RankBoost	30%	20%	10%	30%	22.5%
RankNet	40%	30%	20%	40%	32.5%
S_4	**60%**	**50%**	**30%**	**50%**	**47.5%**

Table 4.3: Precision at One of the different algorithms

These results show that the new measures introduced greatly improved the results of each of the learning rank algorithms used. The best performing learning to rank algorithm found in this test (RankNet) has a P@1 difference of more than half of the best performing algorithm found in Table 4.2. This time, our combination of weights setting achieved the best performances of the five tested. One possible explanation for this is that with such a small train set (comprising of only 80 questions), the learning to rank algorithms are not able use the values of the four measures as effectively as we do in our setting, and need much more information to achieve better results.

4.5.1 Discussion

Although these are very promising results, they do not necessarily translate to good answers by SSS. We tested the models of these algorithms and found that, when dealing with 200 possible answers, they perform much worse, and sometimes are not able to answer correctly to very common questions. One of the reasons for this may be that measure M_2 functions best with more data, because it is calculating the answer frequency. When training the learning to rank models, however, this measure is only looking at 20 possible answers, so the scores may be very different. To have a model suitable to our needs we would need many more annotated triggers and answers, which is the main problem with this approach. We are left with a trade-off between arbitrarily defined weights and a considerate annotating effort.

Chapter 5

Final evaluation

In this chapter, we will present the results of our last evaluation. We will compare our results with the ones obtained by Ameixa et al. in his MSc [1].

5.1 Evaluation setup

To compare both versions of SSS we will perform the same evaluation in this new version of SSS as the one performed by Ameixa et al. [1]. Like in the aforementioned work, we will use exactly the same set of user requests, which comprises of requests posed to Edgar that he was not able to answer, in two different languages: Portuguese and English. After SSS has all the answers, we will divide them into Discarding (when the system was unable to provide any answer) and Non Discarding (when an answer was returned). We will then subdivide the latter group into Suitable (answers that were manually analyzed and considered as properly addressing the user requests), and Non Suitable answers (answers that were not considered as properly addressing the user requests). The main goal of this evaluation is to see if we can get more Suitable answers than SSS.

In this evaluation, we used the best setting found in Section 3.6.3, setting S_4. It is important to note that, in his thesis, Ameixa et al. had very clear examples of what he considered to be Suitable (Figure 5.1) and Non Suitable (Figure 5.2) answers, for both the English and Portuguese languages. This made our evaluation easier, because we could use the same criteria as he used in his evaluations.

1 – Como está? Está bom? (How are you? Are you OK?)
1 – *Sim.* (Yes.)

2 – Olá. Eu sou o Pedro. (Hello. I'm Pedro.)
2 – *Olá, prazer.* (Hello, nice to meet you.)

3 – Tens fome? (Are you hungry?)
3 – *Não, estou bem, obrigada.* (No, I'm fine, thank you.)

Figure 5.1: Examples of answers from the Portuguese corpus considered Suitable, taken from [1]

1 – Você é muito bonito! (You are very handsome!)
1 – *Sou o Dr. Robert Trenton.* (I'm Dr. Robert Trenton.)

2 – Boa tarde. Como está? (Good Afternoon. How are you?)
2 – *O que faz aqui?* (What are you doing here?)

3 – Você deve estar a gozar comigo. (You must be kidding with me.)
3 – *Ouviram-me. Vamos embora.* (You heard me. Let's go.)

Figure 5.2: Examples of answers from the Portuguese corpus considered Non Suitable, taken from [1]

5.2 Evaluation using the English Corpus

In this evaluation we use the corpus of 58 different OOD requests posed to Edgar in English (Annex 8). Table 5.1 shows a comparison between the results obtained in the previous version of SSS and the current version.

		Previous	Current
Discarding		16	0
Non Discarding	Suitable	27	37
	Non Suitable	15	21

Table 5.1: Comparison between the results of the previous version of SSS with the current for the English corpus

Like we said in Section 3.5, one of the advantages of adopting a scoring system is that no answers are thrown away, which means that, as long as Lucene is able to return at least one answer to the given input, we can return that answer to the user. This is the reason why we were able to answer all of the 58 requests, while the previous version would give a discarding answer to 16 requests. As for the Non Discarding answers, the current version was able to answer, in a suitable manner, 10 more requests than the previous version, giving also 6 more Non Suitable answers. This means that our version gave Suitable answers to 63.8% of all user requests compared to 46.6% in the previous version, a difference of 17.2 percentage points.

If we compare only the number of Suitable answers divided by the number of Non Discarding ones, the current system achieves the same score of 63.8% (because it had an answer to every question), while the previous achieves a score of 64.3%, an increase of 0.5 percentage points. This means that, even though our version, by trying to answer all of the requests, was probably bound to have more Suitable answers (even if it picked them randomly), we were able to attain a score very close to the one obtained by the previous version of SSS.

5.3 Evaluation using the Portuguese Corpus

In this evaluation we use the corpus of 99 different OOD requests posed to Edgar in Portuguese (Annex 9). In this case, we also compare with the results of the original version obtained using LSH, that for the Portuguese language was able to achieve slightly better results than Lucene. Table 5.2 shows the comparison between the three different versions.

		Previous with Lucene	Previous with LSH	Current
Discarding		64	70	1
Non Discarding	Suitable	11	15	58
	Non Suitable	24	14	40

Table 5.2: Comparison between the results of the previous version of SSS with the current for the Portuguese corpus

For the same reason as we explained in the previous section, the current version of SSS was able to greatly decrease the number of Discarding answers. It only gave 1 Discarding answer, compared to 64 in the previous version using Lucene and 70 using LSH. Regarding the Non Discarding answers, our version was able to get 47 more Suitable answers than the previous attempt with Lucene and 43 more than with LSH. Because it tried to answers all the requests it also had much more Non Suitable answers, namely: 16 more than the original version with Lucene and 26 more than LSH's version. In terms of percentages, our version gave Suitable answers to 58.6% of all user requests compared to 11.1% in the previous version using Lucene, and 15.1% in the version using LSH. The current version was able to answer suitably to 43.5 percentage points more requests than the previous best version.

Comparing the number of Suitable answers divided by the number of Non Discarding ones instead of all, the current system achieves a score of 59.2%, while the previous Lucene version achieved a score of 31.4%, and the version using LSH got a score of 51.7%. In this case, even when not considering the answers discarded, the current system performed better than the previous best, with a 7.5% increase in percentage points.

5.4 Discussion

With this results, we feel that this work was a nice step forward for SSS. The Portuguese version, in particular, was greatly improved, and can now achieve results comparable to its English counterpart. We are confident that, in light of these results, and considering the number of available subtitles, we could extend SSS to work for other languages.

One thing we need to consider is the increase of Non Suitable answers resulting from trying to answer every single request. Although the main goal of SSS is to deal with OOD requests, returning nonsensical answers frequently might hurt this purpose more than admitting that sometimes it is better to just give a discarding answer like "I'm sorry, i didn't understand that". We could have some kind of *minimum score threshold*, that, in case the best answer found for a given user input fell bellow this threshold, then a discarding answer would be returned.

Chapter 6

Conclusions

As it is impossible to handcraft responses to all the possible OOD turns that can be posed by humans to virtual conversational agents, we hypothesize that conversations between humans can provide some plausible answers to these turns.

In this work we describe the process of building an improved version of the Subtle corpus, composed of pairs of interactions from movie subtitles. A preliminary evaluation shows that the Subtle corpus does include plausible answers. The main challenge is to retrieve them. We introduced a new way of choosing an answer, by using a combinations of measures in order to score each of the possible answers, and then choosing a one with the highest score. This way, we made it very easy for anyone to add new measures and combine them with the current ones. We have tested several measures in SSS that take into consideration the similarities between the user input and the trigger/answer of each retrieved interaction, as well as the frequency of each answer. Also, the time elapsed between the subtitles is taken into consideration. Different weights were given to the different measures and the best results were attained with a combination of the measures: 21 users considered that 61.67% of the answers returned by SSS were plausible; the time elapsed between the turns did not help in the process.

To relieve the burden of having to fine-tune weights for each measure used in SSS, we research the viability of using Learning to Rank in our process of choosing an answer. We found that this approach is feasible with very promising results. Coordinate Ascent, the learning to rank algorithm that performed best in our tests, was able to get 35 percentage points more of good answers than our previous best solution.

We also compare the current version of SSS with the previous one and find that the current version is able answer almost all of the requests, while maintaining a good percentage of suitable answers. In the English version we achieve an increase of 17.2 percentage points of suitable answers compared to the previous version, and in Portuguese, we achieve an increase of 43.5 percentage points of suitable answers compared to the previous best version.

Using the work developed for this thesis (before researching our learning to rank approach), we wrote a paper [22], that was accepted in this year's SemDial (DialWatt)[1]. We also presented Filipe, using SSS,

[1] http://www.illc.uva.nl/semdial

in this year's *Researchers' Night*[2].

6.1 Contributions

- Reviewed the state of the art of techniques and systems that handle OOD interactions, and how they choose an answer.
- Improved the way the interactions are built in Subtle as well as adding the time difference of the subtitles and DialogId, an identifier used by SSS to build a conversation.
- Developed two new ways of choosing an answer from a set of possible ones: the combination of several measures in a linear way and through the use of the learning to rank paradigm.
- By implementing the combination of measures to choose an answer, it is now very easy to add new measures to SSS.
- Adapted SSS to work for Portuguese.
- As the result of the previous contribution, a new version of Filipe is now on-line, working with the Portuguese language[3].

6.2 Future Work

There is still much room from improvement. First, the context of the conversation should be taken into consideration. When comparing two sentences, in our measures, we could use part of speech tagging to give different weights to different classes, for example, giving verbs a higher weight than pronouns. Semantic information, such as the one presented in synonyms, could also be used in the similarity measure. As of now we compare two words by simply seeing if they have the same letters, but if we could use synonyms we could probably achieve better results. Besides this, information regarding dialogue acts could also be used in this process.

Also, responses that refer to idiosyncratic aspects of the movie should receive a lower score. Although M_2 can be seen as an indirect metric for this domain-independence (a frequent response is less likely to come with a strong contextual background), responses that include names of particular persons, places or objects should be identified. However, this strategy is not straightforward, as some particular responses containing named entities should not be discarded. This is the case not only to address factoid questions, but also for inputs such as "*Where do you live?*" or "*What is your mother's name?*" whenever a pre-defined answer was not prepared in advance.

Regarding learning to rank, the corpus used is very small, comprising only of 80 triggers and only 20 questions for each trigger. We are confident that with a large enough corpus we can achieve better

[2] http://www.noitedosinvestigadores.org/
[3] http://www.l2f.inesc-id.pt/~pfialho/sss/pt/

results and use this paradigm in SSS. Besides this, more features can be add to the train corpus like unigrams and bigrams to see if with them we can achieve better results.

With respect to Subtle, there are also many things that can still be improved. Tags like the ones found in the Personage Corpora could be used. We could also use corpora that do not have the problem of not knowing if two interactions are in fact a T-A pair (like the ones presented in Section 2.3.1 and Section 2.3.2) and use them to train a classifier that could be used in Subtle to build T-A pairs.

Bibliography

[1] David Ameixa. Sss: Say something smart. Master's thesis, Instituto Superior Técnico, October 2013.

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 722–735, Berlin, Heidelberg, 2007. Springer-Verlag.

[3] Rafael E. Banchs. Movie-dic: A movie dialogue corpus for research and development. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2*, ACL '12, pages 203–207, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[4] Rafael E. Banchs and Haizhou Li. Iris: a chat-oriented dialogue system based on the vector space model. In *Proceedings of the ACL 2012 System Demonstrations*, pages 37–42, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

[5] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 89–96, New York, NY, USA, 2005. ACM.

[6] J Cohen. Weighted kappa: nominal scale agreement with provision for scaled disagreement or partial credit. *Psychological bulletin*, 1968.

[7] Cristian Danescu-Niculescu-Mizil and Lillian Lee. Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics, ACL 2011*, 2011.

[8] Laila Dybkjær, Niels Ole Bernsen, and Wolfgang Minker. Evaluation and usability of multimodal spoken language dialogue systems. In *IN: SPEECH COMMUNICATION*, pages 33–54, 2004.

[9] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 363–370, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[10] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, December 2003.

[11] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997.

[12] Zhiheng Huang, Marcus Thint, and Zengchang Qin. Question classification using head words and their hypernyms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 927–936, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.

[13] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[14] John Judge, Aoife Cahill, and Josef van Genabith. Questionbank: Creating a corpus of parse-annotated questions. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 497–504, Sydney, Australia, July 2006. Association for Computational Linguistics.

[15] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[16] Ian Lane, Tatsuya Kawahara, Tomoko Matsui, and Satoshi Nakamura. Out-of-domain utterance detection using classification confidences of multiple topics. *Trans. Audio, Speech and Lang. Proc.*, 15(1):150–161, January 2007.

[17] Anton Leuski, Jarrell Pair, David Traum, Peter J. McNerney, Panayiotis Georgiou, and Ronakkumar Patel. How to talk to a hologram. In *Proceedings of the 11th International Conference on Intelligent User Interfaces*, IUI '06, pages 360–362, New York, NY, USA, 2006. ACM.

[18] Anton Leuski and David Traum. Practical language processing for virtual humans. In *Innovative Applications of Artificial Intelligence*, 2010.

[19] Anton Leuski and David Traum. NPCEditor: a tool for building question-answering characters. In *International Conference on Language Resources and Evaluation (LREC)*, Valletta, Malta, May 2011.

[20] Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[21] Diane Litman, Satinder P. Singh, Michael Kearns, and Marilyn Anne Walker. Njfun: a reinforcement learning spoken dialogue system. In *ConversationalSys '00: Proceedings of the ANLP-NAACL 2000 Workshop on Conversational Systems*, page 17–20, Morristown, NJ, USA, 2000. Association for Computational Linguistics, Association for Computational Linguistics.

[22] Melo F. Magarreiro D., Coheur. Using subtitles to deal with out-of-domain interactions. In *Proceedings of SemDial 2014*, pages 98–106, 2014.

[23] François Mairesse and Marilyn A. Walker. Towards personality-based user adaptation: Psychologically informed stylistic language generation. *User Modeling and User-Adapted Interaction*, 20(3):227–278, August 2010.

[24] François Mairesse and Marilyn A. Walker. Controlling user perceptions of linguistic style: Trainable generation of personality traits. *Comput. Linguist.*, 37(3):455–488, September 2011.

[25] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[26] Manish Mehta and Andrea Corradini. Handling out of domain topics by a conversational character. In *Proceedings of the 3rd International Conference on Digital Interactive Media in Entertainment and Arts*, DIMEA '08, pages 273–280, New York, NY, USA, 2008. ACM.

[27] Ana Cristina Mendes, Luísa Coheur, João Pedro Carlos Gomes da Silva, and Hugo Rodrigues. Just ask – a multi-pronged approach to question answering. *International Journal on Artificial Intelligence Tools*, 22(1):34, February 2013.

[28] Donald Metzler and W. Bruce Croft. Linear feature-based models for information retrieval. *Inf. Retr.*, 10(3):257–274, June 2007.

[29] Catarina Moreira. Learning to rank academic experts. Master's thesis, Instituto Superior Técnico, November 2011.

[30] Pedro Mota. Lup: A language understanding platform. Master's thesis, Instituto Superior Técnico, July 2012.

[31] Ronakkumar Patel, Anton Leuski, and David Traum. Dealing with out of domain questions in virtual characters. In *Proceedings of the 6th International Conference on Intelligent Virtual Agents*, IVA'06, pages 121–131, Berlin, Heidelberg, 2006. Springer-Verlag.

[32] Sérgio Curto Pedro Cláudio Ângela Costa Alberto Abad Hugo Meinedo Pedro Fialho, Luísa Coheur and Isabel Trancoso. Meet edgar, a tutoring agent at monserrate. In *51st Annual Meeting of the Association for Computational Linguistics*, ACL '13, pages 61—-66, 2013.

[33] Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *In HLT-NAACL '07*, 2007.

[34] S. Seneff and J. Polifroni. Dialogue Management in the Mercury Flight Reservation System. In *Dialogue Workshop, ANLP-NAACL*, Seattle, April 2000.

[35] David Traum, Priti Aggarwal, Ron Artstein, Susan Foutz, Jillian Gerten, Athanasios Katsamanis, Anton Leuski, Dan Noren, and William Swartout. Ada and Grace: Direct interaction with museum visitors. In *Proc. 12th Int. Conf. Intelligent Virtual Agents*, pages 245–251, 2012.

[36] Marilyn Walker, Grace Lin, and Jennifer Sawyer. An annotated corpus of film dialogue for learning and characterizing character style. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey, may 2012. European Language Resources Association (ELRA).

[37] Ulli Waltinger, Alexa Breuing, and Ipke Wachsmuth. Interfacing virtual agents with collaborative knowledge: Open domain question answering using wikipedia-based topic models. Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI–11), pages 1896–1902. AAAI Press, 2011.

[38] Erik Weitnauer, Nick M. Thomas, Felix Rabe, and Stefan Kopp. Intelligent agents living in social virtual environments — bringing max into second life. In *Proceedings of the 8th International Conference on Intelligent Virtual Agents*, IVA '08, pages 552–553, Berlin, Heidelberg, 2008. Springer-Verlag.

[39] Jun Xu and Hang 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 '07, pages 391–398, New York, NY, USA, 2007. ACM.

Chapter 7

Questionnaire used to evaluate combination of measures

Edit this form

Say Something Smart

Pretendem-se avaliar possíveis respostas de um agente virtual/chatbot a um conjunto de perguntas feitas por vários utilizadores. Por favor, indique se considera que fazem sentido ou não cada uma das respostas dadas, tendo em conta a pergunta realizada.
Este teste dura menos de 5 minutos.
Obrigado!

* Required

Personal Information

How old are you? *

- 18 - 25
- 26 - 38
- 39 - 60

Gender *

- Female
- Male

Respostas do agente/chatbot

Where is the capital of japan? *

	Does not make sense	Makes sense
Don't tell me, you must be Ricky Monroe?		
You are, i believe, a visitor in the neighborhood.		
It's not a capital either.		
Thanks, I know where Helsinki is.		

Where is your mother? *

	Does not make sense	Makes sense
I have no mother		

	Does not make sense	Makes sense
What are you doing?	○	○
Do you know where she is?	○	○

Where are you living? *

	Does not make sense	Makes sense
At the mansion Ekling where you found me.	○	○
I'm in the hotel Ibis.	○	○
Right here.	○	○
I live in Brooklyn.	○	○

Do you have brothers? *

	Does not make sense	Makes sense
I'll manage the business	○	○
I'm not your brother.	○	○

What country do you live? *

	Does not make sense	Makes sense
We used to live in the one above you.	○	○
I'm going to kill your mother or your father.	○	○
I heard you were the best.	○	○
I got some things stored away.	○	○

What's your mother's name? *

	Does not make sense	Makes sense
What's her name? Don't you want to tell me?	○	○
The mother's name isn't important.	○	○

Are you married? *

	Does not make sense	Makes sense
No.	○	○
Uhm, no, I'm not married.	○	○
No, I never married.	○	○
Married? No.	○	○

What do you like? *

	Does not make sense	Makes sense
Why my heart should feel such sadness	○	○
I don't know.	○	○
I don't like anything.	○	○

What's your name? *

	Does not make sense	Makes sense
My name is Fanny	○	○
Jim. Jim Nesbitt.	○	○
What's your name?	○	○

Are you joking? *

	Does not make sense	Makes sense
I'll try to be on time. - And Pampers.	○	○
No, I'm not joking.	○	○
No, you're not.	○	○

Are you a loser? *

	Does not make sense	Makes sense
One cursed of a useless one.	○	○

	Does not make sense	Makes sense
You're a loser. Bitch.	○	○

Where are you from? *

	Does not make sense	Makes sense
I'm from Oklahoma.	○	○
Uganda.	○	○
Where am I from?	○	○

What about your wife? *

	Does not make sense	Makes sense
She's going to divorce me.	○	○
She's your wife?	○	○

What is your job? *

	Does not make sense	Makes sense
Oh, you want it to be a surprise? Are you working at a restaurant?	○	○
It's my job.	○	○
No, you are, but I have to do this now.	○	○
I thought you were a freelancer.	○	○

What is the building behind you? *

	Does not make sense	Makes sense
Look what's behind you. I only believe in myself.	○	○
That's not what I showed you!	○	○
I'm hiding. Hiding? What do you mean?	○	○
Um...	○	○

What's your father's name? *

	Does not make sense	Makes sense
My father's name? Peter.	○	○
What's her name? Don't you want to tell me?	○	○
Subramanyam, when young lovers want to get married, how dare you put conditions for their marriage?	○	○

I wasn't looking for a ride. Do you have a car? *

	Does not make sense	Makes sense
This is a car?	○	○
You're lucky I didn't run you over.	○	○
That's not for you to decide.	○	○
Do you think that truck belongs to me?	○	○

What is the result of one plus one? *

	Does not make sense	Makes sense
The report came out.	○	○
Just asking. You don't have to be so cool!	○	○
That's my undersea dome. It's just one of the first things I ever did.	○	○
Plus One More, you know what I'm saying?	○	○

How are you? *

	Does not make sense	Makes sense
Good; How are you?	○	○
Sam Hardy.	○	○
I'm fine.	○	○

Who is the leader of facebook? *

	Does not make sense	Makes sense
Rajputs!	○	○
No, I'm the leader!	○	○

Comments

Here you can give us feedback about this form

Submit

Never submit passwords through Google Forms.

Chapter 8

OOD English requests posed to Edgar

1. Huge library.
2. Hello Edgar, how are you doing?
3. How are you?
4. How are you doing, Edgar?
5. Clean your nose.
6. Who is Tiago?
7. Stupid.
8. Kill.
9. Are you joking?
10. Fuck you!
11. Why you talk so funny dude?
12. I said hello!
13. O.K. if you want!
14. We're supposed to be friends, Edgar!
15. Have you ever had sex?
16. You are a dumb shit?
17. Hello, how are you?
18. Hi Edgar!
19. Hello Edgar, how are you?
20. How are you? How are you Edgar?
21. Who created you?
22. Sorry.
23. I try again.
24. Are you a donkey?
25. Do you know Maureen Fisher?
26. That's very usefull. Do you know Maureen Fisher?
27. Hello Edgar!
28. Hello Edgar, are you ready?
29. How are you today?
30. Are you done?
31. Hi Edgar, how are you?
32. What the fuck are you talking about, Edgar?
33. Yeah, right!
34. Hello Edgar, bring me a beer.
35. You have a big nose.
36. Are you a coconut?
37. Edgar, a coffee, please.
38. Can you bring me some coffee, please?
39. Thanks.
40. Be quiet!
41. Edgar, I love you!
42. Are you pregnant?
43. Fuck you, Edgar!
44. Hi Edgar, how is life?
45. Are you gay?
46. Who is your master?
47. Who's your boss?
48. Very good!
49. Are you sexy?
50. You can go home now. You're fired!
51. How are you doing these days?
52. Fine, I hope.
53. I don't want to know, how are you doing?
54. What's my name?
55. Okay.
56. You, you.
57. Can you tell me something?
58. Hi, Edgar! How are you?

Chapter 9

OOD Portuguese requests posed to Edgar

1. Casa bonita, linda sala.
2. Para que servem tantos livros?
3. 9 x 5
4. Olá Edgar! Compreende-me?
5. Quanto custas, Edgar?
6. Estás bom?
7. Então e o Luís gosta de mim ou não?
8. Onde é que tu estás?
9. Não percebeste nada. Eu estou a perguntar onde é que tu estás?
10. És pouco esperto.
11. De que marca é o teu uniforme?
12. Usas cuecas?
13. És um nabo.
14. Olá Edgar. Chato!
15. E então és burro!
16. Tudo bem?
17. Então está tudo?
18. Está bem tótó.
19. E tu gostas de rabos?
20. Mimimimimimi
21. O teu nariz é de que tamanho?
22. Que tamanho é que tem o teu nariz?
23. Nós não queremos saber dessas lamechices.
24. Conhece a Andreia?
25. E o castelo de Leiria conhece, o castelo de Leiria?
26. Gostas de sopa?
27. Estás bom?
28. Edgar, tu és um burrinho!
29. Olá Edgar! Estás bom?
30. Já almoçaste hoje?
31. Tens fome?
32. Olá! Tudo bem?
33. Não sabes a resposta, burro.
34. Estás a xonar?
35. Eu não te perguntei isso.
36. Epá, onde é que está a cerveja?
37. É mentira pá.
38. Vá para o diabo.
39. Como estás Edgar?
40. Tens uma grande lata!
41. Gostas de raparigas?
42. Gostas de falar comigo?
43. És homossexual?
44. Olá. Está tudo bem?
45. Não queres falar.
46. Boa tarde. Como está?
47. Edgar, está bom?
48. Cala-te, eu mato-te.
49. Olá. Eu sou o Pedro.
50. Que engraçado.
51. Tótó!
52. Olá! Onde é que há comida?
53. Olá Edgar. Estás bom?
54. A que horas é que se almoça aqui?
55. Almoço a onde?
56. Gostas deste sítio?
57. Em que ano é que os primeiros reis nasceram?
58. Isso aqui é tão ruim assim mesmo?
59. Estás bom?
60. Você deve estar a gozar comigo.
61. Compreendes português?
62. Fala sobre a tua vida.
63. Não me estás a perceber?
64. Mas és um boi!
65. Eu sou o pai Natal!
66. Estava a dizer boa tarde.
67. Cala-te!
68. Tu não sabes nada!
69. Tinha quanto dinheiro?
70. Carlitos maricon!
71. Então meu puto, como é, tudo bem?
72. Está-me a dar baile?
73. Como está? Está bom?
74. Você gosta de piscina?
75. Percebesses!
76. Gostei muito de o ouvir, apesar de ser muito desagradável.
77. Desculpa mas não percebi. Você é um hipópotamo!

78. Há quanto tempo existe este castelo?
79. Posso fazer qualquer pergunta?
80. A Margarida é gira?
81. Você é muito bonito!
82. Quem sou eu?
83. Olá senhor! Como é que está?
84. Tu és tótó, não és?
85. O que é persistente?
86. O que é que se passa contigo?
87. Igualmente.
88. O que fazes fechado nessa máquina?
89. O que foi o teu almoço?
90. De que tamanho é o teu rabo?
91. Como estás?
92. Qual é a origem das árvores?
93. Tu és surdo!
94. Conheces-me pá?
95. Porque é que estás a olhar assim para mim?
96. Só tem isso, mais nada?
97. O que é que é o jantar hoje?
98. Cala-me essa boca!
99. Tudo bem?