Universiteit van Amsterdam
Faculteit der Natuurwetenschappen, Wiskunde en Informatica

MSc Thesis

Learning in Dialogue Interactions

Author:
Dario Chiappetta

Supervisor:
Prof. Raquel Fernández

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Institute for Informatics
Institute for Logic, Language and Computation

July 2013

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

UNIVERSITEIT VAN AMSTERDAM

Abstract

Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Institute for Informatics
Institute for Logic, Language and Computation

Master of Science

Learning in Dialogue Interactions

by Dario Chiappetta

A basic fact about human-human dialogue is that there is often more than one way of talking about any domain. For example, instead of saying *"Turn right after 200 meters"*, a route giver may say *"Turn right at Barclays"* given that *"Barklays"* is an enstabilished referring expression. Human speakers efficiently align terminology and associated ontology in talking about any specific domain (in this example we can say that ontologically there is only one entity – a location – but there are different linguistic terms we can use to refer to it). In contrast, current dialogue systems typically have a static ontology and a static vocabulary, which is used in both generation and interpretation, requiring users to formulate their utterances using the terminology known by the system. The goal of this project is to work towards a system that adapts its own linguistic resources (including possible ontology, vocabulary and grammar) to the interlocutor. The system should be able to learn new concepts by assigning new meanings to known words, as well as new words to talk about concepts known by the system in the domain.

Acknowledgements

The acknowledgements and the people to thank go here…

Contents

List of Figures

List of Tables

Abbreviations

AI	**A**rtificial **I**ntelligence
ASR	**A**utomatic **S**peech **R**ecognition
CA	**C**onversational **A**gent
DM	**D**ialogue **M**anager
ISU	**I**nformation **S**tate **U**pdate
MT	**M**achine **T**ranslation
NLP	**N**atural **L**anguage **P**rocessing
SLU	**S**poken **L**anguage **U**nderstanding
TTS	**T**ext **T**o **S**peech

Symbols

c	chunk of text
m	meaning
s	sentence
σ	chunk similarity
τ	word similarity

Introduction

The **user interface**, or human-computer interface (HCI) is the component of a computer system that provides a space of interaction between the human user and the resources offered by the machine; such a space defines a bridge language which human intentions can be translated into, to be converted into computational procedures for the machine; vice versa, the result of the computation is then presented to the user in the same language, which he or she is assumed to understand.

In the early days of computing, the so-called **batch interfaces** were non-interactive: the user/programmer was supposed to feed the machine with a software, punched on cards using the *machine's assembly language* directly, and retrieve the result of the computation printed on paper. The third and fourth generations of computing brought **text-based** interfaces for operating systems (UNIX, DOS, CP/M), that were later took over by **Graphical User Interfaces** (GUI), moving the human-machine interaction on a new, visual language made of windows, icons and buttons. Recently, touch screen and camera devices allowed for the implementation of even more natural means of interaction based on **gestures** and physical actions.

From this brief spot on computing history, and in a way from common sense, we can draw the rather trivial, yet crucial, conclusion that the trend in user interfaces development is to **close the gap** between humans and machines by moving the needle of the interface languages from a machine-centered space towards the human language itself. To this respect, the studies on Natural Language Processing (NLP) assume a dramatically central role, as dramatically central is natural language in the interaction of humans with each other.

Dialogue Systems

A spoken dialogue system, or conversational agent (CA), allow humans and machines to interact through an intermediate language which is as close as possible to the **human language**, and through conversational episodes that implement as close as possible the human dialogue modalities.

Research in dialogue systems has been carried on since the **early days** of Artificial Intelligence. A milestone in the early work on this field is ELIZA Weizenbaum [1966], which provides the user with a basic human-like interaction based on pattern matching; another example is the SHRDLU system , which interfaces the user with a simple spatial domain being able to ambiguous or implicit references to the entities in it.

According to Jokinen and McTear [2009], modern dialogue systems can be divided in **two main types**: task-oriented and nontask-oriented. Intuitively, systems in the first category are meant to deal with a specific task such as making a hotel booking, or booking a plane ticket; an example in this category is the MIT Mercury system, a vocal interface to a flight database Seneff and Polifroni [2000]. On the other hand, nontask-oriented systems are meant to engage in conversations without a specific purpose to fulfill, but the one of delivering a realistic simulation; ELIZA itself is an example of nontask-oriented dialogue system.

Task-oriented systems can be very simple, as simple and well-formalized the task is; many applications, such as travel service or call routing, can be successfully solved by **slot-based** systems: each step of the conversation requires some pieces of information, modeled as slots, to be filled in by the user (departure city, arrival city, date, and so on); given the slots to be filled, the dialogue task can be solved with a formal grammar of interaction. As the complexity increases, more phenomena of human interaction have to be modeled, such as turn-taking, multimodality or grounding , as well as semantic structures such as quantification and negation; slot-based systems are not sufficient to model these scenarios Gabsdil [2003], that require more advanced frameworks such a the Information State Update (ISU) one.

Learning to talk

One of the constituent features of humans' ability to speak is that such an ability is **not innate**, but is rather learned through interactions.

Ever since the power of computers grew enough to allow for intensive statistical analysis of significant amounts of data, **Machine Learning** approaches to Artificial Intelligence tasks got more and more prominent in the scene, often outperforming static methods (i.e. where the solution procedure for a task is explicitly coded by the programmer). One of the clearest examples is the field of NLP itself, where the most important tasks, like parsing or machine translation, are dominated by Machine Learning methods based on corpora, meaning that, for instance, a Machine Translation system will first be trained on a corpus of aligned sentences. Statistical structures will be extracted from this corpus, like the most likely word-by-word alignment, and later be used to process new examples.

However, the task of learning dialogue interactions brings some **peculiar** challenges. First of all, humans learning to talk do not go through two separate phases of learning and processing, but rather improve their abilities episode by episode; as Fernández et al. [2011] point out, this incremental learning structure is nowadays not implemented in state-of-the-art systems. Furthermore, ...

- Learning to produce or understand new surface forms, or **realizations**, for a given meaning – eg. the sentence "Bill eats an apple" for the action of Bill eating an apple.
- Learning to produce new **meanings** from the existing ones and their respective realizations – eg. the concept of motor home, sharing the features of a house and a car.
- Learning a **grammar of conversation**, to place the correct utterance at each step of a conversational episode. – eg. an appropriate answer to the utterance "My name is Bill" is "Nice to meet you", whereas "I like cookies" would not sound as much appropriate.

Lastly, we can point out that the definitions of actions, meaning and episodes are somehow arbitrary and not sharply bounded, as it can be argued for recursive and compositional structures at any level of their interpretation.

This thesis project

The aim of this thesis project is to design and implement language learning capabilities for an existing dialogue system, focusing on the **realization level**. That is, given a fixed list of meanings, the system should be able to classify every sentence into its correct meaning.

The **domain** I am considering is the one of a music player application, being able to get natural language input from the user and translate it into an appropriate corresponding behaviour. For example, when the input is "Play Pictures at an exhibition", the system should start playing the famous suite by Modest Mussorgsky.

In this domain, each **meaning** is an action that a player is performed, and is defined by a set of representative sentences, being its surface forms. For instance, the action of increasing the volume level can be defined by the following set of sentences:

- *Increase the volume*
- *Increase the volume level*
- *Raise the volume*
- *Increase the volume please*

Note that, even though one would be likely to think of such meanings in compositional terms, such compositionality have not been explored for the sake of simplicity.

The **task** for the application is, given an arbitrary input sentence and a context (the point of the conversational episode being realized), to perform the correct action, that is, associate that sentence with its correct meaning. Furthermore, the system should be able to **learn** new realizations for each meaning, as unknown sentences are given in input and processed.

Note that such processing might be more or less **semantically intensive**. As an example, it can be argued that, given the above definition of the action to increase the volume, matching *"raise the volume please"* is an easier task than classifying *"Turn up the volume"*.

Also, an unknown input sentence should be given a **confidence score** for each candidate meaning it is associated to; the system should be able to narrow possible needs for clarification down to single sentence components, eventually asking the user for disambiguation as specifically as possible.

This document is structured as follows. Chapter ...

Chapter 1

Related work

The goal of this thesis, to design and implement a learning-capable dialogue system, combines different disciplines within the fields of Artificial Intelligence and Linguistics. This chapter reviews the work that has been previously done, and that contributed to the realization of this project.

1.1 Sentence similarity

As it has been mentioned in 6, the core task of the system is to associate an unknown sentence to its correct meaning, where each meaning is defined by a set of sentences realizing it. Therefore, one of the constituent capabilities that the system must implement is the ability to tell whether two sentences **share the same meaning** or not.

The problem of scoring the similarity between two sentences is not new in the literature. Achananuparp et al. [2008] suggest to classify the existing measures in **three categories**: word overlap measures, TD-IDF measures and Linguistic measures. **Word overlap** scores are computed taking into account only the number of words that are shared between the two input sentences; a basic measure of this kind is the Jaccard coefficient, which is defined as the size of the intersection of the words in the two sentences compared to the size of the union of the words in the two sentences. Banerjee and Pedersen [2003] extended the concept to include a special treatment of phrasal n-word overlaps, motivated by the fact that they are much rarer than single word ones. **TF-IDF** measures are based on term frequency-inverse document frequency, hence the name. Those are common measures to express the importance of a term of a document in an indicized corpus; respectively, they represent the frequency of the term in the document, and the frequency of the term across all documents. TF-IDF can be used to score the similarity between two sentences, for instance, computing the cosine similarity in a vector-space approach. Lastly, **linguistic** measures are meant to exploit, intuitively, the linguistic information contained in the input sentences. Such information consists of semantic relations between words, and the syntactic structure that connects them.

The way sentences are compared in MPlay tries to take into account all the aspects of this three type of measures, which are combined together in a feature-oriented fashion; the specific algorithm for sentence comparison is described in Chapter 3.

1.2 Machine Learning for Language Processing

The task of labeling an unknown sentence with its correct meaning can be easily expressed in terms of Machine Learning. In fact, is is a standard supervised classification problem to learn a class' model from examples, and later use that model to label new data points. In this view, a data point is a natural language sentence, and a label is its meaning.

Particularly inspiring for the development of this thesis was the work done by IBM on Watson. Watson is ...

CL&DOP ...

1.3 Information State Update Dialogue Management

...

Chapter 2

Architecture

The outcome of this thesis project is MPlay , a music player application that accepts English utterances as commands, adapts to utterances it has never been exposed to, and learns from them, thus expanding its initial knowledge of the language. The application has been written in Python 2.7, and consists of a client application for the existing OpenTDM dialogue management library. Such a library supports basic dialogue management based on the Information State Update approach, but has no support for grounding or flexible understanding of unknown sentences. Therefore, TDM has been extended with the Language Unit module, that introduces these capabilities.

Section 2.1 describes the MPlay client, Section 2.2 describes the OpenTDM library, and Section 2.3 describes the Language Unit module.

2.1 The client application

MPlay is a client application for the OpenTDM library. An OpenTDM application can be seen as a container for domain-specific **parameters** for the dialogue manager. These parameters consist of:

- A **device** class, containing variables and methods that directly control the actions of the application. The *device* file of a music player application will contain, for instance, variables holding the current playlist, or the current volume level, and methods to play/stop the music, lookup for a song and so forth.

- An **ontology**, which main purpose is to define predicates an actions that will be used at dialogue management level. In the case of MPlay , an example of predicate is `current_song(X)`, that identifies the song currently being played (note that such a predicate must be mirrored in the device file as a variable[1]); an example of action is `increase_volume` that, intuitively, identifies the action of increasing the volume (actions are mirrored as well in the device class, in the same way predicates are).

- A **domain** file, which main purpose is to contain the list of plans that will control the dialogue episodes. For instance, the TOP plan of an application is to find out what the user wants to do. Another example of a plan in MPlay is the `increase_volume` plan: step 1 of the plan is ask the user for how much the volume should be increased, step 2 is to perform the actual action through the device.

- A **grammar** implemented using the Grammatical Framework[2]. This part will not be discussed, since, as it is explained in Section 2.3, the Grammatical Framework in OpenTDM has been replaced with a specifically created module called Language Unit.

The dialogue management logic is left to OpenTDM, that will be discussed in the next section.

2.2 OpenTDM

OpenTDM is a dialogue management library developed and maintained by Talkmatic[3] based on the Information State Update framework

. . .

[1]The actual implementation consists of an inner `current_song` class of the device class, which access the proper, private, variable of the device file through a `perform()` method. The reasons that led to such an implementative choice were not made known by the authors of OpenTDM.

[2]http://www.grammaticalframework.org/

[3]http://talkamatic.se/

2.3 The Language Unit

The Language Unit (LU) is a Python library that have been specifically created to support MPlay . The purpose of this library is to perform the classification task as it has been defined so far.

2.3.1 Language

The `Language` class is the main class of LU, its purpose is to model a natural language under the following abstraction: a Language is defined by a finite set of Meanings (labels); a Meaning is defined by a finite set of Sentences expressing that Meaning. As an example, the MPlay language consists of a number of meanings, each of them defining an action for the application to perform; one of this meaning can be the one to pause the current song; this meaning will be realized by a number of English sentences, like “Pause the song”, “Suspend the music”, or “Pause the current track”. For the purpose of language understanding, a Sentence can be brougth down to any of its (linguistic or non-linguistic) constituents called Chunks or Phrases. A chunk is formed by one or more Words.

The following are the main capabilities implemented in the Language class:

- **Load** and **save** languages. Each OpenTDM application must define a `language/` folder where the language is stored, in the form of a `.l` file. Such a file is just a dump of all the meanings and their example sentences. Since the language can evolve through learning, applications’ language files are updated every time the application is run and dialogue interactions are performed.
- **Learn** a sentence. When a new labeled example (a sentence with its meaning label) is provided, the knowledge of the language is extended. This is done adding the new sentence to the list of sentences realizing that meaning, and drawing statistics (e.g. the frequency of a certain word/phrase in the given meaning) to improve the model of the meaning. Note that, when a language is loaded for the first time, every sentence in it is run through the learning procedure to initialize the statistics.

- **Understand** an input sentence. The core task of the Language Unit is associate an input sentence to its correct meaning. While this operation is trivial when the input sentence is already present in the language, it becomes hard for unknown examples. In this latter case the LU computes a score for the sentence against each of the meanings that are present in the language; the meaning that achieves the best score is given as an output, along with the score itself, representing the degree of confidence for the output to be correct. The way sentences are scored is presented in the next section.

2.3.2 Scores

2.3.2.1 Meaning Score

2.3.2.2 Sentence Score

2.3.2.3 Chunk Score

Chunks are compared using the M2 algorithm, that is explained in detail in Chapter 3.

2.3.2.4 Word Score

Word scores are as well explained in Chapter 3.

2.3.3 Machine Learning

...

Chapter 3

The M2 matching algorithm

Computing a **similarity score** between two natural language **sentences** is not a trivial problem. Many features can be taken into account to solve it, such as the syntactic structure of the sentence, or the meaning of the single words, or their order; all of this features are informative respect to the meaning of the sentence, and it is reasonable to assume they are all considered by humans solving the same task. Here I describe a recursive, dynamic programming **algorithm** for sentence comparison aimed to exploit word-based information, as well as the syntactic one, expressed in the form of unsupervised learned binary trees.

3.1 Introduction

In the context of sentences **classification**, where the task is to label an unknown input sentence with the correct meaning, and each meaning is defined by a set of representative natural language sentences (as it was defined in 6), one crucial point is to measure the similarity of the input sentence with each of the sentences defining each meaning.

The problem of scoring the similarity between two sentences is not new in the **literature**, and a number of different approaches already exist to tackle it. Achananuparp et al. in **?** classify these approaches in *word overlap* measures (based on the number of words shared by the two sentences), *TF-IDF* measures (based on term frequency-inverse document frequency) and *linguistic* measures (based on semantic relations between words and their syntactic composition).

Another source of inspiration for this work is represented by **statistic**, corpus-based methods in Computational Linguistics; a significant example comes from The IBM models for Statistical **Machine Translation ?**, that first introduced the idea of feeding statistically intensive **Machine Learning** algorithms with big data from corpora, which nowadays is the dominant paradigm in MT; insightful is also the work on Data Oriented Parsing, and particularly the U-DOP model for **Unsupervised Language Learning ?**, which core idea is to initially assume all the possible syntax trees for a set of sentences as equally possible, and then use all the possible sub-trees of them to compute the most probable parse trees, letting the structure of the language emerge from the data.

The algorithm presented in this paper aims to provide a similarity measure for two sentences s_1 and s_2 featuring many of the ideas that have proven to be successful in recent developments of Computational Linguistics. The following are some of the key insights of the algorithm:

- **Every possible** sub-sentence of s_1 is compared with every possible sub-sentence of s_2.
- A score between two chunks of text is build **recursively** on the best matches of smaller chunks, and always brought down to scores between single words.
- **Dynamic programming** is used to increase the efficiency of the algorithm, preventing it from computing the same result two times.
- Other than the similarity score, the algorithm outputs the most probable chunking of each sentence in the form of an unlabeled binary **parse tree**.
- Other than the similarity score and the parse trees, the algorithm outputs the most likely **alignment** between the chunks of the two sentences.
- Every score is expressed as a linear combination of **features**, designed to be independent and extensible.
- The partial computations of the algorithm can be stored and further used to implement a learning model.

This paper is **structured** as follows. Section 3.2 describes the algorithm theoretically. Section 3.3 describes possible extensions and improvements. Section 3.4 draws the conclusion of the paper.

3.2 The algorithm

Algorithm 1: The main algorithm

input : c_1, c_2 chunks of text; T partial results table
output: $\sigma(c_1, c_2)$ Similarity score between c_1 e c_2

1 **if** *c1,c2 are words* **then**
2   **return** $\tau(c_1, c_2)$
3 $S \leftarrow [\,]$;
4 **for** $i \leftarrow 1$ **to** `Length`(c_1) **do**
5     $C_1 \leftarrow$ `Split`(c_1,i);
6   **for** $j \leftarrow 1$ **to** `Length`(c_2) **do**
7   **if** i=`Length`(c_1) *and* j=`Length`(c_2) **then** break;
8         $C_2 \leftarrow$ `Split`(c_2,j);
9   **foreach** sc_1 **in** C_1 **do**
10   **foreach** sc_2 **in** C_2 **do**
11   **if** $T[sc_1, sc_2] = \emptyset$ **then**
12                     $T[sc_1, sc_2] \leftarrow \sigma(sc_1, sc_2, T)$
13   **end**
14   **end**
15   `Append`(S,$\tilde{\sigma}(C_1, C_2, T)$)
16   **end**
17 **end**
18 **return** `Max`(R)

Algorithm 1 contains the main loop of the sentence scoring algorithm. Before entering the details of it, it is necessary to say that:

- The input c_1, c_2 can be any **chunks** of text because, while the algorithm is called with two sentences in input, it will recursively call itself on every possible chunk extracted from the two sentences.
- The input T (T for Table) is a data structure holding the intermediate results of the algorithm. As we can infer from line 12 of Algorithm 1, $T[c_1, c_2] = \sigma(c_1, c_2)$
- $\tau(c_1, c_2)$ returns a similarity score between two words. This score is a linear combination of independent features. At the moment, the following features are taken into account:
  - Equality (boolean) – 1 if $c_1 = c_2$, 0 otherwise
  - Edit distance – $1 - d$, where d is the Levelshtein distance Levenshtein [1966] between c_1 and c_2, as it is implemented in NLTK Loper and Bird [2002]

  - Difference in position – $(1/|p_{c_1} - p_{c_2}|)^\alpha$, where p_{c_i} is the position of c_i in the sentence, and α is a free parameter.
  - Wordnet Miller [1995] Path Length similarity, as it is implemented in NLTK

  The weights of the single features, now static, are meant to be trained through Machine Learning

- `Length`(c) returns the number of words contained in the chunk c
- `Split`(c, i) returns a list of two chunks, one containing the words of c from the first to the ith, and the other containing the words of c form the $(i + 1)$th to the last.

  If the ith word is the last word of c, it returns a list containing the only element c.
- `Append`(L, x) is the standard *append* operation for lists, adding the element x in the last position of the list L.
- $\hat{\sigma}(C_1, C_2, T)$ returns the score of c_1, c_2 **given** a particular 2-split of the two chunks. Such a score is computed combining the ones of the smaller constituents of c_1, c_2, which are required to be already present in the table T.

  The score is again the linear combination of independent features, leaving its implementation open and extensible.

  At the moment the only feature being employed is the average of the scores of the best alignments of the sub-chunks in C_1 with the ones in C_2.

  For example, given C_1 = [”increase”,”the volume”] and C_2 = [”raise”, ”the volume”], ”increase” is likely to achieve the best score with ”raise” (eg. T[”increase”,”raise”] = 0.8), as well as ”the volume” will be aligned with ”the volume” (T[”the volume”,”the volume”] = 1)). The score of ”increase the volume” and ”raise the volume”, in this particular splitting, will thus be the average of the two, 0.9.

After these premises it is relatively easy to walk through the pseudo-code of Algorithm 1. Lines 1 and 2 handle the base case, that is, when the algorithm is run on **two words**; in this case the result of the Word Similarity function, which is described above, is returned.

At line 3 the list of **candidate results** is initialized as an empty list. This is because more than one score will be computed for c_1 and c_2, the maximum of which will be returned as a result.

Lines 4 and 6 define a nested loop structure, which purpose is to update the two indexes i (from the first to the last word of c_1) and j (from the first to the last word of c_2) to cover **every possible combination** of word positions in the two input strings. These indexes are used to **split** the input chunks in smaller parts (lines 5 and 8). As an example, if the input is given by $c_1 =$"increase the volume", $c_2 =$"raise the volume", the outer loop will produce the three possible values of C_1: ["increase","the volume"], ["increase the","volume"] and ["increase the volume"]; each of these values will be combined, in the inner loop, with the three possible values of C_2: ["raise","the volume"], ["raise the","volume"] and ["raise the volume"].

Note that, while it is reasonable to consider the entire c_1 and c_2 in the comparisons (eg. if $c_1 =$"quit",$c_2 =$"quit please", we'd want the whole c_1 to be associated with the "quit" sub-part of c_2), it is necessary to prevent the two entire c_1 and c_2 to be associated at the same time, thus producing an **infinite loop**; this is done in line 7.

The next part of the inner loop, from line 9 to line 10 combines every sub-chunk of c_1 ($sc_1 \in C_1$) with every sub-chunk of c_2 ($sc_2 \in C_2$); note that C_1 and C_2 contain either one or two elements. Every combination that is not present in the table is scored with a **recursive** call to the algorithm itself, and the resulting score is saved in the table (line 12).

The last operation done in the loop, at line 15, is to combine the sub-scores that have just been scored in the table to compute the final score of this particular subdivision of c_1 and c_2. This is a **candidate score** for c_1 and c_2, and thus is appended to the list of candidate scores. As we already said, the definition of this combination is open; however it is most reasonable for the $\hat{\sigma}$ function to find the **best alignment** between the input sub-chunks, and provide a score based on it. It is clear that the definition of $\hat{\sigma}$ is crucial for getting sensible results from the algorithm. Furthermore, $\hat{\sigma}$ is the point where the algorithm can be extended with Machine Learning capabilities, that will be discussed in Section 3.3.

At line 18 the function returns the **maximum of the candidate** score, that is, the split that produced the best score (eg. for ”increase the volume” vs ”raise the volume”, the score of ”increase | the volume” vs ”raise | the volume” is likely to produce a better score than ”increase | the volume” vs. ”raise the | volume”). Note that this decision determines the branching of one level of the recursion tree; once the algorithm is done computing the score between two sentences, a traceback of the selected splits can be interpreted as a parse tree for the two input sentences.

3.3 Machine Learning Possibilities

The basic algorithm described in section 3.2 only deals with **static features** to measure similarities: the score of two words is derived from equality, edit distance, Wordnet path length and so on; the score of chunks is the average of the scores of the best alignments of their components.

In this section I address the question whether it is possible, and how, for the algorithm to **learn** from the sentences it is parsing. In other words, given a sentence of which I know the meaning, how can I use this information to improve the way I process new sentences?

Here I consider **three features** based on Statistical Language Processing methods to address this problem: chunk likelihood, class-conditional chunk weight and alignments likelihood.

3.3.1 Chunk likelihood

Let’s consider the following pair of sentences:

> Pump up the volume
> Turn up the volume

As an English-speaking human being I immediately associate the phrase ”pump up” with ”turn up” and ”the volume” with its analogue. One of the reasons I make this association rather than, for instance, associating ”pump” with ”turn” and ”up the volume” with its analogue, is that I have good experience of the phrase ”the volume” being used in different contexts, while fewer times I encounter ”up the volume”.

Thus a feature that can be useful to **enforce good chunking** of the input sentences is the likelihood of a chunk of text (where a chunk is defined as either a word or a combination of chunks). This feature can be modeled as the relative count of each chunk in the whole pool of the ones that have been recorded:

$$l(c) = \frac{\#\text{ of } c}{\#\text{ of total chunks}}$$

Note that this feature is completely unsupervised, in that it does not require a Meaning label to be computed.

3.3.2 Class-conditional chunk weight

Let’s consider the following three sentences:

Increase the volume
I would appreciate if you could increase a bit the the volume
Decrease the volume

With a reasonable approximation, the first two sentences have the same meaning, while the last says the opposite, although the first and the last sentence are much more similar from a syntactic point of view. Nevertheless, no English speaker would bother wondering about hidden meanings behind phrases like ”I would appreciate”, or ”if you could”: everyone could tell that the second sentence is no more than the first one, with some more formal dressing.

Especially, when I listen to the second sentence, I can immediately locate ”increase” and ”the volume” as the most informative phrases, while filtering out the rest as less relevant. One of the reasons I am able to do this is that I have memory of ”I would appreciate if you could” used in sentences conveying lots of different meanings, while ”increase” is likely to appear only in situations where something is increased.

A feature that could be useful to spot the **informativeness** of chunks can be the conditional probability of the chunk, given a certain meaning:

$$l(c|m) = \frac{\#\text{ of } c \text{ in } m}{\#\text{ of } c}$$

This feature can be used to weight the score of different chunks in the sentence, so for more informative chunks to influence more of the total score mass.

3.3.3 Alignment likelihood

Let's consider the following pair of sentences:

Skip this track
Jump over this song

Again, English speakers can easily associate the meaning of "skip" with the one of "jump over", and the meaning of "track" with the one of "song".

One device that the algorithm uses to solve this problem is the **Wordnet** Path Lenght measure for word comparison. However, it has been noticed that this measure has its limits. In particular

- Not all the words are included in Wordnet
- Being a static measure, it cannot be adapted to the domain
- It has difficulties spotting antonyms

The alignment likelihood measure can be used to **enforce correct alignments** of different chunks.

$$l(c_2|c_1) = \frac{\#\text{ of } c_1 - c_2 \text{ alignments}}{\#\text{ of } c_1 \text{ alignments}}$$

Note that, since no alignment labels are provided in the dataset (each sentence is labeled with a meaning, but its syntactic structure is hidden), this measure have to be trained in an unsupervised fashion. This problem shares a lot with the alignment likelihood in **Machine Translation**, even though here we can take advantage of the chunks sharing the same language (thus allowing some clever initialization of unsupervised methods with sensible guesses based, for instance, on Wordnet). On the other side our problem presents less precise training data, in that, while MT alignments are trained on sentence pairs, here we have a group of sentences sharing a same meaning; it can be argued that this cannot allow for strong syntactic assumptions.

3.4 conclusions

Here I described an algorithm to compare the similarity of two input sentences. Its main **advantages** are the concurrent consideration of many features like word meanings, position and overlapping, as well as the structure of the sentences, having the result in the form of aligned binary trees. The algorithm is recursive and exhaustive in that considers all the possible alignments of all the possible combinations of all the possible binary trees of the two input sentences, maximizing the efficiency with dynamic programming. Due to its modular structure, the algorithm can be easily extended with new features, allowing its integration with well-known Machine Learning techniques in Computational Linguistics (eg. Expectation-Maximization for computing word/phrase alignment probabilities).

There are some assumptions and **weak points** as well. First of all, the algorithm is inherently binary, in that every recursion step is made of a 2-split of the two input strings; this may have an effect in the way partial scores are combined. More generally, this ”two times binary” recursive structure makes the features design task hard, since the effect of one operation is not easy to control over a potentially unlimited number of recursion steps. Also the number of parameters can become high, since each score is a linear combination of single features, requiring a separate Machine Learning training.

Lastly, this paper only describes the **backbone** of the algorithm, which is not yet ready for more formal tests on existing corpora. The main steps ahead before expecting good results from it include a better tuning and possible extension of the existing features and the implementation of Machine Learning techniques to include alignment probabilities and more advanced statistical features for the chunks being compared (eg. their degree of informativeness).

Appendix A

Appendix Title Here

Write your Appendix content here.

Bibliography

Joseph Weizenbaum. Eliza, a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, January 1966. ISSN 0001-0782. doi: 10.1145/365153.365168. URL `http://doi.acm.org/10.1145/365153.365168`.

Kristiina Jokinen and Michael F. McTear. *Spoken Dialogue Systems*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2009.

Stephanie Seneff and Joseph Polifroni. Dialogue management in the mercury flight reservation system. In *Proceedings of the ANLP-NAACL 2000 Workshop on Conversational Systems*, ConversationalSys '00, pages 11–16, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1605285.1605288`.

M Gabsdil. Clarification in spoken dialogue systems. In *In AAAI*, 2003.

Raquel Fernández, Staffan Larsson, Robin Cooper, Jonathan Ginzburg, and David Schlangen. Reciprocal learning via dialogue interaction: Challenges and prospects. Proceedings of the IJCAI 2011 Workshop on Agents Learning Interactively from Human Teachers (ALIHT 2011), 2011.

Palakorn Achananuparp, Xiaohua Hu, and Xiajiong Shen. The evaluation of sentence similarity measures. In *Proceedings of the 10th international conference on Data Warehousing and Knowledge Discovery*, DaWaK '08, pages 305–316, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85835-5. doi: 10.1007/978-3-540-85836-2_29. URL `http://dx.doi.org/10.1007/978-3-540-85836-2_29`.

Satanjeev Banerjee and Ted Pedersen. Extended gloss overlaps as a measure of semantic relatedness. In *In Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 805–810, 2003.

Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.

Edward Loper and Steven Bird. Nltk: the natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1118108.1118117. URL `http://dx.doi.org/10.3115/1118108.1118117`.

George A. Miller. Wordnet: a lexical database for english. *Commun. ACM*, 38(11): 39–41, November 1995. ISSN 0001-0782. doi: 10.1145/219717.219748. URL `http://doi.acm.org/10.1145/219717.219748`.