Chatbots Enhancing with Word Embeddings

Semester Project Thesis

By

ZAKARIA EL KHAMAR
LIONEL SOMÉ

Supervised by:

RAPHAËL TRONCY
THIBAULT EHRHART

A Report submitted to the D2K Lab department at Eurecom in accordance with the requirements of the final semester project

FEBRUARY 2019

Abstract

As part of its research activity in the domain of NLU, the D2KLab team at Eurecom has developed several virtual assistants (Chatbots). One of those chatbots is called "Minotour". It is specialized in the domain of tourism and has the vocation to give recommendations as accurate as possible to tourists that are looking for events, places, activities and so on. This virtual assistant is available on popular messaging platforms (Slack, Facebook Messenger, Skype, Telegram) and on smart speakers (Alexa, Google Home). Moreover, it is trained in English and in French using off-the-shelves Natural Language Understanding (NLU) frameworks such as Google, DialogFlow, Facebook WIT.AI, Microsoft LUIS, or Amazon LEX.

The performance of a NLU engine depends strongly on its capacity to recognize entities. The component responsible for this feature in Minotour is called FADE (Filling Automatically Dialog Events). It is a tool that extracts information from knowledge graphs, applies permutation rules, and builds dictionary of entities that must be recognized by a NLU engine.

The main objective of this project is to figure out new techniques that can be used to automatically enhance this dictionaries. Another objective is to run training experiments of the same bot (Minotour) on RASA, which is an unsupported tool, in order to give an insight into the performance gain a potential migration can provide.

Firstly,we led a research work in order to elaborate a comparative analysis of the different variants of word embedding algorithms on the one hand, and of different NLU engines on the other hand. Secondly, we drew a comparison between Chatito and FADE workflow. And thirdly, we run experiments of Minotour bot training in RASA in order to compare its performance on it to the training in supported frameworks (mainly DialogFlow). And finally, we explored algorithmic ways in which word embeddings can be used to improve the overall new data entries collection and classification process.

TABLE OF CONTENTS

Chapter 1

Introduction

In a world where everything is being automatized and where the volume and the velocity of the data are achieving unprecedented levels, applications and algorithmic methods that aim to extract insight and knowledge from these information are gaining ground. Natural language processing (NLP) is one of these fields concerned with the interactions between machines and human languages, commonly referred to as natural languages. A subtopic of this latter field that deals specifically with machine reading comprehension is called Natural-language understanding (NLU). There is a very big commercial interest in the topics related to this domain because of its several applications: news-gathering, text categorization, large scale content analysis, machine translation, automated reasoning, and question answering. However, It is considered as an AI-hard problem. [19]
When a NLU is used for virtual assistance, one of its most important challenges is entity & intents (cf.section 3.1) recognition. In fact, a virtual assistant has the capability to gather much more information related to a specific query (if the sources are provided of course) than what the human assistant can provide with its limited knowledge, yet it needs to understand the query first in order to be able to take profit of the way it aggregates data from multiple sources.
This project aims at exploring general techniques whose results can be in a way or another used to improve Minotour entity recognition and/or classification. In what follows, we present the following sections:

- State-of-the Art of word embedding techniques and NLU engines
- Comparative analysis between Chatito and Fade
- Minotour training comparison on RASA and dialogflow
- Words embeddings for unsupervised clustering.
- Documents embeddings for similarities detection and classification

Chapter 2

State-of-the-art of Word Embeddings Algorithms and NLU Engines

2.1 Word Embeddings Algorithms

Word embedding is one of the most common representation of a text-dataset vocabulary. It refers to a set of language modeling and feature learning techniques where each element of the vocabulary (word or phrase) is mapped to a specific vector of real numbers. The figure 2.1 illustrates those vectors. [20]

There are many methods to generate such a vectors, such as neural networks [G2], co-occurrences matrix, lexicon-based or probabilistic models, etc. But in the following sections and for the sake of clarity and coherence, we will only focus on neural networks based techniques.

2.1.1 Introductive Examples

In what follows, we will explain the need for such a mapping when dealing with texts. Consider the two following sentences: "I wish you a good luck", "I wish you a great luck". They have more or less the same meaning. If a basic data set contains only those two sentences, its corresponding vocabulary will be {I,wish,you,a,good,great,luck}.

If we want to map each word of this vocabulary to a different vector, one of the most intuitive Ideas is to do a one-hot encoding [G1]. So we'll end up having the following vector representation:

I = [1,0,0,0,0,0] ; wish=[0,1,0,0,0,0] ; you=[0,0,1,0,0,0] ; good=[0,0,0,1,0,0] ; great=[0,0,0,0,1,0] ; luck = [0,0,0,0,0,1]

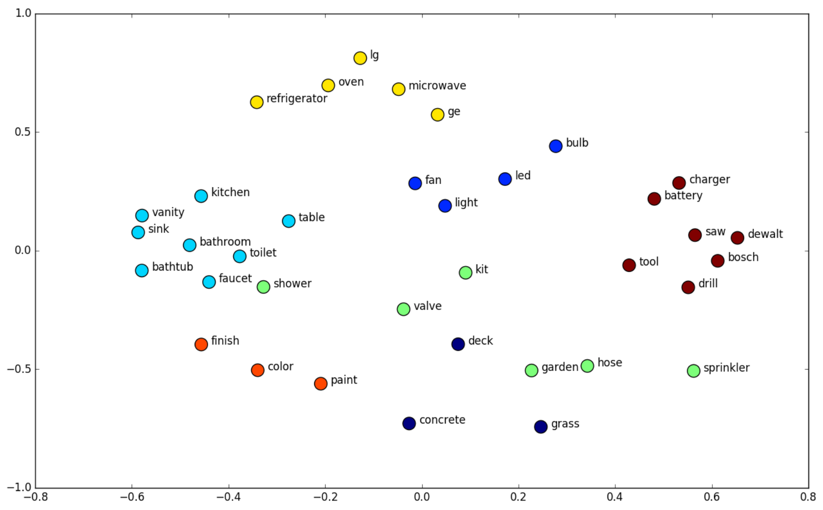


FIGURE 2.1. Example of words in the vector space.[10]

One of the difficulties of such a mapping is that it doesn't discriminate between words based on meaning, context, etc. For instance, in the previous example, "Great" and "Good" are just as different as "You" and "Wish", which is not accurate.

So the objective is to end up with a word representation with which similar words (i.e that have similar context or meaning) are close to each other from a spatial point of view. Thus, we need to define a spatial measure of similarity. The most common measure is the cosine of the angle between the two vectors in question. The more the angle is close to 0 (i.e vectors tend to be aligned), the more the cosine is close to 1, which means the more the words are similar. [8]

Now that we understand the motivations, we will take an in-depth look at some of the methods that are used to initialize those vectors, and to improve them progressively throw a training process in order to make them describe as much similarities as possible and in the most accurate way.

2.1.2 Word2vec algorithm

Word2vec algorithm was originally introduced in this paper [12]. It was a novel technique that outperformed all existing solutions meant to build word representations based on neural networks. In fact, the most common limitation that those techniques were facing was that they were hardly trained on huge data sets, because of the big complexity of the models used. So the main purpose of word2vec is to provide efficient word representations while minimizing the computational complexity, at the cost of decreasing the depth of the neural networks used.

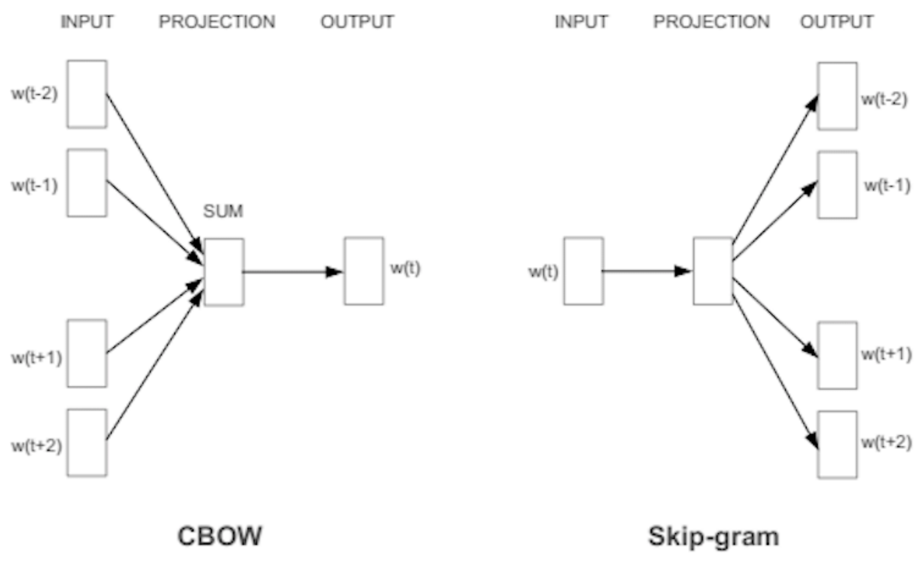


FIGURE 2.2. Architecture of Word2Vec models: CBOW and Skip-Gram[7]

There are two main different variants of Word2vec: CBOW and Skipgram. We will present their training processes separately in the next subsections. A general architecture is shown in figure 2.2

2.1.2.1 Continuous Bag of Words: CBOW

The CBOW model is based on a simple neural network that is trained to solve the task of mapping the context of a word to the word in question. For a given word, the context is defined as the n surrounding words in the text, n being a parameter that the user can tune. In other words, if we take the sentence "I wish you a good luck" from our initial example, the context of window size 4 of the word "you" is ['I','wish','a','good']. So the neural network will take the context as an input, and should learn to predict the word ['you'].

The architecture of the neural network is composed of: an input layer, a Hidden layer, referred to as the projection layer, and an output layer. The figure 2.3 gives a simple drawing of this architecture along with the different data structures involved. While the model trains the weights matrices to perform the mapping task, the embedding vector is created in the output of the projection layer, and the more the model is well trained, the more this vector is accurate.

It is important to note that we don't explicitly provide the model with (data point, target) couples, that's why the word2vec models are said to be self-supervised.

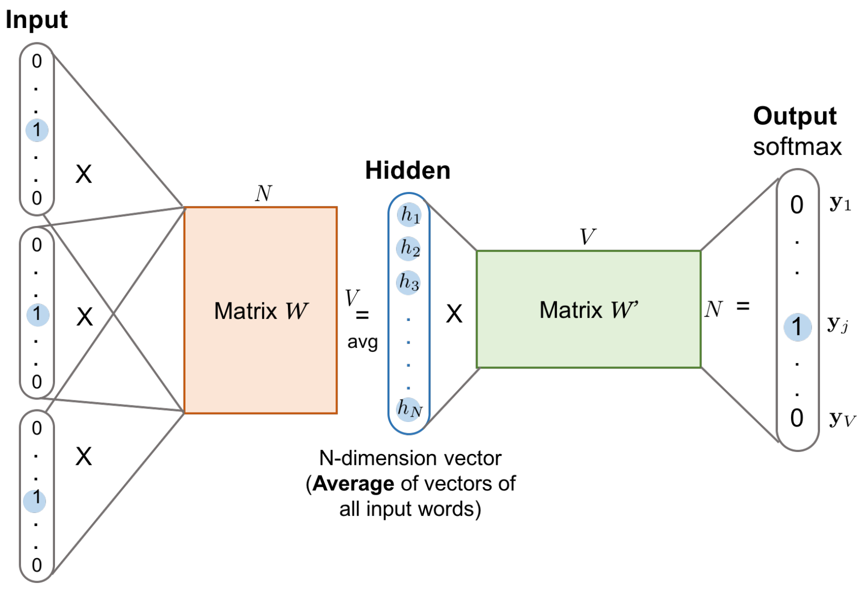


FIGURE 2.3. The CBOW model. Word vectors of multiple context words are averaged to get a fixed-length vector as in the hidden layer.[18]

2.1.2.2 Skip-gram

The second variant of word2vec is skip-gram. It has more or less the same architecture as CBOW, with the main difference that the neural network in skip-gram model learns to map the word to its context. As shown in the figure 2.4, to each word in the input will correspond several words in the output, which are those forming its context. From a computational point of view, each word is seen as a one-hot encoding, so the output will be a vector of 0s and 1s, each position j in which there is a 1 corresponds to a word in the context.(see figure 2.5)

The objective function to be maximized in the skipgram model is the average log probability:

$$\frac{1}{T} * \sum_{t=1}^{T} \sum_{j \in [-c,c], j \neq 0} \log P(w_{t+j} \mid w_t)$$

where T being the length of a sequence of training words, [-c,c] denoting the index interval of the context of word w_t, and the conditional probability defined as follows:

$$P(w_o \mid w_i) = \frac{\exp({v'_{w_o}}^\top v_{w_i})}{\sum_{w=1}^{W} \exp({v'_w}^\top v_{w_i})}$$

where v_w and v'_w are the “input” and “output” vector representations of a word w, and W is the number of words in the vocabulary (Note that the denominator is computationally expensive).

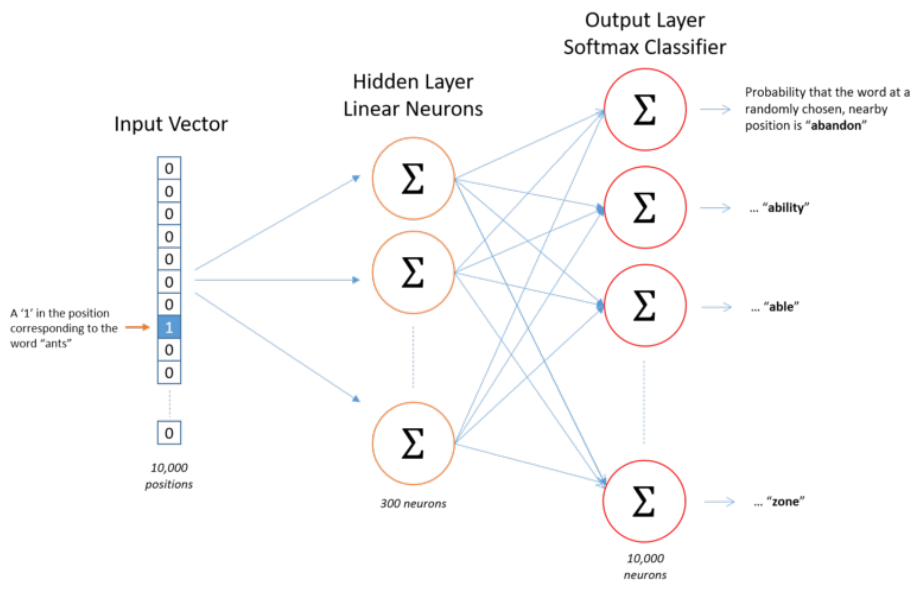


FIGURE 2.4. The general skip-gram model architecture[2]

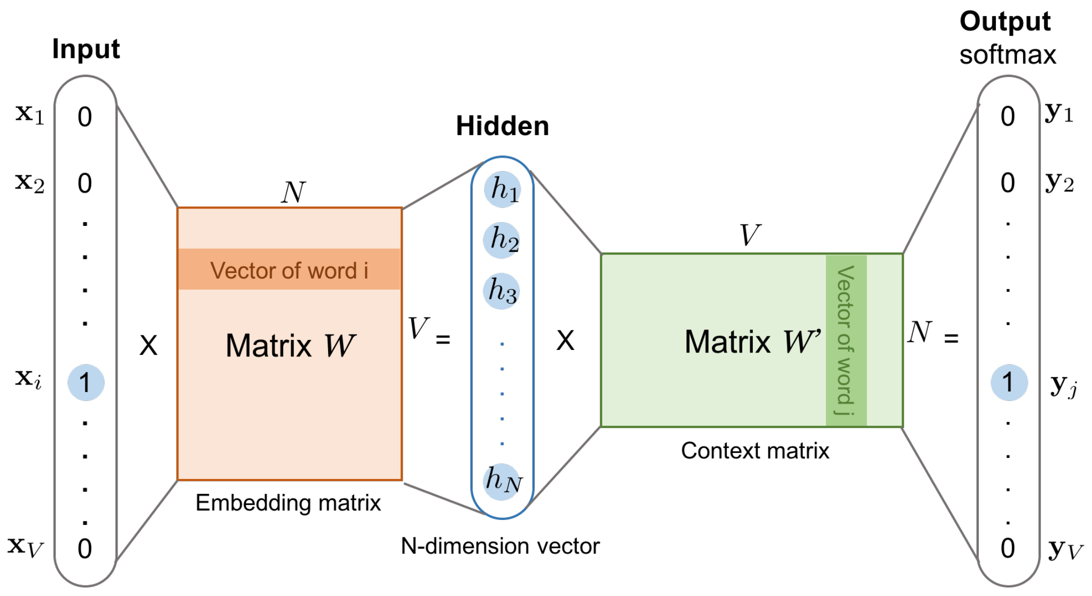


FIGURE 2.5. The Skip-gram model with computational details[18]

2.1.3 Negative Sampling

The Negative sampling technique is an alternative to word2vec algorithm having the following three main innovations: [3]

- Treating Common word pairs or phrases as single words in the model.

- Subsampling frequent words to decrease the number of training examples.
- Modifying the optimization objective in a way that each training sample updates only a small percentage of the model's weights.

The negative sampling objective function is the following:

$$\log\sigma({v'_{W_0}}^{\top} v_{W_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)}[\log\sigma(-{v'_{W_i}}^{\top} v_{w_I})]$$

with $P(w_i)$ the probability of selecting a word, defined as:

$$P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=0}^{n}(f(w_j)^{\frac{3}{4}})}$$

Where f the frequency of the word w_i (word count).

2.1.4 Glove Algorithm

Glove algorithm aims at optimizing the Word2vec algorithm [13]. In fact, Word2vec doesn't fully take advantage of statistical information related to co-occurrences. So Instead, Global vectors (Glove) combines the benefits of the word2vec skip-gram model when it comes to word analogy tasks, with the benefits of matrix factorization methods [G3] that can exploit global statistical information.

Mathematically speaking, Glove defines the Co-occurrences Matrix $(X_{ij})_{1in,1jn}$, where each X_{ij} is the number of times word i appears in the context of word j. Then, the objective (or cost) function is adapted accordingly:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^{\top}\widetilde{w}_j + b_i + \widetilde{b}_j - \log X_{ij})$$

where f is a function defined by:

$$f_{x_{max},\alpha}(x) = \begin{cases} (x/x_{max})^{\alpha} & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases}$$

And:

- wi is word vector i.

- wj is context vector j.
- bi and bj additional biases of wi and wj.

2.1.5 FastText Algorithm

Another interesting algorithm of word embeddings that was initially developped by facebook research team is fastText [1]. Unlike all previous algorithms, fastText makes use of the morphology of words instead of assigning a distinct vector to each word.

In fact, It is based on the ingenious idea of making the model learns vectors for each character n-gram instead of directly learning vectors for words.

Each word is then represented as a bag of character n-grams. And the corresponding vector will be the sum of the vectors of its n-grams.

So if we take as example the word w="where" with n=3, the corresponding n-grams bag will be : <wh,her,ere,re>

The objective function of FastText is defined as follows:

$$\sum_{t=1}^{T}\Big[\sum_{c\in C_t} l(s(w_t,wc)) + \sum_{n\in N_{t,c}} l(-s(w_t,n))\Big]$$

Where s the scoring function defined as:

$$s(w,c) = \sum_{g\in G_w} z_g^{\mathsf{T}} v_c$$

And l the logistic loss function:

$$l : x \mapsto \log(1+\exp(-x))$$

C_t the context of word w_t, $N_{t,c}$ a set of negative examples sampled from the vocabulary, G_w the set of n-grams of word w, z_g the vector representation of n-gram g, and v_c the context vector.

Since in the cost function we only multiply the n-grams vectors of the word in question by the context vector, this speeds up the computations. Thus, FastText is a fast algorithm, allowing to train models quickly on very large corpora. In addition, an other main advantage of FastText is the fact that it allows computing vector representations for words that doesn't exist in the training data.

To sum up, here are the main characteristics of each algorithm:

- **Skip-gram:** Works well on small datasets, Represents well rare words, Computationally expensive.
- **CBOW:** Faster than Skip-gram, Good Accuracy for frequent words.
- **Negative Sampling:** Faster than word2vec, Subsampling frequent words.
- **Glove:** Takes into account statistics of word co-occurrences in the corpus.
- **FastText:** Doesn't require a lot of data, Allow constructing vectors for non existing words in the training data.

2.2 NLU Engines

2.2.1 DialogFlow NLU

Dialogflow [5] provides a powerful natural language understanding (NLU) engine to process and understand language input. This allows building conversational interfaces on top of the clients products and services.

Unlike traditional interfaces that require structured input to work properly, Dialogflow makes the interface easier and more natural for users as it can adapt to multiple language variants and queries order. Hence, users don't have to figure out the structure of the input to interact correctly with the engine.

Let's take an example based on our Chatbot Minotour. The agent is supposed to answer queries like:

"What can I do in Nice Today?"

But the same question, or slightly different ones can be asked in a complete linguistically different manners, for instance:

"Any Interesting activities in Nice Today ?"

"What can a visitor do in Nice ?"

"Activities nearby ?"

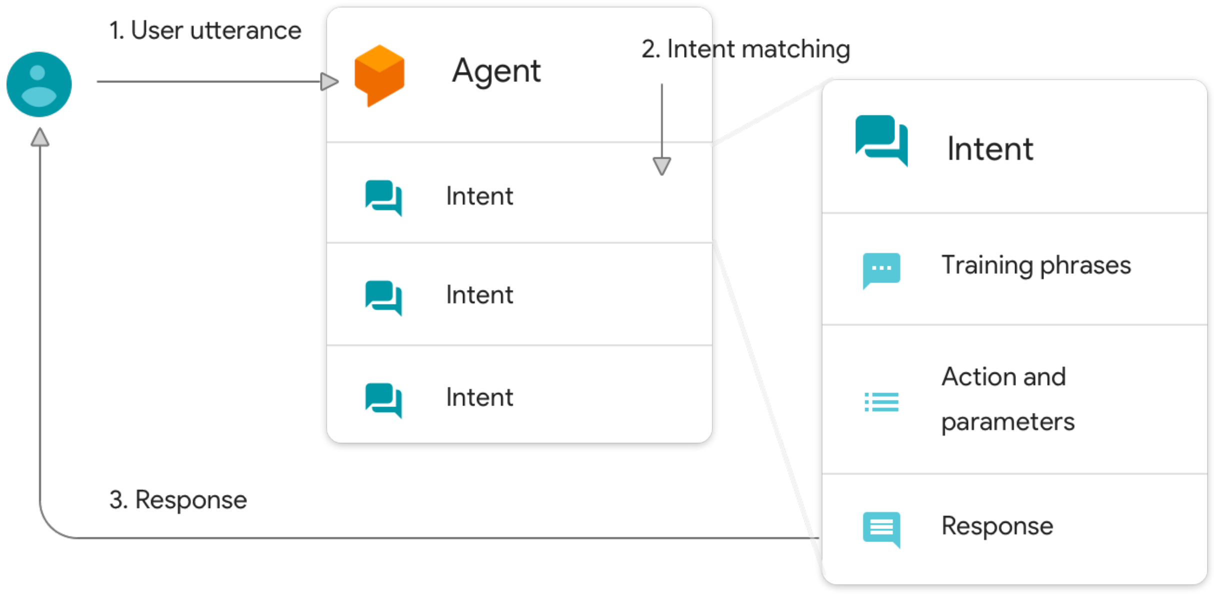


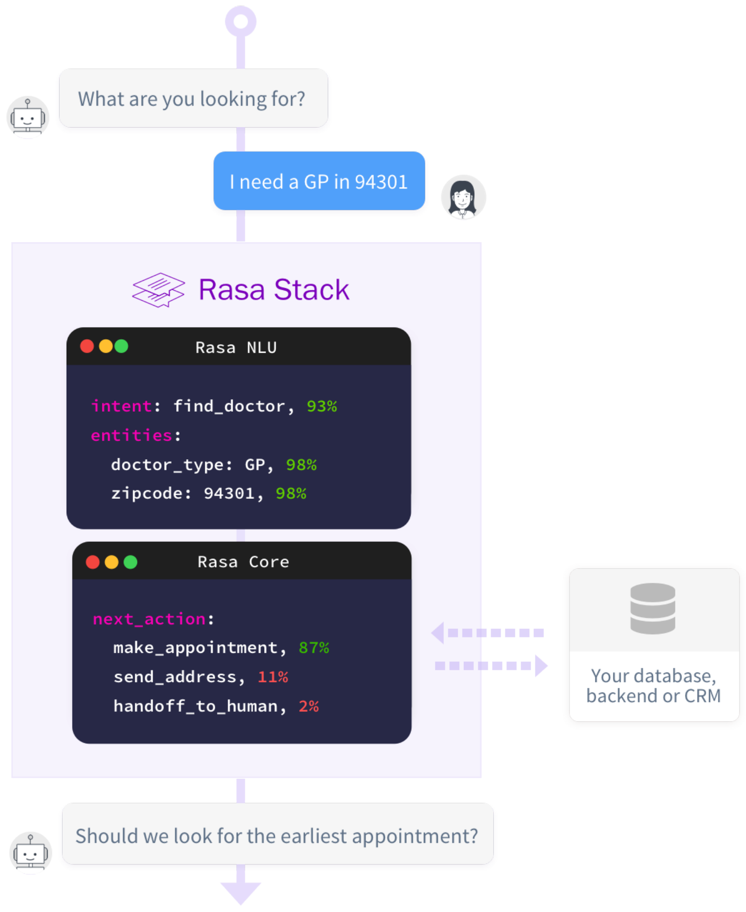
FIGURE 2.6. Example of how Dialogflow handles a user utterance[5]

So it's evident that even such a simple question is difficult to implement from a conversational point of view. Thus, traditional computer will need to force the users to follow a well-know pre-defined language structure for their queries, which deteriorates user experience.

However, Dialogflow lets the service provider easily achieve a good conversational user experience by handling internally all the natural language understanding part. In other words, the agents created will be able to understand a wide range of nuances of human language and map them to standard and structured units that apps and services can understand.

Let's take a look at how Dialogflow might have handled the previous examples for activities in Nice. The figure 2.6 shows an example of how Dialogflow handles a user utterance.

To look up an activity, you need more pieces of information, such as date and location. And even though this information can be provided by users in different order and with various expressions, Dialogflow can deal with these differences as mentioned previously. Once understood, the queries will be translated into intents (cf.chapter 3) in order to get the required response.Technically speaking, It parses the query to get only the key information needed for the action that it's supposed to accomplish. In our case, these information are "activities", "Today", and "Nice".Then, it uses this information to look up activities available today in Nice using the REST API [G4]. And finally, the answer to the query (e.g: Concert today in Nice at 6 PM) is returned to the user, in the form of a response.


What are you looking for?
I need a GP in 94301
Rasa Stack
Rasa NLU
Rasa Core
Your database, backend or CRM
Should we look for the earliest appointment?

```
intent: find_doctor, 93%
entities:
  doctor_type: GP, 98%
  zipcode: 94301, 98%
```

```
next_action:
  make_appointment, 87%
  send_address, 11%
  handoff_to_human, 2%
```

FIGURE 2.7. Example of how RASA handles a user utterance[16]

2.2.2 RASA NLU

RASA engine [16] aims at providing developers with a service similar to what Dialogflow offers. However, while Dialogflow encloses all the natural language understanding part in a black-box to which the developer don't have access, RASA offers a set of open source machine learning tools for developers to create their own contextual AI assistants. It has two main components:

- **RASA Core:** A Chatbot framework with machine learning-based dialogue management
- **RASA NLU:** A library for natural language understanding, based on intent classification and entity recognition & extraction

The two previous components are independent. The developer can either use only one of them or both of them, but it is recommended to use both. The figure 2.7 presents the high-level functioning of RASA NLU, to answer a query.

Technically speaking, RASA processing happens in two main steps:

- **First step:** the NLU component starts by understanding the message of the client based on the training data it has been provided with. The understanding process is composed of two sub-steps:
  - **Intent Classification:** The meaning of the message is interpreted based on predefined intents (figure 2.7 example: find_doctor).
  - **Entity Extraction:** Once a specific intent detected, the next NLU's mission is recognizing structured data (figure 2.7 example: GP is a doctor_type, 94301 is a zipcode).
- **Second Step:** The Core component decides what happens next in the conversation. It's based on machine learning algorithms for dialogue management that predicts the next best action, taking into consideration the NLU component's output, the conversation history and the training data (figure 2.7 example: next_action is make_appointment)

Note that more details about Intents & entities supported by concrete examples are provided in the next chapter(cf.chapter 3).

The figure 2.8 provides a more detailed diagram of the RASA utterance handling process. In the diagram, we can see a third component called RASA NLG (Natural Language Generation) in the user side. Its role is to fill the human-machine communicative gaps. The technology accepts as an input a non-linguistic format and transforms it into one of the human understandable formats such as reports, documents, and text messages.

To sum up, the table 2.1 presents the main characteristics of RASA and Dialogflow.

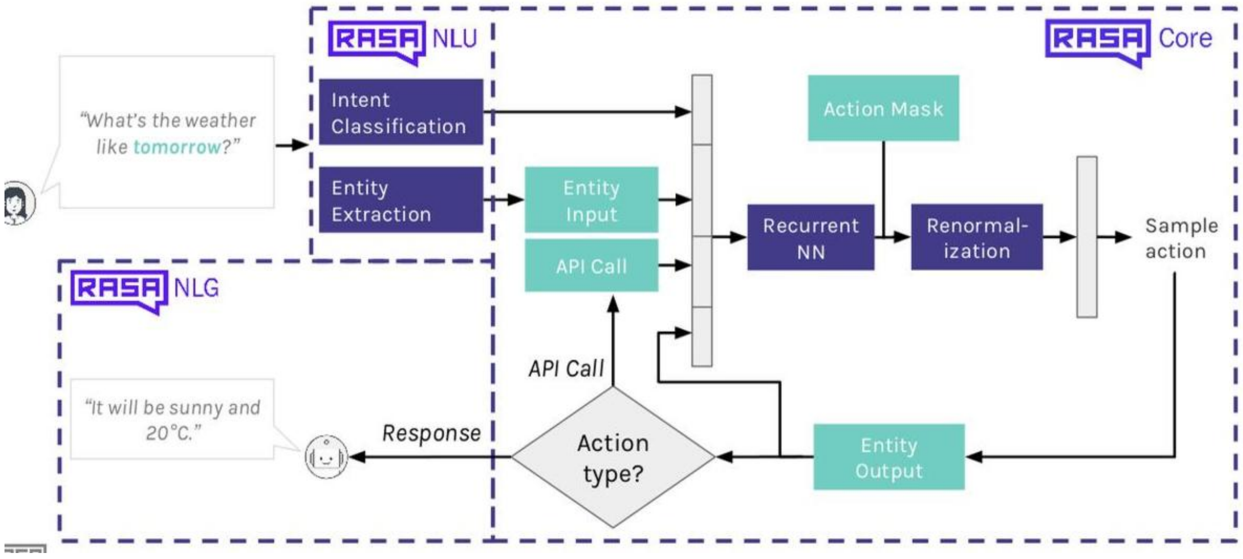


FIGURE 2.8. Detailed Example of how RASA handles a user utterance[9]

NLU engine	**Main Characteristics**
Dialogflow	Easy to Understand, can be used by non-techies Out of box integration with Google Assistant, Skype, Slack, Messenger, ... Web-based interface for building bots Data is hosted only on the cloud, can't be hosted on specific servers or on-premise Closed System
RASA	Open-source, code available in Github Can be hosted on a specific servec No interface provided, needs to write JSON or markdown files No hosting possibility Requires installation of multiple components Requires tech knowledge

Table 2.1: Dialogflow vs RASA [11]

CHAPTER 3

CHATITO & FADE

Before presenting some tools and mechanisms to generate training data files that would be used by the NLU engines (cf.section 2.2), It is primordial to define the notions of Intent and entities.

An **intent** is the intention or the purpose of the user interacting with the Chatbot. In the context of Minotour and tourism assistance, an intent can be "find_activity", "find_places", "find_events", etc.

An **entity** is a term or an object in the input message that provides additional information or specific clarification related to a particular intent. For instance, "find_activity" can have as an entity "Date_time", "Location", etc.

3.1 FADE

FADE (Filling Automatically Dialog Events)[17] is a tool that extracts information from knowledge graphs, apply permutation rules and build dictionary of entities that must be recognized by a NLU engine. It has been designed specifically for D2KLab usage. Somehow, it plays the same role as Chatito (cf.section 3.2) by generating training and test sets that will be used for training models on NLU engines, however FADE offers more functionalities, especially by supporting many engines such as Dialogflow, Wit.ai, and Amazon Lex. FADE retrieves the list of entries related to a chosen category from a knowledge base, and uploads them to the chosen platform. In addition, the knowledge base to use can also be specified by the user, otherwise a default one is provided.

FADE mechanism to retrieve data and upload it to a specific platform can be divided into three

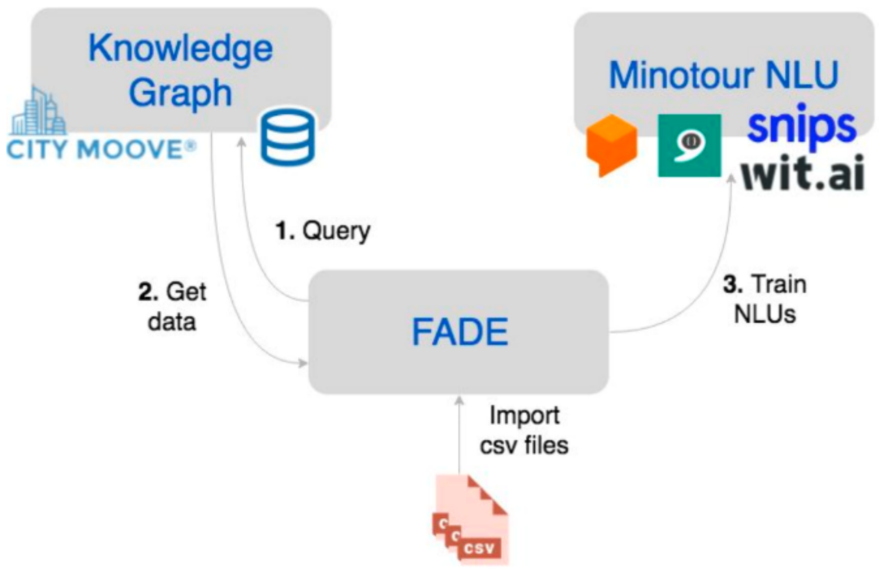


FIGURE 3.1. FADE Architecture [4]

main phases:

- Retrieving the list of entries corresponding to a chosen category from the knowledge base.
- Clean the data and format it in order to match the requirements of the platform in question.
- Upload the data files to the chosen platform via its specific API.

FADE uses the following configuration for intent & entities classification:

```
{"name": "find.activity",
"languageCode": ["en", "fr"],
"settings": {
    "dialogflow": {
        "id": "29d1e8ba-82a0-423b-a384-901d4549cc79"}},
"entities": [{
    "name": "@city"
    }, {
    "name": "@activity"
    }, {
    "name": "@date-time",
        "mapping": {
            "dialogflow": {
                "name": "datetime",
                "type": "@sys.date-time"}}}]}
```

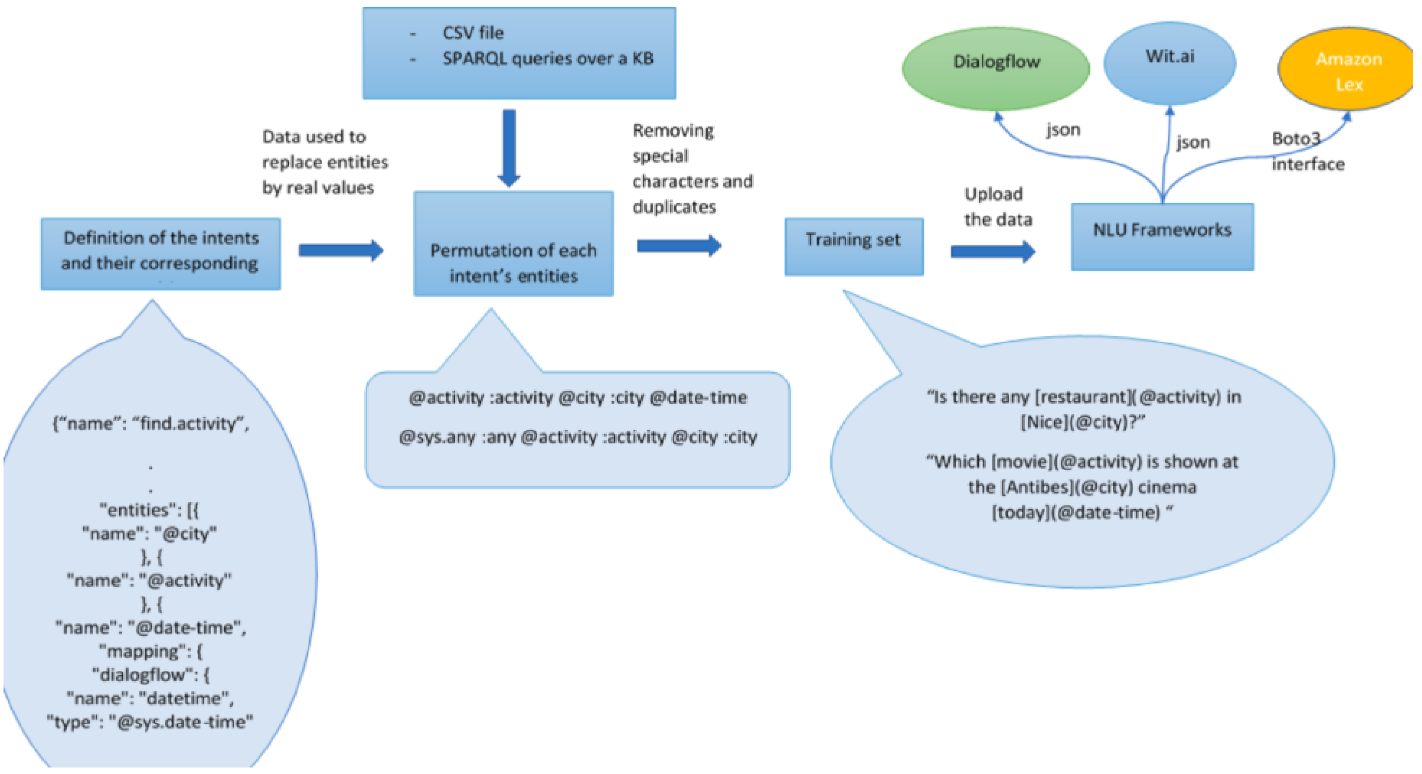

- CSV file
- SPARQL queries over a KB
Dialogflow
Wit.ai
Amazon Lex
Data used to replace entities by real values
Removing special characters and duplicates
json
json
Boto3 interface
Upload the data
Definition of the intents and their corresponding
Permutation of each intent's entities
Training set
NLU Frameworks
{"name": "find.activity",
"entities": [{ "name": "@city" }, { "name": "@activity" }, { "name": "@date-time", "mapping": { "dialogflow": { "name": "datetime", "type": "@sys.date-time"
@activity :activity @city :city @date-time
@sys.any :any @activity :activity @city :city
"Is there any [restaurant](@activity) in [Nice](@city)?"
"Which [movie](@activity) is shown at the [Antibes](@city) cinema [today](@date-time) "

FIGURE 3.2. FADE Process

The figure 3.1 represents FADE architecture, and figure 3.2 details the process followed by FADE starting from data retrieving until uploading training sets to the platforms.

3.2 Chatito

Chatito [14] is a domain specific language designed to simplify the process of creating, extending and maintaining datasets for training natural language processing models. These latter can be dedicated to various tasks: text classification, named entity recognition, slot filling, etc. the figure 3.3 represents a simple diagram of Chatito mechanism.

Chatito was designed around three key principles:

- **Simplicity:** Anyone can use it and understand it with minimum effort.
- **Speed:** Generate samples by pulling them from a cloud on demand
- **Practicality:** The design of Chatito aims to meet the community needs, as the main objective is helping people building their Chatbots.

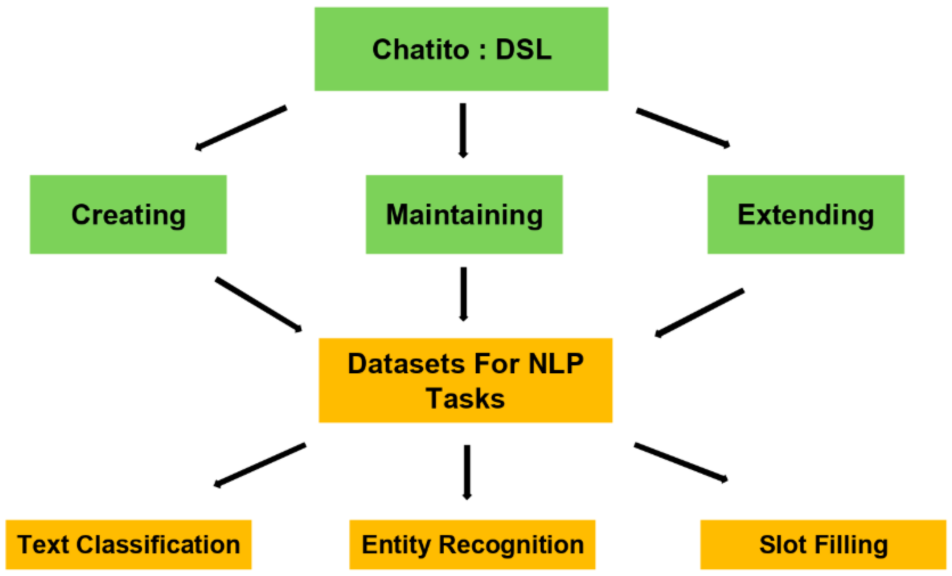


FIGURE 3.3. Chatito tasks and applications

Chatito has a different terminology for "Entity" definition. In Chatito language, entities are all types of objects dedicated to defining keywords that wrap sentence variations and attach some properties to them. There are three main types of entities: Intent, Slot, and Alias.

- **Intent:** the intent entity is defined by %[symbols at the start of a line, followed by the intent name and] symbol. The intent is the classification tag that will be added to all the sentences generated inside it. For example, the intent 'greet' can be defined as follows:

  ```
  %[greet]('training': '2', 'testing': '1')
            hello
            hi
            salut
            hola
            hallo
  ```

  The previous example can generate a maximum of 5 examples. However, the entry only requests 3. The training dataset will contain 2 utterances for greet intent and the testing dataset only 1.

- **Slot:** The slot entity is defined by the @[symbols at the start of a line, followed by the name of the slot and] symbol. The slot is the tag that will be added to the relevant words in a generated sentence. Slots referenced within sentences can have ? symbol (e.g: @[name?]). It this case, the ? symbol means that adding a combination from the slot name to the sentence is optional and could be omitted at generation. In addition, slots have the variations property. These variations are defined by # symbol after the slot name. For example:

```
%[ask_for_delivery]
    my parcel should be delivered in @[delivery_time#time_in_hours]
    my parcel should be delivered @[delivery_time#relative_time]

@[delivery_time#time_in_hours]
    3 days
    5 hours

@[delivery_time#relative_time]
    as fast as possible
    quickly
```

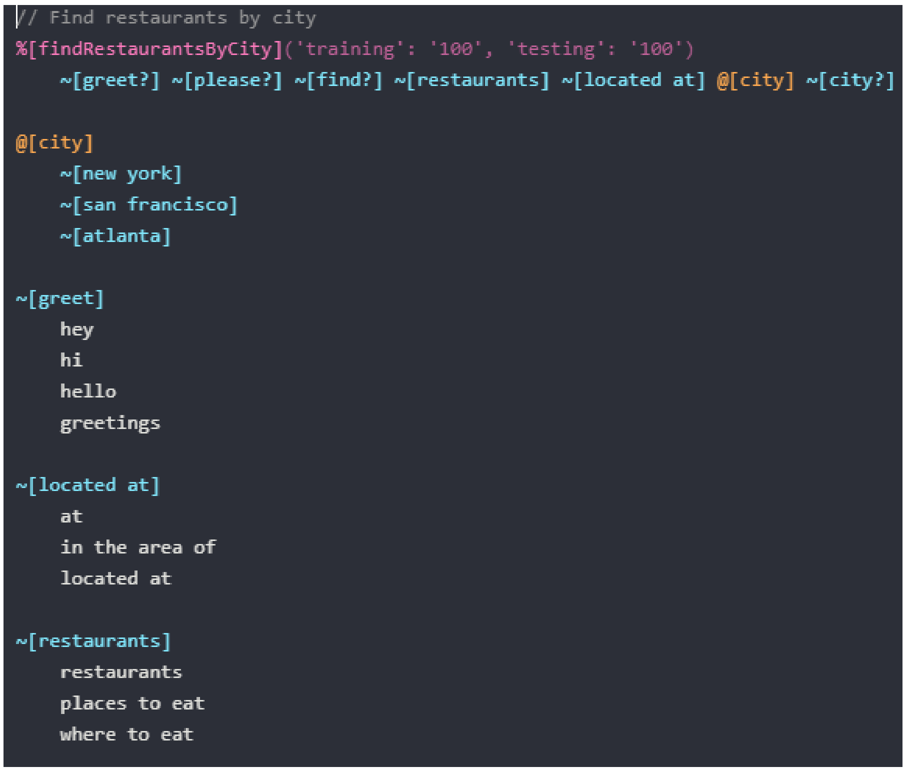
In this example, time_in_hours and relative_time are variations of the slot delivery_time

- **Alias:**The alias entity is defined by the ˜ [symbols at the start of a line, followed by the name of the alias and] symbol. Alias represents just variations of a specific word, and does not generate any tag. If an alias is referenced in the defining structure sentence but not defined (e.g: "how are you" in the next example), by default it uses the alias key name. the symbol "?" has the same role as in the slot entities.

```
%[greet]
    ~[hi] ~[how are you?]

~[hi]
    hi
    hey
```

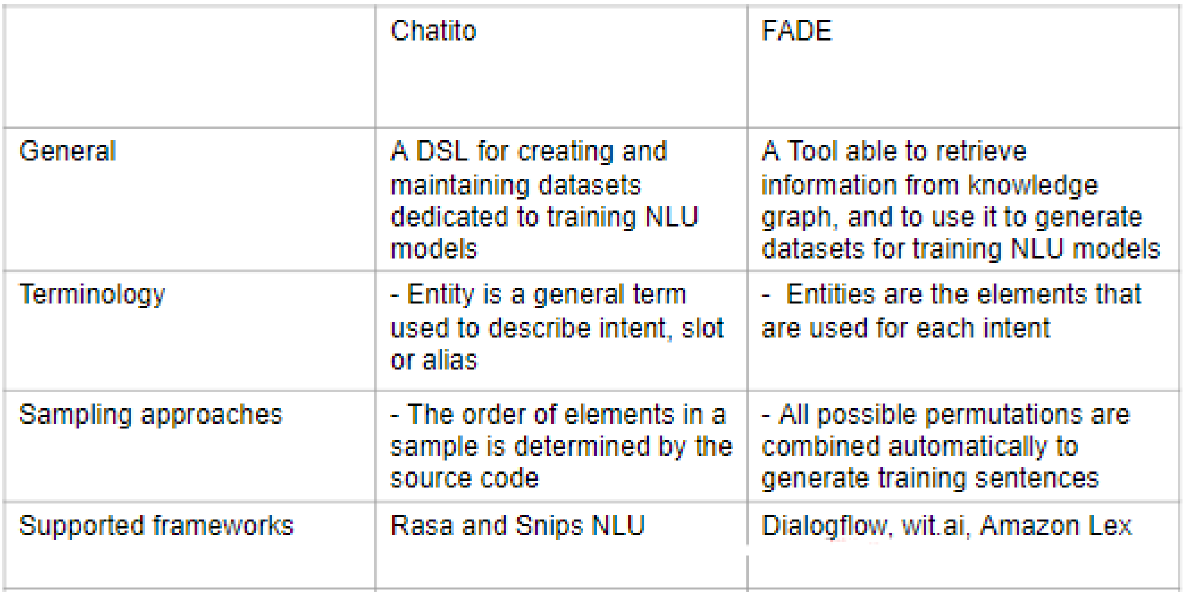
The figure 3.4 represents the screen shot of a real Chatito file.

To sum up, the Table 3.5 compares Chatito and FADE characteristics.



```
// Find restaurants by city
%[findRestaurantsByCity]('training': '100', 'testing': '100')
    ~[greet?] ~[please?] ~[find?] ~[restaurants] ~[located at] @[city] ~[city?]

@[city]
    ~[new york]
    ~[san francisco]
    ~[atlanta]

~[greet]
    hey
    hi
    hello
    greetings

~[located at]
    at
    in the area of
    located at

~[restaurants]
    restaurants
    places to eat
    where to eat
```

FIGURE 3.4. Content of a real Chatito file



	Chatito	FADE
General	A DSL for creating and maintaining datasets dedicated to training NLU models	A Tool able to retrieve information from knowledge graph, and to use it to generate datasets for training NLU models
Terminology	- Entity is a general term used to describe intent, slot or alias	- Entities are the elements that are used for each intent
Sampling approaches	- The order of elements in a sample is determined by the source code	- All possible permutations are combined automatically to generate training sentences
Supported frameworks	Rasa and Snips NLU	Dialogflow, wit.ai, Amazon Lex

FIGURE 3.5. Table 3.5 : Chatito vs FADE

CHAPTER 4

TRAINING MINOTOUR: DIALOGFLOW VS RASA

4.1 Logs Analysis:

In this section, we will present the results of the Minotour logs analysis we performed before training Minotour on RASA. The logs to be studied correspond to the period between Octobre and Novembre 2018. This analysis will give us an Idea about the intents of the most frequent messages, as well as the intents that are frequently being misidentified by the bot.

The figure 4.1 shows a bars chart of the different intents matches in October logs. We can see that the intent that has been matched the most is "find.event". This means that either most of the people that interacts with Minotour ask for events, or most of the requests that are misidentified end up being recognized as "find.event". By taking a look at the messages, we can say that it's more likely to be the first possibility. Moreover, the second most frequent match corresponds to the default welcome intent. This is quite normal since almost each conversation starts with greetings.

The table 4.2 shows some examples of the messages whom intents were misidentified.

4.2 Training Minotour on RASA:

Currently, Minotour doesn't support RASA NLU. So in order to test its training on this engine, we exported the same project that exists under Dialogflow, processed it, and fed it to RASA. In this section, we will present the process that has been followed along with the results that have been obtained. A comparison between RASA and Dialogflow will be established in the next section.

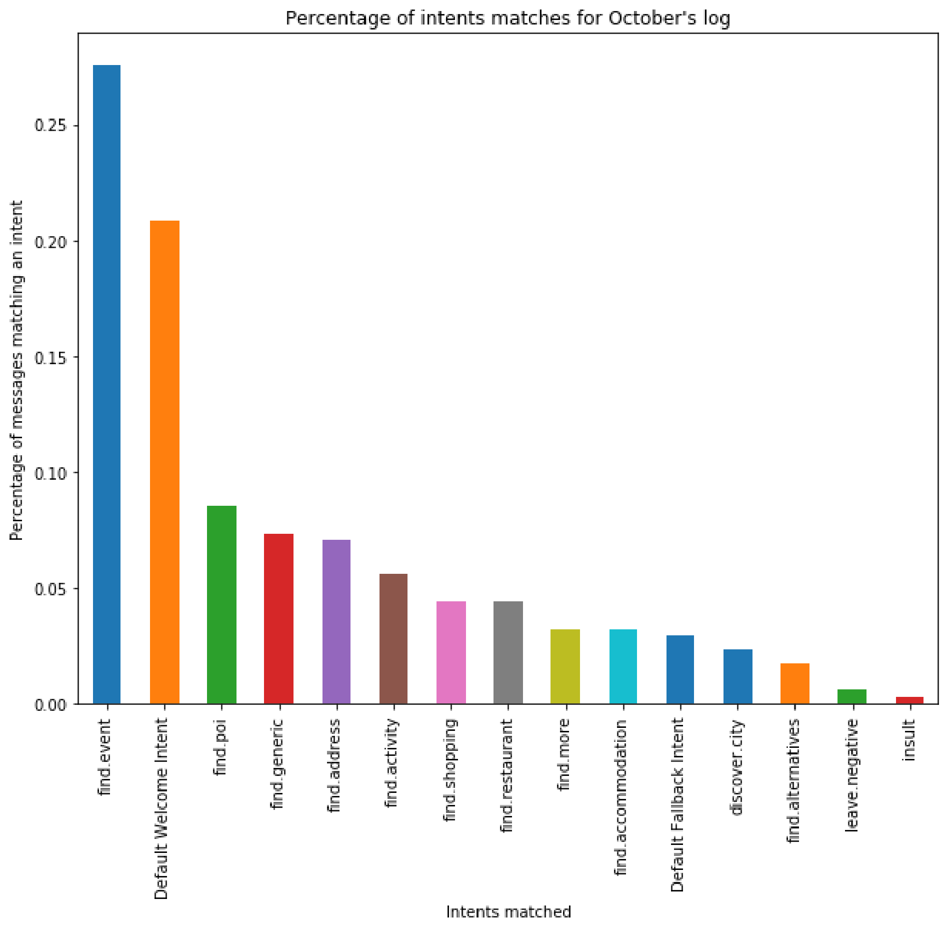


FIGURE 4.1. Intents matches in October 2018

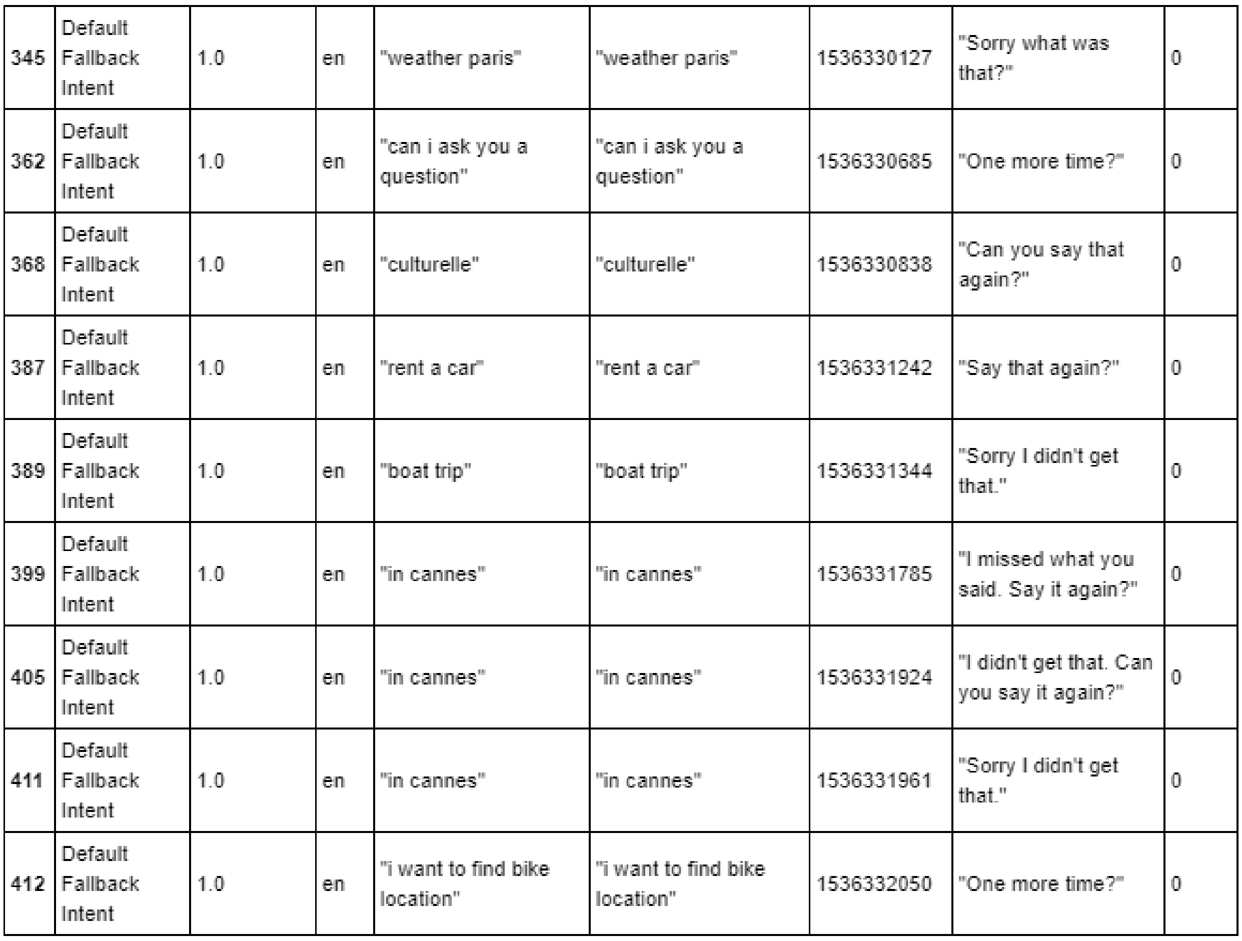
Let's take a look at the training data. The figure 4.3 shows a bar chart representing the number of elements per intent.In total, the initial training data (that is currently used under Dialogflow) contains **3411** examples for **17** distinct intents and **7** distinct entities.

We trained several models, with mainly different training/test sets size (from 548 to 2721 examples for training, and a quarter of this number for testing), and different pipelines [G5] (tensorflow_embedding or Spacy_sklearn).

The figure 4.4 represents the evolution of the accuracy curve with number of elements using tensorflow pipeline.

The figure 4.5 represents a histogram of the hits and misses with 2721 examples for training and 689 examples for testing using tensorflow pipeline.

The figure 4.6 represents a histogram of the hits and misses with 2721 examples for training and



345	Default Fallback Intent	1.0	en	"weather paris"	"weather paris"	1536330127	"Sorry what was that?"	0
362	Default Fallback Intent	1.0	en	"can i ask you a question"	"can i ask you a question"	1536330685	"One more time?"	0
368	Default Fallback Intent	1.0	en	"culturelle"	"culturelle"	1536330838	"Can you say that again?"	0
387	Default Fallback Intent	1.0	en	"rent a car"	"rent a car"	1536331242	"Say that again?"	0
389	Default Fallback Intent	1.0	en	"boat trip"	"boat trip"	1536331344	"Sorry I didn't get that."	0
399	Default Fallback Intent	1.0	en	"in cannes"	"in cannes"	1536331785	"I missed what you said. Say it again?"	0
405	Default Fallback Intent	1.0	en	"in cannes"	"in cannes"	1536331924	"I didn't get that. Can you say it again?"	0
411	Default Fallback Intent	1.0	en	"in cannes"	"in cannes"	1536331961	"Sorry I didn't get that."	0
412	Default Fallback Intent	1.0	en	"i want to find bike location"	"i want to find bike location"	1536332050	"One more time?"	0

FIGURE 4.2. Table 4.2 Misidentified intents in October 2018 logs

689 examples for testing using spacy pipeline.

As we can see from the results, tensorflow_embedding pipeline performs way better than spacy_sklearn pipeline. This can be explained by the fact that the spacy_sklearn uses pre-trained word vectors from either GloVe or fastText while the tensorflow_embedding pipeline doesn't use any pre-trained word vectors, but instead fits the vectors specifically for the dataset it is trained on.

Another problem with spacy is related to scalability. Indeed, with a few 1000 examples, the training takes an extremely long time. This seems to be due to the component intent_classifier_sklearn that uses sklearn.svm.SVC, a C-Support Vector Classification whose fit time complexity is more than quadratic with the number of samples.

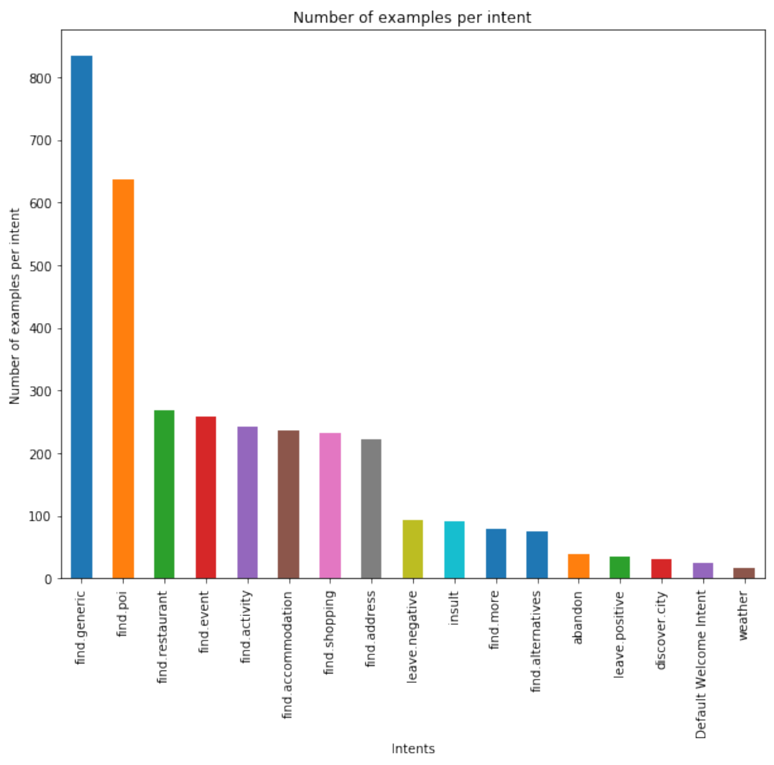


FIGURE 4.3. Distribution of training data per intent

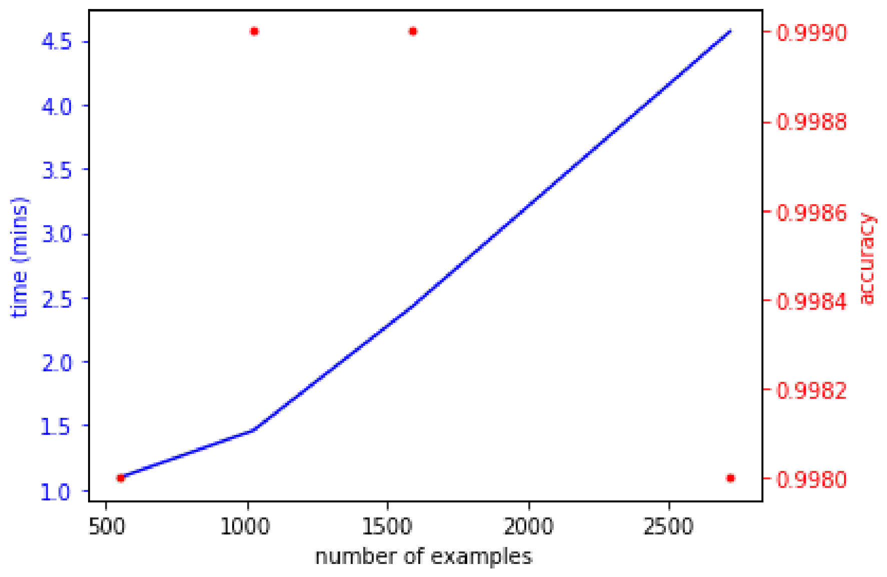


FIGURE 4.4. Accuracy evolution with training data size (pipeline: tensorflow)

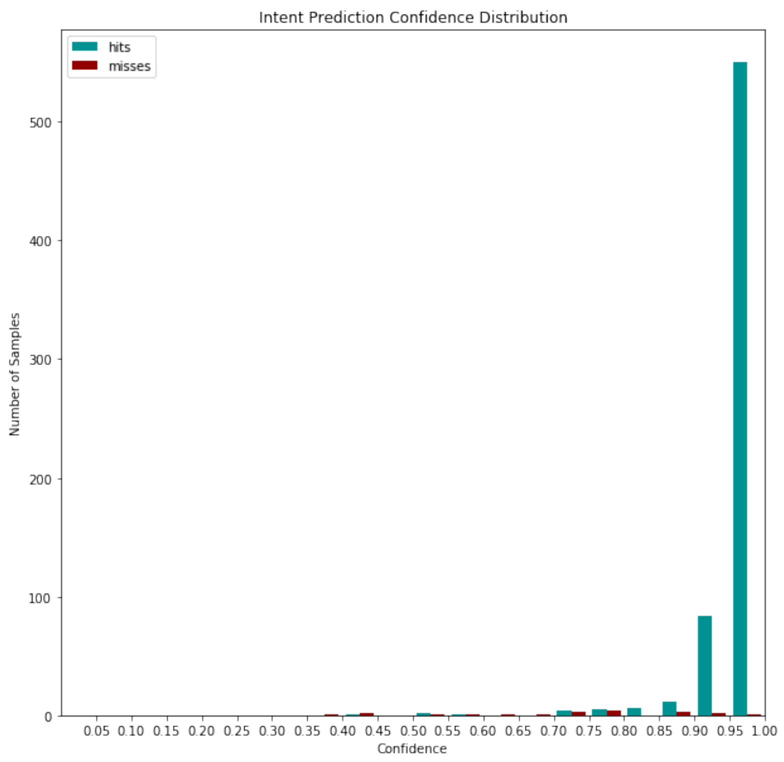


FIGURE 4.5. Distribution of hits and misses (2721 training - 689 testing) (pipeline: tensorflow)

4.3 Minotour: RASA vs Dialogflow

The table 4.7 shows the results we get for intent recognition and the confidence with which it was recognized for both RASA and dialogflow NLU.

As we can see, unlike RASA, Dialogflow has a confidence of 1 for almost all the examples. The model seems to overfit on the examples it receives, eventually because they are somehow similar to the data it has been trained on. However, analyzing the confidence differences is not really the most accurate way for comparing the models, and that's for two main reasons:

- Dialogflow has a context that helps to understand and make, if any, a link between two consecutive messages while RASA NLU doesn't support that. Indeed, we need the RASA core part and another component to handle that in RASA;

- According to RASA documentation [6], the confidence is not a true probability that the prediction is correct, it's just a metric defined by the model that approximately describes how similar your input was to the training data. Therefore, it is not accurate to use it as a metric to compare the models.

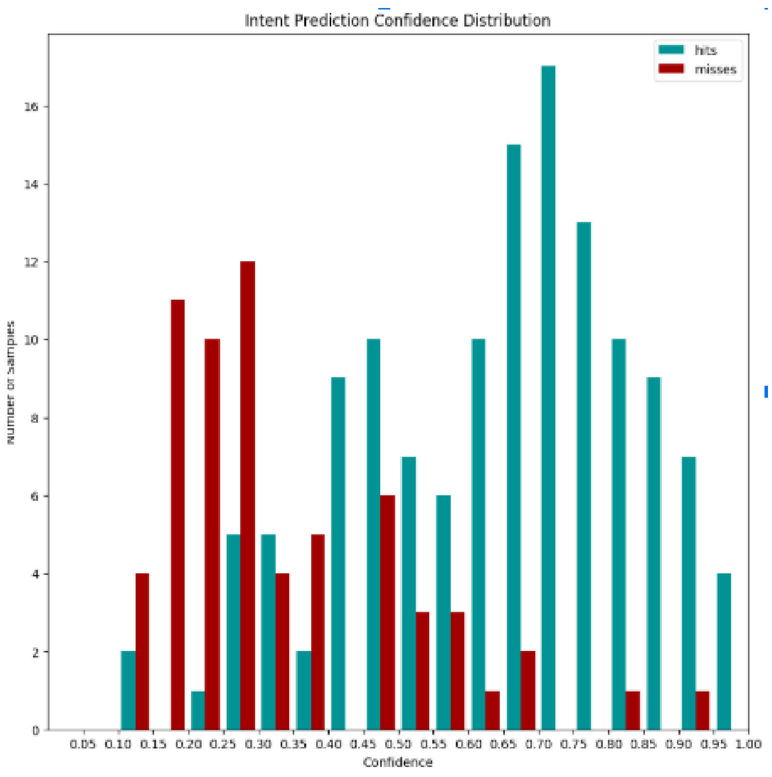
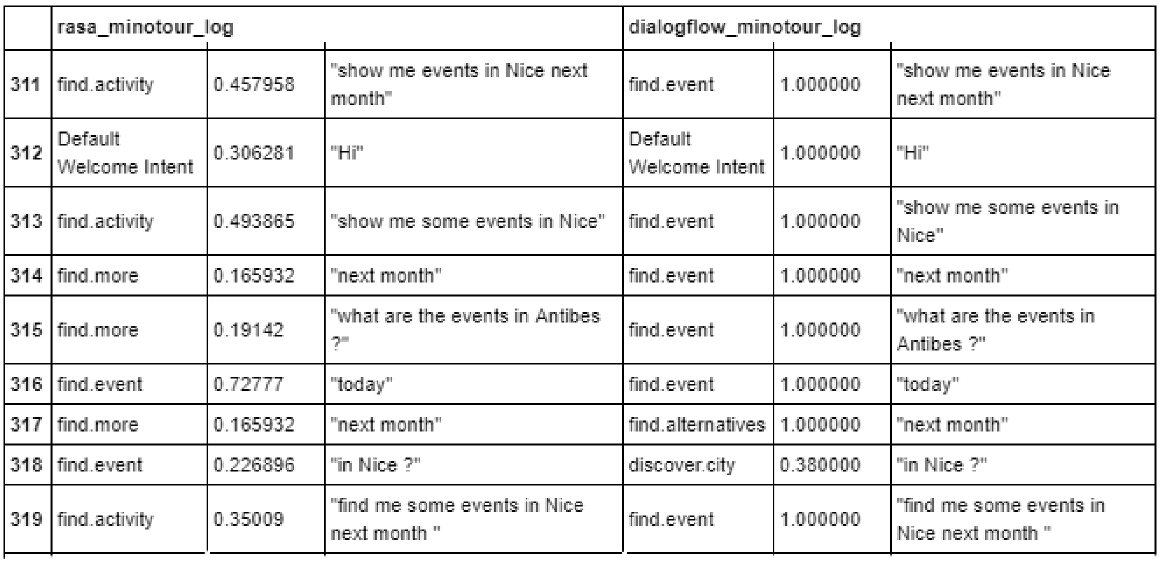


FIGURE 4.6. Distribution of hits and misses (2721 training - 689 testing) (pipeline: Spacy)



	rasa_minotour_log			dialogflow_minotour_log		
311	find.activity	0.457958	"show me events in Nice next month"	find.event	1.000000	"show me events in Nice next month"
312	Default Welcome Intent	0.306281	"Hi"	Default Welcome Intent	1.000000	"Hi"
313	find.activity	0.493865	"show me some events in Nice"	find.event	1.000000	"show me some events in Nice"
314	find.more	0.165932	"next month"	find.event	1.000000	"next month"
315	find.more	0.19142	"what are the events in Antibes ?"	find.event	1.000000	"what are the events in Antibes ?"
316	find.event	0.72777	"today"	find.event	1.000000	"today"
317	find.more	0.165932	"next month"	find.alternatives	1.000000	"next month"
318	find.event	0.226896	"in Nice ?"	discover.city	0.380000	"in Nice ?"
319	find.activity	0.35009	"find me some events in Nice next month "	find.event	1.000000	"find me some events in Nice next month "

FIGURE 4.7. Table 4.7 : Intent Recognition using RASA and Dialogflow

CHAPTER 5

Word embeddings applied to Places & Events data

Taking into consideration the purpose of Minotour, we used data related to tourism in côte d'azur in France. The data was scrapped from multiple websites, and is regrouped in two big main CSV files: **Places** file and **Events** file. In the first subsection and without loss of generality, we will focus on the Events CSV file.

Each row in this file corresponds to a specific event. In addition to event title and descriptions, each entry has several other features, such as language, city, address, and many others that are irrelevant for us.

In the following subsections, we will present the purpose and the results of our implementation of word embedding algorithms on this data.

5.1 FastText & Unsupervised Clustering

The first Idea we got when thinking of word embedding algorithms applied to such a data is the following: Since FADE is missing data and a developer still need to add entries manually (even though FADE was initially meant to automatize the whole process), we will try to use word embedding algorithms to help with this task.

We processed the data and extracted the descriptions of the events file. Then, we applied fastText algorithm on the vocabulary built from this data in order to get vectors highly influenced by the context from which words were taken (events).

Then we applied K-means [G6] algorithm on the output in order to get an unsupervised clustering of the words. We wanted to see if this may give any insight (such as synonyms or word belonging to the same context clustered together, etc) that can be therefore useful for enriching the entities entries. Here are the words that some clusters contained:

```
cluster n 5 elements :  ['montagne,', 'montagnes,',
'ensemble,', 'd tente,', "l'ann e,", 'titanesques,', ... ]

cluster n 6 elements :  [':', 'h', '-', '6', 'g', '/',
'7', '"', '9', 'no', 'bra', '\na', '26', ...]'
```

As we can see, there is some small resemblance between few words belonging to the same lexical field in cluster number 5, but there is no overall strong correlation that can make this results useful.

Instead, the only thing that seems to be done correctly is noise filtering. In fact, we can see that cluster number 6 contains almost only noise, which can be useful for filtering out most of the noise, since we only have to remove from our dataset all the words belonging to certain clusters.

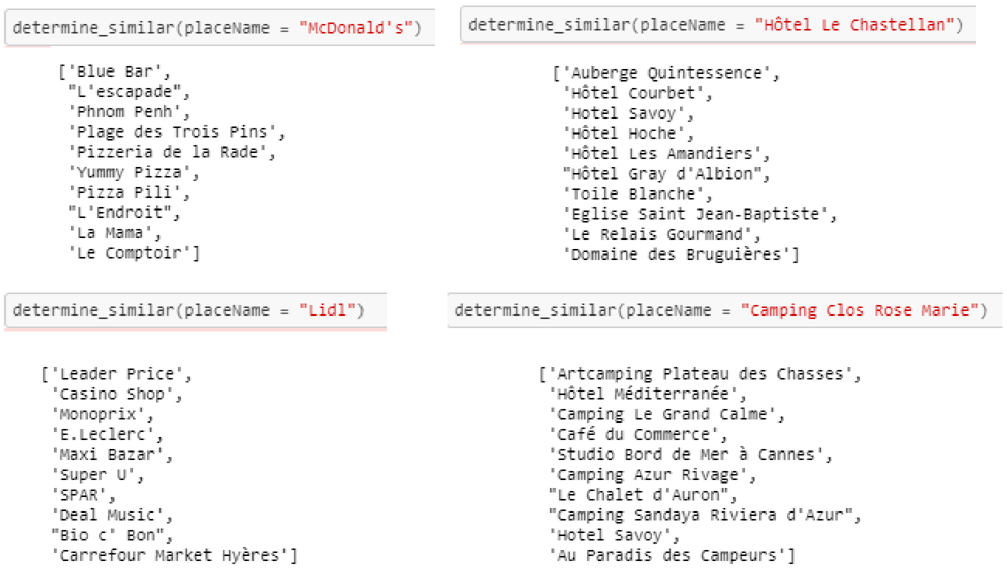
We can see that using this process doesn't seem to provide any add-value to our problem, which is a bit intuitive since the vocabulary is big, noisy and uncontrollable, so unsupervised clustering is not beneficial in this case. Instead, we've thought of a more developed and structured process that can help addressing our problem, and that's what will be presenting in the next and last subsection.

5.2 Doc2vec Algorithm applied to Clients Reviews

In this subsection and without loss of generality, we will focus on the Places CSV file. Each row in this file corresponds to a specific place with a specific review, so basically the same place can be found multiple times in the CSV file, depending on the number of reviews it has. In addition to place name and reviews, each entry has several other features, such as language of the reviews, city, address, and many others that are irrelevant for us. The file contain more than 1 million entries, but since we are interested only in places with french reviews, the number of such entries is around 32K.

The goal of the following study is to explore the potentiality of doc2vec algorithm applied to customers reviews in the task of places classification. The Idea is to create one file/place, and to regroup in that file all the reviews in our possession that have been written about this place, then perform a similarities analysis, and finally define and train a classifier.

Doc2vec algorithm [15] is based on the word2vec model, but instead of constructing numeric



```
determine_similar(placeName = "McDonald's")

['Blue Bar',
 "L'escapade",
 'Phnom Penh',
 'Plage des Trois Pins',
 'Pizzeria de la Rade',
 'Yummy Pizza',
 'Pizza Pili',
 "L'Endroit",
 'La Mama',
 'Le Comptoir']
```

```
determine_similar(placeName = "Hôtel Le Chastellan")

['Auberge Quintessence',
 'Hôtel Courbet',
 'Hotel Savoy',
 'Hôtel Hoche',
 'Hôtel Les Amandiers',
 "Hôtel Gray d'Albion",
 'Toile Blanche',
 'Eglise Saint Jean-Baptiste',
 'Le Relais Gourmand',
 'Domaine des Bruguières']
```

```
determine_similar(placeName = "Lidl")

['Leader Price',
 'Casino Shop',
 'Monoprix',
 'E.Leclerc',
 'Maxi Bazar',
 'Super U',
 'SPAR',
 'Deal Music',
 "Bio c' Bon",
 'Carrefour Market Hyères']
```

```
determine_similar(placeName = "Camping Clos Rose Marie")

['Artcamping Plateau des Chasses',
 'Hôtel Méditerranée',
 'Camping Le Grand Calme',
 'Café du Commerce',
 'Studio Bord de Mer à Cannes',
 'Camping Azur Rivage',
 "Le Chalet d'Auron",
 "Camping Sandaya Riviera d'Azur",
 'Hotel Savoy',
 'Au Paradis des Campeurs']
```

FIGURE 5.1. Four Examples of Most-similar (cosine based) results using documents vectors

representation for words, it aims at creating numeric vectors for documents. The nuance between the two is that word2vec exploits the Idea that the prediction of neighboring words for a given word strongly depends on this latter, while doc2vec considers that this prediction depends on the document also.

Let's now give the detailed description of what we've done:

1. Read data from places scrapped CSV file and select places that have french reviews (at least one french review).

2. Group the reviews by places and create 1 document per place containing all of its french reviews.

3. Training a Doc2vec model to generate 1 vector per document.

4. Study similarities detected and analyse the patterns. The figure 5.1 presents some of the results obtained. We can notice that for the different tests performed, most of the similar places correspond to the same category of the place in question (Restaurant, Hotel, Hyper/super market, Camping, etc), which means that the model detects patterns that can be fed to a classifier and potentially performs well.

5. Based on the previous analysis, create generic place classes and Label the data. In fact, The scrapped data comes with a column called categoryNames. This column contains a lot of noise, empty values,a lot of redundancy, same element having multiple labels,other

elements without labels, and so on. So it would be impossible to feed it to a classifier as it is. Instead, we tried to rely on the information it provides in order to automatize the labeling process. We defined 7 generic labels: Restaurant, Hotel, Shopping, Grocery, Divertissement, Public, and other.

6. Prepare data for classification : mapping Vectors to place Labels. The Idea is to map each place vector to each new label. So the features that would be fed into the classifier are the places vectors components.

7. Define training and test sets. In fact, because of class imbalance, it would be hard to train the classifier on that data as it is:

```
Number of elements with Label "Restaurant": 1298
Number of elements with Label "Hotel": 153
Number of elements with Label "Grocery": 5
Number of elements with Label "Shopping": 534
Number of elements with Label "Divertissement": 0
Number of elements with Label "Public": 16
Number of elements with Label "Other": 807
```

   That's why we defined a classification problem with less classes: Restaurant, Hotel and Shopping only, and we tried to gather the maximum elements we can for each of them while ensuring that there isn't a huge different between them in terms of number of training examples.

8. Train a classifier (Gradient Boosting [G7]) to perform the mapping. For this first classifier, we used a training data composed of 500 elements with label "Restaurant", 400 elements with label "Shopping", and 100 elements with label "Hotel", and a test data containing 200 elements with label "Restaurant", 134 elements with label "Shopping", and 53 elements with label "Hotel"

9. Analyze the overall accuracy and intra-classes accuracy. For this model, the overall accuracy is around 80%, while intra-classes accuracies are **85%** for "Shopping", **85%** for "Restaurant", and **43%** for "Hotel". We expected these results since the class hotel is underrepresented.

10. Improve the input data: we decided to feed the model with data having equal distribution among classes. We trained a second classifier with a training data composed of 100 elements of each one of the three classes, and a test data with 51 elements again for each class.

11. The new overall accuracy is **75%**, and intra-classes accuracies are **75%** for "Restaurant", **80%** for "Shopping", and **71%** for "Hotel". There is a significant improvement in "Hotel" class accuracy since it is not underrepresented anymore.

Chapter 6

Conclusions & Future work

The results obtained in the last section seems to be promising. Indeed, the reviews provide more information about the places than we expected. The classifier is able to gain knowledge of the similar places by taking as input only the vector representations that were associated to the reviews document corresponding to the place in question. Thus, it can be very beneficial to implement somewhere in Minotour/FADE global process an add-on based on this classification solution. It will help categorizing the new data entries automatically without the need for a human to label manually every single entry. Aside from this, We conducted training experiments of Minotour on RASA in order to see whether a migration could be a good option. And we saw that Minotour trained on RASA NLU, especially with tensorflow pipeline, gave a good performance.

In order for the team to make an efficient use of our results and to exploits our analysis, further steps are required:

- In the doc2vec part, the final results seem good but due to our constraints(taking only french reviews, dealing with class imbalance,..), the datasets on which our classifiers were trained are of limited size. So in order to ensure the stability of the result, and potentially to make the model learns much more patterns (and thus improve the results), more data that meets our requirements should fetched, and more training and model optimization should be conducted. Moreover, the developer(s) that will take over the project should think of how this process can be integrated to FADE from a practical point of view.

- In the RASA part, since FADE doesn't currently support RASA, an add-on to FADE should be implemented in order to generate Chatito files. Once this is done, RASA will be added to the engines supported by FADE.

Appendix A

Glossary

1. **One-hot Encoding:** One-hot is a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0).
2. **Neural Network:** Computing systems vaguely inspired by the biological neural networks that constitute animal brains.The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs.
3. **Factorization methods:** Class of collaborative filtering algorithms used in recommender systems. Matrix factorization algorithms work by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices.
4. **REST-API:** is a software architectural style that defines a set of constraints to be used for creating web services.
5. **Pipeline:**a linear sequence of specialized modules used for pipelining.
6. **K-means:** K-means clustering aims to partition the n observations into k sets S = {S1, S2, . . . , Sk} so as to minimize the within-cluster sum of squares(WCSS) (i.e. variance).
7. **Gradient Boosting:**machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

BIBLIOGRAPHY

[1] P. BOJANOWSKI, E. GRAVE, A. JOULIN, AND T. MIKOLOV, *Enriching Word Vectors with Subword Information*, https://arxiv.org/pdf/1607.04606.pdf, 2017.

[2] M. CHABLANI, *Word2Vec (skip-gram model): PART 1 - Intuition*, https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b.

[3] ——, *Word2Vec (skip-gram model): PART 1 - Intuition*, https://towardsdatascience.com/word2vec-skip-gram-model-part-1-intuition-78614e4d6e0b.

[4] D2KLAB, *Minotour Technical Documentation*.

[5] DIALOGFLOW, *DialogFlow Documentation: Overview*, https://dialogflow.com/docs/intro.

[6] R. O. DOCUMENTATION, *Confidence and Fallback Intents*, https://rasa.com/docs/nlu/fallback/.

[7] G. FUTIAA, A. VETRO, A. MELANDRIB, AND J. C. D. MARTIN, *Training Neural Language Models with SPARQL queries for Semi-Automatic Semantic Mapping*, https://www.researchgate.net/publication/328373466_Training_Neural_Language_Models_with_SPARQL_queries_for_Semi-Automatic_Semantic_Mapping.

[8] D. KARANI, *Introduction to Word Embedding and Word2Vec*, https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa.

[9] KUNAL BHASHKAR, *Conversational AI chatbot using Rasa*, https://medium.com/@BhashkarKunal/conversational-ai-chatbot-using-rasa-nlu-rasa-core-how-dialogue-handling 2018.

[10] S. LYNN, *Get Busy with Word Embeddings – An Introduction*, https://www.shanelynn.ie/get-busy-with-word-embeddings-introduction/.

[11] D. D. MAMGAIN, *Dialogflow vs RASA: Which One to Choose?*, https://hackernoon.com/dialogflow-vs-rasa-which-one-to-choose-844c42117cb2, 2018.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient Estimation of Word Representations in Vector Space*, https://arxiv.org/pdf/1301.3781.pdf.

[13] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, *Distributed Representations of Words and Phrases and their Compositionality*, https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[14] R. Pimentel, *Chatito Spec*, https://github.com/rodrigopivi/Chatito/blob/master/spec.md.

[15] Q. L. QVL and T. Mikolov, *Distributed Representations of Sentences and Documents*, https://cs.stanford.edu/~quocle/paragraph_vector.pdf.

[16] RASA, *RASA Documentation: Step 1*, https://rasa.com/docs/get_started_step1/.

[17] M. Trinelli, *Developing a virtual assistant for booking your holidays*, 2018.

[18] L. Weng, *Learning Word Embedding*, hhttps://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html.

[19] Wikipedia, *Natural-Language Understanding*, https://en.wikipedia.org/wiki/Natural-language_understanding.

[20] ——, *Word embedding*, https://en.wikipedia.org/wiki/Word_embedding.