Final report

Anna - The movies-based chatbot

Author :	Mael Ogier - Adrian Delmarre - Léo Vetter
Supervisor :	Behrang Qasemizadeh
Institutions :	INSA Lyon - Universität Passau
Date :	August 27, 2015

Table of Contents

1 Introduction

1.1 Abstract

This project is done in the context of our semester at the University of Passau which is part of our double master degree INSA Lyon Informatique (INSA IF) - Uni Passau Information und Kommunikation (IFIK). The goal here is to make use of what we learned in the first weeks of the *Text Mining Project* course by developing a project. To do so, we took inspiration from the ideas and suggestions from our supervisor Behrang Qasemi Zadeh, some of which are presented in a document on his website [3].
Our idea is to exploit movies' scripts in order to create a bot which you can speak with. This bot is called Anna in reference to the Basshunter's song Boten Anna[1].
This report will be organized as follows. First we will describe our motivations, then present the chosen dataset as well as some enrichments and statistics and finally we will describe how our solution works.

1.2 Motivations

We first chose this course in order to discover a new programming language (Python) as well as a new domain of IT : Natural Language Processing.
NLP is an important IT domain nowadays. More and more NLP-based applications are developed and there are many different use-cases. The simplest one is in our pocket since every smartphone now supports predictive text, Apple's SIRI and Android's "Ok Google" understand pretty much everything the user is saying and handwriting recognition is also a thing. All of this is provided by technologies based on NLP and we can safely say that, in our multilingual society, these technologies will soon be a must-have.
Regarding our project's topic, we all are movies aficionados and the idea of interacting with a bot using our favorite movies' punchlines was very tempting. This is also the occasion to learn more about the english language (grammatically and syntactically speaking) which isn't our mother tongue.

1.3 Acknowledgments

We would like to thank our supervisor Behrang for his precious help, ideas and advices during the whole project phase.
A special "Thank you !" goes to Rafael E. Banchs from the Institute for Infocomm Research of Singapore who authorized us to use his movie dialogue corpus *Movie-DiC*.

[1]Boten Anna on YouTube : https://www.youtube.com/watch?v=RYQUsp-jxDQ

2 Dataset

This section describe the chosen dataset and explain how we enriched it. Statistics about data are also presented.

2.1 Initial Dataset

2.1.1 Description

The first challenge we had to face was finding a corpus of movies' dialogues. Luckily, our supervisor provided us one called Movie-DiC. This corpus has been built by R. Banchs [1] for research and development purposes. See below table for some statistics about the corpus.

Total number of scripts collected	911
Total number of scripts processed	753
Total number of dialogues	132,229
Total number of speaker turns	764,146
Average amount of dialogues per movie	175.60
Average amount of turns per movie	1,014.80
Average amount of turns per dialogue	5.78

Table 1: Main statistics

The corpus consist on a XML document of 2,249,053 lines, its structure isn't so complex since his maximal depth is only of 3 (movie - dialogue - utterance). See Code 1 for an example of a dialogue unit.

```
<dialogue id="2" n_utterances="3">
   <speaker>SECURITY GUARD</speaker>
      <mode></mode>
      <context>K The Guard glances up as Jacket Man gets into an ELEVATOR.</context>
      <utterance>Crump... I'm sorry, no one by that name.</utterance>
   <speaker>SECURITY GUARD</speaker>
      <mode></mode>
      <context>The Security Guard rushes to intercept him.</context>
      <utterance>Hey! You can't go up there!</utterance>
   <speaker>SECURITY GUARD</speaker>
      <mode></mode>
      <context></context>
      <utterance>We've got an intruder in the express elevator!</utterance>
</dialogue>
```

Code 1: Dialogue unit

This dialogue is pulled out the "xXx" movie[2] produced in 2002 by Rob Cohen.

[2]xXx on IMDB : http://www.imdb.com/title/tt0295701/

2.1.2 Remarks

This dataset is really interesting but some details rose small issues during the project.
First, as you can see above in Code 1, one dialogue is in fact a succession of tags which follows a specific pattern :

1. Speaker : who is speaking ?
2. Mode : is he happy/suspicious/...?
3. Context : description of the shot
4. Utterence : the speaker's speech

We found it strange that all those 4 tags aren't children of another one, let say "utterance". It would make more sense given the XML spirit and would also be easier to parse. Here we had to iterate on the dialogue's children and group every group of 4 tags together in a single entity.
Then, some of the movies' titles are modified. For instance "X Files Fight the Future The" in fact refers to the movie "The X Files Fight the Future". These little modifications are a problem when it comes to the enrichments phase (see section 2.2).
Finally, the XML document wasn't totally correct, there were special characters such as "&" which blocked the python XML parser. A root tag was also missing. We fixed these errors manually before any operation on the corpus.

2.2 Enrichments

The Movie-DiC corpus only contains data about the content of the movie, but no data about the movie itself apart from its title. In order to build statistics, presented in section 2.4, we enriched the corpus with the help of the Open Movie Database (short OMDb) API[3]. This API is maintained by its users and isn't endorsed by or affiliated with the more famous IMDB website[4].
The use of the OMDb is really simple. It provides a web service which allows us to search for a movie given its title. It then returns lot of data about the movie such as its released year/date, genre, director's and actor's names, language, country. It also pulls out some data from imdb such as the number of votes or the average rating.
Here's an example of usage :
Query : http://www.omdbapi.com/?t=Whiplash&plot=short&r=json
Answer :

```
1 {
2     "Title": "Whiplash",
3     "Year": "2014",
```

[3]http://www.omdbapi.com/
[4]http://www.imdb.com/

```
    "Rated": "R",
    "Released": "15 Oct 2014",
    "Runtime": "107 min",
    "Genre": "Drama, Music",
    "Director": "Damien Chazelle",
    "Writer": "Damien Chazelle",
    "Actors": "Miles Teller, J.K. Simmons, Paul Reiser, Melissa Benoist",
    "Plot": "A promising young drummer enrolls at a cut-throat music conservatory where
        his dreams of greatness are mentored by an instructor who will stop at nothing to
        realize a student's potential.",
    "Language": "English",
    "Country": "USA",
    "Awards": "Won 3 Oscars. Another 87 wins & 95 nominations.",
    "Poster": "http://ia.media-imdb.com/images/M/
        MV5BMTU4OTQ3MDUyMV5BMl5BanBnXkFtZTgwOTA2MjU0MjE@._V1_SX300.jpg",
    "Metascore": "88",
    "imdbRating": "8.6",
    "imdbVotes": "244998",
    "imdbID": "tt2582802",
    "Type": "movie",
    "Response": "True"
}
```

Code 2: OMDb answer

So we processed the whole corpus through this API and saved only the following fields : Country, imdbRating, Director, Actors, Year, Genre, Runtime, imdbVotes.
The "Genre" field is also used in one of our chatbot version, see section 3.4.3 for more details.

We also enrich the corpus by processing each sentence in it. We use the nltk library to tokenize and tag all sentences in our corpus. The purpose of the processing is to identify for each sentence its type and the most relevant tokens. These informations will be then used by anna to answer with the sentence wich is the most relevant given previous sentences.
We identify four categories of sentence : interrogative, exclamative, affirmative and greeting. Moreover a sentence can be either negative or positive. To determine if a sentence is interrogative or exclamative we check the presence of a question mark (?) or a exclamation point (!) at the end of the sentence. If no such marks is found we assume that the sentence is affirmative. To determine if a sentence is negative we check if the sentence contain a "n't" token. Finally to detect a greeting we check if the sentence contain keywords such as "Hi", "Hello", "Good Morning"...
Regarding the tokens we assume tokens tagged as noun and verb are the most relevant to understand the meaning of the sentence and thus we only keep these and eliminate others.
The figure 20 illustrate the informations stored in the database.

2.3 How to use our data ?

The Movie-DiC corpus isn't public so you will need to contact its author in order to have his approval and to get the XML file. All the enrichments we worked with are available on our Github[5]. It is also possible to generate the file yourself, the function to call is *generate_all_infos_json* in the *categorization.py* file.

This file is organized as a key-value JSON with the key being the movie's ID and the value being a JSON containing all the details we kept from the OMDb's API response. Therefore it is very simple to link our enrichments to the main file : you just need to join the movies using their IDs. Here's a simple example :

```
import xml.etree.ElementTree as ET
# Open the JSON file containing all details
with open(theJSONFilePath) as data_file:
  allMoviesDetails = json.load(data_file)
  # Parse the movies XML file
  tree = ET.parse(theXMLFilePath)
  root = tree.getroot()
  # Browse the elements tree
  for movie in root.findall('movie'):
    currentDetails = allMoviesDetails[movie.get('id')]
    # Do something now and have fun
```

Code 3: Usage example

[5] https://github.com/Vongo/anna/blob/master/src/pre-processing/movies-categorization/outputs/allMovies.json

2.4 Statistics

Motivation The idea which lays behind these statistics is to get a better understanding of the data we added to the corpus. In this section, we are going to stress our data, aggregate it, and try to extract some meaning. We have no pretension to be cinema experts, and there are obvious bias to any statistics which would be built on this data :

- First, we are working on a dataset which contains data about less than a thousand movies. This is a lot, but at the same time, this is very few, when you know that more than 1450 movies have had an international publication in 2014.
- The ratings we base our rankings on are the ratings of ImDB users. Users are not experts. Their ratings show how they feel about the movies, but they are not a proof (nor even a direct indicator) that a movie is good or bad.

Modus Operandi All the aggregated statistics you are going to see in this section are based on data aggregated on the condition that there is enough of it. For instance, in the next section 2.4.2, we are going to describe which film directors have made the movies that were most (and least) appreciated by the public. In order to avoid outliers, especially in such a small dataset, we decided not to evaluate the movies from directors of whom only one movie is represented. This method is not perfect, and to have stronger results, we should have increased this threshold to 5 movies at least. But the author of our dataset decided not to focus on some specific directors, and to represent as many of them as possible.

2.4.1 Topology of the data

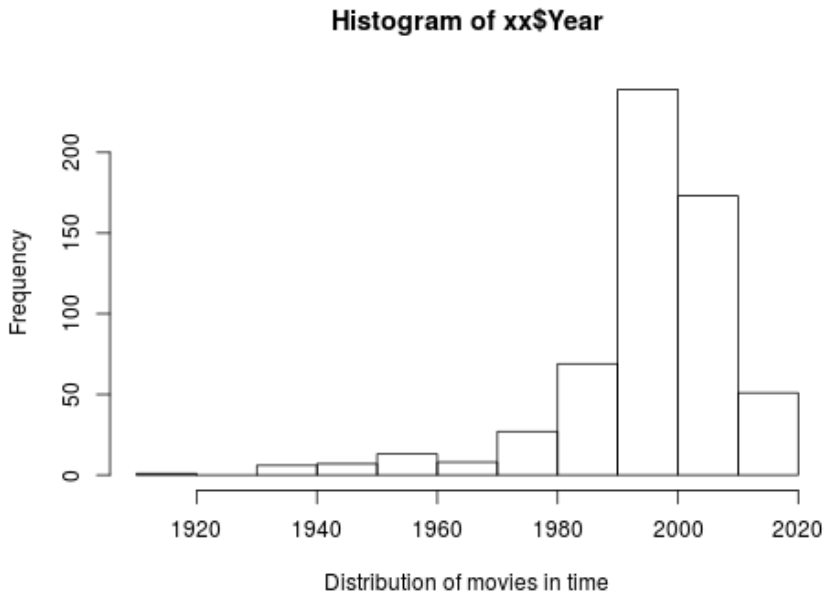

Figure 1: Distribution of movies in time

As we can see in Figure 1, most of the movies in the dataset come from the 1990-2010 period.

genres.png

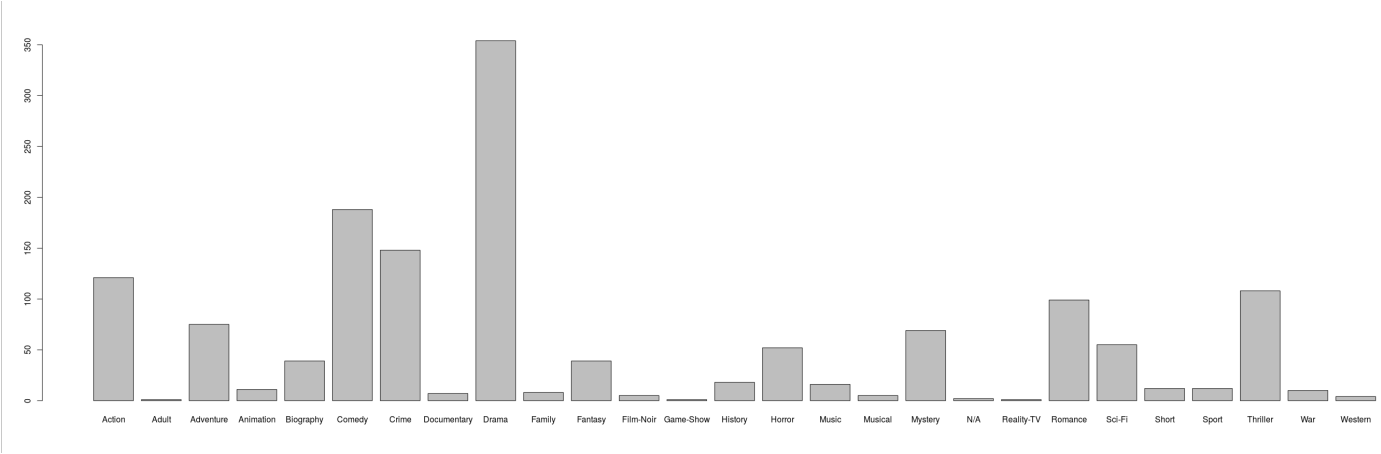

Figure 2: Distribution of movies by genre

Figure 2 shows that most of the movies are Drama. Action, Comedy, Crime, Thriller and Romance also have an acceptable amount of data. But Adult movies, Game-Show and

Reality-TV, alongside with Western, are definitely not represented enough to build reliable stats on it.

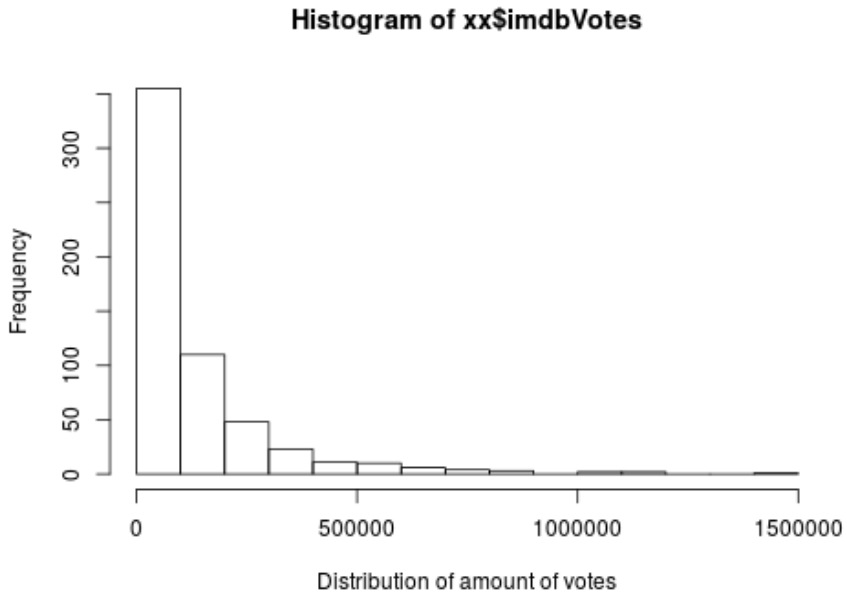

Figure 3: Distribution of amount of OMDb ratings

Besides, before taking into account our stats on the quality of movies, actors or directors, we have to be aware of the fact that most of the time, few people voted (see Figure 3). See also Figure 11 to check how the amount of votes correlates with the "quality" of the movie.

The ratings themselves follow a Gaussian distribution centered around 7.25, as we can see in Figure 4.

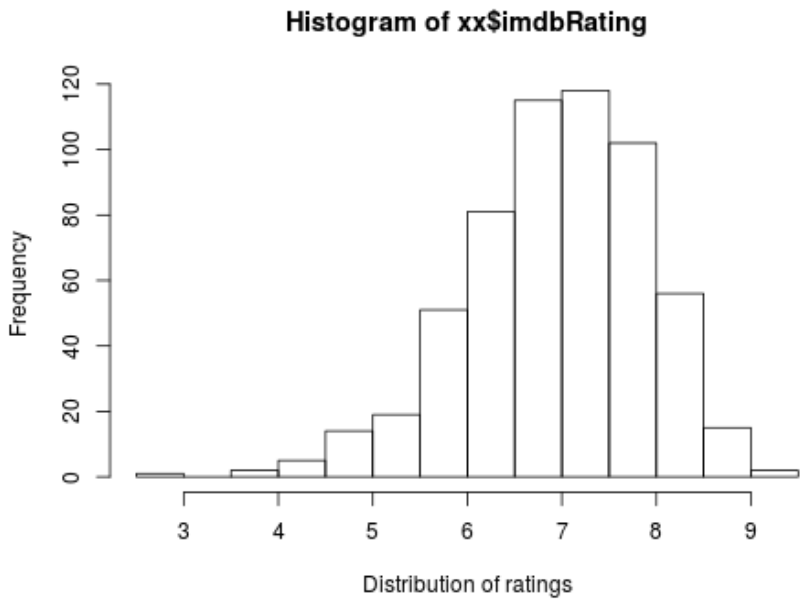

Figure 4: Distribution of OMDb ratings

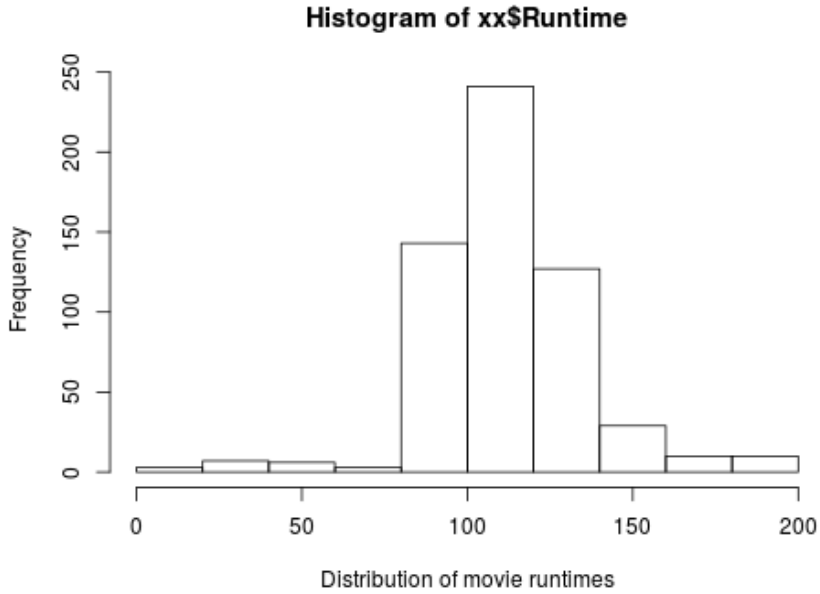

Figure 5: Distribution of runtimes

The movie duration also follows a Gaussian distribution, centered around 100-120 minutes

(see Figure 5).

2.4.2 Directors

Directors rankings that follow in Figures 6 and 7 don't rate the talent of the directors mentioned. They only represent how the ImDB users rated these directors' movies (the ones that are in our dataset, if there are at least two movies for each director in the dataset). Although they are quite consistent with global directors' fame, we cannot help noticing that Brian de Palma, who is quite famous for movies such as *Scarface* or *The incorruptibles*, is ranked very poorly. This is because, although his movies are famous, they are not always highly rated. We could have expected the same from Steven Spielberg's movies, which are also quite mainstream, but the latter is, on the contrary, among the most successful directors. That's one downside of the movie sampling that was made to build this dataset.

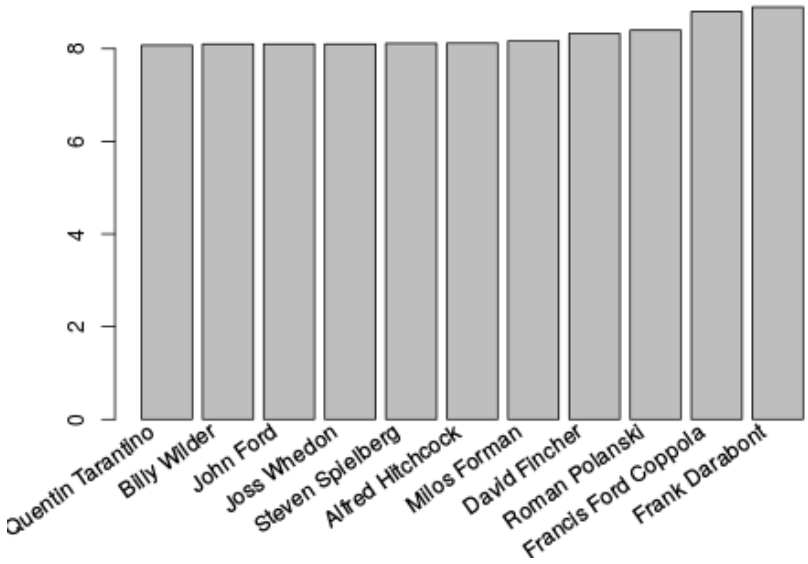

Figure 6: Best directors

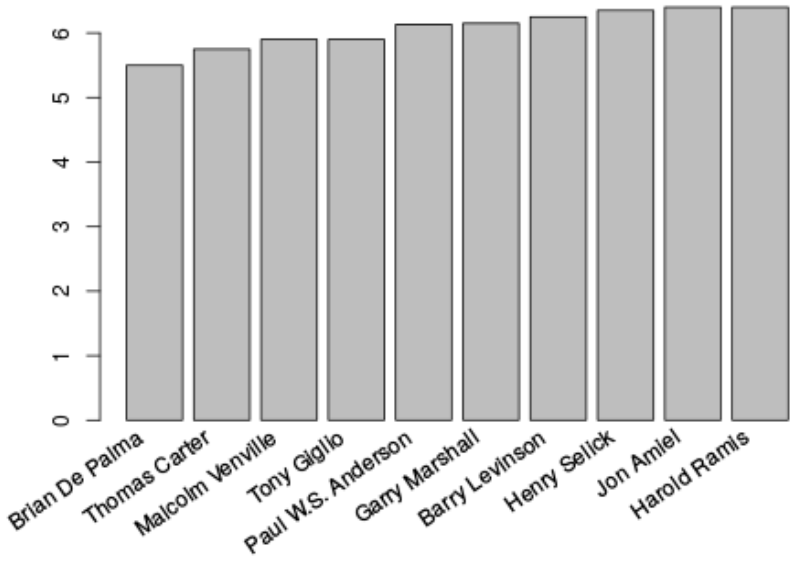

Figure 7: Worst directors

2.4.3 Countries

For ranking countries (see Figure 8), there is also one bias with sample size and representativity. Besides, from one country to another, the standards are note the same, and we must keep in mind that ImDB is an American Website, with American members. Not all Swedish movies are excellent, and not indian movies are mocked by the occidental readers of ImDB.

One more interesting thing to measure is the interaction between countries in co-producing movies. We can see that there is a large historical occidental cluster (USA, UK, France, Germany) around which the co-producing is articulated. This is of course also a biased result, because most of the movies in the dataset are occidental ones.

2.4.4 Ratings

Is *the old better than the new* ? Not for movies, at least. Although we would have expected some bias that would play in favor of old movies (Selection[6], *Hipstering*, Quantity), it seems that the average rating is getting better with time (see Figure 10) How to interpret that ? Well, an interesting phenomenon is that when you look at the all-time best movies ranking on ImDB, the top movies are either very old, or, most of the time, very recent. The very old thing can be explained by the combination of two phenomenon :

- Quantity : there are more and more movies right now, with bigger and bigger budget.

[6]The movies in this dataset were obviously selected for their diversity, but also for their quality : for an old movie not to be forgotten, it takes more than for a recent one.

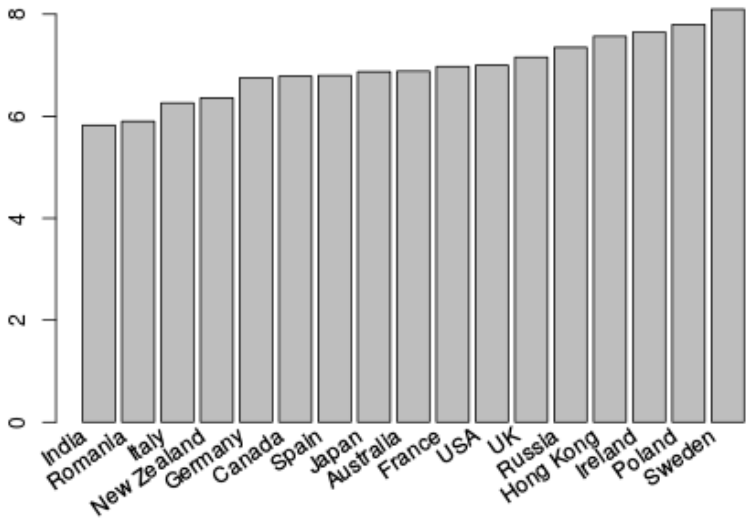

Figure 8: Average movie rating for each country

So, basically, now, everyone can make his own movie, and try and hide his lack of talent behind massive special effects or famous actors.

- Hipstering : for some reason, it is *cool* to like old movies and to promote them as better than the recent blockbusters, which are so *mainstream*.

On the other hand, the fact that recent movies are well rated can be explained with a more psychological approach : people tend to like what is new. Besides, the novelty of a movie like *Avatar* (J.Cameron, 2009) was the mind-blowing special effects, which seem now quite disappointing, even though the critics were dithyrambic at the time. For a movie to survive old-fashioned special effects, it takes talent (Confer the fist *Star Wars* trilogy). That's why recent blockbusters are often disavowed a few years later.

Figure 11 is also quite interesting. On the one hand, we could say that there is no direct correlation between the quality of a movie and the quantity of users that evaluated it, because there are movies evaluated as good with few evaluations, and there are some with many[7]. But on the other hand, movies rated as bad by ImDB users are never evaluated by many users.

[7]Although there are more with few than with many.

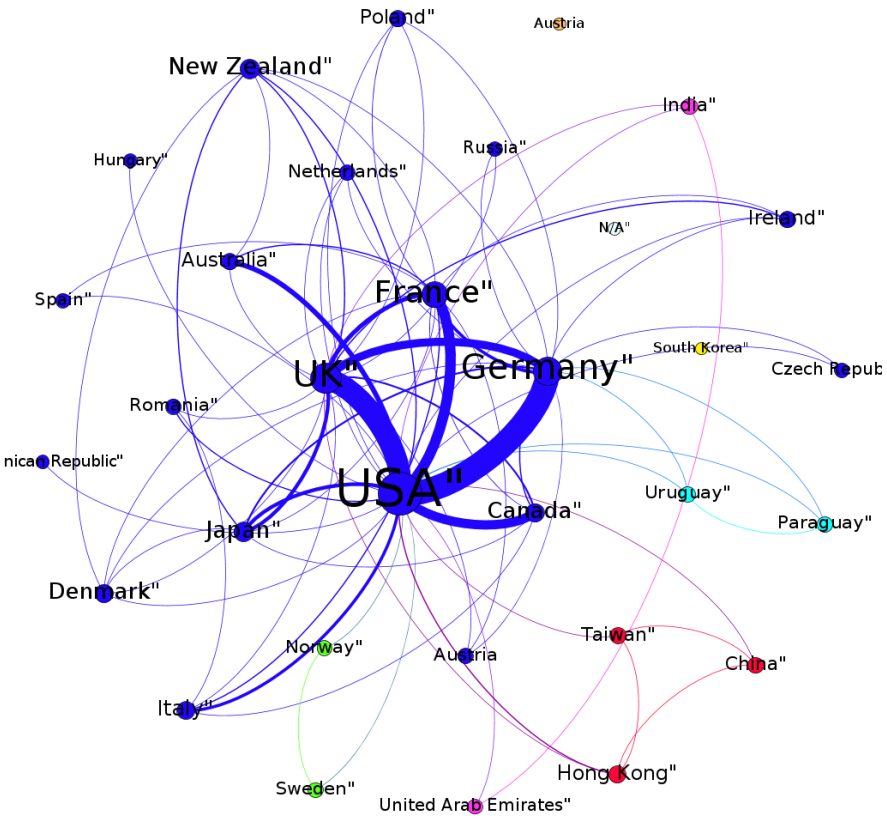

Figure 9: Network of countries co-producing movies

Figure 12 raises the issue of a correlation between rating and runtime. Well, this is definitely not obvious. By proceeding a simple least-square linear regression, we can say that the trend is for longer movies to be more appreciated.

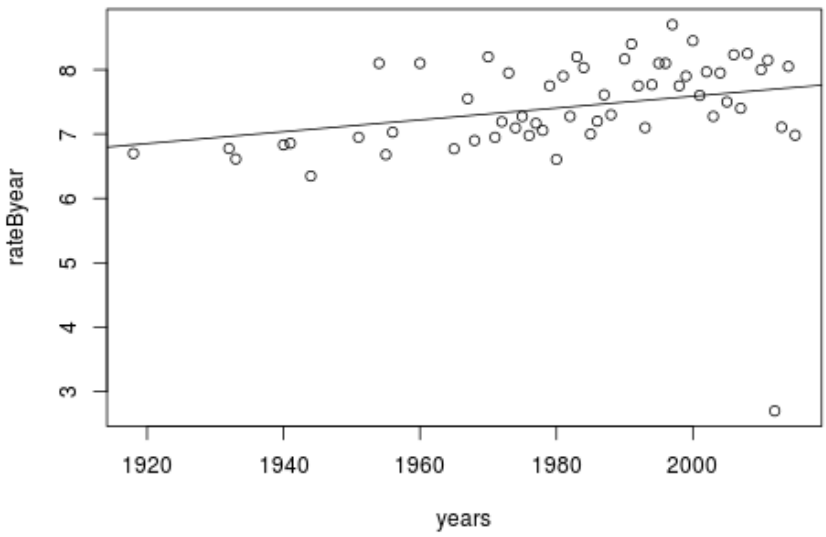

Figure 10: Evolution of rating over time

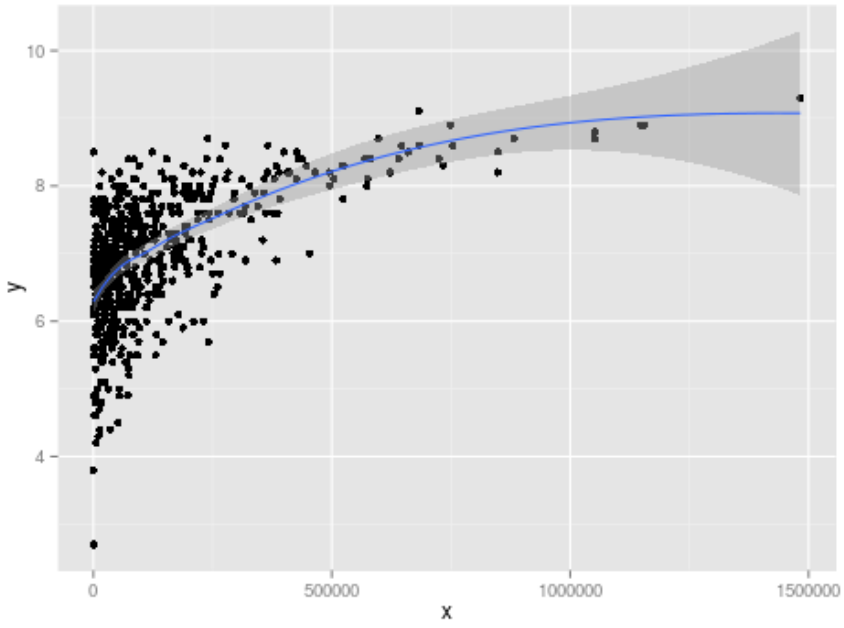

Figure 11: Rating related to amount of ImDB users who voted

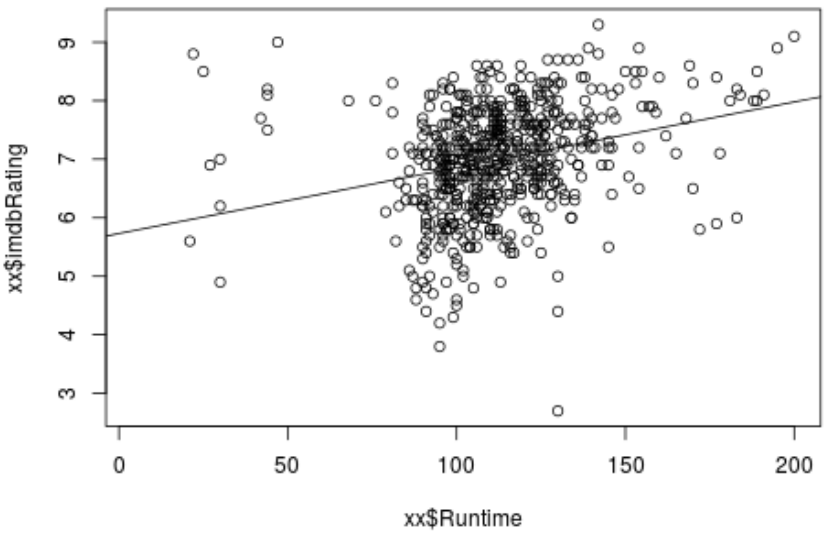

Figure 12: Rating related to film duration

2.4.5 Genre

We can see in Figure 13 that usually, the best ratings are given to movies that deal with serious topics (*History*, *Biography*, *War*). On the other hand, large-audience topics, such as *Horror* or *Action* movies, or *Comedies*, even though they are very popular, seem to be evaluated as less "good" by the audience. One surprising result is that *Documentaries* are unexpectedly low in this ranking, when the popular *Drama* is quite high.

In Figure 14, we see that the ranking is quite similar as in Figure 13. The longer the better : that's a result we already saw in Figure 12 where we evaluated the correlation between rating and duration. Let's note, though, that most *Films-Noirs* are shorter than the average, and still they get the best ratings.

We can also study (see Figure 15) the proximity between the different genres.

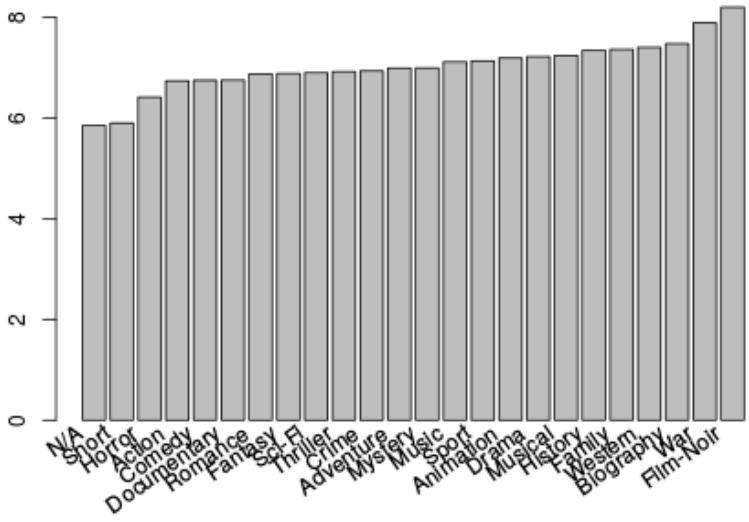

Figure 13: Average movie rating for each genre

2.4.6 Actors

As for directors, actors rankings that follow in Figures 16 and 17, once again, don't rate the talent of the actors mentioned. They only represent how the ImDB users rated these actors' movies (that are in our dataset, if there are at least two movies for each actor in the dataset). But they are quite consistent with global actors' fame.

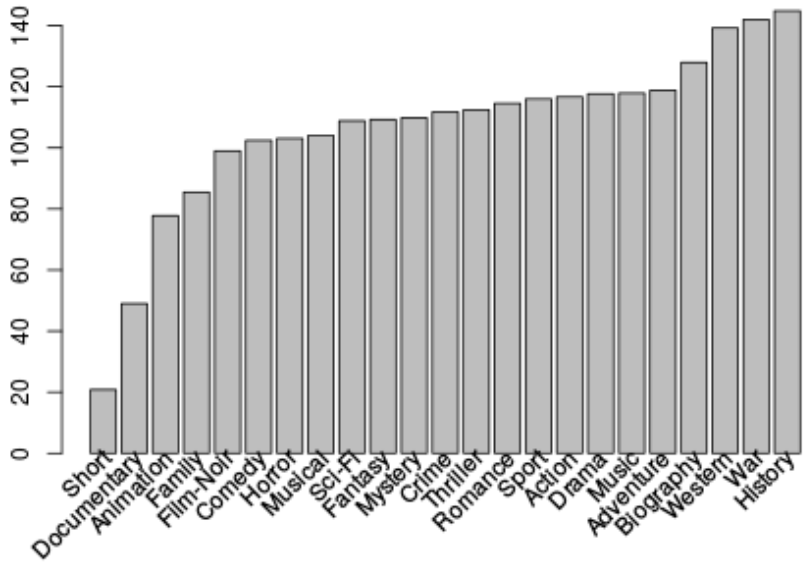

Figure 14: Average movie duration for each genre

Figure 18 represents the network of actors playing together. We can see that there is a very huge connected component in the center of the graph, built around famous actors like Robert de Niro or Georges Clooney. On the other hand, there is a lot of little independent cliques (for instance, there is one composed of Levar Burton, Patrick Stewart, Jonathan Frakes and Brent Spiner). They represent actors who played together but not with other actors. The theory behind this kind of graph is very strong[8]. Thus we know that this graph should be fully-connected, thanks to less-famous actors, who get minor roles in many movies. Here, we only get the main actors for each movie when we query ImDB APIs. That makes the graph all the more so interesting to read.

[8]Thanks to *Six Degrees of Kevin Bacon(https : //en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon)and many movie fans.*

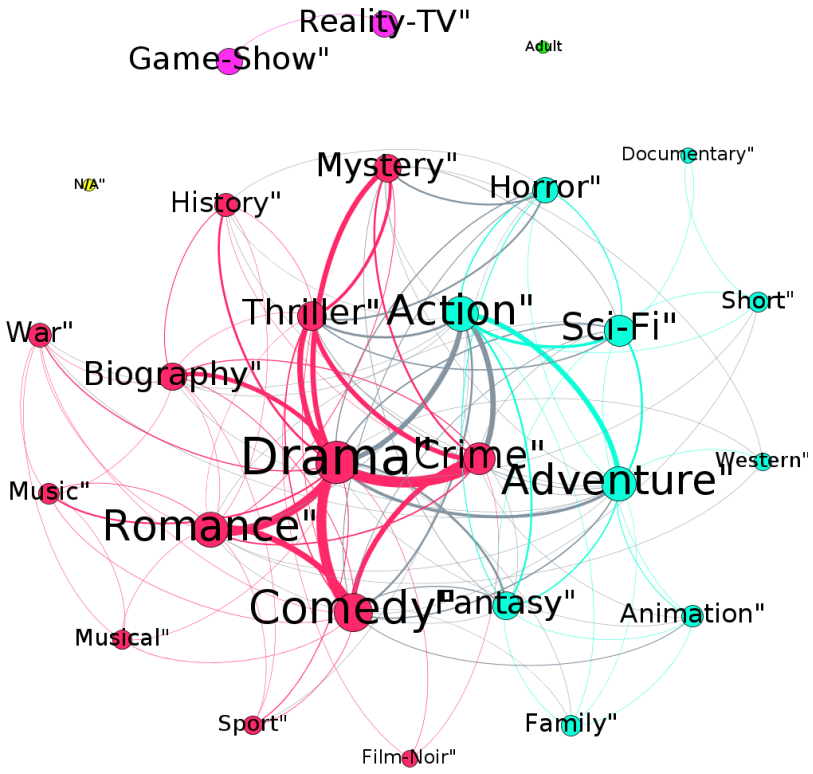

Figure 15: Proximity between all genres

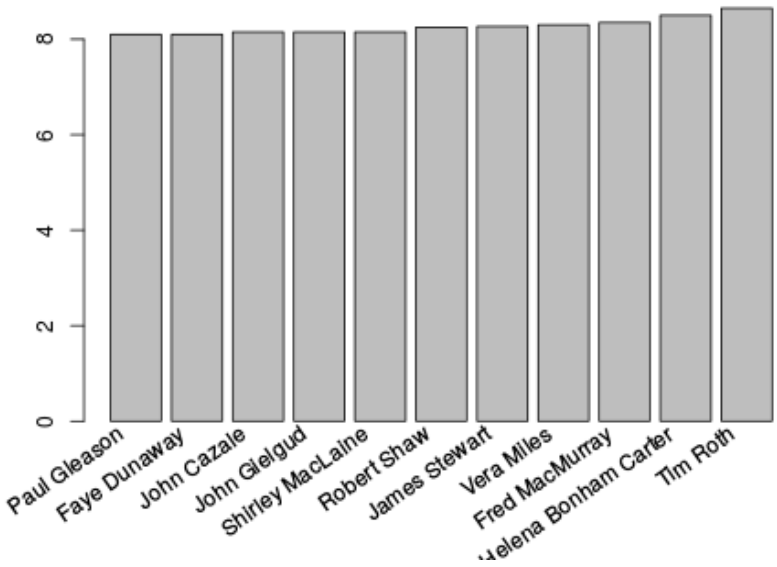

Figure 16: Best-ranked actors

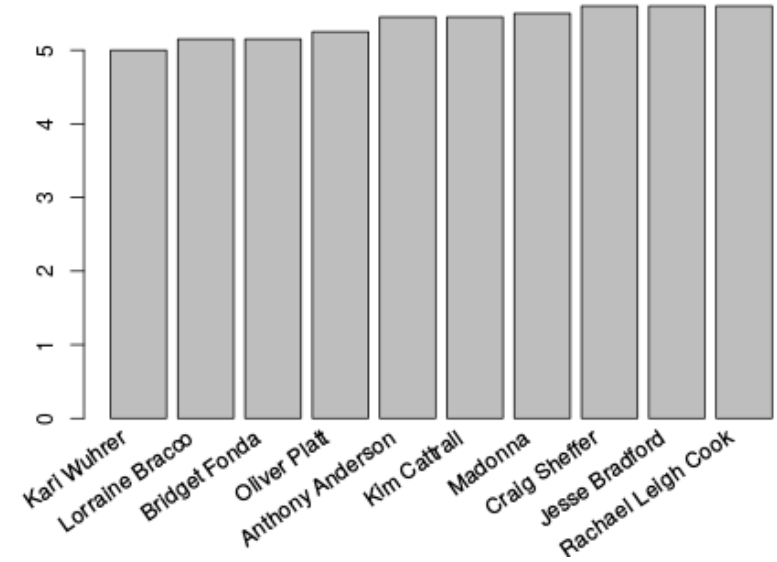

Figure 17: Worst-ranked actors

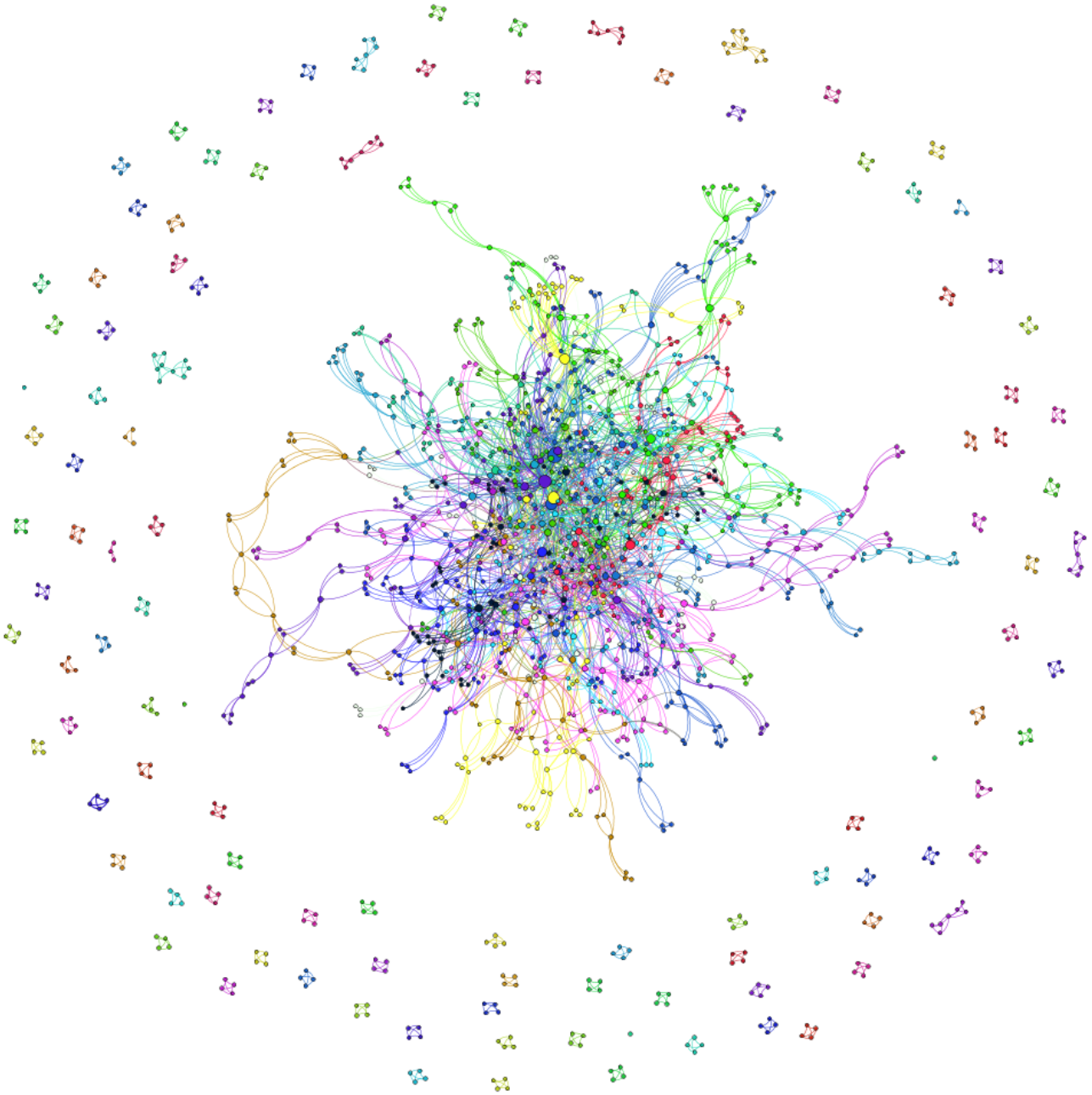

Figure 18: Network of actors playing together

3 Proposed ChatBot

3.1 DataModel

It seems obvious that browsing our 2,249,053 lines-long corpus for each query wasn't such a great idea, that's why we stored the data in a re-usable DBMS.

3.1.1 Chosen technology

We considered 2 options : a traditional DBMS such as MySQL or a graph-oriented DBMS. We finally chose to use a graph model for several reasons :

- For a corpus such as ours, with a lot of n-1..n relations, a graph allows the user to build queries which look like "paths". It is both more efficient (no table join, less computation time) and more natural for the user.
- Graph-base DBMS doesn't require a rigid schema, which means that we can create new kinds of entities and relationships "on the go".
- It supports indexes and unique constraints.
- Most of the graph-based systems implement useful algorithms such as the shortest path between two nodes.

Regarding the technology, Neo4j[9] was chosen in regards of those facts :

- One of us (Mael) was already working with it for his MasterArbeit and was familiar with the language used to build Neo4j queries (Cypher).
- Neo4j is cross-platform.
- The community is strong, there are even french groups.
- The technology is mature, 15 years old and currently running version 2.2
- It provides a nice browser interface.

3.1.2 Structure

Here, we will describe the actual structure of the corpus-graph. Remember that it isn't rigid so it might vary over the project's progress.
Other nodes and relationships exist in our database but are used for other purposes such as the history of the conversation or some probabilities (see section 3.4.2).
Basically we can say that in our case, one vertex equals one XML tag. Instead of listing all the vertexes and the edges, we provide here an example based on a dialogue from the movie[10] "12 Monkeys".

[9] http://neo4j.com/
[10] 12 Monkeys on IMDB : http://www.imdb.com/title/tt0114746/

```xml
<movie id="1" title="12 Monkeys">
  <!-- 70 other dialogues before this small one -->
  <dialogue id="71" n_utterances="2">
     <speaker>COLE</speaker>
        <mode></mode>
        <context>But RAILLY'S total attention is on their dilemma.</context>
        <utterance>I've seen that...before.</utterance>
     <speaker>RAILLY</speaker>
        <mode></mode>
        <context>Very confused, COLE lets her drag him along the sidewalk, past ONLOOKERS.
             She looks crazier than he does. ANGLE ON THE CHEVY, making a sudden, urgent u
            -turn, almost colliding with a passing car. BRAKES SQUEAL and a HORN BLARES.</
            context>
        <utterance>James, trust me. We're in terrible trouble. We have to run.</utterance>
  </dialogue>
  <!-- 32 others after -->
</movie>
```

Code 4: XML extract

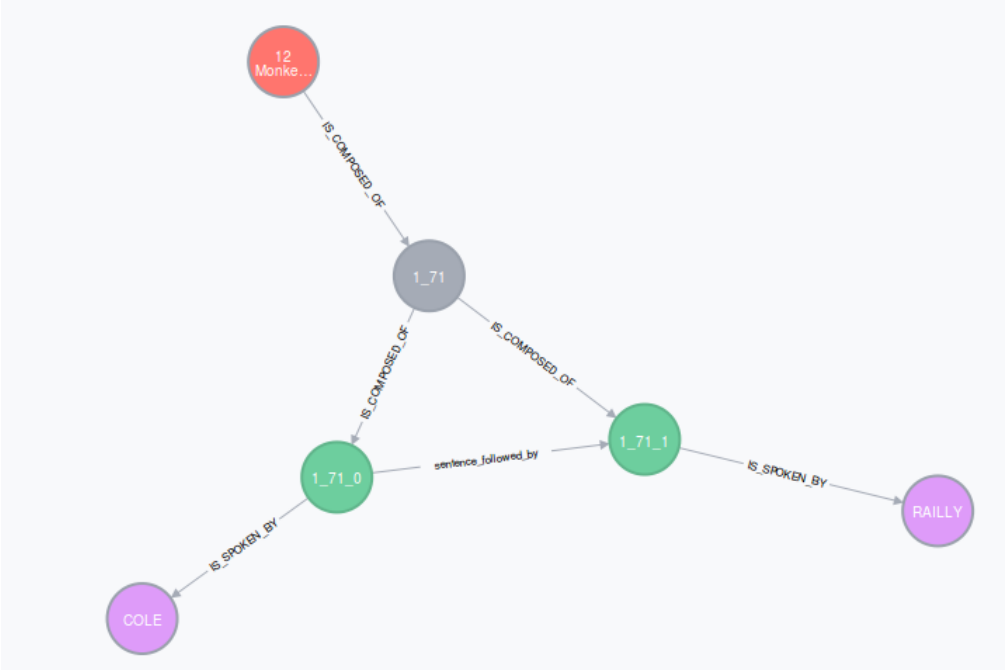

Figure 19: Neo4j modelisation

With the above example we see that, just like in the XML (Code 4), a **Movie** (pink) *IS_COMPOSED_OF* **Dialogues** (gray, only one displayed here) which *IS_COMPOSED_OF* several **Sentences** (green). Each sentence *IS_SPOKEN_BY* a **Character** (purple).
We store the full sentence as an attribute of the Sentence node. The labels of the dialogues and sentences are an aggregation of the id fields of their parents :

Sentence_Node_id = idMovie_idDialogue_idSentence

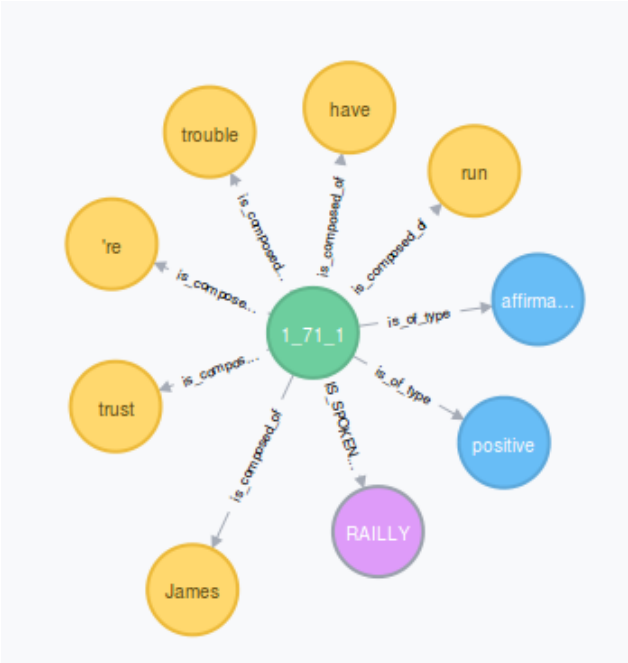

Figure 20: Sentence details

3.2 Architecture

If we take a look at Figure 21, we can see that the responsibilities given to R and Python are not the same. R is in charge of :

- The User Interface, both for the cosmetic and the behavioural part. These components will also be responsible for calling Python API through an elegant R API, thanks to the library *rPython*[11].
- Building the statistics on the new data we get on the web. For that matter, we use built-in R, as well as *ggplot2*[12] library. For cooccurrences graphs, we build cooccurrence matrices, that we read and display with Gephi[13].
- Taking into account the user's evaluations[14].

On the other hand, Python is in charge of :

- Retrieving information about the movies through OMDb API[15].

[11] *rpython.r − forge.r − project.org/*
[12] *ggplot2.org/*
[13] *gephi.github.io/*
[14] In a later version.
[15] *www.omdbapi.com/*

- Parsing the XML dataset[16] to build the neo4j graphbase[17].
- Finding the best answer for a given conversation (Confer 3.4).

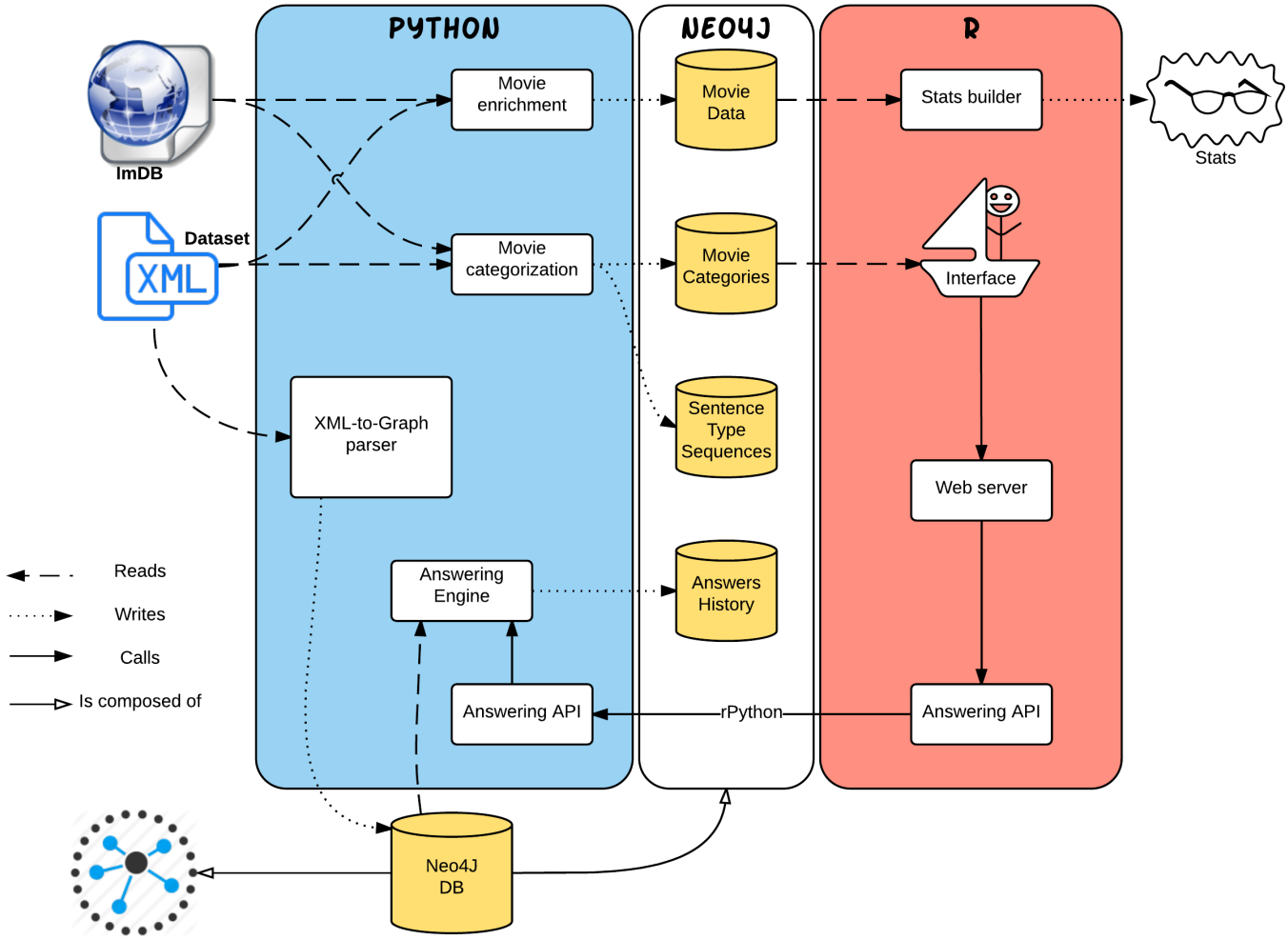

Figure 21: Anna's Architecture.

3.3 User Interface

3.3.1 Chosen technology

[16] *docs.python.org/2/library/xml.html*
[17] *py2neo.org/*

State of the Art Obviously, this is a Python project, because the point of the course is to discover NLP and Text-mining by using NLTK. So, we decided to naturally go and try our way in Python to get an interface for our chat-bot. In the litterature, there are two main approaches for doing that :

- Interfacing our bot on an IRC server, and using an IRC client as a User Interface.
- Developing our own interface.

As we wanted the user to be able to rate our bot's answers, the first solution was never really an option. An IRC (or so) client's interface doesn't give us room for non-textual interaction, and it is something which was important for us. The second solution, developing our own interface, is quite easy in Python, as long as you have more than just notions in web development. We explored two Python frameworks, *Django* and *Flask*. *Django* is obviously way too dense and heavy for our needs, so we tried our best with Flask. Considering that the user interface is not a very important component of our application, we decided that Flask was also a waste of time for us, and that we should do what we could already do.

Shiny Therefore we decided to propose a R user interface. R is, like Python, an open-source interpreted language, but it is specialized in Data Mining. Its main IDE, RStudio, proposes a package for building easily interactive web interfaces.

Interfacing R with Python The point of the project still being to develop a chat-bot based on NLP and text-mining (in Python, using NLTK), we figured out a way of connecting both, and built APIs to do it easily. Our app is called through R interpreter, which launches the User Interface. Then, the R server calls for the answering API, which eventually calls Python answering engine.

3.3.2 Result

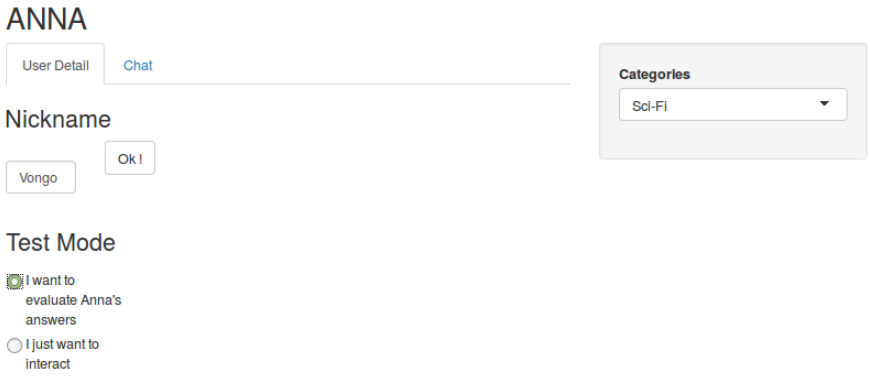

Figure 22: LogIn Interface, where you can also decide whether you want to evaluate Anna's answers.

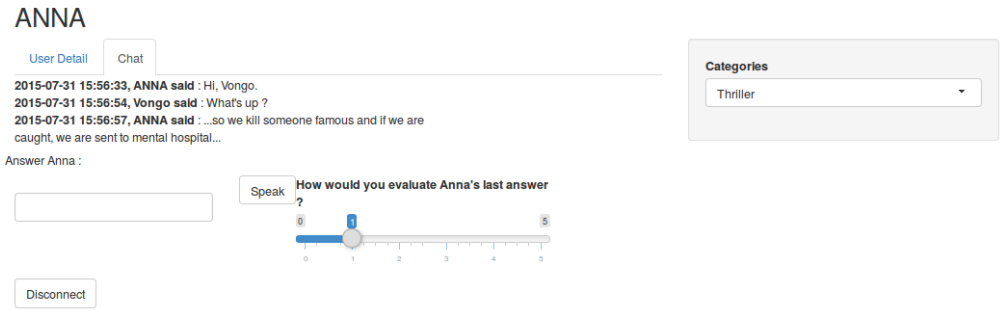

Figure 23: Chat Interface : a simple chat session.

3.4 Anna's answers

3.4.1 Version 1 : Random answers

There is not much to say about this first release. Like in all projects, we implemented a basic and stupid version to make sure that all communications processes (between R and Python for instance) are correctly working and that our queries to the graph are successful.
Anna's answer is selecting with the help of 3 random functions. First, we randomize the amount of movies in the graph : idMovie.
Then, the average of dialogues per movie beeing 175.60, we get a random dialogue by randomizing between 0 and 100 : idDialogue.
The sentence's id is also random, between 0 and the number of outgoing *IS_COMPOSED_OF*

relationships from the above dialogue : idSentence.
And the winner is : idMovie_idDialogue_idSentence !
This results on conversations which make no sense at all, see Figure 24 for an example.

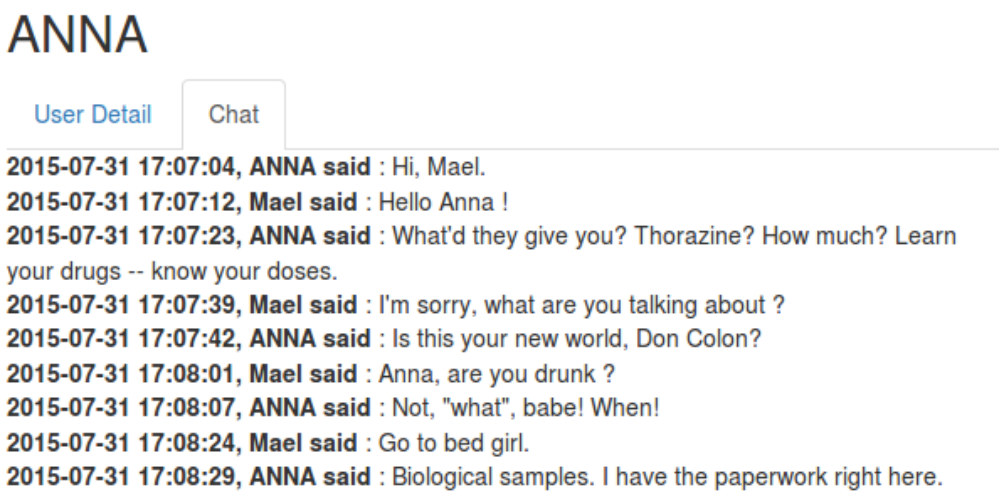

Figure 24: Random conversation from V1

3.4.2 Version 2 : Determined SentenceType

For the second release of the project we use likelihood to determine the best type of sentence to answer.
The first step is to run through all dialogues in our corpus and compute for each sequence of sentence's type the probabilities of the types of the following sentence. To do so we first start by computing probabilities with a sequence of length 1 meaning for each possible sentence's types (affirmative positive, affirmative negative, interrogative positive...) we look at the types of the following sentences and compute probability of occurrence for each type. Then we repeat the process for a sequence of length 2 (affirmative positive - affirmative negative, affirmative positive - interrogative positive...) then sequence of length 3 and so on.
The results are stored as a graph where each node represent a sentence's type associated with its probability of occurrence given previous sentence's type represented by its ancestors in the tree.
Figure 25 displays a 3-depth graph of probabilities in which we can see that, after the sequence "affirmative positive – interrogative positive", the probability of the SentenceType "affirmative positive" is of 69%.
Then when Anna is speaking with an user we use these statistics to answer a sentence whose type is the most likely to occur given the types of the previous sentences in the dialogue. It results on conversations which are correctly structured. See figure 26 for a creepy example.

Figure 25: Probabilities tree

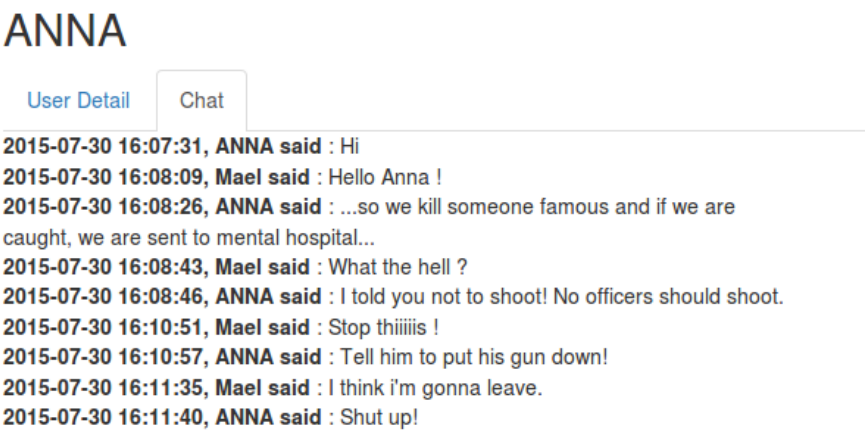

Figure 26: Conversation from V2

3.4.3 Version 3 : Determined movie genre

This version propose a new feature which allows the user to choose which kind of movie should be queried by Anna. The movies' categories have been added to the dataset through

the OMDb API, see 2.2 for more details about the API and the enrichment process.
The user can modify the chosen category at any time since the drop-down list is available on the discussion page of the UI.

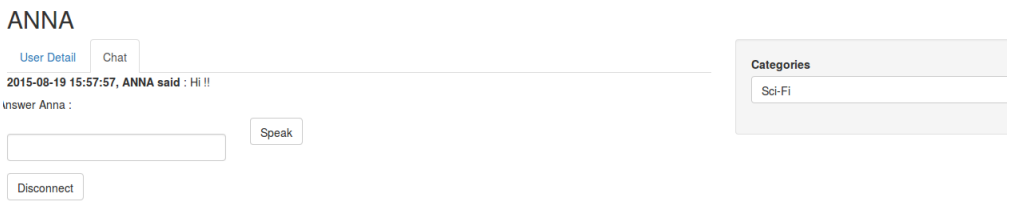

Figure 27: Movie's genre selection

3.4.4 Version 4 : Replace proper nouns

One of the NLP applications we wanted to try with this tool was to build generic sentences. Here, we tried to detect proper nouns and to replace them with some chosen values. The first problem we faced concerned NLP library which have some problems to handle correctly the process of classifying words into their right lexical categories (Noun, Verb...). For example all words beginning by a uppercase letter is considered by the library as a proper noun. Moreover the online documentation giving the tag associated to each lexical category contains some errors. Then, we figured out that replacing proper nouns isn't that simple: it depends of the structure of the sentence. For instance, in the sentence "Hello Jack, I'm Bob ! What do you think about our friend Lola ?" we got 3 NNP (Jack, Bob and Lola) but we can't replace them with the same value, it wouldn't make any sense. For a general overview on how we decide how to replace NNPs, see Figure 28.
We first split the initial utterance into sub-sentences, or naive propositions. This is done by detecting the presence of a grammatical separators such as "**.**", "**that**", "**:**", "**;**", "**?**", "**!**", "**,**", "**what**", "**-**" or "**...**".
Once we isolated the sub-sentences, we determine whether the NNP is the subject of the sentence ("I'm Bob !"), if someone is talking to him ("Hello Jack,") or if he's a third party ("What do you think about our friend Lola ?"). Given these three cases, we replace as follows :

Case 1 : Subject NNP becomes "ANNA".

Case 2 : Interlocutor NNP becomes "USERNAME"

Case 3 : Third party NNP becomes "EASTER"

Then the interface engine replaces those special chars by matching ones. Note that *ANNA* always becomes "Anna", so we could have avoided this step for this token. But we thought it was important to separate the grammatical identification of the NNP from the replacement with the proper noumns.

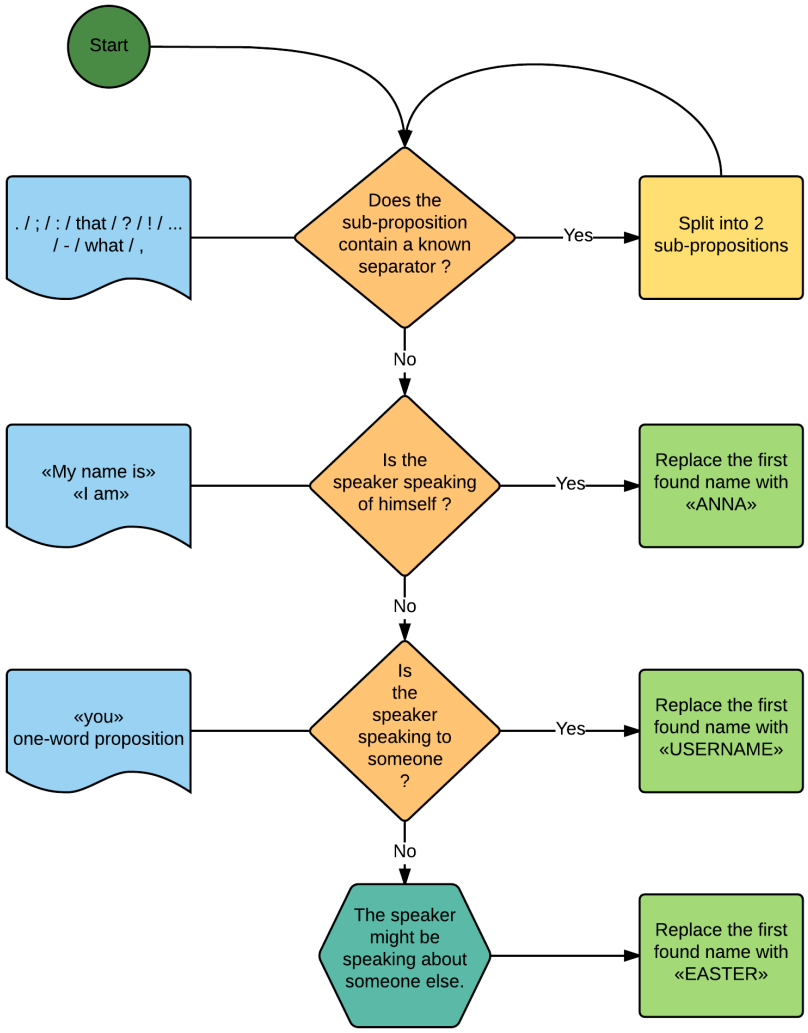

Figure 28: Decision tree on how to replace the names in a given utterance.

3.4.5 Version 5 : Tokens frequency

The last version we implemented make use of the tokens frequency in the historic sub-graph. We first compute a distribution of the tokens with a Part-of-speech tag of "NN*". It means that we consider :

NN Noun, singular or mass

NNS Noun, plural

NNP Proper noun, singular

NNPS Proper noun, plural

The tokens are ordered by their frequency and we process the list from the most frequent to the least. We then query our model and search for sentences containing the first token. If none is found, we search for the second one and so on ...

The whole dialog process try to use the "best" version (currently the fifth one), if it doesn't return a sentence, we jump to the second best, then the third and so on ... It results on the following code organization :

```
# Get Anna's answer
# Good SentenceType + movie category + tokens frequency version
labels = db.findNextSentenceType(5,3).split()
sentence = self.getAnswerWithGoodSentenceTypeAndCategoryAndSemanticRelevancy(5, category,
     labels)
if sentence is None:
    # Good SentenceType + movie category version
    sentence = self.getAnswerWithGoodSentenceTypeAndCategory(category, labels)
    if sentence is None:
        # Good SentenceType version
        sentence = self.getAnswerWithGoodSentenceType(labels)
        if sentence is None:
            # Random version
            sentence = self.getRandomAnswer()
```

Code 5: Answer code

4 Conclusion and further work

This project was a great opportunity for us to discover the NLP universe and to enhance our knowledge of the Python language. We also became more familiar with the graph-oriented databases and especially with the Neo4j solution and its query language Cypher.

We found out that NLP is a wide and really interesting topic which would be worth a more detailed study. This wasn't possible during this project and regarding our master-thesis preparation but we all agree on the fact that this subject could be a master-thesis itself.

We have some ideas for future work. It would be nice to further study the content of previous sentences in order to get a more consistent dialog, for instance upgrade our last version of Anna and try to find sentences composed of more than the most frequent token (the first two ones could be a nice start). We also didn't make use of the *MODE* node in the XML file, this could give the user the choice to dialog with an angry Anna or a happier one. Eventually, an evaluation of Anna's answer is possible and it would be nice to use it. We could imagine to plug a Machine-Learning solution so that Anna could learn from her mistakes (irrelevant answers). Another way we could try to explore is to represent the sentences as Markov chains and to set n-grams size to 3 or 4. This way, we could create relevant composite punchlines and be less dependant on the movie corpus.

References

[1] Banchs, Rafael E. "Movie-DiC: a movie dialogue corpus for research and development." Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2. Association for Computational Linguistics, 2012.

[2] Bird, Steven, Ewan Klein, and Edward Loper. Natural language processing with Python. " O'Reilly Media, Inc.", 2009.

[3] Behrang QasemiZadeh, Siegfried Handschuh : Text Mining Practical: Project Ideas (2009)

[4] Madnani, Nitin. "Getting started on natural language processing with Python." Crossroads 13.4 (2007): 5-5.