

Report - TeamLab Phonetics

Fabian Fey

Mtr.-Number: 2802772

fabian.fey@ims

Matthias Wilke

Mtr.-Number: 2865476

matthias.wilke@ims

Abstract

During this semester, we explored the basic functionality of *PyDial*, an open-source, multi-domain dialogue system toolkit written with Python. We encountered advantages and disadvantages of rule-based dialogue systems while integrating an entirely new domain (actors, producers, directors etc. from the renowned **Internet Movie Database** - the *IMDb* (Needham, 1990)) into the system. Trying to add a large-scale database (~1.5m entries) to *PyDial* revealed severe limitations of the system regarding file size and non-standardized input. We present the challenges we faced, as well as the most important conclusions for generally integrating new Domains into *PyDial*.

1 Introduction

The goal was to work with *PyDial* (Ultes et al., 2017) and improve or expand the System. *PyDial* is an open-source *Python Multi-domain Statistical Dialogue System Toolkit* developed by the *Cambridge University*¹ and can be used to create a dialogue system which is able to provide information and answer specific questions.

For this project we decided to add a new domain to the *PyDial* system and chose "movies" as the general theme of the new domain. The largest and most renowned resource for information about the movie industry is the *IMDb*² which includes multiple interconnected datasets about celebrities, movies, tv shows, and even video games. In this report we first explain the required steps to preprocess the gathered data and what limitations are set

by *PyDial* in the choice of data and concerning its expandability.

After the integration of our new domain, *PyDial* will be able to answer question like "When was Leonardo DiCaprio born" or "What movies did Bruce Lee star in?".

2 Data

Since the new domain was supposed to handle movie related information we chose the **Internet Movie Database** (*IMDb*) for two reasons, first its datasets can be used for personal and non-commercial use without any restrictions and second it is the most comprehensive and complete database for movie titles and information about the persons connected to them.

The *IMDb* database contains seven different datasets that hold information about the titles, the episodes, the ratings, the persons and much more, these datasets are additionally interconnected with an unique alphanumeric identifiers for each entry which enables the combined use of all datasets.

For this project we used two of the available datasets, the *title.basics.tsv* which holds movie related data like the title and the runtime, and the *name.basics.tsv* which holds the professions and on which movies the person has worked on.

2.1 Title Dataset

The first dataset is the *title.basics.tsv* (~430MB) file which contains a list of movies and the respective information, if an information is not available the missing data is labelled with a "\N". This file includes, in the order given in the file (See Table 1 for a short example) the following information:

¹<http://www.camdial.org/pydial/>

²<https://www.imdb.com/interfaces/>

tconst	tt1663202
titleType	movie
primaryTitle	The Revenant
originalTitle	The Revenant
isAdult	0
startYear	2015
endYear	\N
runtimeMinutes	156
genres	Adventure,Drama,History

Table 1: Example entry for the movie "The Revenant" taken from the *title.basics.tsv* file.

- An unique alphanumeric identifier of the title to cross reference to the other datasets (*tconst*)
- The type of the title, e.g. movie, short, tv series, video game, etc. (*titleType*)
- The primary title (*primaryTitle*)
- The original title (*originalTitle*)
- Adult film status (*isAdult*)
- The release year (*startYear*)
- The end year, mostly for TV series (*endYear*)
- The runtime of the title (*runtimeMinutes*)
- Up to three genres of the title (*genres*)

2.2 Name Dataset

The second dataset is the *names.basics.tsv* (~520mb) file which contains a list of persons and their respective information, if an information is not available the missing data is labelled with a "\N". This file includes, in the order given in the file (See Table 2 for a short example) the following information:

nconst	nm0000138
primaryName	Leonardo DiCaprio
birthYear	1974
deathYear	\N
primaryProfession	actor,producer,writer
knownForTitles	tt0993846,tt1375666, tt0120338,tt0407887

Table 2: Example entry for the actor "Leonardo DiCaprio" taken from the *names.basics.tsv* file.

- An unique alphanumeric identifier of the person to cross reference to the other datasets (*nconst*)
- The name of the person (*primaryName*)
- The birth year of the person (*birthYear*)
- The death year of the person (*deathYear*)
- The top three professions of the person (*primaryProfession*)
- Four of the most known titles of the person, encoded with the tconst from the *title.basics.tsv* file (*knownForTitles*)

3 Data Preprocessing

If a new domain is added to *PyDial* it is necessary to create a sql database which holds all the needed information, this database will then be imported into *PyDial*. In this Section we will explain the necessary steps to create the database for our domain and what steps are needed to import the database into *PyDial*.

Since not all the data provided by the *IMDb* datasets were needed for the goal of our project we were able to discarded some of the information (see Section ??). Additionally, due to encoding limitations on part of *PyDial*, all non-ASCII characters needed to be converted to ASCII characters.

All of the data preprocessing was done with *Python* version 3.6.

3.1 Preprocessing the title file

The *title.basics.tsv* file is a tab separated file with 5,112,841 entries and nine columns per entry. The first step was to replace every "\N" with "-1" which is easier to handle within the sql format. Since the collected data was too much for *PyDial* (see Section 5.1.4), we needed to excluded a lot of the data. Excluded were, all titles which were an adult film, if the title was published before the year 1980, if it had a total runtime of less than 10 minutes or no runtime at all, if it had no start year or if the title had no genres appended. After this pre selection, all the information per entry that was kept for later processing were the *tconst*, *titleType*, *primaryTitle*, *startYear*, *runtimeMinutes* and the *genres* entries.

All of the data was then stored, for later use, in a dictionary with the *nconst* as the key.

3.2 Preprocessing the names file

The *names.basics.tsv* file is a tab separated file with 8,659,623 entries and 6 columns per entry. The first step was to replace every "\N" with "-1". Again, due to the limitations of *PyDial* (see Section 5.1.4), we needed to discard most of the data provided by the dataset. We chose to excluded all persons that matched the following constraints: The person had only one or no movie title, no birth year, or no professions appended. After this pre selection, all the information per entry that was kept for later processing were the *nconst*, *originalName*, *birthYear*, *deathYear*, *primaryProfession* and the *knownForTitles* entries. This data was then stored in a dictionary with the *nconst* as key and the rest of the information as value, stored in a list.

Since each person has four movie titles appended in from of the unique alphanumeric identifier *tconst* we were able to easily append the information about each movie, compiled in Section 3.1, to each corresponding title per person.

3.3 Creating the database with sqlite

To be able to use the extracted information in *PyDial* a database needed to be created. All of the previously collected data could now be added to a sql database, the final database had the following six columns, "id" filled with the *nconst*, "name" which held the actor name in lower case letters, "birthyear", "deathyear", "professions" which held a string with 3 professions separated by a comma and "movieinfo" which was a string with the information of four titles the person worked on, also separated by a comma. An example entry for "Leonardo DiCaprio" is shown in Table 3.

Due to size limitations on part of *PyDial* we again needed reduce the amount of data in our database, wo chose to only use the first 1000 entries of our database. The reason and implications of size limitation will be discussed in Section 5.1.

3.4 Unicode

As mentioned before, *PyDial* only supports ASCII characters in the database, if a non-ASCII character was stored in the database *PyDial* would crash. We tried two different approaches to solve this problem, the first was to try and change the way *PyDial* handles the input from the database. Since the documentation for *PyDial* is limited this approach was quickly abandoned since it did not

id	nm0000138
name	leonardo dicaprio
birthYear	1974
deathYear	-1
professions	actor,producer,writer
movieinfo	the wolf of wall street : movie, 2013 (biography,comedy,crime, 180 minutes) inception : movie, 2010 (action,adventure,sci-fi, 148 minutes) titanic : movie, 1997 (drama,romance, 194 minutes) the departed : movie, 2006 (crime,drama,thriller, 151 minutes

Table 3: Example entry for the actor "Leonardo DiCaprio" as it would appear in the database. At this stage of preprocessing every string was converted to lower case letters.

yield the desired results. The second approach, which was consequently the used method, was to change every non-ASCII character to it's nearest ASCII equivalent. This meant that we needed to create a non-ASCII to ASCII mapping from scratch and consequently resulted in a total of 245 non-ASCII to ASCII conversions, some examples of theses conversions are listed in Table 4, the list of all conversions can be found in the provided code under *IMDB-data/merger.py* from line 23 to line 309.

'À' → 'A'	'Ž' → 'z'
'Á' → 'A'	'Û' → 'u'
'Œ' → 'OE'	'ř' → 'r'
'œ' → 'oe'	'İ' → 'I'
'ž' → 'z'	'þ' → 'th'

Table 4: Ten examples for some of the non-ASCII to ASCII conversions implemented in the preprocessing steps.

All non-ASCII characters in the database were looked up and replaced by ASCII characters from the mapping table. The code used to convert all non-ASCII characters to ASCII characters can be found in Listing 1.

```

1 def fold_to_ascii(text):
2     if isascii(text):
3         return text
4     text = ''.join([ASCII_REPLACEMENTS.
5         get(c, c) for c in text])
6     return unicode(unicodedata.normalize
7         ('NFKD', unicode(text)).encode('ascii
8         ', 'ignore'))

```

Listing 1: Converting non-ASCII characters to ASCII characters.

3.5 Importing the database

To import the created database into *PyDial*, we needed to pass the database to the *ontologyTool.py* script. This provides an interface in which we could choose which slot types will be linked to which of the six columns in our database. The available slot types are:

- Informable: information the user can inform the system about
- System requestable: information the system can actively ask the user
- Requestable: information the user can request from the system
- Binary: information which is in the form of a yes/no or true/false

The "birthyear", "deathyear" and "name" columns were selected to be *Informable* and *System requestable*, the "birthyear", "professions", "deathyear", "name" and "movieinfo" were selected to be *Requestable* and no column was selected as *Binary* since we do not have binary data in our database. The script then creates a *.json* which holds the information which column is linked to which slot type.

4 Additions to the code

When adding a new domain to *PyDial* some additional code and configurations need to be added to the source code. The first addition is the configuration file *IMDB.cfg* in the */config* folder, this configuration file must be provided with the execution of *PyDial* and provides *PyDial* with the name of the *Messages.txt* file.

The *Messages.txt* file, in our case named *IMDBMessages.txt*, provides *PyDial* with a set of responses for the system. These range from simple

statements like "Thank you, goodbye." and "What are you looking for" to more complex statements like "I've found \$A (*\$B). They're known for the following movies: \$E." which provides the user with the requested person, his/her birthyear and some information about the titles they worked on.

The last addition needed to be implemented in the *semi/RegexSemi_IMDB.py* file which provides the system with an extensive set of regular expressions. These help the system to identify which information is requested by the user, for example the profession of a person is checked with the regular expression "(profession|work|role)" and "(movie)|(info)|(information)|(film)" to tell the system to recall information about the titles a specific person has worked on. An example dialogue that uses all these additions can be found in Table 8.

5 Discussion

5.1 Limitations

Even though *PyDial* offers some sophisticated mechanics and is generally well-engineered, its most significant drawback is its poor performance. Furthermore, the documentation provided on the official *PyDial* website is rather sparse, making the practical use of *PyDial* a tedious trial-and-error task. This is the main reason we failed to simply interconnect our two databases instead of having to merge them into a single one. For the small-scoped applications *PyDial* is targeted at, the system even seems somewhat over-engineered and overly complicated, reducing its accessibility even further. Of course, we did not only encounter difficulties with *PyDial*: The *IMDb* itself is not fully compatible with what we wanted to achieve with our system.

5.1.1 The IMDb

Browsing the data in the *IMDb* quickly reveals the redundancy of the huge set of titles, as Table 5 shows. Furthermore, the star ratings, which are one of the core features of the *IMDb*, are not included in the title database, but in a separate file. To reduce the required amount of preprocessing, we did not merge these ratings into our database. In hindsight, we believe that the ratings could have provided a good way of assessing 'relevance' for the included titles³.

³Of course, even some movies with low quality ratings are very well-known by broad audiences, which would induce even more fine-grained inspection of the data.

tconst	type	original title	start year	genres
tt1752318	tvEpisode	Countdown Show: 6 Health Plagues in Women Over 40	2010	Talk-Show
tt0660506	tvEpisode	Heiße Reifen, Scharfe Kurven	2005	Crime
tt0449193	video	Big Guns	2001	Adult
tt0451003	videoGame	Brothers in Arms: Road to Hill 30	2005	Action,Biography,Drama
tt0451095	movie	Kolechko zolotoe, buket iz alykh roz	1994	Drama
tt0451252	tvSpecial	UFC 47: It's On!	2004	Sport
tt0451427	short	Moines et guerriers	1909	History,Short,War
tt7285726	video	O Fresh Ft. Slim Thug: We Up to Something	2017	Music,Short

Table 5: Examples of irrelevant titles in the *IMDb*.

The following peculiarities in the title dataset become obvious from Table 5;

- For TV shows, every single episode is listed separately.
- The database contains adult films, video games, sports shows, music videos, and other entries that are completely irrelevant to our task.
- Non-English titles are rather unlikely to be searched for by international users of our system.
- Due to the sheer age of some productions, many entries are potentially unknown to audiences nowadays.

On the other hand, the examples from Tables 1, 2, and 3 show another problem of the *IMDb*: Only four popular movies are referenced for each celebrity’s entry in the name dataset. In the shown example, *The Revenant*, the movie which earned *Leonardo DiCaprio* his first *Oscar*, remains entirely unmentioned because of the high number of successful productions DiCaprio had previously starred in. While four movies are a feasible number for getting a quick overview of a person’s previous work, we would have wished for a more complete movie index in our system due to its otherwise rather limited informational benefit.

"Stone" Robert A Stone III
'Carl' Nai Peng Wang
A. Chandler Warren Jr.
James P. O'Connor
Hélène Ninerois
Lajos Kathy Horváth
Miasha

Table 6: Poorly standardized names in the *IMDb*.

Regarding the name dataset, we identified a fundamental design flaw in the *IMDb*: All names are stored as a single string (see Table 6). Thus, users of our system need to precisely enter all names including abbreviations and nicknames with the exact punctuation in a single prompt, which is quite a considerable source for errors since *PyDial* does not account for spelling errors. This is especially aggravated by the punctuation inconsistencies in the *IMDb*, which can also be seen in Table 6.

5.1.2 Storing conversational context

The slot values for information previously entered by the user is sometimes not accessed correctly by *PyDial*. In our case, the system forgot about the name, lifespan, and professions of the current person as soon as the movie info was requested by the user. The system then printed the slot variables instead of their values, as can be seen in Table 8.

5.1.3 PyDial development

It seems that the authors of *PyDial* have somewhat abandoned their project in spite of a general lack of alternatives and its promising architecture. The fact that *PyDial* neither supports unicode strings nor runs in Python 3 supports this observation. Some features of Python 3 even cause syntax errors when trying to run *PyDial* on machines without Python 2 installed.

5.1.4 Database access

When running the built-in Ontology tool, *PyDial* creates a redundant, identical copy of the provided database. Additionally, all slots are copied into a JSON (Crockford, 2002) file⁴ for inexplicable reasons instead of harnessing the speed and accessibility of SQLite databases on the fly.

⁴For the informable slots, every single database entry is duplicated to the JSON file, while only the slot reference is copied for other slots.

We believe that this architectural peculiarity is the main factor responsible for the bad performance of *PyDial*. The sample databases distributed with *PyDial* are ridiculously small (1kb to 30 kb) compared to the 1.5M entries that we originally had in the *IMDb* dataset, resulting in a file size of around 700mb. Since running a dialogue on the original 700mb *IMDb* dataset revealed that not even a single query could be answered within 20 min of waiting time, further preprocessing of our database was required (See Table 7). The next iteration resulted in a database size of 30mb with 98312 entries, yet *PyDial* response times are still high at half a minute per query. Reducing the file size this drastically was achieved excluding the following entries during data selection.:

- productions from before 1980
- movies with a runtime less than ten minutes
- actors having starred in less than two movies

feature restriction	database v1	database v2
length	< 5 min	< 10 min
start year	none	< 1980
runtime	none	none
genres	none	none
starred movies	none	< 2

Table 7: Entries excluded from the database (first vs. second iteration).

5.1.5 Expandability

Despite all efforts to make *PyDial* as expandable as possible, trying to add an entirely new domain revealed some architectural flaws of the system:

- The primary key of the database always needs to be called '*name*'. We don't see any necessity for this convention, since the system could simply look for the primary key independently from its identifier. This approach requires needless preprocessing of existing databases. This issue could be circumvented by simply accessing the database on the fly instead of an external JSON file.
- For some reason, the creators of *PyDial* designed their system entirely for ASCII text. Non-ASCII characters in any of the database's slots result in a fatal encoding

error crashing the program. We encountered this problem early on in our endeavours, since many international actors and motion pictures are featured in the *IMDb*. Due to the international popularity of people like '*Daniel Brühl*' or '*Céline Dion*', we couldn't simply exclude all non-ASCII names from the database. Instead, we mapped unicode characters to their corresponding ASCII symbols (i.e. 'ü' to 'ue' or 'é' to e) as mentioned in 3.4.

- Considering the rather modular approach of *PyDial*, adding all the necessary regular expressions and rules for new keywords proved to be enormously time consuming, once more questioning the feasibility of rule-based approaches. While the system works perfectly well within the boundaries of the pre-defined questions and answers, it completely fails when encountering typographical alternatives to known words or even entirely unknown ones.

6 Conclusions

Our attempts of including an entirely new, large-scale domain into *PyDial* allow some suggestions for the future development and practical use of *PyDial* and the *IMDb* datasets in general.

6.1 Working with *PyDial*

- We urgently suggest the implementation of unicode support in *PyDial*. This is a rather minor change with huge effect on the amount of possible applications for *PyDial*.
- *PyDial* currently processes all slot values as lower case strings. While this makes sense for taking user input, it would look more natural if names etc. were output with their original capitalization.
- Python 3 has been around for long enough to finally consider upgrading *PyDial* from Python 2 to 3, benefiting from various new language features and maintaining support on newer machines.
- Even though the original authors had consciously made the design decision to rely on external JSON files instead of directly accessing the sqlite database, we propose trying to omit the JSON files. *PyDial* greatly suffers

from performance issues when ran on large databases, which we in part attribute to the indirect data access.

- The documentation about *PyDial* is still very sparse. In order to raise popularity and comprehensibility of the framework, proper documentation, tutorials, and a lively community forum are required. The lack of documentation was the main reason we had to merge our datasets instead of using the interconnected database files, which would have saved a lot of preprocessing.
- Even though the problem of *PyDial* forgetting conversational context may have been due to errors on our end, we think that the system needs to store more information from previous user input. In a real-world conversation, speakers don't respecify affected entities in every sentence following their mention. This is an interesting prospect which was also tackled by another group in this TeamLab.
- To us, the convention of having to name the primary key of a database 'name' for use with *PyDial* seems entirely unnecessary and contradictory to the whole point of primary keys in sqlite databases. This restriction should be abolished for more flexibility.
- In our opinion, adding a new domain required too many code modifications in *PyDial*. Providing external configuration files is a more pleasant and user-friendly way of modifying standard behaviours of a system. While the prompt templates can be provided with text files, regular expressions need to be constructed within the code for keyword recognition. Covering all possible case discriminations for different spellings etc. is a tedious task and a major source of errors. We suggest circumventing this problem by the implementation of a spellcheck system suggesting similarly spelled items if no entries are found for the user's input.⁵

⁵For example, one could use the Levenshtein distance (Levenshtein, 1966) as a very simple approach for spellchecking.

6.2 Using IMDb datasets

- Even though it is unlikely for the *IMDb* to change how entries are stored, it is important to be aware of the particularities of the dataset.
- The name dataset needs thorough standardization regarding punctuation, abbreviations, nicknames, and even the order of names.
- It would have been convenient if the average movie ratings were saved directly in the title file. This information could save lots of preprocessing, since the ratings constitute a good indicator of relevance.
- For the full set of movies per celebrity, costly cross-referencing to the *title.principals.tsv* is required. The name dataset only contains 4 titles per person that are not chosen in any publicly transparent way.

With these suggestions and conclusions, we hope to facilitate the entry to working with both *PyDial* and the *IMDb*, fuelling more sophisticated dialogue systems and making information about the movie business more accessible.

Notes

The *PyDial* code for this Project is located in /mount/arbeitsdaten31/studenten1/team-lab-phonetics/2018/groups/3/pydial_Project/pydial. The *IMDb* datasets and the script used to create the database are located in /mount/arbeitsdaten31/studenten1/team-lab-phonetics/2018/groups/3/pydial_Project/IMDB-data.

Acknowledgements

Information courtesy of IMDb (<http://www.imdb.com>). Used with permission.

```
-bash-4.4$ python pydial.py chat config/IMDB.cfg
*** logfile: IMDB-seed397162272-00.0.chat.log ***
*** Chatting with policies ['IMDB']: ***
```

Prompt > Welcome to 'Who' sInTheBiz', the IMDb (internet movie database) dialogue system. Ask me about actors, directors, producers, composers, etc. from the movie industry.

User > I am looking for an actor named Leonardo DiCaprio

Prompt > I've found a person named leonardo dicaprio. Do you want to know when they were born or died, what they worked as and what movies they are famous for?

User > Can you tell me when he was born?

Prompt > I've found a person named leonardo dicaprio (* 1974).

User > Can you tell me what he worked as?

Prompt > I've found the actor, producer, writer named leonardo dicaprio .

User > Can you tell me which movies he worked on?

Prompt > I've found the \$D named \$A. They' re known for the following movies:| the wolf of wall street : movie, 2013 (biography,comedy,crime, 180 minutes) || inception : movie, 2010 (action,adventure,sci-fi, 148 minutes) || titanic : movie, 1997 (drama,romance, 194 minutes) || the departed : movie, 2006 (crime,drama,thriller, 151 minutes) |.

Table 8: An example for a dialogue using the new database.

References

- Douglas Crockford. 2002. [Json: Javascript object notation](#).
- Vladimir Levenshtein. 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707.
- Col Needham. 1990. [Internet movie database](#).
- Stefan Ultes, Lina Rojas-Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gašić, and Steve Young. 2017. [Pydial: A multi-domain statistical dialogue system toolkit](#). In *Proceedings of ACL 2017, System Demonstrations*, pages 73–78, Vancouver, Canada. Association for Computational Linguistics.