Four Different Ways to Build a Chatbot About Movies

SPINNINGBYTES

Erland Xhoxhaj, Yusuf Koc, Sandro Panighetti, Matteo Togni

Dirk Von Grünigen, Martin Weilenmann, Hans Daniel Graf, Daniel Zürrer

Fernando Benites, Jan Deriu, Nico Neureiter, Pius von Däniken

Mark Cieliebak, Walter Eich, Stephan Neuhaus, Kurt Stockinger

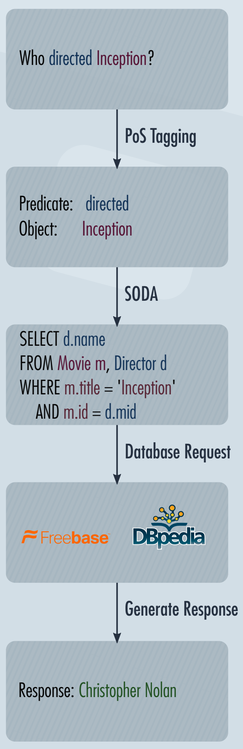
Zürcher Hochschule für Angewandte Wissenschaften

zhaw

Chatbots have recently become a focus of broad interest. Many systems claim to provide technology to develop your personal bot. However, the technologies behind the scene are either kept secret or trivial.
We examine four different, technologically challenging approaches to dialogue systems. Each system engages the problem from a different perspective: two rule-based factoid question answering systems, showing the ability of an agent to interact with an existing knowledge base; a conversational agent, able to be trained on any pair of sentences by using deep learning; finally, we explore in this context the impact of micro-services on system performance.

Rule Based Question Answering

We developed a bot which is able to answer basic factoid questions on movies, like "Who directed Inception?" or "When was Brad Pitt born?". It parses the question using natural language processing and machine learning techniques and extracts the relevant information from an existing database. This project shows a possible way to replace formal database queries by natural language questions. In that way we could provide an interface for non-technical users to databases from virtually any domain.

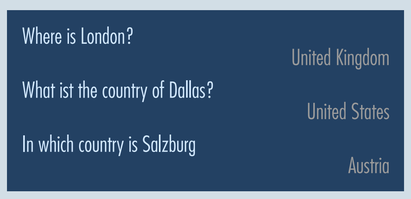


System Description

- Part-of-Speech Tagging (Stanford CoreNLP):
  We first extract the subject, object and predicate from the input sentence. We also use synonyms for these words.
- Bag-of-Words similarity (Gensim):
  We compare the input sentence to a set of known questions. If we find a match, we can directly infer the question type.
- Simple Object Data Access (SODA):
  SODA allows to search an entire database for keywords. They are matched to the database's metadata and transformed into an SQL query. We pass the previously found keywords (e.g. subject and predicate) to SODA and use the extracted data to generate our response.
- Database:
  We use the well known knowledge bases Freebase and DBpedia as our information sources.

Results

This chatbot demonstrates a way to implement a natural language interface to structured information. To the right you can see one example from the movie domain (flow chart) and three examples on geographical information.

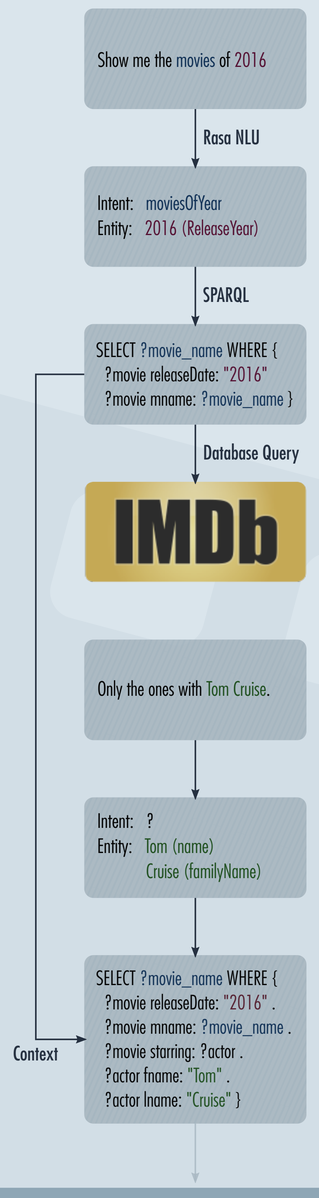


Example questions (left) and the answers of our chatbot (right).

Modelling Conversation Context

In a second project we approached the same problem - automatically answering factoid questions about movies - with different tools and a different focus. We developed a system that keeps track of the context of the current conversation and is able to answer follow-up questions.

Imagine a conversation where you are talking about a specific actor and want to know his date of birth. Instead of restating the title of the movie, you will probably just ask "Who is the director of this movie?" and expect your dialogue partner to remember which movie you were talking about. We present a simple approach to resolve this kind of state dependency in certain scenarios, by simply combining database queries.



System Description

- Rasa NLU:
  Entity extraction and intent classification.

```
Input: Show me movies by Clint Eastwood!
Intent: moviesOfDirector
Entities: (Clint, name), (Eastwood, familyName)
```

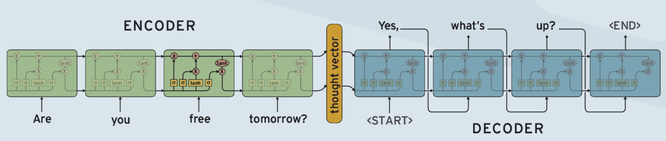
- GraphDB, SPARQL:
  We use data from IMDb, stored as RDF Triples in GraphDB:

```
636307, firstName, Clint
636307, lastName, Eastwood
Million Dollar Baby, director, 636307
```

- Context Tracking:
  We keep track of the context by combining the current SPARQL query with the previous one. E.g. if the previous query asked only for movies from a specific director, we can keep this filter for the new query as well.
  It is not obvious, when we want use this context and when to treat a new query independently. We decided to only use the context when the current input shows no clear intent.

Learning Dialogues End-to-End

We developed a neural network which learns how to respond to a dialogue partner. In contrast to the two chatbots above, the focus was not on correctly answering factoid questions, but rather on learning end-to-end to interpret input and generate appropriate output without any external source of information.

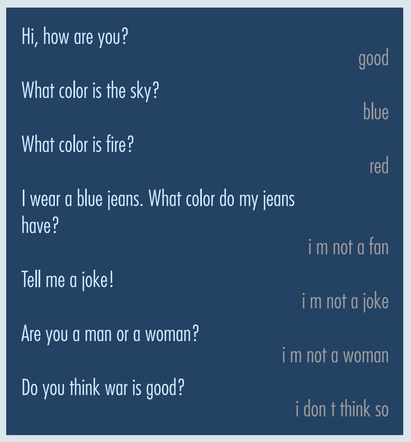


System Description

- Word embeddings:
  The input sentence is transformed into a sequence of word vectors.
- Sequence-to-sequence model:
  The model consists of two long short-term memory networks (LSTMs): an encoder, which maps the input sequence to a thought vector, and a decoder, which maps the thought vector to the output sentence. The encoder and decoder are trained together on a set of training dialogues.
- Training set:
  We trained the model once on a set of reddit.com comments and once on the OpenSubtitles dataset. Empirically the results obtained from the OpenSubtitles set were more convincing.

Results

In contrast to the other presented chatbots, this one does not extract the information for a response from a given knowledge base, but learns how to answer questions in general. This makes it hard to evaluate the performance of the chatbot. We think the fact that the model learns to give sound responses, which fit the type of the question, is exciting by itself. Also, in many cases the system actually gives correct or interesting responses (see examples).



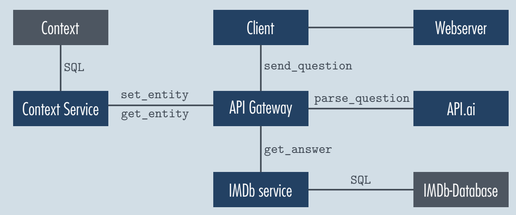
Example questions (left) and the answers of our chatbot (right).

Microservice Architectures

The recent developments and interest in cloud computing lead to increased use of microservice architectures, where classical monolithic applications are replaced by independent services, which rely on very little communication. This kind of architecture can have great advantages in scalability, robustness and innovative potential.

We implemented three different versions of a chatbot and investigated how suitable each approach is in different settings. The three versions were:

1) a classical monolithic implementation.
2) a microservice implementation with an aggregated data interface (few services, diagram).
3) a microservice implementation with atomic data access (more services).



Component diagram of the microservice architecture with aggregate data interface. The atomic variant replaces the IMDb service by separate interfaces for different kinds of requests (get_movies_by_director, get_movies_by_genre,...).

Results

We performed experiments in four different settings:

- Low or high workload: The chatbot is confronted with 2 or 100 queries per second respectively.
- Simple or complex queries: Queries filtering by one or multiple criteria (e.g. "Show me action movies from 2016") respectively.

Our results show that the performance of each architecture strongly depends on the setting. In particular splitting the data-interface into atomic services, requires an application-level-join for complex queries. This results in a significant drop in performance. Overall, we conclude that there is no general answer to the question monolith vs. microservices. The appropriate architecture depends on the requirements of the application.