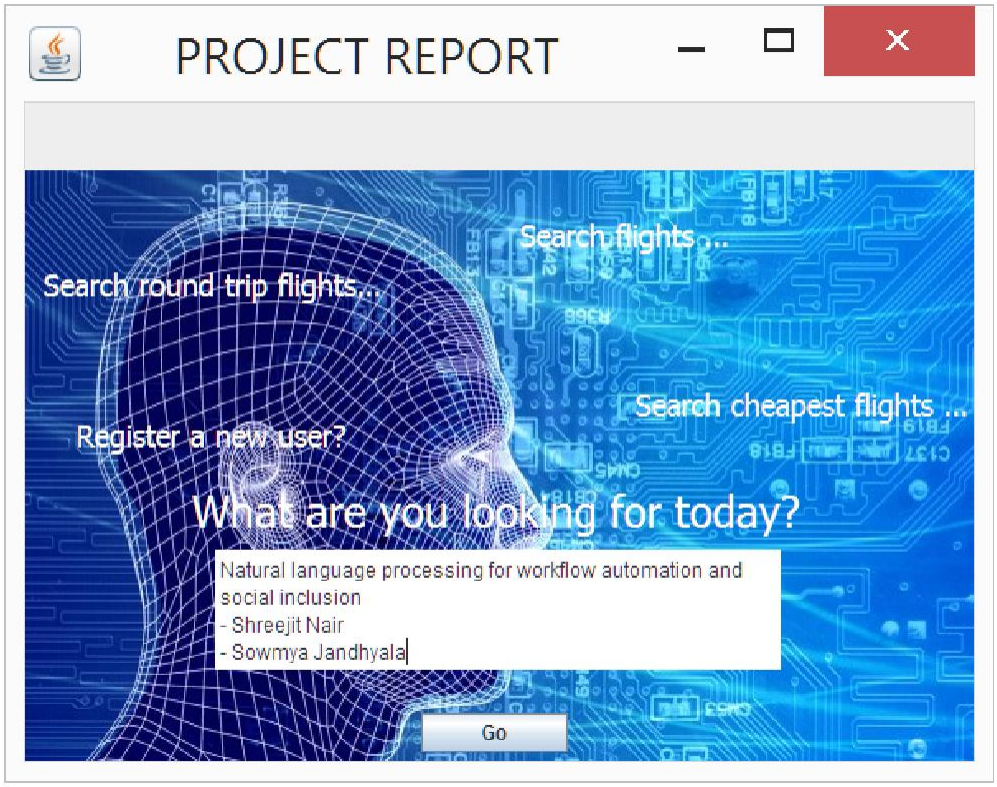
Natural language processing as a tool for workflow automation and social inclusion



- Shreejit Nair
- Sowmya Jandhyala

Problem Statement and Motivation:

The project is an attempt at building conversational interfaces using Natural Language processing techniques towards building a more human like approach in interacting with an underlying application. We attempt at building a simple interface where humans could key in what they need from the application instead of having to laboriously understand on directly interacting with the application.

For our sample application we use a Flight reservation system on this dummy website for our testing purposes:

http://newtours.demoaut.com/

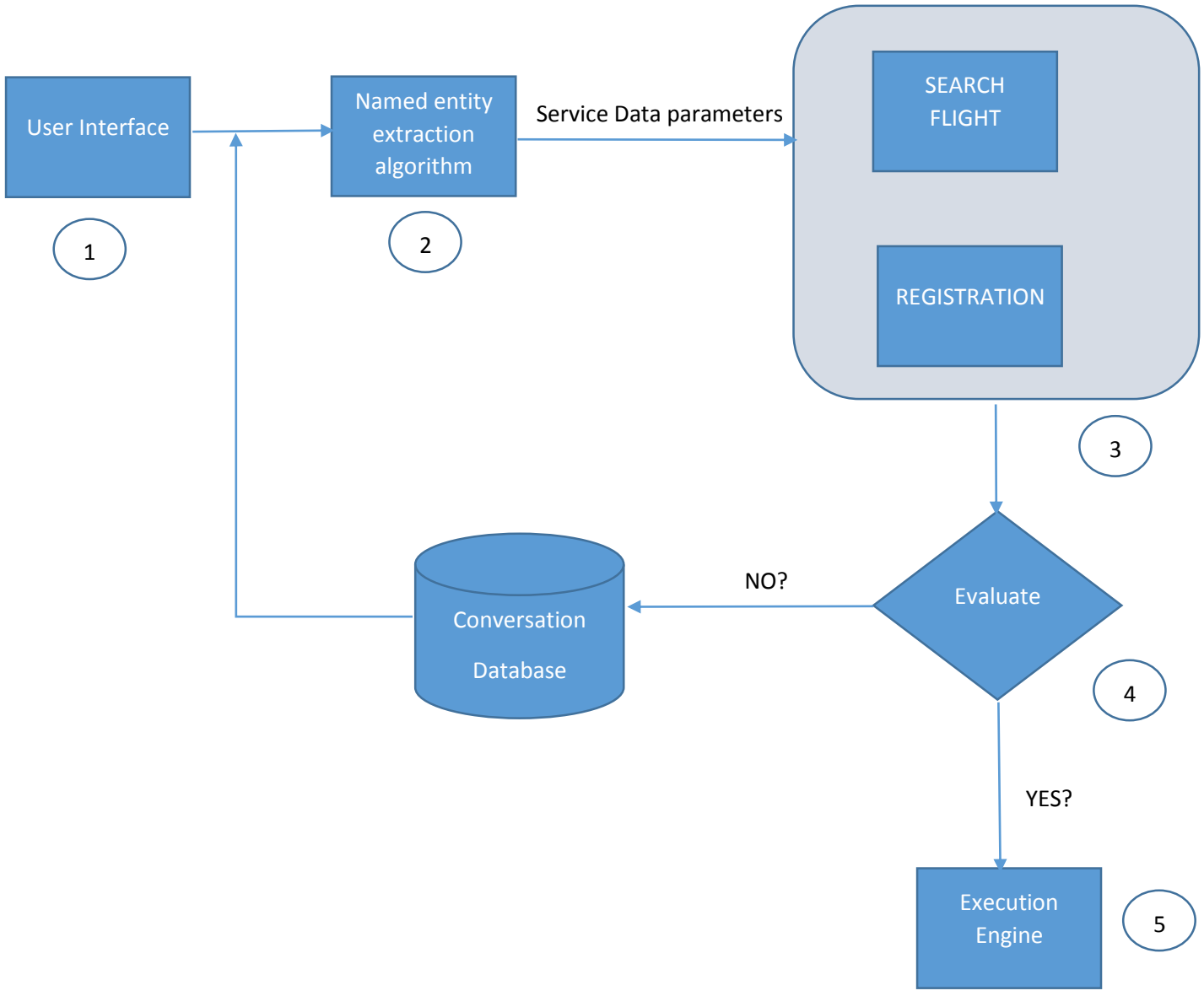
The project acts as a window to social inclusion since the moment we allow people to communicate directly with an application in natural language we remove the additional challenges of understanding the intricacies of an application and allows for more users from many walks of life to be able to consume the services provided by the application.

High Level Description:

1. The NLP algorithm designed is responsible for performing named entity based extraction of meaningful information from the given input text provided by the user.
2. User does not have to directly interact with the application but uses a conversational interface provided to him where he can key in information regarding his search request.
3. The conversational bot algorithm now converts this natural text and maps it into several entities that allow it to call an underlying service in the application and passing the required data parameters to it. E.g. in our case it would call a service like SearchFlight() routine and pass in data parameters to that service.
4. We now run an evaluation algorithm that determines if we have all the required input data from the user to successfully execute the routine. For doing this we split every service into Mandatory parameters (we cannot execute the service without getting this information from the user), Optional parameters (e.g. number of passengers, if not mentioned we assume the value is one).
5. For any missing mandatory parameters to the service, the conversational interface will ask input from the user for the missing data and save the previous conversation context in the database and run the evaluation algorithm on it again.
6. Once all the mandatory parameters to a service are acquired, the bot executes the underlying instruction set and runs an execution engine to perform the required steps as required by the user.

7. The bot then provides a response back to the user stating that the required action has been completed or not.

Technical workflow diagram:



1. User enters natural text on a user interface describing his request. Eg. Search a flight for me from New York to Boston.
2. We run a named entity extraction algorithm on this data using a custom trained Stanford NER Tagger which uses CRF for Named entity recognition and try to identify the underlying service requested by the user and the required data parameters. e.g. SearchFlight(source,destination,date of journey).
3. We execute this service on the underlying software ecosystem and map the service entities to services exposed by the software. In our case we are exposing services like SearchFlight and User Registration.
4. Next, run an evaluation engine which executes these against instructions fed to the system which have a mapping for the required mandatory and optional parameters for every service.
5. If successfully evaluated, we executed the underlying instructions for the service.
6. If the execution fails, we save the conversation context to a database and run the named entity extraction algorithm once again to get more data from the user.

Named Entity Recognition CRF Algorithm:

We use the Stanford NER algorithm that uses CRF and custom train our corpus of hand crafted annotated data and teach it to identify our services and data parameters.

Below is a sample tok file representation that was used to train the algorithm.

Sample .tok file representation:	
Get	O
flight	SERVICE
from	O
New York	SOURCE
to	O
Chicago	DESTINATION

Classes : SERVICE, SOURCE, DESTINATION, LAST NAME, FIRST NAME, PHONE, EMAIL ADDRESS

Conditional random fields (CRFs) are a probabilistic framework for labeling and segmenting structured data, such as sequences, trees and lattices. The underlying idea is that of defining a conditional probability distribution over label sequences given a particular observation sequence, rather than a joint distribution over both label and observation sequences.

To determine whether a token is a name, our system uses weak evidence from a number of sources. The main consideration in deciding which information sources to use was the difficulty associated with creating and maintaining such sources. A secondary consideration was to keep the learned models as small as possible. The knowledge sources, encoded as a set of 29 features, are: Word Level Features: Language- or genre-specific cues can sometimes be exploited to provide evidence for name detection (e.g., in English, names are often capitalized). The following features are used in encoding tokens and the system learns which of these correlate strongly with names: (1) all-uppercase, (2) initial-caps, (3) all-numbers, (4) alphanumeric, (5) single-char, (6) single-s (if the token is the character “s”), and (7) single-i (if the token is the character “I”). Individually, none of the local word-level features are very effective: the strongest individual feature is all-caps; it flags 474 tokens in the training set (containing 50,416 tokens, of which 3,632 are names). Of these, 449 are actually names, the rest being non-name acronyms such as “CEO” and “PC”, yielding an F1-score2 of only 21 (Precision = 94; Recall = 12). The feature initial-caps is similar, flagging 5,650 words, of which 3,572 are names, leading to an F1-score of 82 (Precision = 73; Recall = 94). Note that not all capitalized words are names, and that not all names are capitalized (e.g., “van Gogh”). For English text, capitalization, in conjunction with reliable end-of-sentence boundary detection, is a good indicator for names. However, determining sentence boundaries is difficult since common boundaries such as periods, question- and exclamation-marks can occur in many different contexts [16; 13]. While the system does not explicitly contain rules for sentence boundary analysis, by using contextual cues, it can account for many sentence boundaries. Dictionary Look-Up: Another weak heuristic for determining whether a particular token is a name is to check whether it can be found in a dictionary. Since many names are not valid English words, this resource can identify some potential names. The dictionary used in our experiments was the standard spelling dictionary available on most UNIX systems; it contained 45,402 words, of which 6,784 were capitalized, and were discarded as names. The remaining 38,618 tokens contained multiple morphological variants of the same word (further decreasing the number of unique root forms). Finally, since a number of English names are also part of the regular vocabulary (e.g., “mark”, “baker” and “stone”), name detection using only evidence from the dictionary is not very reliable: the F1-score for the dictionary module alone on our training set was only 64. Part-of-Speech Tagger: Part-of-Speech (POS) tags can be used by other modules to reason about the roles and relative importance of words/tokens in various contexts. In this system, we used the Brill tagger for POS tagging3 . Brill reports approximately 97% overall accuracy for words in the WSJ corpus for the tagger [6; 7]. Its performance is lower on the named-entity task: on our training data, the tagger obtained an F1-score of 83 (P = 81, R = 86); consistent results are reported in [1]. The following POS tags were used as features by the machine learning component of our system: (1) determiner, (2) foreign-word, (if the token is one that the tagger has not seen), (3) preposition, (4) adjective, (5) noun, (6) proper-noun, (7) personal-pronoun, (8) possessive-pronoun, (9) verb, (10) WH-pronoun (which, what, etc.), (11) unknown-POS. Punctuation: Robust name detection probably requires that the system capture contextual syntactic information. (At the very least, to disambiguate capitalization cues due to

sentence boundaries.) The system learns syntactic patterns that may indicate named entities. Section 5 discusses the effects of varying the size of this contextual window. The following punctuation characters are encoded as features: (1) comma; (2) period; (3) exclamation mark; (4) question mark; (5) semi-colon; (6) colon; (7) plus or minus sign; (8) apostrophe; (9) left parenthesis; (10) right parenthesis.

Experiments and Results:

One of the major challenges was the lack of training data to represent how humans would enter text to an underlying service. To achieve this both students worked separately on building a training + testing corpus of data unknown to each other to reduce bias or overfitting towards the testing inputs.

We handcrafted around 30-40 different ways of training data to train the CRF algorithm to identify proper services and data parameters for the underlying service calls. The second major challenge was incomplete data from the human user. Our evaluation engine used to repeatedly fail since the required mandatory parameters for a service were not provided by the human user. We had to build a conversation layer on top of this to help bump up our accuracy and not expect the user to provide all the necessary information at the very first instance of interacting with the conversational bot.

Service/Action	Training Set	Test Set	Accuracy
Search Flight	34	43	64%
Register User	24	32	52%

We observed that depending on the complexity of the service model the accuracy of text initially entered by the user deteriorates. For extremely complex services the accuracy of the input data tends to taper downwards. More or less once users understand what the service provides the accuracy tends to stabilize across various services.

Overcoming lower accuracy by building conversational layers:

One of the major challenges as already suggested was the the incomplete information provided by the user to execute an underlying service. For e.g. if we want to successfully invoke a service called as SearchFlight() at the minimum we need the source, destination and date of journey information to successfully execute the steps for the user. To overcome this gap we prompt the user with responses to fill in this missing information by storing his conversation in a database and only prompting the user for the missing mandatory information.

This is achieved by storing the conversational information in a sqlite database table.

Service Mapping Table:

SERVICE ID	SERVICE NAME
1001	SearchFlight
1002	RegisterUser

The above table provides a mapping of the services available for the user exposed as a natural language service. When our input data annotates a service tag as one of the values mapped to the service name we are able to identify that the service required by the user is being as a natural language service.

Service Mandatory Parameter mapping:

SERVICE ID	DATA PARAMETERS
S1	{mandatory: source,destination} {optional: no. of passengers}
S2	{mandatory: name, address} {optional: gender}

This table provides a mapping in terms of the mandatory and optional parameters tagged with every service which is maintained as a json textual information.

Conversation table

CONVERSATION ID	USER PARAMETERS	SERVICE ID
1001	{parameter name: value}	S1
1002	{parameter name: value}	S2

Conversation entered by the user which is annotated by our CRF algorithm is split into parameter values with a service id mapping and saved into a conversational table which now has a mapping of the parameters received from the user, values of these parameters and the service name the user wants to execute. Our evaluation engine now picks up these parameter values and tries to map it into the mandatory parameters required for the service and does a lookup from the service and data parameter mapping table. The conversational bot now is able to identify the missing parameter names and prompts the user to fill in this information. Once it receives the next batch of information it again updates the conversation table and the evaluation engine is executed again. Once, the evaluation engine receives all the required parameters it calls the Execution engine which executes the instructions for the service along with the data passed by the user. This conversation layer build up acts as a major factor for bumping up our accuracy and not expecting the user to provide all the information on the very first go and gradually communicates with the user to build up and extract all the required information from the user.

Technologies used:

Python Tkinter for the user interface, custom trained Stanford CRF algorithm for named entity recognition, Python routines for evaluation and execution engine, Sqlite database to maintain mappings and saving conversations.

Further Improvements:

1. One of the major improvements could be speech recognition and machine translation where a user is not expected to enter text but can converse directly with the conversational layer using audio signals and a machine translation layer which can convert input from the user into language the machine understands. This would act as a major boost towards social inclusion where users can communicate in a language of their choice.
2. Building more intelligent responses from the conversational layer. As we noticed from the data above the current responses from the conversational layer are rigid and strictly asks for prompts from the user asking them to fill out the missing mandatory parameter information for the underlying service engine to get executed. We can make use of Information Retrieval techniques to send out more intelligent responses to the user. For.e.g while booking a flight from New York to Boston wouldn't it be great if the bot provides us with information on weather forecasts from Boston?

Summary:

There has been a proliferation of devices all around us. Mobile apps, Kiosk based systems, desktop, web solutions and connected devices. We sincerely believe that building conversational interfaces is the future when it comes to interacting with theses underlying complex systems where the user can communicate with a device completely in a language that he understands without having to deal with the internal complexities of the machine he is interacting with.

Natural language process combined with Information retrieval can work wonders. We are in exciting times.