Roman Urdu Chat-bot

Roman Urdu Chat-bot

Session 2016-2017

Project Supervisor

Sir Farhan Ahmed Siddiqui

Submitted By

Amaan Ullah (EP-1450008)

Syed Ahmer Mashhadi (EP-1450106)

Fardan Akhter (EP-1450020)

Department of Computer Science

University of Karachi

Karachi

Table Of Contents

A report submitted to the

Department of Computer Science

In partial fulfillment of the requirements for the

Degree

Bachelors of Science

In

Software Engineering

By

Farhan Ahmed Siddiqui

University of Karachi

Aug 15, 2017

Acknowledgements

We truly acknowledge the cooperation and help by Sir Farhan Ahmed Siddiqui, Student Advisor at Department of Computer Science University of Karachi. He has been a constant source of guidance throughout the course of this project. We would also like to thank Sir Badar Sami and Ms. Humaira Tariq for their help and guidance in understanding Natural Language Processing and Artificial Intelligence. We are also thankful to each of our group members for their hard work and dedication in completing this project.

(Signed)

Farhan Ahmed Siddiqui

Date

Aug 15, 2017

Abstract

Time keeps progressing and so does technology. The situation we are in right now, is the time when Artificial Intelligence is the work in progress. With each step a human expects ease in his life by creating substitutes for the repetitive jobs that can be handled by an intelligence itself. Meanwhile this being the focus of some, others are using AI to create solutions of real world problems by creating chat assistants and chat-bots. A chat-bot is a piece of software that handles responding to a human or another bot by guessing and learning intelligent answers. Each time it gets to interact, it either learns or perform the specified query. Artificial Intelligence isn't just for creating solutions but for bringing the future closer this era. A chat-bot will answer for your query and perform what for you, what would take you a little more time to perform. Meanwhile, the chat-bot isn't smart enough to work on a general domain, it does have to ability to integrate itself into a bigger more complex domain to solve multiple problems and multiply for larger domains. Each time a bot interacts, it will create a response, while the learning deals with telling the bot how not to respond to a situation. Behind the bot runs multiple data mining and analytical algorithms which create its decision making on the basis of previous and on coming replies. There is no limit to a bot, by claiming it to be perfect in some or all situations. The results we conducted according to this project were based on the intelligence and learning of a chat-bot in a specific domain. The bot responded well and we trained it as much as we could to handle the possible cases that it was making mistakes at.

Chapter 1: Introduction

1.1 Artificial Intelligence (AI)

Artificial intelligence (AI) is an area of computer science that emphasizes the creation of intelligent machines that work and react like humans. Some of the activities computers with artificial intelligence are designed for include:

- Speech recognition
- Learning
- Planning
- Problem solving

Another way to explain AI can be as follows.

Artificial intelligence is a branch of computer science that aims to create intelligent machines. It has become an essential part of the technology industry.

Research associated with artificial intelligence is highly technical and specialized. The core problems of artificial intelligence include programming computers for certain traits such as:

- Knowledge
- Reasoning
- Problem solving
- Perception
- Learning
- Planning
- Ability to manipulate and move objects

1.2 Machine Learning

Machine learning is a type of artificial intelligence (AI) that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. The basic premise of machine learning is to build algorithms that can receive input data and use statistical analysis to predict an output value within an acceptable range.

Machine learning algorithms are often categorized as being supervised or unsupervised. Supervised algorithms require humans to provide both input and desired output, in addition to furnishing feedback about the accuracy of predictions during training. Once training is complete, the algorithm will apply what was learned to new data. Unsupervised algorithms do not need to be trained with desired outcome data. Instead, they use an iterative approach called deep learning to review data and arrive at conclusions. Unsupervised learning algorithms are used for more complex processing tasks than supervised learning systems.

1.3 What are Chat-bots (Chatterbots)?

A chatterbot is a program that attempts to simulate the conversation or "chatter" of a human being. Chatterbots such as "Eliza" and "Parry" were well-known early attempts at creating programs that could at least temporarily fool a real human being into thinking they were talking to another person. A chatterbot can be thought of as the spokesperson for an artificial intelligence (AI).

Chatterbots can be text-based or voice-based. Typically, chatterbots use machine learning algorithms to improve the accuracy of their natural language and voice recognition capabilities. As the end user interacts with the bot, deep learning programming gets better at predicting what the appropriate response is when communicating with an end user. Chatterbots can be stateless or stateful. A stateless chatterbot approaches each interaction as if it was with a new user. A stateful chatterbot is more sophisticated; it can review past interactions and frame new responses in context.

Chatterbots have been used in instant messaging (IM) applications and online interactive games for many years. Chatterbots can be added to a buddy list or provide a single game player with an entity to interact with while awaiting other "live" players. If the bot is sophisticated enough to pass the Turing Test [1], the person may not even know they are interacting with a computer program.

1.4 The Turing Test

In artificial intelligence (AI), the Turing Test is a method for determining whether or not a computer is capable of thinking like a human. The test is named after Alan Turing, an English mathematician who pioneered artificial intelligence during the 1940s and 1950s, and who is credited with devising the original version of the test. According to this kind of test, a computer is deemed to have artificial intelligence if it can mimic human responses under specific conditions. In Turing's test, if the human being conducting the test is unable to consistently determine whether an answer has been given by a computer or by another human being, then the computer is considered to have "passed" the test.

In the basic Turing Test, there are three terminals. Two of the terminals are operated by humans, and the third terminal is operated by a computer. Each terminal is physically separated from the other two. One human is designated as the questioner. The other human and the computer are designated the respondents. The questioner interrogates both the human respondent and the computer according to a specified format, within a certain subject area and context, and for a preset length of time (such as 10 minutes). After the specified time, the questioner tries to decide which terminal is operated by the human respondent, and which teminal is operated by the computer. The test is repeated many times. If the questioner makes the correct determination in half of the test runs or less, the computer is considered to have artificial intelligence, because the questioner regards it as "just as human" as the human respondent.

The Turing Test has been criticized, in particular because the nature of the questioning must be limited in order for a computer to exhibit human-like intelligence. For example, a computer might score high when the questioner formulates the queries so they have "Yes" or "No" answers and pertain to a narrow field of knowledge, such as mathematical number theory. If response to questions of a broad-based, conversational nature, however, a computer would not be expected to perform like a human being. This is especially true if the subject is emotionally charged or socially sensitive.

1.5 Applications of Chat-bots

Chatbots are incredibly valuable for Marketers! A few reasons:

- **They increase customer lifetime value:** Chatbots help drive incremental revenue through your highest engagement channels, Facebook Messenger for example, by offering targeted promotions to your audience. They also enable clear insight into customer issues, and provide your brand the opportunity to provide immediate and automated solutions to their problems (this is a Marketing issue as well as a Support issue).
- **Chatbots help maintain brand consistency:** Powered by natural language processing (NLP) technology, conversational chatbots provide a consistent, personalized brand experience, as well as answers to routine customer questions.
- **Monitor your brand's perception:** Use engagement analytics to track what your customers say about, and want from, your brand.
- **Collect and integrate product feedback:** Thanks to NLP technology, chatbots "listen" for market feedback—such as product feedback, content needs, or emerging trends—and can automatically aggregate this information for stakeholder access and action.
- **They help save development cycles:** Stop wasting money trying to pull people into your ecosystem—instead, push your content where your audience is already active and through apps they already use and trust.

Chapter 2: Natural Language Processing (NLP)

2.1 Natural Language Processing and AI

Natural language processing (NLP) [2] is the ability of a computer program to understand human speech as it is spoken. NLP is a component of artificial intelligence (AI).

The development of NLP applications is challenging because computers traditionally require humans to “speak” to them in a programming language that is precise, unambiguous and highly structured or, perhaps through a limited number of clearly-enunciated voice commands. Human speech, however, is not always precise -- it is often ambiguous and the linguistic structure can depend on many complex variables, including slang, regional dialects and social context.

Current approaches to NLP are based on machine learning, a type of artificial intelligence that examines and uses patterns in data to improve a program's own understanding. Most of the research being done on natural language processing revolves around search, especially enterprise search.

Common NLP tasks in software programs today include:

- Sentence segmentation, part-of-speech tagging and parsing.
- Deep analytics.
- Named entity extraction.
- Co-reference resolution.

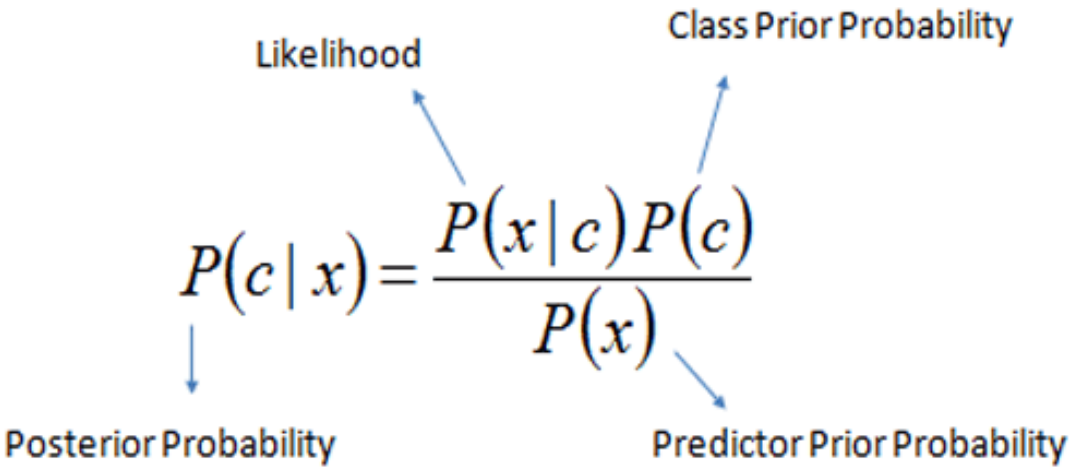
The advantage of natural language processing can be seen when considering the following two statements: "Cloud computing insurance should be part of every service level agreement" and "A good SLA ensures an easier night's sleep -- even in the cloud." If you use national language processing for search, the program will recognise that *cloud computing* is an entity, that *cloud* is an abbreviated form of cloud computing and that *SLA* is an industry acronym for service level agreement.

2.2 The Naïve Bayes Algorithm

Commonly used in Machine Learning, Naive Bayes is a collection of classification algorithms based on Bayes Theorem [3]. It is not a single algorithm but a family of algorithms that all share a common principle, that every feature being classified is independent of the value of any other feature. So for example, a fruit may be considered to be an apple if it is red, round, and about 3″ in diameter. A Naive Bayes classifier considers each of these “features” (red, round, 3” in diameter) to contribute independently to the probability that the fruit is an apple, regardless of any correlations between features. Features, however, aren’t always independent which is often seen as a shortcoming of the Naive Bayes algorithm and this is why it’s labeled “naive”.

Although it’s a relatively simple idea, Naive Bayes can often outperform other more sophisticated algorithms and is extremely useful in common applications like spam detection and document classification.

In a nutshell, the algorithm allows us to predict a class, given a set of features using probability. So in another fruit example, we could predict whether a fruit is an apple, orange or banana (class) based on its color, shape etc. (features).



$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

The Naive Bayes classifier relies on a small number of parameters to make predictions. In training, it learns the probability of each class label present in its training set, and the conditional probability of each value of each feature present in its training given each class label. Consequently, the total number of parameters it learns is upper bounded by the sum of the number of class labels present and the product of the number of class labels present and the number of feature values (not features) present.

Some other classifiers, such as Support Vector Machines and the Logistic Regression Classifier, learn even smaller numbers of parameters. These two learn just one parameter per feature present in the training set (not feature value). Fortunately, the Naive Bayes classifier can get away with not storing a very large number of the conditional probabilities it uses for prediction. Rather than explicitly storing a parameter corresponding to each possible combination of class label and feature value, it can simply decline to store any parameter that corresponds to the conditional probability of a feature value given a class label for which that feature value did not appear in training. Later when making predictions, it can assume that the conditional probabilities corresponding to all combinations of class labels and feature values for which it did not store a parameter in training is 0. Note that handling such combinations requires smoothing to avoid predicting probabilities of 0 for all class labels. This property makes the Naive Bayes classifier extremely efficient at modeling sparse feature sets, such as n-grams derived from corpora of natural language.

2.3 Natural Language Processing Models

The Natural Language Processing models or NLP models are a separate segment which deals with instructed data.

Automatic summarization

Produce a readable summary of a chunk of text. Often used to provide summaries of text of a known type, such as articles in the financial section of a newspaper.

Coreference resolution

Given a sentence or larger chunk of text, determine which words ("mentions") refer to the same objects ("entities"). Anaphora resolution is a specific example of this task, and is specifically concerned with matching up pronouns with the nouns or names that they refer to. The more general task of coreference resolution also includes identifying so-called "bridging relationships" involving referring expressions. For example, in a sentence such as "He entered John's house through the front door", "the front door" is a referring expression and the bridging relationship to be identified is the fact that the door being referred to is the front door of John's house (rather than of some other structure that might also be referred to).

Discourse analysis [4]

This rubric includes a number of related tasks. One task is identifying the discourse structure of connected text, i.e. the nature of the discourse relationships between sentences (e.g. elaboration, explanation, contrast). Another possible task is recognizing and classifying the speech acts in a chunk of text (e.g. yes-no question, content question, statement, assertion, etc.).

Machine translation

Automatically translate text from one human language to another. This is one of the most difficult problems, and is a member of a class of problems colloquially termed "AI-complete", i.e. requiring all of the different types of knowledge that humans possess (grammar, semantics, facts about the real world, etc.) in order to solve properly.

Morphological segmentation

Separate words into individual morphemes and identify the class of the morphemes. The difficulty of this task depends greatly on the complexity of the morphology (i.e. the structure of words) of the language being considered. English has fairly simple morphology, especially inflectional morphology, and thus it is often possible to ignore this task entirely and simply model all possible forms of a word (e.g. "open, opens, opened, opening") as separate words. In languages such as Turkish or Manipuri,[6] a highly agglutinated Indian language, however, such an approach is not possible, as each dictionary entry has thousands of possible word forms.

Named entity recognition (NER)

Given a stream of text, determine which items in the text map to proper names, such as people or places, and what the type of each such name is (e.g. person, location, organization). Note that, although capitalization can aid in recognizing named entities in languages such as English, this information cannot aid in determining the type of named entity, and in any case is often inaccurate or insufficient. For example, the first word of a sentence is also capitalized, and named entities often span several words, only some of which are capitalized. Furthermore, many other languages in non-Western scripts (e.g. Chinese or Arabic) do not have any capitalization at all, and even languages with capitalization may not consistently use it to distinguish names. For example, German capitalizes all nouns, regardless of whether they refer to names, and French and Spanish do not capitalize names that serve as adjectives.

Natural language generation

Convert information from computer databases into readable human language.

Natural language understanding

Convert chunks of text into more formal representations such as first-order logic structures that are easier for computer programs to manipulate. Natural language understanding involves the identification of the intended semantic from the multiple possible semantics which can be derived from a natural language expression which usually takes the form of organized notations of natural languages concepts. Introduction and creation of language metamodel and ontology are efficient however empirical solutions. An explicit formalization of natural languages semantics without confusions with implicit assumptions such as closed-world assumption (CWA) vs. open-world assumption, or subjective Yes/No vs. objective True/False is expected for the construction of a basis of semantics formalization.[7]

Optical character recognition (OCR) [5]

Given an image representing printed text, determine the corresponding text.

Part-of-speech tagging

Given a sentence, determine the part of speech for each word. Many words, especially common ones, can serve as multiple parts of speech. For example, "book" can be a noun ("the book on the table") or verb ("to book a flight"); "set" can be a noun, verb or adjective; and "out" can be any of at least five different parts of speech. Some languages have more such ambiguity than others. Languages with little inflectional morphology, such as English are particularly prone to such ambiguity. Chinese is prone to such ambiguity because it is a tonal language during verbalization. Such inflection is not readily conveyed via the entities employed within the orthography to convey intended meaning.

Parsing

Determine the parse tree (grammatical analysis) of a given sentence. The grammar for natural languages is ambiguous and typical sentences have multiple possible analyses. In fact, perhaps

surprisingly, for a typical sentence there may be thousands of potential parses (most of which will seem completely nonsensical to a human).

Question answering

Given a human-language question, determine its answer. Typical questions have a specific right answer (such as "What is the capital of Canada?"), but sometimes open-ended questions are also considered (such as "What is the meaning of life?"). Recent works have looked at even more complex questions.[8]

Relationship extraction

Given a chunk of text, identify the relationships among named entities (e.g. who is married to whom).

Sentence breaking (also known as sentence boundary disambiguation)

Given a chunk of text, find the sentence boundaries. Sentence boundaries are often marked by periods or other punctuation marks, but these same characters can serve other purposes (e.g. marking abbreviations).

Sentiment analysis

Extract subjective information usually from a set of documents, often using online reviews to determine "polarity" about specific objects. It is especially useful for identifying trends of public opinion in the social media, for the purpose of marketing.

Speech recognition

Given a sound clip of a person or people speaking, determine the textual representation of the speech. This is the opposite of text to speech and is one of the extremely difficult problems colloquially termed "AI-complete" (see above). In natural speech there are hardly any pauses between successive words, and thus speech segmentation is a necessary subtask of speech recognition (see below). Note also that in most spoken languages, the sounds representing successive letters blend into each other in a process termed coarticulation, so the conversion of the analog signal to discrete characters can be a very difficult process.

Chapter 3: Roman Urdu and Urdu Grammar

3.1 Roman Urdu

Urdu has borrowed significant number of words from Arabic and Persian glossary. The borrowed words are written faithfully in Urdu as those are written in its original language. But Urdu speakers do not pronounce the words in the same way as they are pronounced by an Arab or Persian person. For example, an Arabic speaker can differentiate between Alif "ا" and Ain "ع", but for an Urdu speaker both have the same sound. As we will see in next section, it causes many problems in Roman to English transliteration [6].

The roman script is not an official standard for writing Urdu text, but it is widely used. The reason is influence of English language in Urdu speaking community. English is widely used in offices, business circle and education sector.

English is also the default language of user interfaces of computers. Any person can install Urdu support on the computer and can write email and use chat applications in Urdu, but roman script is widely used when there comes a need of a language for the informal communication over internet.

Unlike Urdu script, the roman script for Urdu does not have any standard for spelling the words. A word can be written in various forms not only by distinct writers but also by the same writer at different occasions. Specially, there is no one to one mapping between Urdu letters for vowel sounds and the corresponding roman letters. The support of Urdu script on the computers is getting better by the usage of Unicode character set and Open Type fonts. The availability of Urdu script support demands for roman to Urdu translator that can convert the text written in roman Urdu into proper Urdu script.

3.2 Issues in Roman-Urdu Transliteration

- The roman script for Urdu does not follow any standard.
- Roman to Urdu transliteration cannot be implemented by simple one to one replacement of characters.
- Consonants in Urdu script are mostly not to be found in Roman script.
- The transliteration of roman vowel letters is also a complex issue as there is no one to one mapping in roman and Urdu script.
- We cannot predict the syllable boundary either in roman script or in Urdu script.

3.3 Multiple Urdu letters for Roman Consonants

Both 'common' and 'mango' are written as aam in roman script. This implies that Roman script does not have characters for all consonants that are used in Urdu script

The same sound Persio-Arabic characters are not the only problem in roman to Urdu mapping. Different letters (or letter sequences) for different Urdu consonant can map to

single roman equivalent. Urdu letter(s) "چ" (chay) and "چه" (chay-do_chasmi_hay) both are written as ch in roman script as in chor چور 'thief' and churi چھری 'knife'.

Table 1.a gives the list of same sound Urdu characters and corresponding roman characters. The first column of the table has a roman character or character sequence. The second column has all the Urdu letters that are used as equivalent of the roman character in the previous column. The last column has the most frequently used Urdu character against each roman character(s). These characters can be used as the default transliteration for the roman character(s) specified in front of the each equivalent symbol.

Roman Letter(s)	Equivalent Urdu Letters	Most Common Urdu Equivalent
a	"ا" (alif), "ع" (ain), "ء" (hamza), "آ" (alif_mad)	"ا" (alif)
ch	"چ" (chay), " چه" (chay-do_chasmi_hay)	"چ" (chay)
d	"د" (daal) , "ڈ" (ddaal)	"د" (daal)
gh	"غ" (ghain), "گھ" (gaaf-dochasmi)	"غ" (ghain)
h	"ح" (hay), "ہ" (hay_gol), "ھ" (hay_dochasmi)	"ہ" (hay_gol)
kh	"خ" (khay) , "کھ" (kaf-dochasmi)	"خ" (khay)
k	"ک" (kaaf), "ق" (qaaf)	"ک" (kaaf)
r	"ر" (ray), "ڑ" (rray)	"ر" ray
s	"ث" (say), "س" (seen), "ص" (suad)	"ر" ray
t	"ت" (tay), "ط" (tuay), "ٹ" (ttay)	"ت" (tay)
y	"ی" (chooti-ye), "ء" (hamza)	"ی" (chooti-ye)
z	"ذ" (zaal), "ز" (zay), "ض" (zuaad), "ظ" (zuay), "ژ" (zay)	"ز" (zay)
n	"ن" (noon), "ں" (noon-ghunna	"ن" (noon)

Table 1.a

The above table shows that the roman letter ‘h’marks Urdu “ح” (hay), “ہ” (gol_hay) and “ھ” (do chasmi hay). When it comes after an aspiratable consonants like ‘b’, ‘p’ or ‘k’, it represents aspiration i.e. do chashmi hay. The Urdu letter “ق” (qaf) is written as Roman ‘q’. Occasionally, it can be written as ‘k’ according to the choice of the writer. Similarly, frequently used mapping of all roman characters to Urdu characters is listed in the above table.

3.4 Multiple roman letters for Urdu vowels

Like many other issues, roman Urdu inherits this problem from roman orthography of English. In English script, a roman vowel character can map to different vowel sounds e.g. ‘a’ maps to short vowel in the English word ‘about’ and to long vowel in the word ‘father’.

Table 2: Roman letters for Urdu Vowels

Urdu letter	**Roman Equivalents**
Zabar	a
Zer	I
Paish	U
alif	a, aa
bari-ye	ai , ay, ei, e
chooti ye	ee, ey, i , ie
vao	oo, au, ou, o, u

Table 1.b

3.5 Morphology

The concept of Morphology comes from the statement “every word in every language is composed of one or more morphemes”.

For example the English word builder consists of two morphemes: build (with the meaning ‘construct’) and –er (which indicates the entire word functions as a noun with the meaning ‘one who builds’). Similarly the word horses is made up of the morphemes horse (name of an animal) and –s (with the meaning ‘more than one’). Examples of Urdu words and their morphemes are given below

Word	Corresponding Morpheme
Kursiyan	Kursi(chair) + yan (meaning ‘more than one’)
Naliaq	Na(indicates negation) + liaq(intelligent)
Tameezdar	Tameez(manners) + dar(indicates presence of)

Table 1.c

3.6 Word Formation

The morphemes of a word are constrained to appear in certain combinations and orders.

These constraints need to be considered while forming rules.

For example, English words 23 such as piti-less-ness and un-guard-ed-ly are valid, while *piti-ness-less and *un- guard-ly-ed are not valid due to incorrect order. Similarly, insaan-i-yat is a valid Urdu word, but *insaan-yati is not.

3.7 Roots, Affixes and Bases

Complex words typically consist of a root and one or more affixes. The root morpheme carries the major component of the word’s meaning and belongs to what is known as the lexical category

By definition affixes do not belong to the lexical category and are always bound morphemes. Morphemes which occur only before other morphemes are called prefixes. Similarly, suffixes are those morphemes which occur only after other morphemes.

Prefix	**Suffix**
Prefix + Root	Prefix + Root
Na-liaq	kam-tar

Chapter 4: Rule-Based Chat-bots

4.1 Machine Learning Capabilities of a Bot

Rule-based systems [7] provide an adaptable method, suitable for a number of different problems. Rule-based systems are appropriate for fields, where the problem area can be written in the form of if-then rule statements and for which the problem area is not extremely too great. In case of too many rules, the system may become difficult to maintain and can result in decreased performance speeds.

A classic example of a rule-based system is a domain-specific expert system that uses rules to make deductions or narrow down choices. For example, an expert system might help a doctor choose the correct diagnosis based on a dozen symptoms, or select tactical moves when playing a game. Rule-based systems can be used in natural language processing or to perform lexical analysis to compile or interpret computer programs.

A major difficulty in evaluating segmentation algorithms is that there are no widely-accepted guide- lines as to what constitutes a word, and there is therefore no agreement on how to "correctly" segment a text in an unsegmented language.

4.2 Rule-Based Chat-bots

Rule based chatbots have already achieved commercial success stories such as (i.e. KLM's chatbot in the Facebook messenger). Rule based chatbots do not create grammatical mistakes, but the disadvantage is that they cannot answer new questions that are not identified in the data warehouse.

We can have a rule-based bot that understands the value that we supply. However, the limitation is that it won't understand the intent and context of the user's conversation with it. For example: If you are booking a flight to Paris, you might say "Book a flight to Paris" and someone else might say that "I need a flight to Paris" while someone from another part of the world may use his native language to have the same context.AI-based or NLP-based bot identifies the language, context and intent and then it reacts based on that. A rule based bot only understands a pre-defined set of options.

4.3 Natural Language Understanding (NLU)

Natural Language Processing (NLP) and Natural Language Understanding (NLU) [8] attempt to solve the problem by parsing language into entities, intents and a few other categories. Different NLP platforms may have different names however the essence is more so the same.

Below are mentioned the categories:

- ✓ **Agents** correspond to applications. Once you train and test an agent, you can integrate it with your app or device.

- ✓ **Entities**: represent concepts that are often specific to a domain as a way of mapping natural language phrases to canonical phrases that capture their meaning.
- ✓ **Intents** represent a mapping between what a user says and what action should be taken by your software.
- ✓ **Actions** correspond to the steps your application will take when specific intents are triggered by user inputs. An action may have parameters for specifying detailed information about it.
- ✓ **Contexts** are strings that represent the current context of the user expression. This is useful for differentiating phrases which might be vague and have different meaning depending on what was spoken previously.

4.4 Training

The transformation-based algorithm involves applying and scoring all the possible rules to training data and determining which rule improves the model the most. This rule is then applied to all applicable sentences, and the process is repeated until no rule improves the score of the training data. In this manner a sequence of rules is built for iteratively improving the initial model. Evaluation of the rule sequence is carried out on a test set of data which is independent of the training data.

4.5 Some key concepts

Agents, NLU (Natural Language Understanding) modules for applications. Their purpose is to transform natural user language into actionable data and can be designed to manage a conversation flow in a specific way. Agents are platform agnostic. You only have to design an agent once and then can integrate it with a variety of platforms using our SDKs and Integrations, or download files compatible with Alexa or Cortana apps.

Machine Learning, allows an agent to understand user inputs in natural language and convert them into structured data, extracting relevant parameters. In the API.AI terminology, the agent uses machine learning algorithms to match user requests to specific intents and uses entities to extract relevant data from them. The agent learns from the data you provide in it (annotated examples in intents and entries in entities) as well as from the language models developed by API.AI. Based on this data, it builds a model (algorithm) for making decisions on which intent should be triggered by a user input and what data needs to be extracted. The model is unique to the agent. The model adjusts dynamically according to the changes made in agent and in the API.AI platform. To make sure that the model is improving, the agent needs to constantly be trained on real conversation logs.

Intent represents a mapping between what a user says and what action should be taken by software.

Entity represents concepts and serves as a powerful tool for extracting parameter values from natural language inputs. The entities that are used in a particular agent will depend on the parameter values that are expected to be returned as a result of agent functioning.

Chapter 5: API.ai

5.1 Basics

The process an API.AI [9] agent follows from invocation to fulfillment is similar to someone answering a question, with some liberties taken of course. In the example scenario below, the same question is being asked, but we compare the "human to human" interaction with a conversation with an API.AI agent.

- Bill's friend Harry wants to ask him a question. So as not to be rude, Harry says "Hello" to Bill first
- In order to start a conversation with an agent, the user needs to invoke the agent. A user does this by asking to speak with the agent in a manner specified by the agent's developer.
- Harry asks Bill "What's the weather supposed to be like in San Francisco tomorrow?" Because Bill is familiar with the city and the concept of weather, he knows what Harry is asking for.
- A user asks the agent "What's the weather supposed to be like in San Francisco tomorrow?" In API.AI, an intent houses elements and logic to parse information from the user and answer their requests.
- For the agent to understand the question, it needs examples of how the same question can be asked in different ways. Developers add these permutations to the User Says section of the intent. The more variations added to the intent, the better the agent will comprehend the user.
- The API.AI agent needs to know what information is useful for answering the user's request. These pieces of data are called entities. Entities like time, date, and numbers are covered by system entities. Other entities, like weather conditions or seasonal clothing, need to be defined by the developer so they can be recognized as an important part of the question.
- Armed with the information Bill needs, he searches for the answer using his favorite weather provider. He enters the location and time to get the results he needs.
- API.AI sends this information to your webhook, which subsequently fetches the data needed (per your development). Your webhook parses that data, determines how it would like to respond, and sends it back to API.AI
- After scanning the page for the relevant info, Bill tells Harry "It looks like it's going to be 65 and overcast tomorrow."
- With the formatted reply "in hand", API.AI delivers the response to your user. "It looks like it's going to be 65 and overcast tomorrow."
- Now that the conversation is on the topic of weather, Bill won't be thrown off if Harry asks "How about the day after that?" Because Harry had asked about San Francisco, follow up questions will more than likely be about the same city, unless Harry specifies a new one.
- Similar to Bill's scenario, context can be used to keep parameter values, from one intent to another. Contexts are also used to repair a conversation that has been broken by a user or system error, as well as branch conversations to different intents, depending on the user's response.

5.2 What is an Entity?

An entity is a concept you want your personal assistant to understand when it's mentioned by the user in conversation. Each entity has a range of values and properties that contain the terms the assistant will need to understand to respond to this concept.

There are three types of entities in Api.ai:

- **System:** Entity types defined by Api.ai such as date, color, email, number and so on, which Api.ai already understands. You can find a full list of these entities in Api.ai's documentation on System Entities.
- **Developer:** Entities which we create for our individual needs. These are what you'll be focused on in this article.
- **User:** These are created for individual users while they use the assistant and can be generated by the Api.ai API to be used within a single session. I won't be covering these in this article, but if there's enough reader interest, I might explore this in future!

Api.ai's small talk is an example of a range of statements with no entities whatsoever. They're listening for statements whose meaning doesn't change based on keywords in the sentence. When someone asks "How are you?", there's no variation Api.ai understands. However, if we adjusted that to "How is Jimmy Olsen?", we're moving this sentence into the realm of entities. "Jimmy Olsen" would be an entity that represents a person. Teaching your agent that entity would mean if someone asks "How is Bruce Wayne?", your agent would know that they mean to ask how a person is, and that they want to know about Bruce Wayne specifically.

As another example, an entity of "superhero" is not something Api.ai knows about. You could train your assistant to understand a range of superheroes and their various names — "Superman", "Batman", "The Flash", "Green Lantern", "Wonder Woman", "Santa" and so on. It could then understand that these are specific concepts you want to trigger actions with, such as contacting these heroes when villains strike via an API when you say things like "We need The Flash!"

5.3 Using Your Entity in an Intent

You now need to create a new intent that will train your personal assistant to recognize the sentences that trigger your sleep-related requests. Start by heading to the "Intents" page and creating a new intent as you've done a few times by this point in the series.

On your new intent page, include your entity within "User Says" statements. In the case of your sleep entity, enter in a statement that includes a generic word to represent an example of your entity. In this case, type in "How many hours of sleep did I get last night?" Your entity in this case is "sleep".

To teach Api.ai that "sleep" is your entity, highlight that word and a dropdown will appear. One of the options should be your new sleep entity — @sleep. Choose that and the word will be highlighted to represent that it can potentially change in statements spoken to your agent.

You also have a field for the action name. This is the name that will be passed to your web app to show what Api.ai thinks the user wants to do. Name your action “sleepHours”.

It might also find other pre-built entities that Api.ai understands, such as time period. In the screenshot above, Api.ai picked up that “Last night” was a time period and highlighted that too. This means that Api.ai should still understand the sentence “Last night’s sleep hours please” if it instead hears “Last week’s sleep hours please.”

To finish up your intent, set up some responses to your intent about sleep hours. The assistant itself in Api.ai can’t look up the stats, so you’ll need to use your own web app for that. However, it’s nice for the assistant to at least keep up the illusion that it’s doing all the work. To do this, your responses say things like “I’ll retrieve your sleep stats for you now, one moment!” and “Looking up your sleep hours now.” It also gives us a bit of time for our web app to retrieve that data.

5.4 Expanding Your Entity

You have a working entity that lets your assistant understand when you want to look up how many hours of rest you’ve had, but the entity is still quite simple. Sleep is just sleep. Rest. Shut-eye. In reality, there are specific types of sleep that a user might ask about. What if the user asks “How many hours of REM sleep did I get last night?” “REM sleep”, “Deep sleep” and “Light sleep” are all different types of “sleep” that should be understood by your sleep entity. You’ll add those in.

Return to the Entities page and open the @sleep entity. Underneath “sleep” and its synonyms of “sleep, rest, doze, shut-eye”, add new types of sleep such as “REM sleep” (also just called “REM”), “deep sleep” and “light sleep”. Include these as new rows, as they have distinct meanings and aren’t exactly the same as the generic term of “sleep”. To add a new row, click “Add a row”. Once you’ve added your new forms of sleep, click “Save” to save the changes:

If you click the “Show JSON [10]” button underneath, you’ll see where the power of this truly comes into play

Chapter 6: References

6.1 References

[1] Chatterbots, Tinymuds, and the Turing Test (http://www.aaai.org/Papers/AAAI/1994/AAAI94-003.pdf)

[2] Artificial Intelligence, Expert Systems, Computer Vision, and Natural Language Processing (https://ntrs.nasa.gov/search.jsp?R=19860033961)

[3] Naïve Bayes text Classification (https://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html)

[4] Methods for Critical Discourse Analysis (https://books.google.com.pk/books?hl=en&lr=&id=BENCiEez4MsC&oi=fnd&pg=PP2&dq=Discourse+analysis&ots=wsxj4rPDaG&sig=qpWR0t4THiX397UdiJp4n_jfQyM#v=onepage&q=Discourse%20analysis&f=false)

[5] Optical character recognition (OCR) (http://www.ijemr.net/DOC/OpticalCharacterRecognitionOCR.pdf)

[6] Roman to Urdu Transliteration (https://www.researchgate.net/profile/Tafseer_Ahmed/publication/237821067_Roman_to_Urdu_Transliteration_using_word_list/links/54253b280cf238c6ea73f1db/Roman-to-Urdu-Transliteration-using-word-list.pdf)

[7] The Rise of Social Bots (http://dl.acm.org/citation.cfm?id=2818717)

[8] Natural Language Understanding (NLU) (http://infolingu.univ-mlv.fr/Bibliographie/Saetre_Unitex_Thesis.pdf)

[9] API.ai official documentation (https://api.ai/docs/getting-started/basics)

[10] JavaScript Object Notation (http://www.cse.iitd.ac.in/~cs5090250/JSON.pdf)