RESEARCH ARTICLE OPEN ACCESS

Topic Trees And Chatbot Conversational Context

Peter Waksman

ABSTRACT: This article discusses the relation between topic-specific word trees ("ontologies") and the retrieval of information from a chatbot's recent conversation – the conversational context. I assume the conversational context includes the sequence of recently visited nodes of the topic tree. So we caninterpret context retrieval as a problem of selection from this sequence, in reverse order. We will see that a surprisingly small set of selection methods support a diversity of indeterminate words and phrasesthat may need to be handled by a chatbot: like "it", "them", "what's the difference?", "the third", etc.An essential part of the solution is the way context is filteredto contain only the kinds of items we are expecting to find in the current input but that are missing. What is **expected but missing** drives context retrieval. In practice when an input sentence is encountered that is not understood and contains an indeterminate word, a chatbot may be allowed a second pass after retrieving and inserting recent context into the input.Theseideas are implemented in Python at: https://github.com/peterwaksman/Narwhal

Date of Submission: 02-08-2018 Date of acceptance: 17-08-2018

I. INTRODUCTION

It is interesting that chatbot development requires more than just language understanding algorithms. Achatbotneeds to organize and keep track of a conversation and vary its responses depending on previous exchangesand the current "state" of the conversation. Also a chatbot may be required to understand the context of the most recent exchanges with its client and may need to handle indefinite words, like "it" or "both", that refer to things that were just said. This article discusses one way to do that: keeping a list of recently mentioned concepts and retrieving them as needed to disambiguate indefinite words in a current input.

Let me use the word **indeterminate**for words or phrases that get their content dynamically, by retrieving recently used information. Many of the indeterminate words are pronouns (e.g. "it", "his"). Others are classified as determiners, predeterminers, and adverbs (e.g."what" and "both"). In the case of the indeterminate phrase "what is the difference" the word "difference" is a noun. Identifying indeterminate words by their parts of speechclassification is not useful;andanalyzing them as a diagrammed sentence using Natural Language Processing (**NLP**) does not tell us that these particular wordsrequire special handling in a chatbot. For actual Natural Language Understanding (**NLU**)one needs additional operations for seeking the appropriate words from the context and inserting them retroactively into the input (or something comparable). Thus NLU needs more than justa reformatting of the input into the structure of a diagrammed sentence –it needs more than what is available in NLP.

Indeterminate words have a short term scope. The word "it" is often abused in real conversation and hard to follow, even for a person. Some chatbots are able to disambiguate complex combinations of indeterminates[1]but my goal here is consider how to retrieve context successfully in the simplest situations,that would be easy for a person to understand; so the discussion is limited to the simplest kinds of context retrieval. I hope the outlines are clear but the reader will easily note the incompleteness of this story.

The following tries to be neutral about how you implement NLU. A minimum requirement is to havea metric forwhen an input expression isunderstood or not[2]. This tells the program when context may be needed. Simply put, when there is little or no understanding of the input and indeterminate words are present in the input, then retrieving and inserting context mightconvertthe input into something that is understood. The chatbot program needs to know when it should try this and it needs to know when it has succeeded.

In section 1 "Trees" I discuss some basic ideas about how words can be stored in a tree structure for the purposes of matching against incoming text. I introduce a simple tree notation for

[1] See Don Patrick's Winograd Schema Challenge

[2]For example Narwhal, mentioned in the abstract, implements a GOFor "goodness of fit"score that measures understanding as a value between 0.0 and 1.0.

the discussion.In section 2 “Chatbot Conversational Context” I discuss how to save copies of the nodes as a sequence and how to retrieve from this sequence and insert into the input. Finally, in section 3 “When to Use Context” I discuss detecting when context retrieval may be needed and when it has succeeded.

We will see that short term context retrieval depends on three factors:

- the list of recently exchanged tree nodes,
- the indeterminate word that enables retrieval, and
- what is expected but missing from the current input, triggering the need for retrieval and determiningwhat kind of thing to retrieve.

1. Trees – Word and Concept Ontologies

An "ontology" or "taxonomy" is a classification using a hierarchical tree structure with branch points, or **nodes**, corresponding to the entities being classified. The idea is that similar entities can be grouped as sub nodes of a single 'parent' node in a useful way. In the field of language processing, it is typical to organize words or concepts as an ontology, usually in a way where parent-child relations are from a more general ‘parent’ to a more particular ‘child’. Chatbot developer may need to spend significant time organizing ontologies.

Organizing words into a hierarchical structure is straightforward and one finds that simple parent-child relations arise naturally. Unlike the indeterminate words, topic specific words have well defined, constant, meanings within the chatbot program. We could call these **determinate** words.

Because a node represents something in common to the words that define it, I may usedescriptions like“concept” or “sub-topic” instead of “node” for such a word grouping.One tries to name the node in a way that is consistent with the conceptand, here, I will write this as a label, followed by a comma separated list of defining words,within curly braces ‘{}’.
For example:
COLOR {color, tint, hue, shade}
We can represent the parent-child relation between nodes by indenting the names of children, underneath the parent. For example:
COLOR {color, tint, hue, shade}
RED {red, scarlet, rose, pink, ruby, vermillion}
BLUE {blue, azure, indigo }

Here there is one COLOR node with two particular children: RED and BLUE. If, for example, an input expression contains the word “azure”, then this is **match** for the BLUE node. In some circumstances “azure” may also be considered a match to the COLOR node, since“azure” is a **match to one its children**. Note that although COLOR will be a constant in the program, it may be treated like a variable in expressions that are used to find matches among the children, as discussed more below.

The word groups that define a node are not the same thing as the parent child relations between nodes, these groupings are application specific. Thewordsin a group defining a node do not need to be the same kinds of parts of speech or even single words. For example, here are words used in hotel reviews to describe extremes of GOOD and BAD:
GOOD { very pleased,soothing,a gem, love ,home away from home, wow,wonderful,bliss,happy,superb,gorgeous, spotless, immaculate,phenomenal,fantastic,perfect,relax,excellent,spectacular,peaceful,lovely,beautiful,amazing,impressive, impressed,heaven,magical,fabulous,serene,paradise,special,cozy,oasis }
BAD { unacceptable,filthy,dump, mildew, lousy, awful, horrible, horrid, poison, gross, ick, icky, disgusting, hairs, yuck, ug, ugh , unhelpful, unpleasant, disappoint, disappoint, terrible, bugs, spider }

It is important to note that although, for example, ‘spider’ and ‘hair’ are not identical concepts in general English usage, both words express revulsion in the English used for hotel reviews – so they can be treated as synonyms for a hotel review application.

Reasons for grouping words and phrases together include:

- The words may be synonymous or functionally equivalent in your chatbot’s world.In this case the node label can be chosen to represent the shared concept.
- The words may be different miss-spellings of the same word, or variations on a phrase. In this case the node label can be chosen to represent the shared word.
- The groupings may be for simple convenience. For example the ‘root’ node of a tree may be a parent to all the other word groupings – without being related to a specific concept. In this case the node creates a formal grouping but its label is arbitrary.

I use the term **topic tree** when an ontology contains terms that are all related to a particular topic. Typically, a chatbot intended for a narrow business function will use multiple topic tress grouped under a single parent for convenience.When no confusion arises, I will also use the term “topic tree” for a grouped collection of related sub trees.

In practice you are never done extending your word lists. The different ways people express themselves, the variety of miss-spellings, etc. and it is hard to see how one could automate the extension process – say using **machine learning** from new examples. The overall question is: how to generate and extend ontologies using supervised or unsupervised machine learning?

That is the basic idea of an ontology. A number of examples can be found in the Narwhal project on GitHub.

Additional examples of nodes

Some of the nodes will appear to define 'attributes' - with word lists that mostly contain adjectives; while other nodes appear to define 'things' - with word lists that mostly contain nouns. You may also want to define concepts that define an action or relation – with word lists that mostly contain verbs. It should be emphasized that the part of speech classifications for the words used in these lists is irrelevant. Some other examplesfrom hotel reviews are:

Attribute-type nodes:
OPEN { open, thin, paper thin }
NEAR { near to , in the heart of, over , outside, next to,…}

Thing-type nodes:
WINDOW { window, balcony, balcony door }
EQUIPMENTSOURCES { ac, a/c, air con, tv, hvac, clanking, …}
QUESTION { ? , ask about, question, question about , questions about , need to know , want to know, want to find out, help with , is there, info, information, information, tell me about }
CROWN { crown, full contour, cut back, temporary, temp }
ABUTMENT[3] {abutment, abut, abutmen }

For example one might have several different Thing-types as children of a more general thing-type parent:
OPENING {…}
WINDOW{…}
DOOR { …. }

Action-type nodes:
KEEPOUT { keep out, filter, cut down on, shut out, cut out, block, keep out, escape from… }
REQUEST {use, build, ask for, sell me, fabricate, produce, provide,….}

Parent- or Category-type nodes:

Often, natural concept nodes are categories that group sub categories or singular entities as their children. For example the COLOR node above refers to the category of colors, it is not a thing or an attribute but a parent of such entities.

One observes somewhat arbitrary or abstract categories can have other categories as children. For example if COLOR and SHAPE were to define different dimensions of an item, they could be grouped, somewhat arbitrarily, under a node like this:
DIMENSION { dimension, attributes }
COLOR {…}
SHAPE {…}
A more complicated example (from dental product description):

PRODUCT { … }
RESTORATION { … }
ABUTMENT {…}
CROWN { … }
MATERIAL { … }
TITANIUM { … }
ZIRCONIA { … }
TOOTH { #, tooth }
TYPE { anterior, posterior, canine, premolar, …. }
TOOTHNUMBER { }
ONE {1, one}
TWO { 2, two }
…etc.
THIRTY_TWO { 32, thirty two, thirty-two }

These trees are application specific. It is worth observing that the hierarchical structure of a word tree is analogous to the hierarchical definition of a floating point numbers. For our current purposes, the **word trees can be thought of as an application-specific numbering system, with text matching considered an act of measurement.**

Node formulas

Node combinations are elements of simple meaning. Consider the difference between asking a question about a window versus making a request for a window. Let me use the '*' symbol to

[3]In dentistry, an 'abutment' is a short post screwed into a patient's jawbone, as a substitute for tooth material, intended to support a crown. A given tooth location could have an abutment, a crown, or both.

represent a combination[4], so we can consider two combinations:

QUESTION*WINDOW versusREQUEST*WINDOW.

Or, we could have a combination with several terms, like a request for a restoration on a particular tooth:

REQUEST*RESTORATION * TOOTHNUMBER

Such simple combinations of nodesbegin to behave like the meanings of expressions, such as "please open the window" versus "is there a window?" One concludes that a simple form of NLU is possible just by keeping track of node combination, matching them to an input expressions, and providing a metric of understanding[5].

If the nodes of a node combination match the input, they are said to be **filled**. It gets interesting when all but one node is filled. In that case the context can be searched, looking for instances of the missing node or its children.

Parent Nodes Act as Variables when matching a Combination

Suppose one expresses a combination using a parent node like 'COLOR', for example:

DRESS * COLOR

This is a match for "dress with color" but it can also serve to match more particular expressions like "the dress is blue", or "scarlet skirt". For those purposes, the parent node functions as a variable in the combination "formula", with a range of values equal to the children nodes of the parent.

The Expected but Missing – Pattern Completion

I assume context retrieval is triggered when one part of a node combination is missing: instead of what is expected, an indeterminate word occurs. In this case the context may contain diverse nodes but only the ones that are children of what was expected are of interest.

For example the chatbot may be programmed to recognize when a client asks for crowns on tooth #3 and #4". It can do this by trying to fill in the combination:

REQUEST * RESTORATION * TOOTHNUMBER

However, when those two numbers are already in the recent context, the client might simply say "I want crowns on both teeth".In this case the chatbot finds and incomplete match to the combination, indicated by '?':

REQUEST*RESTORATION*OOTHNUMBER(?)

In this case, the only part of the context worth retrieving is the partcontaining tooth numbers. When the tooth numbers are expected but missing, this tells us to filter the context to only contain tooth numbers.Choosing two items from a list (after filtering) is straightforward.

Borrowing a metaphor from computer science, the indeterminate words, together with the 'expected-but-missing' behave likedynamic casting operations that act upon the sequence of past nodes, filtered by what was expected.

2. Chatbot Conversational Context

Text Processing and Segmented Text

Conversation processing requires matching of input language to program entities. For developers focused on Natural Language Processing, text matching is done using "regular expression" pattern matching; and the necessary regular expression patterns are defined by the chat program. Another approach is to break the text up into separate words or **tokens** and subject the tokens to matching against some other kind of patterns that are defined by the program. I find great efficiency and simplification in replacing words of text with tokens,followed by replacing tokens, one-for-one, with tree node copies. When an unknown token occurs, one can use a "null node" placeholder. I call this process **segmenting** the text and the result **segmented text**. To repeat: The process of segmenting text is this: Text is converted to tokens, tokens are matched to nodes of the topic tree, the sequence of nodes is copied into memory. The result is called segmented text.A typical processing sequence, leading to application specific code is illustrated in Figure 1.

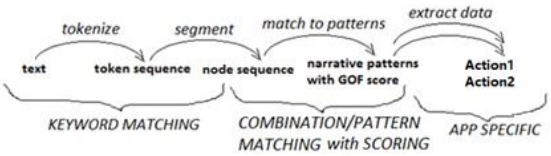

Figure 1A processing sequence converts text to segmented text, with two levels of pattern matching, followed by applications specific actions.

[4]The '*' is deliberately vague, intended to represent more than one type of combination.

[5]A good NLU library will also provide sentiment (good/bad value) parsing. So node combinations may behave like signed quantities.

Saving segmented input and output as context

In a chatbot dialog, the context needs to include both the client's input and the chatbot's output. For example to handle this dialog:

CLIENT: I want an abutment on tooth #8
CHATBOT: Do you want titanium or zirconia?
CLIENT: What is the difference?

Here the client's last sentence is an abbreviated form of "What is the difference between titanium and zirconia?" In this case, the previous chatbotresponsecontains the relevant contextto interpret the question.

In another example, the context comes from the client :

CLIENT: I want abutments on tooth #8 and 9
CHATBOT: Is there anything else?
CLIENT: I want crowns on both.

Here the first line of client inputcontains the relevant context for interpreting the word "both" as "both #8 and #9".

Hence the context should include input from the client and also response from the chatbot. One can segment the input as a sequence of tree nodes and one can also segment the outputthat is passed back as a response to chatbot client. However it is easier to decouple the response text from the underlying segmented version of the response; and save a segmented version of the response directly without re-processing the response text. It is simpler to just generate segmented text in parallel with the response text, and store that as an additional segment in the context.

Thus we can create context for a chatbot by saving an extended sequence of segmented texts, following steps like this:

- input text → tokenized text → segmented text (and save the segmented text in the 'context')
- process and generate response output (and generate a segmented version of the response to store in the 'context')

Although there is more to "context" than this, for now we consider context to be the and array of segmented texts, alternating between the clients input and the chatbots output, as illustrated in Figure 2.

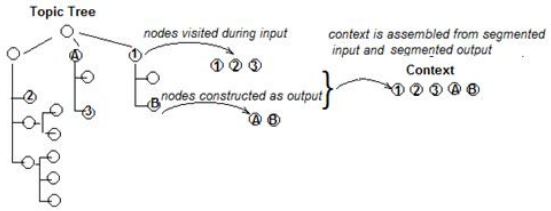

Figure 2–Here a topic tree is represented with circles for nodes and line segments for the parent-child relationships. Nodes from the tree are visited during input (1,2,3) and the sequence (the **segmented text**) is combined with another sequence, corresponding to the chatbot response (A,B). Segments for both input and response are saved as part of the context.

How much context to save

The question arises as to how much context should be saved? I find that the last three or so exchanges between client and chatbot, are enough for short term context. However there are other aspects to context - such as the current topic and who the client is. Such longer term aspects of context may not be part of the linguistic history of client-chatbot exchanges. Some developers treat context as including hundreds of previous client-chatbot exchanges[6].

Filtering context to contain nodes that are missing from a current combination

In the example where the client says "what is the difference", we need only retrieve the most recent alternatives found in the context. But in the example where the client says "I want crowns on both" the word "crown" tells us what kind of thing to retrieve with the 'both', specifically whatever we expect to see in combination with crowns.

Words like 'difference' can retrieve directly from context, while ignoring the other words in input. But words like "both' need to retrieve from a context that has been filtered to contain only sub nodes of the missing but expected sub tree node (in this case tooth numbers) to be combined with 'crown' in an expression.

In one application, a dental ordering system, the set of allowed actions is constant and finite, and each action is triggered by understanding a specific combination of nodes. Thus for example nodes like REQUEST or QUESTION, are combined with product specific nodes, like ABUTMENT or CROWN, and might be used in combinations like:

QUESTION * MATERIAL (e.g. "what material do you want?")

[6]Don Patrick at chatbots.org

REQUEST*RESTORATION* TOOTHNUMBER ("I want a crown on #8")

These explicit forms are anticipated and they lead to specific actions. If the client says "What is the difference between titanium and zirconia?" then that input is a good match for the QUESTION * MATERIAL combination. So MATERIAL is expected but missing and should be what is retrieved from context when the client says only "What is the difference?".

Similarly if the client says "I want a crown on both" this is a reasonable but incomplete match to
REQUEST*RESTORATION*TOOTHNUMBER.
So the indeterminate combination: REQUEST*RESTORATION*BOTH needs to resolve the 'BOTH' against the expected but missing TOOTHNUMBER of the desired combination. Hence the handler function for the indeterminate node 'BOTH' could take the expected but missing information of TOOTHNUMBER as an argument.

In general, handler functions should be able to take arguments defining the missing type of information, depending on what node combination we are trying to resolve.

Methods for extracting information from the context

We now have most of what is needed to describe a version of context retrieval. The context itself will be a sequence of tree nodes, arising from either segmented input or from a segmentedversion of response.

So we consider the problem of going back through the context to find the last occurrences of a particular type of node. We want these sorts of "get" functions, which we will call **context handlers**.

- getAttribute() - get one, a pair, or severalrecent 'attribute' nodes
- getCategory() - get most recent 'category' node
- getThing() - get one, a pair, or several recent 'thing' nodes

In each case, the 'get' method can iterate through the nodes of the segments, saved in the context in reverse order, until the item, pair, or group is found (or not). It may make sense to limit how far apart the occurrences are allowed to be and, as mentioned to limit how far back in the past to look within the current conversation.

It also may make sense for the context handlers to retrieve only from nodes within a subtree of a given topic tree. In other words, when a context handler is applied to past context, the context can first be filtered through a subtree, by replacingany non-subtree node with a null node so, later, the context handler can be applied to a filtered version of the context. Below, sub tree filtering is done when a particular concept node is expected but missing from a client's input. In that case the context can be filtered to only contain children or self nodes in the subtree of the expected but missing node.

Nodes that "take"

The indeterminate words can be put into their own topic tree or at least treated as nodes, in the same way that topic keywords are. For example:
IT {it , the one , that}
BOTH {both, the pair, the two}
DIFF {difference, compare }
Rather than labeling these as 'thing', 'category', or 'attribute', we can specialize them by assigning a context handler to each. For example:
IT is assigned the getThing( ) handler, looking for one 'thing' node in the context.
BOTH is assigned the getPair( ) handler, looking for a pair of thing nodes.
DIFF is assigned the getAttribute( ) handler, looking for a pair of attributes.

Nodes, corresponding to indeterminate words and assigned a context handler might be called **nodes that take** because of their correspondence with nodes that give, found in the context. In Python, for example, a node class can contain a 'self.contextFn' to act as a handler which is set to 'None' except when it is one of these special nodes that take.
Thus we have two coordinated ways of adding information to a tree of nodes:

- Nodes that "give" are labeled as 'thing', 'category', or 'attribute'
- Nodes that "take" are assigned a non-trivial context handler method.

3. When to use Context

The presence of an indeterminate should not always trigger context retrieval. For example 'both' can occur in an explicit request like "I want crowns on **both**#8 and 9" without triggeringcontext retrieval. Only when the node combination REQUEST*RESTORATION*TOOTHNUMBER occurs and TOOTHNUMBER is missing do we try to replace the empty spot with what can be retrieved using the handler assigned to BOTH. Specifically the getPair() is applied to the context, filtered to contain only sub nodes of TOOTHNUMBER.
The whole question of when to try context retrieval depends on the chatbot program knowing when an input is incomplete. As mentioned in the introduction, I assume your NLU has a metric that gives a score per desired node combination. Each

such combination will receive its own version of the score and each may be incomplete in a different way. So the same indeterminate word might be retrieve differently for each one. For example, "both" could mean "both #8 and #9". Or, in another node combination, it could mean "both a crown and an abutment."

Peter Waksman "Topic Trees And Chatbot Conversational Context "International Journal of Engineering Research and Applications (IJERA) , vol. 8, no.8, 2018, pp. 56-62