Anna: A Nutrition-Facts Dialogue System

Eduardo Alvarez-Godinez
Department of Linguistics
University of Washington
Seattle, WA 98195, USA
eduara@u.washington.edu

Sumithra Bhakthavatsalam
Department of Linguistics
University of Washington
Seattle, WA 98195, USA
sumithra@u.washington.edu

Abstract
We introduce Anna, a nutrition-facts dialogue system built with a combination of freely available components and some ad hoc code. In this paper, we describe its domain, its overall architecture, and its major components.

1 Introduction
With the rising health consciousness due to the sedentary lifestyles in most of the urban world, people are increasingly curious about nutrition and seek data to help them make better food choices. A dialogue system specializing in nutrition could play an important role in filling this need. We have developed Anna as a prototype for such a dialogue system.

Although this is a text-based prototype, we believe that a fully developed system paired with speech recognition and synthesis components could have great potential in supermarkets or restaurants to help customers make informed decisions on the fly. It could also be extended into an advisory service that can be used to get recommendations on what to eat for a specific health issue or a set of vital statistics.

In the last few years, many natural language processing (NLP) software resources have been made freely available to the public. We wanted to test the practicality of building a dialogue system for our domain in a relatively short time (less than three months) using only such components, publicly available data, and some ad hoc code. Anna is the result of this effort.

2 The Domain
While nutrition data is readily available on the internet and through government publications, the data can be difficult to navigate due to its large size. For example, a simple query for egg on nutritiondata.com returns 164 results. A dialogue system could help users find the information they need, as shown in the following sample dialogue.

2.1 Sample Dialogue
S: How may I help you?
U: I'd like to know how much cholesterol is there in egg.
S: Ok, cholesterol in egg. Is the egg white, whole, or yolk?
U: Whole
S: Is it cooked, dry, or raw?
U: Raw
S: Whole, raw egg has 423 mg of cholesterol per 100 g.
U: Ok, and how much sodium does it have?
S: It has 140 mg of sodium.
U: And how about raw, white egg?
S: Raw, white egg has 166 mg of sodium.

As we will see in subsequent sections, unequivocally identifying nutrient and food in such a way that this data can be retrieved from the backend

1 http://www.nutritiondata.com
database is what makes this a challenging and interesting dialogue system project.

2.2 Tasks

- Give information regarding the nutritional content of food.
- Work with the user to specify a particular nutrient and food item given many alternatives through precision questioning.

2.3 Users

This would be useful for people seeking nutrition information but who would find difficult to browse a web site, particularly one that presents the users with what could be an overwhelming number of choices at the same time. For example:

- Restaurant and market patrons accessing the system through a speech interface.
- Visually-impaired people using a Braille terminal.

3 The Architecture

The architecture of the dialogue system is illustrated in Figure 1. It is a more specific version of the typical architecture presented by Jurafsky and Martin (2008.)

The natural language understanding (NLU) component takes natural language text from the user and produces a semantic representation appropriate for the dialogue manager. To do this, it uses grammars and state information from the dialogue manager. Conversely, the natural language generation (NLG) component takes semantic input from the dialogue manager and generates a natural language response for the user.

The dialogue manager, which in our case is also the task manager, controls the whole process. It interacts with the backend to get task domain data.

Additionally, any input from the user that is not recognized by the NLU component and is hence considered to be out-of-domain, is routed through the dialogue manager to a general-domain chatbot, whose replies are equally routed through the dialogue manager and the NLG component to the user. We have labeled this exchange between the user and the chatbot as pass-through, NL text since it just passes through the other components, monitored but unaltered. This is what the dotted line in Figure 1 represents.

A note on the implementation: With the exception of the semantic parser discussed in the NLU section, the system is written in Python\(^2\) (ver. 2.5) and uses the following additional libraries:

- The Natural Language Tool Kit (NLTK)\(^3\)
- PyAIML\(^4\), an AIML interpreter

4 The Dialogue Manager

The dialogue manager’s main task-related goal is to answer the question how much of a nutrient is there in a given type of food. If unequivocally identifying a nutrient and food type were easy, a user-initiative, frame-based system to fill a form and translate questions into SQL queries would suffice. As we discussed in section 2, this is not the case, and this identification requires the system to ask clarifying questions, present choices, understand acceptances and rejections, and so on. To handle this complexity, ours is a mixed-initiative system that while frame-based, leans toward the information-state model described by Traum and Larsson (2003.) Since logic and dialogue specification are coded together, it is also what Pieraccini and Huer-

\(^2\) http://python.org
\(^3\) http://nltk.org
\(^4\) http://pyaiml.sourceforge.net
ta (2005) refer to as a programmatic dialogue manager.

The dialogue manager uses semantic and dialogue act information encoded in feature structures. For example, the following would represent a sentence like “how much vitamin A do strawberries have?”

```
type = 'request_nutrient_quantity'
nutrient = [ desc = 'vitamin A' ]
food = [ desc = 'strawberries' ]
```

The user could have chosen to omit one or both of the elements and the system would have asked for them in turn according to its task context and update rules. In this particular case, and to illustrate the need for clarifying questions, the system would first have to ask the user to select between the two kinds of vitamin A:

```
type = 'request_nutrient_selection'
nutr_choices = [ choice1 = u'Vitamin A, IU' ]
[ choice2 = u'Vitamin A, RAE' ]
```

The answer from the user could be one of the two options (probably a partial match like “IU”) or an ordinal number (e.g. “the first one”), and the dialogue manager needs to refer to its dialogue context to interpret this answer.

For any reference to nutrients or food, the dialogue manager needs to retrieve data from the back-end via function calls, and update its information state and next action accordingly.

The dialogue manager is responsible for the response planning or deep-structure language generation. The main design consideration in deciding what to say to the user is to avoid overwhelming her with too much information. This is particularly important when the system needs to ask the user to narrow down the food item, given that the potential choices are usually in the tens or hundreds. To accomplish this, we use equivalence classes, search trees, and choice pages, as described next.

If a whole category of foods will have the exact same nutrient content (for a particular nutrient), we consider them to be part of an equivalence class and we do not require narrowing them down further. For example, if the user asks about vitamin D in apples, the system does not need to ask what type of apples because all types have the exact same vitamin D content (0 mg.)

The backend structures the results of its food queries in a search tree. The dialogue manager uses this tree structure to ask the user to make a select between only the number of choices needed to navigate the search tree, until it reaches a leaf or an equivalence class.

In cases where even the number of choices needed to navigate the tree is large (greater than 7 in Anna), the choices are presented in pages, or groups of 7, from which the user can select one or request to see more choices.

We should mention that the order in which the choices are presented is according to word count, from fewer words to more words. We noted that the more generic food categories had one or two words, whereas the more specific ones, usually related to a specific brand and thus less likely to be of interest, had multiple words.

User input not recognized by the NLU comes to the dialogue manager as an unknown type of dialogue act with the unparsed input as content. This is passed to the chatbot and its reply, which is framed in a feature structure but otherwise unaltered, is passed back to the user via the NLG component.

The following is the set of the dialogue act types used by the dialogue manager.

**User dialog act types:**
- Accept
- Reject
- Select
- Clear
- Request_intro
- Request_nutrient_quantity
- Request_more_choices
- Unknown

**System dialog act types:**
- Intro
- Request_intro
- Request_nutrient
- Request_nutrient_confirmation
- Request_nutrient_selection
- Request_nutrient_specification
- Request_food
- Request_food_feature_selection
- Answer_nutrient_quantity
- Inform_no_more_choices
- Inform_no_data
- Inform_starting_over
- Unknown (chatbot reply)
5 The Components

5.1 The NLU Component

The natural language understanding (NLU) component determines dialogue acts and extracts semantic entities using semantic grammars and dialogue context.

The semantic grammars, as described by Jurafsky and Martin (2008), are CFGs in which some of the non-terminals correspond to the semantic entities being expressed. Our system uses Phoenix (Ward and Issar, 1994), a robust semantic parser based on Recursive Transition Networks. As mentioned before, the Phoenix semantic parser is the only component not available as a Python library, so we interact with it via its command-line interface through a sub-process call and temporary files.

The Phoenix parser matches input strings into a sequence of semantic frames. Each frame is a named set of slots which correspond to the semantic entities we want to extract. Each slot has its own CFG. In our case, each frame corresponds to a dialogue act from the user, and each slot corresponds to the data we need such as food and nutrient. We pick the parse with the most likelihood to determine the dialogue act.

Most user input will actually be parsed by the request-nutrient-quantity frame, which is the main task of our application. It contains many variants of the nutrient request question, along with hundreds of entries for the nutrient and food lexicon. The request-nutrient-quantity frame allows slots to be empty. At the beginning of the conversation, it will be common for the user to specify both nutrient and food, as in “how much calcium is there in milk?” However, after receiving an answer to this original question, the user will likely only change the nutrient or the food, and the grammar is capable of parsing inputs like “how much of it is there in butter?” or “how much vitamin d is there in it?”, or even “how about bananas?” The system will decide what “it” means based on the task context, or it will ask additional questions to clarify if needed.

Similarly, when specifying food, the frame distinguishes between a food head and food features. The food head corresponds to the main category, such as “apples”, whereas the food features narrow it down, e.g. “canned”, “raw”, etc. A valid food head is required when specifying food, but the food features are optional and the system will ask about the ones it needs in due time.

The select dialogue act is a special case and this is where dialogue context comes in. When the system asks the user to make a selection given some specific choices, it first tries to match the user input to one of those choices using, in this order, (1) digits, (2) textual ordinal references (e.g. “the first one”, “number three”), (3) exact text match, and (4) partial text match. Only if there is no match does the system attempt to parse the input using the other grammars. Dialogue context is important here because the NLU component needs to know that the system just asked a multiple-choice question and what those choices are.

To parse ordinal references, we use a semantic grammar augmented with features, parsed by the Earley feature chart parser from the NLTK. This allows us to readily get the choice as a number from the ORD feature regardless of whether the user says, for example, “the first one” or “number one”. Here are some sample productions:

S -> CHOICE[ORD=?o]
CHOICE[ORD=?o] -> NUM CD[ORD=?o]

Det -> 'the'
ONE -> 'one'
NUM -> 'number'

JJ[ORD=1] -> 'first'
JJ[ORD=2] -> 'second'
JJ[ORD=3] -> 'third'
...

CD[ORD=1] -> '1' | 'one'
CD[ORD=2] -> '2' | 'two'
CD[ORD=3] -> '3' | 'three'
...

Lastly, any user input not recognized by any of the grammars will be passed verbatim to the chatbot via the dialogue manager.

---

5 Although the grammar has a specific set of productions for “calories”, internally they are translated to energy (kcal) and treated as just another nutrient.
5.2 The NLG Component

Natural language generation in Anna follows a two-step process of response planning and response generation similar to the one of the MERCURY system as described by Senenff (2002.) Response planning, or deep-structure generation, is done by the dialogue manager and was discussed in its corresponding section. This section and the NLG component deal with response generation or surface-form generation.

Anna uses the simple template-based generation technique, in which many of the words are fixed and others are variables that are filled by the generator based on the information on the semantic frame, according to several rules. This is better explained through an example. Consider the following semantic frame:

```
type = 'answer_nutrient_quantity'
food_desc = 'beef, brisket, whole'
nutr_desc = 'Vitamin C, total…'
nutr_units = 'mg'
nutr_val = '0'
```

The corresponding output from the system is:

```
There is no vitamin C, total ascorbic acid, in whole, brisket beef.
```

In addition to filling the slots, the generator has made modifications to the content to make it look more natural, among them: using `no` instead of `0 mg`, lower-casing the word `vitamin`, adding a comma after `ascorbic acid`, and writing the food description in the correct order.

To make the dialogue more coherent, the generator uses pronouns and tapered prompts as suggested by Cohen et al (2004.) Pronouns are used when a nutrient quantity request refers to the same food as the previous one, as in this sample dialogue:

```
Anna:
There are 0.125 mg of vitamin B-6 in uncooked, sulfured, dried apples.
> how about vitamin d?
Anna:
There is no vitamin d in them.
```

The system uses heuristics to decide the number for the pronoun.

Tapered prompts are used mostly when asking the user to make a selection:

```
> how many calories does milk have?
Anna:
There are several kinds of milk. Please select from the following choices using the numbers or the descriptions: (1) buttermilk, (2) canned, (3) chocolate, (4) dry, (5) filled, (6) fluid, or (7) goat? If you want to see more choices, let me know.
> dry
Anna:
Ok, dry milk. Please choose from: (1) nonfat or (2) whole?
> 1
Anna:
Nonfat, dry milk. (1) instant, (2) regular, or (3) calcium reduced?
```

The reader will notice that both the food reprise (for implicit confirmation) and the prompt to make a selection became less verbose gradually.

5.3 The Backend

The key element of the backend is the National Nutrient Database from the U.S. Department of Agriculture (2008). We downloaded its data and put it in a SQLite database for offline, programmatic access. It should be noted that the information needed was not readily available as a KB for SIGMA-KEE (Pease, 2003) or any other such tool.

To complement this database, the backend provides a set of functions to generate and execute SQL queries and parse their results. To illustrate this last task, consider for example a query for ‘apples’, which returns the following 13 rows:

```
Apples, raw, with skin
Apples, raw, without skin
Apples, raw, without skin, cooked, boiled
```

The reader will notice that both the food reprise (for implicit confirmation) and the prompt to make a selection became less verbose gradually.

---

6 http://www.sqlite.org
Apples, canned, sweetened, sliced, drained, unheated
Apples, canned, sweetened, sliced, drained, heated
Apples, dehydrated, sulfured, uncooked
Apples, dehydrated, sulfured, stewed
Apples, dried, sulfured, uncooked
Apples, dried, sulfured, stewed, without added sugar
Apples, dried, sulfured, stewed, with added sugar
Apples, frozen, unsweetened, unheated
Apples, frozen, unsweetened, heated

The backend structures these results in a tree so that the dialogue manager can ask the user to select first between raw, canned, dehydrated, dried, or frozen apples, rather than presenting all 13 choices at the same time. 

5.4 The Chatbot

The chatbot is a conversational agent all by itself. Rather than being task-oriented, the chatbot’s main strength is its appearance of being human. Our reason for including it is to make our dialogue system more user-friendly by allowing it to reply to out-of-domain inputs such as what’s your name, etc. properly rather than reporting an error.

Our chatbot is based on AIML, the Artificial Intelligence Markup Language, from the A.L.I.C.E. Artificial Intelligence Foundation. Most of its content (AIML programming) comes from the so-called Standard AIML Set from said foundation. We modified some of the files to make the chatbot less likely to ask out-of-domain questions and to respond to the help universal command. We used PyAIML as the AIML interpreter.

6 Commands and Error Handling

Anna has three universal commands:

- **Help**: Display a general description of the system.
- **Clear**: Clears the dialogue manager task context, i.e. the information about current nutrient and food.
- **Quit**: Terminates the program.

Anna will also attempt to handle the following type of errors: spelling errors, misunderstandings, out-of-grammar errors, and out-of-application errors.

6.1 Spelling errors

The NLU component uses Peter Norvig’s so called toy spelling corrector (2008), trained on our nutrient and food lexicon and some additional words like those included in salutations, interjections, and so on.

6.2 Misunderstandings

Although Anna does not have a speech recognition component, it can still misunderstand the user due to spelling under- or over-correction, or incomplete information. To verify understanding, Anna uses explicit verification for nutrients and implicit verification for foods (which need to be verified much more often.) In both cases, the user can reject Anna’s understanding. This will initiate a small repair dialogue to ask the user again for the nutrient or food.

6.3 Out-of-Grammar Errors

Out-of-grammar errors are hard to recognize because of the robustness of the Phoenix parser and the fact that our main frame, the nutrient-quantity-request frame, allows for partial information. For example, let’s say the user asks: how many calories are there in brownies? and assume brownies is not in the grammar. The frame will still be valid because calories is a valid nutrient, so the system will assume that no food was specified and it will ask for it. This is of course not very good since the user is being asked something he just answered. However, this second time, since the system is asking explicitly for food, if it does not find the answer in it its lexicon, it will recognize it as an out-of-grammar error and will inform the user that it has no data for it.

6.4 Out-of-Application Errors

Out-of-application user inputs, as discussed, will go directly to the chatbot. If the chatbot does not have a good answer for it either, it will display a catch-all message such as I’m confused.

---

7 For a more dramatic example, consider that there are 750 kinds of beef in the database.
8 http://www.alicebot.org

---

9 http://norvig.com/spell-correct.html
7 Evaluation and Discussion

Our dialogue system has not been formally evaluated. In informal tests it performed as expected.

There are of course many things that could be done to improve this system, but given more time, our focus would be in extending its scope to handle serving sizes and more complex caloric content queries.

Acknowledgments

We would like to thank Scott Farrar and Bill McNeil for their guidance throughout this project.

References


