Chatterbot with common sense

Degree Project in Computer Science, First Level
DD143X
School of Computer Science and Communication
Swedish Royal Institute of Technology
Supervisor: Gabriel Skantze

Max Malmgren
Nyodlingsvägen 35
16766 Bromma
0708205661
maxmal@kth.se

Ulf Åhammar
Sjödalstorget 15
14147 Huddinge
0709725539
uahammar@kth.se

Abstract

A semantic network is a database containing information regarding how concepts relate to one another. In this report the effects of consulting such a network to generate responses for a chatterbot based on the ELIZA computer program are examined. Although unsuccessful in creating a program that generates replies that makes sense on a regular basis, the research shows evidence of the technology being useful when creating chatterbots.

1. Introduction

1.1 Background

In 1966 Joseph Weizenbaum published a program called ELIZA, an early example of software using natural language processing to construct a dialog between computer and human, called a chatterbot. ELIZA was able to respond to some predefined queries with a semblance of awareness, but otherwise responded with a generic prompt to give more information. Since then a large number of attempts have been made to improve upon the perceived or actual intelligence of chatterbots, notable examples are Google Cleverbot, A.L.I.C.E and Elbot.

A detracting aspect of any conversation with ELIZA is the fact that the bot offers no new content to the discussion - having no knowledge base, it could not reference any subject other than rote repetition of the user input. This easily leads to an impression of a stale chatterbot.

To provide such content, one possible solution is to use a *semantic network*. Such a network is a database providing relations between different primitives, or keywords. A program using it can use user input as root concept for a query and form responses based on related subjects.

1.2 Thesis objective

This thesis intends to examine the possibilities of using a semantic network to provide a method for a bot to advance the conversation, specifically the online semantic network Open Mind Common Sense(OMCS). Furthermore, the thesis will examine the quality of the provided data in regards to the application, and how that might affect the result. The specific context under which the conversation occurs is that of patient to psychologist.

The material used for this thesis are relevant literature in conjunction with an implementation of an ELIZA-like bot, which will be compared to a custom chatterbot using the Open Mind Common Sense database.

1.3 Restrictions

The thesis will not encompass other implementations of chatterbots

1.4 Audience

The thesis targets any person interested in chatterbots and natural language processing, with a background in Computer Science.

1.5 Methodology

A pre-existing implementation of ELIZA will be used as a base, upon which improvements will be made. These improvements will be partitioned into several versions, each of which will be evaluated in comparison to the other versions.

The original ELIZA was written in the rather obscure programming language SLIP[1], and there have been many implementations since. For that reason we shall not use the original implementation. Our version will be compared to a Python implementation originally made by Steven Bird and Edward Loper, based upon an implementation by Joe Strout, Jeff Epler and Jez Higgins. Test subjects will then have conversations with either implementation, and case studies of the textual logs will be performed to deduce differences in quality and reasons thereof.

2 Extended Background and Basic Concepts

This section will further explain chatterbots and related terminology.

As touched upon in section 1, ELIZA is a very simple chatterbot. Technically, it uses simple text replacement techniques to generate answers. A number of predefined responses exists, using pattern matching with the user input to create the output. Even being so severely limited ELIZA created a stir among the test subjects conversing with the pretend psychologist. Author Joseph Weizenbaum noted early that individuals often asked for him to leave during their conversation, and asking for the conversation to go unrecorded. [1]

2.1 Natural language processing

ELIZA uses simple text replacement techniques to generate answers. No consideration is afforded to properties of the patients input, such as relevant part-of-speech or formulating grammatically correct responses. To enable effective implementation of other improvements such as using a knowledge base it is important to perform some form of processing to determine how to best use the input. It can be argued that performing part-of-speech analysis, differentiating between for example noun or verb, and making decision based on that information will likely make a bot seem more insightful.

2.2 Common sense

The psychologist bot ELIZA has a very small data bank consisting of mapping certain input phrases to certain output phrases, in most cases regurgitating the input in slightly different form to the user. As a result ELIZA can never itself bring the conversation onwards, by adding new concepts or content. One possible way to improve the chatterbots behavior is thus to provide a way for it to move to a tangential topic - for example from the topic ‘family’ to the more general topic ‘relative’. This form of conversational shifting is natural to humans, but a computer has no inherent notion of relations between concepts. The bot needs to access some form of database from which it can make decisions on which direction to move the conversation in. Such a database needs to be specialised to its purpose - to contain the generalised concepts and relations of reality a human will start forming at birth. It is commonly referred to as a *semantic network*.

A semantic network is simply enough a graph of objects with edges denoting some relationship between said objects.[3] In this instance, the semantic network contains knowledge of the intuitive relations between concepts that all humans share. ‘Fire’ is ‘hot’, and can be used to make ‘food’, which is required to sustain ‘life’. Informally, it could be called the ‘common sense’.

Equipping the chatterbot ELIZA with such a knowledge base would allow it to further a conversation on its own.

However, a semantic network does not just come into being. Creating a semantic network is a daunting project. Million tidbits of knowledge humans take for granted must be inserted and recorded and made available.

The material for this thesis will include a supposed improvement upon the original ELIZA based upon using the Open Mind Common Sense semantic network authored by the Media Lab of MIT.

3 Objective satisfiability

In order for the purpose of this thesis to be successfully achieved evidence of whether or not the perceived intelligence of an ELIZA-like bot has the potential of being improved by using OMCS need to be found, and that evidence must be sound.

The bulk of the data to base conclusions on will be had from prototype implementations of using a specific semantic network within a specific bot, and as such, it is imperative to form as objective means of measurement as possible. However, due to the nature of the output (textual conversation), the method of evaluation chosen is performing case-studies on conversations made by test subjects. Since the subject under test is intelligence of computer software as perceived by a human, the immediate concern is defining guidelines as to what should be perceived as intelligence.

Since the purpose of this thesis is to explore the viability of using semantic networks to further a conversation in the context of a patient-psychologist relation there is no need for a fully functional prototype, but only the beginnings of one, assuming that problems faced are not insurmountable flaws in the approach.

Even though an implementation that performs better in an objectively measurable way would conclusively prove that the use of the technology is a valid way to improve the perceived intelligence, it is not a necessity.

4 Design and Implementation

4.1 ELIZA

The original ELIZA will upon receiving an input sentence try to pattern match it to one of its built in sentence-parts. If it finds that part of the sentence matches it will randomise one of a few preset ways of generating a response. Sometimes the whole response is hardcoded into the program itself, ELIZA will then just print out the text it found. There are also times when the preset way of responding includes a part of the input message. ELIZA will then add the part of the sentence that wasn't matched to a pattern to the response, i.e. if the input sentence is 'I feel sad.' ELIZA can respond 'Do you often feel sad?' It uses sad from the input sentence and adds it onto the preset 'Do you often feel' response.

If a part of the input sentence includes certain words, ELIZA will replace them with others from a built-in dictionary before it uses them in the response. This is to reflect statements back at the user. If, for example, the input sentence is 'I feel like I don't belong.', ELIZA will replace 'I' using the its dictionary with 'you' and can if the randomised response sentence is the 'Do you often feel'-response print out 'Do you often feel like you don't belong?'.

4.2 OMCS and ConceptNet

ConceptNet is a semantic network that makes use of data from many different sources, among which OMCS is one. [4] It, unlike OMCS itself, provides an API which can be used to extract information from all or just a specific source.

Data is stored as relations between different concepts, each relation has eighteen different attributes, all of can be used for when searching for a relation. Searches are made using HTML GET requests and results are returned in JSON format.

Looking at the data in Appendix A1, which is one of the resulting relations from querying the network for the word 'feel'. The 'surfaceText'-attribute says that '[[feel]] is a type of [[emotion]]', with the words contained in the double brackets being the different concepts that which the relation connects. The first concept in these kinds of texts is known as the 'start' and the second one is known as the 'end'.

The nodes attribute contains information on not only which concepts are included in the relation, but also how the two relate to one another. In the Appendix A1 we can see that the node attribute contains the concepts 'feel', 'emotion' and that the type of relation is 'IsA'. This means that the 'start' concept is an instance of the 'end' concept.

Another important attribute is the ‘source’ attribute, it contains information regarding where ConceptNet gathered the data from. In the case of the data in Appendix A1, this is from OMCS.

4.3 Implementation one

The original method of adding common sense to ELIZA included using only techniques already present in ELIZA itself. The exception being the sending of queries and handling of responses sent to and from ConceptNet. Instead of just printing a response after matching part of the sentence like ELIZA did, the program would query the ConceptNet semantic network for information from OMCS. And use the response in it’s reply.

Seeing as queries to ConceptNet typically only include one word, it would pick the first word that was not part of the matching part of the input sentence and did not match one of the words that it had been instructed to ignore, such as ‘a’, ‘an’ and ‘like’. Use it to query ConceptNet and then if there were more than one resulting response randomise which word to use in the reply sentence. It had separate preset sentences to use with the results from ConceptNet queries, if the query yielded no result it would default to the old ELIZA answers.

As searches for concepts can be made in two different ways, start to end and end to start, both are done with each query word.

The preset sentences made for use with the resulting responses from ConceptNet include information on how the concept has to relate to the word used in the query. This is done to help with properly formatting the reply sentence.

4.4 Natural Language Toolkit

Natural Language Toolkit(NLTK) is an extensive platform for processing natural language with Python. It is used in our research for its many word class identification features. By using its tagging feature on a sentence, you can identify properties of all the words in it. This includes word class and grammatical number, which are most relevant to our purpose. [2]

There is also an important feature called ‘lemmatize’, which basically returns words to their ‘base’ form. It turns plural words into singular, comparatives into the words they inflect and so on.

4.5 Implementation two

One major difference between the between the first and second implementation is the way the word being used to query ConceptNet is chosen. In the second version, the word is chosen by going backwards through the input sentence and checking the word class of each word. As soon

as a noun, adjective or verb is found, it will be used in the query. A check of the returned results of the query was also implemented, to make sure the word class of the returned word is the same as the word used in the query. The number of words of the returned concept was implemented, because sometimes a result could be in the form of a whole sentence. These results rarely worked well with our sentences.

In order to make the responses from ConceptNet more predictable, and therefore easier to put into sentences, a way of deciding which type of search to be used for each individual preset sentence was added in version two. This was needed because certain sentences needed the word that was to be put in it to be broader than the original word.

Another feature added in the second version is the way the bot handles the number class of the result. NLTK has built-in support of detecting the number class of a word and turning it into singular. However turning a singular word to its plural form is not. To accomplish that a separate function had to be made. It does not work for all words, but for the majority of words simple detection of the last few letters of a word and replacing them with the proper suffix works fine.

The ‘lemmatize’ feature of NLTK not only used to convert plural into singular, it is also used before querying ConceptNet, so that the query is more likely to be successful. This is because concepts are usually stored as the ‘base’ word, there is no separate entry for every way of using each word.

Something that was found to mostly cause bad responses was when long sentences were returned from the query, these were generally very hard to fit into the preset sentences. This led to a limit of two words, excluding small words such as ‘a’ and ‘an’, to be implemented.

A bug with the first implementation that was solved in the second was the problem with queries returning results for all concepts starting with the desired concept. In the first implementation a search for ‘car’ could potentially return results for ‘carp’.

5 Data acquisition

This section describes data acquisition, error sources and test methodology.

This thesis will be supported by data from three different implementations of an ELIZA-like bot, based on each other. These versions are described in the section 4.

5.1 Chatterbot prototypes

Note that the acquired data is found in Appendix B.

The data acquired from the prototypes are textual logs of conversations performed by various persons.

Since the purpose of this thesis is evaluating the viability of using the target form of semantic networks as a way of furthering conversation and not whether our current implementation is superior, more focus will be afforded positive results than negative. We will examine positive and negative results separately, and attempt to discern why an individual response was insightful or nonsensical.

Given the purpose of the thesis, and that the available implementations are such an early prototypes, it is not meaningful to examine full conversations, and thus single statements will be given as input to all systems under test multiple times. The output and chain of execution inside each bot will be examined to determine why a certain result occurred. Results will then be compared, to determine the potential of the version. The original ELIZA-like version will be used as a baseline.

As noted above, the acquired data can be found in Appendix B.

6 Evaluation

This section contains a summary of the data acquired and preliminary comments on the results.

6.1 Version 1

As logs *V1 Log A* to *V1 Log X* shows, without natural language processing, it is difficult to pose a grammatically correct meaning and have the bot query the database with a relevant word.

V1 Log A clearly shows this. The first entry is a longer sentence, but the bot does not pick up ‘love’ as the relevant word. Subsequent entries show responses based upon the correct word.

V1 Log B is also an example of the above phenomenon.

Something else worth of note is that almost all replies are nonsensical.

6.2 Version 2

V2 Log A shows, in contrast to *V1 Log A*, that the bot now picks the correct word to perform the query upon. The statement generated is still of questionable quality due to the data returned from the database (mother -> fathers wife).

V2 Log B show how the bot seems to be better at handling queries about physical objects. Despite seeming contrived in a patient-psychologist setting, it could be argued that the conversation shows a spark of ‘insight’.

Most replies are still nonsensical, but are more often grammatically correct due to the limited algorithm to transform output words and sentences to the correct form.

6.3 Baseline comparison

Baseline Log A shows at once the rogerian-psychologist-type statements that makes the original chatterbot show a glimmer of insight and how those statements quickly go stale as conversation continues. It also shows the inability of the bot to generate grammatically correct sentences.

Comparing the baseline logs to the other versions’ logs shows that the replies from V1 and V2 are generally more nonsensical than that of the baseline, except when favorable data is returned from Open Mind Common Sense.

6.4 Soundness of data

Since the method of evaluation used is performing a number of case studies on conversations made by test subjects all the inherent flaws of that method is present in our results. The number of case studies are naturally severely limited due to time constraints, which means the thesis is supported by a small sample size. Defining and measuring perceived intelligence is also a difficulty and a source of ambiguity. It is difficult to form objective and meaningful guidelines of evaluating perceived intelligence.

7 Epilogue

7.1 Conclusions

This section contains a summary of the acquired results and what conclusions the authors of this thesis draws from them.

Although most of the responses generated by our bot can be considered worse than the original ELIZA, there are a few replies which are of a very high quality. The good replies fulfil the goal we originally had of broadening the subject of the conversation and increase the perceived intelligence of the bot. This leads us to the conclusion that there is potential for this technology to be used with chatterbots.

The nature of the concepts associated with psychology are generally quite hard to grasp, as a nearly all of them have to do with human emotions. This, one would think, makes semantic networks an ideal way to give chatterbots some kind of insight into how the concepts relate to each other. But what we found during the course of the work going into this thesis, is that it seemingly worked much better for physical things and concepts. This leads us to believe that this kind of technology might be better suited for conversations about different subjects than psychology.

7.2 Discussion

This section discusses how to improve the behavior of a bot, grouped by area, including difficulties and challenges faced.

One of the biggest difficulties with using the semantic network to improve the quality of the replies was to adapt the pre made answers to fit the results returned by the query to ConceptNet. Considering the variety of concepts stored in OMCS, this was a nearly impossible task. In order to this properly, the NLP would need to be a lot more advanced and a larger part of the replies would have to be dynamically generated to fit the response from the query.

There are basically three different ways in which our returned sentences can make no sense at all. The first of which is that we use an irrelevant word for the query. The second way is when the returned result does not fit with our preset reply sentence. Both of these can be fixed by utilizing better NLP. With more development time and experience, we are confident these issues can be fixed. The third reason for our replies not making sense is that the data returned from OMCS simply doesn’t. This problem could be fixed by either removing bad data from OMCS or having a scoring system which keeps these kinds of results from being returned. And there actually is one.

However the scoring system does not seem to work very well. As can be seen in Appendix A2 all

the scores are all the same. This creates problems when trying to decide which result to return. There are many issues with the data stored in OMCS, an ex an IsA relation for end type car is '[[this]] is a [[car]]', this data does little more than telling us that car is a noun. If the scoring system was working, relations like this one would probably be scored very low and we could filter them out. Whereas a relation such as '[[car]] is a [[motor vehicle]]' would be scored high. There are more examples more closely related to psychotherapy such as these results for 'feel', a good response would be '[[feel]] is a type of [[emotion]]' and a bad one is [[feel]] is a type of [[with finger]]. The latter of which doesn't make any sense at all. These things would all be solved if the scoring system worked, sadly though it seems it doesn't.

7.2.1 Semantic networks

An inherent difficulty with constructing a semantic network of humanity's reality is that different individuals or societies has radically different perspective on the relations between objects and concepts, and inevitably there must be some form of preconceptions, which makes it difficult to form a globally 'correct' database.

It could be argued that to form a truly meaningful conversation, some value must be attached to any applicable concept. For example, is love 'good' or 'bad'? Is fidelity to one's partner important? Is danger 'good' or 'bad'? Many humans would argue that it depends on the situation, and such decision-making is out of the scope of a semantic network.

The drawback of solely using a semantic network noted above stem from the definition of such a database, regardless of the accuracy of the data held within.

7.3 Future Work

Although we found that there could be uses for this kind of technology when creating chatterbots, we did not manage to create a very good one ourselves. This in combination with the fact that ELIZA is a very old and outdated chatterbot, makes one future task within this subject quite obvious, make a chatterbot that successfully implements using a semantic network to achieve perceived understanding of the subject of the conversation.

There is also a lot to be done with the semantic networks themselves, a lot of the information stored within them is quite irrelevant and sometimes wrong. Although this could be solved by having a working scoring system, as talked about in the discussion part of the thesis, sometimes the information is just not there to begin with. OMCS and similar projects need to continue their work to gather information and keeping said information relevant.

Furthermore, the bot implementations seen here only use one level of relations, using a heuristically chosen word of the input sentence. Given a more solid semantic network, it would be interesting to see the results if more advanced techniques in traversing the graph it represents were used, taking into account more relations to ascribe more meaning.

8. References

[1] http://www.cse.buffalo.edu/~rapaport/572/S02/weizenbaum.eliza.1966.pdf [Retrieved 2013-04-10]
[2] http://nltk.org/book/ [Retrieved 2013-04-10]
[3] http://www.jfsowa.com/pubs/semnet.htm [Retrieved 2013-04-10]
[4] http://conceptnet5.media.mit.edu/ [Retrieved 2013-04-10]

8.1 Summary of Literature

Summary of the literature (reports, papers, web sites, etc) which the author(s) have read in relation to the essay work.
[1] *ELIZA* - The original article in which ELIZA is described, also includes a source code of ELIZA written in the SLIP programming language.
[2] *Natural Language Processing with Python* - The online book describing how to analyse text with the Natural Language Toolkit. In particular the chapter about categorising and tagging words.
[3] *Semantic Networks* - Definition and description of semantic networks and how they work.
[4] *ConceptNet* - The official website of ConceptNet which describes how it works and which sources it uses

Appendix A1

This appendix contains a specific result from a search using ConceptNet.


```
{
    "endLemmas": "emotion",
    "rel": "/r/IsA",
    "end": "/c/en/emotion",
    "features": [
      "/c/en/feel /r/IsA -",
      "/c/en/feel - /c/en/emotion",
      "- /r/IsA /c/en/emotion"
    ],
    "license": "/l/CC/By",
    "sources": [
      "/s/activity/omcs/Verbosity",
      "/s/contributor/omcs/verbosity"
    ],
    "startLemmas": "feel",
    "text": [
      "emotion",
      "feel"
    ],
    "uri": "/a/[/r/IsA/,/c/en/feel/,/c/en/emotion/]",
    "weight": 1.0,
    "dataset": "/d/conceptnet/4/en",
    "start": "/c/en/feel",
    "score": 12.613185,
    "context": "/ctx/all",
    "timestamp": "2013-03-04T22:03:48.223Z",
    "nodes": [
      "/c/en/emotion",
      "/c/en/feel",
      "/r/IsA"
    ],
    "id": "/e/7592d9a85759905a25fdfecf112744ec0510b969",
    "surfaceText": "[[feel]] is a type of [[emotion]]"
  }
```

Appendix A2

This appendix contains raw result data from searches made using ConceptNet.


```
{
  "numFound": 10,
  "edges": [
    {
      "endLemmas": "emotion",
      "rel": "/r/IsA",
      "end": "/c/en/emotion",
      "features": [
        "/c/en/feel /r/IsA -",
        "/c/en/feel - /c/en/emotion",
        "- /r/IsA /c/en/emotion"
      ],
      "license": "/l/CC/By",
      "sources": [
        "/s/activity/omcs/Verbosity",
        "/s/contributor/omcs/verbosity"
      ],
      "startLemmas": "feel",
      "text": [
        "emotion",
        "feel"
      ],
      "uri": "/a/[/r/IsA/,/c/en/feel/,/c/en/emotion/]",
      "weight": 1.0,
      "dataset": "/d/conceptnet/4/en",
      "start": "/c/en/feel",
      "score": 12.613185,
      "context": "/ctx/all",
      "timestamp": "2013-03-04T22:03:48.223Z",
      "nodes": [
        "/c/en/emotion",
        "/c/en/feel",
        "/r/IsA"
      ],
      "id": "/e/7592d9a85759905a25fdfecf112744ec0510b969",
      "surfaceText": "[[feel]] is a type of [[emotion]]"
    },
```

```
{
  "endLemmas": "emotion",
  "rel": "/r/IsA",
  "end": "/c/en/emotion",
  "features": [
    "/c/en/feel /r/IsA -",
    "/c/en/feel - /c/en/emotion",
    "- /r/IsA /c/en/emotion"
  ],
  "license": "/l/CC/By",
  "sources": [
    "/s/activity/omcs/vote",
    "/s/contributor/omcs/verbosity"
  ],
  "startLemmas": "feel",
  "text": [
    "emotion",
    "feel"
  ],
  "uri": "/a/[/r/IsA/,/c/en/feel/,/c/en/emotion/]",
  "weight": 1.0,
  "dataset": "/d/conceptnet/4/en",
  "start": "/c/en/feel",
  "score": 12.613185,
  "context": "/ctx/all",
  "timestamp": "2013-03-04T22:03:48.223Z",
  "nodes": [
    "/c/en/emotion",
    "/c/en/feel",
    "/r/IsA"
  ],
  "id": "/e/5a3ec803842a4ce5ca7975b2b3e1a22ca5934036",
  "surfaceText": "[[feel]] is a type of [[emotion]]"
},
{
  "endLemmas": "fabric",
  "rel": "/r/IsA",
  "end": "/c/en/fabric",
  "features": [
    "/c/en/feel /r/IsA -",
    "/c/en/feel - /c/en/fabric",
    "- /r/IsA /c/en/fabric"
  ],
```

```
  "license": "/l/CC/By",
  "sources": [
    "/s/activity/omcs/Verbosity",
    "/s/contributor/omcs/verbosity"
  ],
  "startLemmas": "feel",
  "text": [
    "fabric",
    "feel"
  ],
  "uri": "/a/[/r/IsA/,/c/en/feel/,/c/en/fabric/]",
  "weight": 1.0,
  "dataset": "/d/conceptnet/4/en",
  "start": "/c/en/feel",
  "score": 12.613185,
  "context": "/ctx/all",
  "timestamp": "2013-03-04T22:03:48.235Z",
  "nodes": [
    "/c/en/fabric",
    "/c/en/feel",
    "/r/IsA"
  ],
  "id": "/e/4616c6b46c60aca2324049cedda77f4eb5760c52",
  "surfaceText": "[[felt]] is a type of [[fabric]]"
},
{
  "endLemmas": "fabric",
  "rel": "/r/IsA",
  "end": "/c/en/fabric",
  "features": [
    "/c/en/feel /r/IsA -",
    "/c/en/feel - /c/en/fabric",
    "- /r/IsA /c/en/fabric"
  ],
  "license": "/l/CC/By",
  "sources": [
    "/s/activity/omcs/vote",
    "/s/contributor/omcs/verbosity"
  ],
  "startLemmas": "feel",
  "text": [
    "fabric",
    "feel"
```

```
  ],
  "uri": "/a/[/r/IsA/,/c/en/feel/,/c/en/fabric/]",
  "weight": 1.0,
  "dataset": "/d/conceptnet/4/en",
  "start": "/c/en/feel",
  "score": 12.613185,
  "context": "/ctx/all",
  "timestamp": "2013-03-04T22:03:48.235Z",
  "nodes": [
    "/c/en/fabric",
    "/c/en/feel",
    "/r/IsA"
  ],
  "id": "/e/44827b449046222b3bbfee3dbf82b059dc4c83fb",
  "surfaceText": "[[felt]] is a type of [[fabric]]"
},
{
  "endLemmas": "nerve sense",
  "rel": "/r/IsA",
  "end": "/c/en/nerve_sense",
  "features": [
    "/c/en/feel /r/IsA -",
    "/c/en/feel - /c/en/nerve_sense",
    "- /r/IsA /c/en/nerve_sense"
  ],
  "license": "/l/CC/By",
  "sources": [
    "/s/activity/omcs/Verbosity",
    "/s/contributor/omcs/verbosity"
  ],
  "startLemmas": "feel",
  "text": [
    "nerve sense",
    "feel"
  ],
  "uri": "/a/[/r/IsA/,/c/en/feel/,/c/en/nerve_sense/]",
  "weight": 1.0,
  "dataset": "/d/conceptnet/4/en",
  "start": "/c/en/feel",
  "score": 12.613185,
  "context": "/ctx/all",
  "timestamp": "2013-03-04T22:03:48.435Z",
  "nodes": [
```

```
    "/c/en/nerve_sense",
    "/c/en/feel",
    "/r/IsA"
  ],
  "id": "/e/51e1ca68fc7bd801f7983b486c5c09a364bfb18c",
  "surfaceText": "[[feel]] is a type of [[nerve sense]]"
},
{
  "endLemmas": "nerve sense",
  "rel": "/r/IsA",
  "end": "/c/en/nerve_sense",
  "features": [
    "/c/en/feel /r/IsA -",
    "/c/en/feel - /c/en/nerve_sense",
    "- /r/IsA /c/en/nerve_sense"
  ],
  "license": "/l/CC/By",
  "sources": [
    "/s/activity/omcs/vote",
    "/s/contributor/omcs/verbosity"
  ],
  "startLemmas": "feel",
  "text": [
    "nerve sense",
    "feel"
  ],
  "uri": "/a/[/r/IsA/,/c/en/feel/,/c/en/nerve_sense/]",
  "weight": 1.0,
  "dataset": "/d/conceptnet/4/en",
  "start": "/c/en/feel",
  "score": 12.613185,
  "context": "/ctx/all",
  "timestamp": "2013-03-04T22:03:48.435Z",
  "nodes": [
    "/c/en/nerve_sense",
    "/c/en/feel",
    "/r/IsA"
  ],
  "id": "/e/a8f67482e36349b2aed90570eccd3ae217c0e777",
  "surfaceText": "[[feel]] is a type of [[nerve sense]]"
},
{
  "endLemmas": "perceive",
```

```
  "rel": "/r/IsA",
  "end": "/c/en/perceive",
  "features": [
    "/c/en/feel /r/IsA -",
    "/c/en/feel - /c/en/perceive",
    "- /r/IsA /c/en/perceive"
  ],
  "license": "/l/CC/By",
  "sources": [
    "/s/activity/omcs/Verbosity",
    "/s/contributor/omcs/verbosity"
  ],
  "startLemmas": "feel",
  "text": [
    "perceive",
    "feel"
  ],
  "uri": "/a/[/r/IsA/,/c/en/feel/,/c/en/perceive/]",
  "weight": 1.0,
  "dataset": "/d/conceptnet/4/en",
  "start": "/c/en/feel",
  "score": 12.613185,
  "context": "/ctx/all",
  "timestamp": "2013-03-04T22:03:48.479Z",
  "nodes": [
    "/c/en/perceive",
    "/c/en/feel",
    "/r/IsA"
  ],
  "id": "/e/d6b92469fa9cd38fd21fd76e2ae6762ee92e885d",
  "surfaceText": "[[feel]] is a type of [[perceive]]"
},
{
  "endLemmas": "perceive",
  "rel": "/r/IsA",
  "end": "/c/en/perceive",
  "features": [
    "/c/en/feel /r/IsA -",
    "/c/en/feel - /c/en/perceive",
    "- /r/IsA /c/en/perceive"
  ],
  "license": "/l/CC/By",
  "sources": [
```

```
    "/s/activity/omcs/vote",
    "/s/contributor/omcs/verbosity"
  ],
  "startLemmas": "feel",
  "text": [
    "perceive",
    "feel"
  ],
  "uri": "/a/[/r/IsA/,/c/en/feel/,/c/en/perceive/]",
  "weight": 1.0,
  "dataset": "/d/conceptnet/4/en",
  "start": "/c/en/feel",
  "score": 12.613185,
  "context": "/ctx/all",
  "timestamp": "2013-03-04T22:03:48.479Z",
  "nodes": [
    "/c/en/perceive",
    "/c/en/feel",
    "/r/IsA"
  ],
  "id": "/e/ce9358c29651114a8b47bc46466edde1461af3a7",
  "surfaceText": "[[feel]] is a type of [[perceive]]"
},
{
  "endLemmas": "with finger",
  "rel": "/r/IsA",
  "end": "/c/en/with_finger",
  "features": [
    "/c/en/feel /r/IsA -",
    "/c/en/feel - /c/en/with_finger",
    "- /r/IsA /c/en/with_finger"
  ],
  "license": "/l/CC/By",
  "sources": [
    "/s/activity/omcs/Verbosity",
    "/s/contributor/omcs/verbosity"
  ],
  "startLemmas": "feel",
  "text": [
    "with finger",
    "feel"
  ],
  "uri": "/a/[/r/IsA/,/c/en/feel/,/c/en/with_finger/]",
```

```
  "weight": 1.0,
  "dataset": "/d/conceptnet/4/en",
  "start": "/c/en/feel",
  "score": 12.613185,
  "context": "/ctx/all",
  "timestamp": "2013-03-04T22:03:48.699Z",
  "nodes": [
    "/c/en/with_finger",
    "/c/en/feel",
    "/r/IsA"
  ],
  "id": "/e/66c2818a6dfa8f20a4d4b78927d82b65b6bdcce4",
  "surfaceText": "[[feel]] is a type of [[with finger]]"
},
{
  "endLemmas": "with finger",
  "rel": "/r/IsA",
  "end": "/c/en/with_finger",
  "features": [
    "/c/en/feel /r/IsA -",
    "/c/en/feel - /c/en/with_finger",
    "- /r/IsA /c/en/with_finger"
  ],
  "license": "/l/CC/By",
  "sources": [
    "/s/activity/omcs/vote",
    "/s/contributor/omcs/verbosity"
  ],
  "startLemmas": "feel",
  "text": [
    "with finger",
    "feel"
  ],
  "uri": "/a/[/r/IsA/,/c/en/feel/,/c/en/with_finger/]",
  "weight": 1.0,
  "dataset": "/d/conceptnet/4/en",
  "start": "/c/en/feel",
  "score": 12.613185,
  "context": "/ctx/all",
  "timestamp": "2013-03-04T22:03:48.699Z",
  "nodes": [
    "/c/en/with_finger",
    "/c/en/feel",
```

```
        "/r/IsA"
      ],
      "id": "/e/fcc564e923129d6e840b125472d366e4ac43b31d",
      "surfaceText": "[[feel]] is a type of [[with finger]]"
    }
  ],
  "maxScore": 12.613185
}
```

Appendix B

This appendix contains conversations or snippets of conversations with different implementations of the chatterbot. Entries made by the human is indented three characters with angular brackets.

Version 1 (Use of Common Sense)

V1 Log A

>>> I need love in my life
Is that because it has much?
>>> I need love
Do you have need of other valentine?
>>> I need love
Do you have need of other feeling we?
>>> I need love
Do you have need of other charity?

V1 Log B

Hello. How are you feeling today?
>>> I am looking to buy a car
So you're true?
>>> I am looking to buy a car
Do you wings?
>>> I am looking to buy a car
Do you named connie?
>>> I need car
Do you have need of other motor vehicle?

V1 Log C

Hello. How are you feeling today?
>>> My daughter won't listen to me. It makes me angry
Is that because it has mother?
>>> I am her mother
Are you this location?

Version 2 (Use of Common Sense and more sophisticated NLP)

V2 Log A

>>> I have problems with my mother
What about other father wifes?
>>> I have problems with my mother
Does that statement hold for other father wifes?
>>> I have problems with my mother
Is that because of its children?

V2 Log B

>>> I am looking to buy a car
What about other motor vehicles?
>>> I might actually buy a motorcycle
How does that make you feel?
>>> I might actually buy a bike
Very interesting.

V2 Log C

>>> My daughter won't listen to me. It makes me angry
Why do you say that your daughter won't listen to me. it makes you angry?
>>> My daughter won't listen to me
What about other girl sons?

Baseline (Use of neither Common Sense or NLP)

Baseline Log A

>>> I need help with my mother
Are you sure you need help with your mother?
>>> Yes. She is hurting me emotionally.
You seem quite sure.
>>> Why would she do this to me?
Why do you think would she do this to me?

Baseline Log B

Hello. How are you feeling today?
>>> I am looking to buy a car
How long have you been looking to buy a car?
>>> For a few months. I am so stressed out, though.
Very interesting.
>>> I need help fighting that anxiety.

Are you sure you need help fighting that anxiety?

Baseline Log C

Hello. How are you feeling today?
>>> I am having trouble with my family. My daughter won't listen to me.
Did you come to me because you are having trouble with your family. your daughter won't listen to you?
>>> Indeed. I just get so angry, and then she closes up.
How do you feel when you say that?
>>> Sad.
I see.

Appendix C

This appendix contains source code of the three different chatterbots used in the research for this thesis.

Baseline

eliza.py

```
#---------------------------------------------------------------------
#  eliza.py
#
#  a cheezy little Eliza knock-off by Joe Strout <joe@strout.net>
#  with some updates by Jeff Epler <jepler@inetnebr.com>
#  hacked into a module and updated by Jez Higgins <jez@jezuk.co.uk>
#  last revised: 28 February 2005
#---------------------------------------------------------------------

import string
import re
import random

class eliza:
  def __init__(self):
        self.keys = map(lambda x:re.compile(x[0], re.IGNORECASE),gPats)
        self.values = map(lambda x:x[1],gPats)

  #---------------------------------------------------------------------
  # translate: take a string, replace any words found in dict.keys()
  #  with the corresponding dict.values()
  #---------------------------------------------------------------------
  def translate(self,str,dict):
        words = string.split(string.lower(str))
        keys = dict.keys();
        for i in range(0,len(words)):
        if words[i] in keys:
        words[i] = dict[words[i]]
        return string.join(words)

  #---------------------------------------------------------------------
```

```
  #  respond: take a string, a set of regexps, and a corresponding
  #       set of response lists; find a match, and return a randomly
  #       chosen response from the corresponding list.
  #---------------------------------------------------------------------
  def respond(self,str):
          # find a match among keys
          for i in range(0,len(self.keys)):
          match = self.keys[i].match(str)
          if match:
          # found a match ... stuff with corresponding value
          # chosen randomly from among the available options
          resp = random.choice(self.values[i])
          # we've got a response... stuff in reflected text where indicated
          pos = string.find(resp,'%')
          while pos > -1:
          num = string.atoi(resp[pos+1:pos+2])
          resp = resp[:pos] + \
          self.translate(match.group(num),gReflections) + \
          resp[pos+2:]
          pos = string.find(resp,'%')
          # fix munged punctuation at the end
          if resp[-2:] == '?.': resp = resp[:-2] + '.'
          if resp[-2:] == '??': resp = resp[:-2] + '?'
          return resp

#---------------------------------------------------------------------
# gReflections, a translation table used to convert things you say
#         into things the computer says back, e.g. "I am" --> "you are"
#---------------------------------------------------------------------
gReflections = {
  "am"   : "are",
  "was"  : "were",
  "i"      : "you",
  "i'd"  : "you would",
  "i've"  : "you have",
  "i'll"  : "you will",
  "my"  : "your",
  "are"  : "am",
  "you've": "I have",
  "you'll": "I will",
  "your"  : "my",
  "yours"  : "mine",
  "you"  : "me",
```

```
  "me"  : "you"
}

#----------------------------------------------------------------------
# gPats, the main response table.  Each element of the list is a
#  two-element list; the first is a regexp, and the second is a
#  list of possible responses, with group-macros labelled as
#  %1, %2, etc.
#----------------------------------------------------------------------
gPats = [
  [r'I need (.*)',
  [  "Why do you need %1?",
          "Would it really help you to get %1?",
          "Are you sure you need %1?"]],

  [r'Why don\'?t you ([^\?]*)\??',
  [  "Do you really think I don't %1?",
          "Perhaps eventually I will %1.",
          "Do you really want me to %1?"]],

  [r'Why can\'?t I ([^\?]*)\??',
  [  "Do you think you should be able to %1?",
          "If you could %1, what would you do?",
          "I don't know -- why can't you %1?",
          "Have you really tried?"]],

  [r'I can\'?t (.*)',
  [  "How do you know you can't %1?",
          "Perhaps you could %1 if you tried.",
          "What would it take for you to %1?"]],

  [r'I am (.*)',
  [  "Did you come to me because you are %1?",
          "How long have you been %1?",
          "How do you feel about being %1?"]],

  [r'I\'?m (.*)',
  [  "How does being %1 make you feel?",
          "Do you enjoy being %1?",
          "Why do you tell me you're %1?",
          "Why do you think you're %1?"]],

  [r'Are you ([^\?]*)\??',
```

```
[  "Why does it matter whether I am %1?",
        "Would you prefer it if I were not %1?",
        "Perhaps you believe I am %1.",
        "I may be %1 -- what do you think?"]],

[r'What (.*)',
[  "Why do you ask?",
        "How would an answer to that help you?",
        "What do you think?"]],

[r'How (.*)',
[  "How do you suppose?",
        "Perhaps you can answer your own question.",
        "What is it you're really asking?"]],

[r'Because (.*)',
[  "Is that the real reason?",
        "What other reasons come to mind?",
        "Does that reason apply to anything else?",
        "If %1, what else must be true?"]],

[r'(.*) sorry (.*)',
[  "There are many times when no apology is needed.",
        "What feelings do you have when you apologize?"]],

[r'Hello(.*)',
[  "Hello... I'm glad you could drop by today.",
        "Hi there... how are you today?",
        "Hello, how are you feeling today?"]],

[r'I think (.*)',
[  "Do you doubt %1?",
        "Do you really think so?",
        "But you're not sure %1?"]],

[r'(.*) friend (.*)',
[  "Tell me more about your friends.",
        "When you think of a friend, what comes to mind?",
        "Why don't you tell me about a childhood friend?"]],

[r'Yes',
[  "You seem quite sure.",
        "OK, but can you elaborate a bit?"]],
```

```
[r'(.*) computer(.*)',
[  "Are you really talking about me?",
        "Does it seem strange to talk to a computer?",
        "How do computers make you feel?",
        "Do you feel threatened by computers?"]],

[r'Is it (.*)',
[  "Do you think it is %1?",
        "Perhaps it's %1 -- what do you think?",
        "If it were %1, what would you do?",
        "It could well be that %1."]],

[r'It is (.*)',
[  "You seem very certain.",
        "If I told you that it probably isn't %1, what would you feel?"]],

[r'Can you ([^\?]*)\??',
[  "What makes you think I can't %1?",
        "If I could %1, then what?",
        "Why do you ask if I can %1?"]],

[r'Can I ([^\?]*)\??',
[  "Perhaps you don't want to %1.",
        "Do you want to be able to %1?",
        "If you could %1, would you?"]],

[r'You are (.*)',
[  "Why do you think I am %1?",
        "Does it please you to think that I'm %1?",
        "Perhaps you would like me to be %1.",
        "Perhaps you're really talking about yourself?"]],

[r'You\'?re (.*)',
[  "Why do you say I am %1?",
        "Why do you think I am %1?",
        "Are we talking about you, or me?"]],

[r'I don\'?t (.*)',
[  "Don't you really %1?",
        "Why don't you %1?",
        "Do you want to %1?"]],
```

```
[r'I feel (.*)',
[  "Good, tell me more about these feelings.",
        "Do you often feel %1?",
        "When do you usually feel %1?",
        "When you feel %1, what do you do?"]],

[r'I have (.*)',
[  "Why do you tell me that you've %1?",
        "Have you really %1?",
        "Now that you have %1, what will you do next?"]],

[r'I would (.*)',
[  "Could you explain why you would %1?",
        "Why would you %1?",
        "Who else knows that you would %1?"]],

[r'Is there (.*)',
[  "Do you think there is %1?",
        "It's likely that there is %1.",
        "Would you like there to be %1?"]],

[r'My (.*)',
[  "I see, your %1.",
        "Why do you say that your %1?",
        "When your %1, how do you feel?"]],

[r'You (.*)',
[  "We should be discussing you, not me.",
        "Why do you say that about me?",
        "Why do you care whether I %1?"]],

[r'Why (.*)',
[  "Why don't you tell me the reason why %1?",
        "Why do you think %1?" ]],

[r'I want (.*)',
[  "What would it mean to you if you got %1?",
        "Why do you want %1?",
        "What would you do if you got %1?",
        "If you got %1, then what would you do?"]],

[r'(.*) mother(.*)',
[  "Tell me more about your mother.",
```

```
        "What was your relationship with your mother like?",
        "How do you feel about your mother?",
        "How does this relate to your feelings today?",
        "Good family relations are important."]],

[r'(.*) father(.*)',
[  "Tell me more about your father.",
        "How did your father make you feel?",
        "How do you feel about your father?",
        "Does your relationship with your father relate to your feelings today?",
        "Do you have trouble showing affection with your family?"]],

[r'(.*) child(.*)',
[  "Did you have close friends as a child?",
        "What is your favorite childhood memory?",
        "Do you remember any dreams or nightmares from childhood?",
        "Did the other children sometimes tease you?",
        "How do you think your childhood experiences relate to your feelings today?"]],

[r'(.*)\?',
[  "Why do you ask that?",
        "Please consider whether you can answer your own question.",
        "Perhaps the answer lies within yourself?",
        "Why don't you tell me?"]],

[r'quit',
[  "Thank you for talking with me.",
        "Good-bye.",
        "Thank you, that will be $150.  Have a good day!"]],

[r'(.*)',
[  "Please tell me more.",
        "Let's change focus a bit... Tell me about your family.",
        "Can you elaborate on that?",
        "Why do you say that %1?",
        "I see.",
        "Very interesting.",
        "%1.",
        "I see.  And what does that tell you?",
        "How does that make you feel?",
        "How do you feel when you say that?"]]
]
```

```
#---------------------------------------------------------------------
#  command_interface
#---------------------------------------------------------------------
def command_interface():
  print "Therapist\n---------"
  print "Talk to the program by typing in plain English, using normal upper-"
  print 'and lower-case letters and punctuation.  Enter "quit" when done.'
  print '='*72
  print "Hello.  How are you feeling today?"
  s = ""
  therapist = eliza();
  while s != "quit":
          try: s = raw_input(">")
          except EOFError:
          s = "quit"
          print s
          while s[-1] in "!.": s = s[:-1]
          print therapist.respond(s)


if __name__ == "__main__":
  command_interface()
```

Version 1

eliza-conceptnet.py

```
#---------------------------------------------------------------------
#  eliza.py
#
#  a cheezy little Eliza knock-off by Joe Strout <joe@strout.net>
#  with some updates by Jeff Epler <jepler@inetnebr.com>
#  hacked into a module and updated by Jez Higgins <jez@jezuk.co.uk>
#  last revised: 28 February 2005
#---------------------------------------------------------------------

import conceptnet
import string
import re
import random
import util
```

```
import logger

class concept_types:
    IsA =  "/r/IsA"
    HasA = "/r/HasA"
    HasProperty = "/r/HasProperty"



def make_response(replacement_words, response):
        return response.replace('%1', replacement_words)

class eliza:
  def __init__(self):
        self.keys = map(lambda x: re.compile(x[0], re.IGNORECASE), gPats)
        self.values = map(lambda x: x[2],gPats)
        self.responses = map(lambda x: (re.compile(x[0], re.IGNORECASE), x[1]), gPats)

  def generate_conceptnet_response(self, words, responses):
        word = conceptnet.get_important_word(words)
        results, response = util.get_first(res for res in ((conceptnet.get_results(word, r[0]), r[1])
for r in responses) if len(res[0]) > 0) or (None, None)

        if results:
        return make_response(util.get_random(results, 10), util.get_random(response))

        return None

  #----------------------------------------------------------------------
  # translate: take a string, replace any words found in dict.keys()
  #  with the corresponding dict.values()
  #----------------------------------------------------------------------
  def translate(self,str,dict):
        words = string.split(string.lower(str))
        keys = dict.keys();
        for i in range(0,len(words)):
        if words[i] in keys:
        words[i] = dict[words[i]]
        return string.join(words)

  #----------------------------------------------------------------------
  #  respond: take a string, a set of regexps, and a corresponding
  #       set of response lists; find a match, and return a randomly
```

```
  #       chosen response from the corresponding list.
  #------------------------------------------------------------------
  def respond(self,str):
          # find a match among keys
          for i in range(0,len(self.keys)):
          match = self.keys[i].match(str)
          if match:
          # found a match ... stuff with corresponding value
          # chosen randomly from among the available options
          resp = random.choice(self.values[i])
          # we've got a response... stuff in reflected text where indicated
          pos = string.find(resp,'%')
          while pos > -1:
          num = string.atoi(resp[pos+1:pos+2])
          resp = resp[:pos] + \
          self.translate(match.group(num),gReflections) + \
          resp[pos+2:]
          pos = string.find(resp,'%')
          # fix munged punctuation at the end
          if resp[-2:] == '?.': resp = resp[:-2] + '.'
          if resp[-2:] == '??': resp = resp[:-2] + '?'
          return resp

  def respond_conceptnet(self, str):
          match, conceptnet_responses = util.get_first(((r[0].match(str), random.sample(r[1],
len(r[1]))) for r in self.responses if r[0].match(str))) or (None, None)

          if match:
          return self.generate_conceptnet_response(match.group(1), conceptnet_responses) or
self.respond(str)

          return None

#------------------------------------------------------------------
# gReflections, a translation table used to convert things you say
#       into things the computer says back, e.g. "I am" --> "you are"
#------------------------------------------------------------------
gReflections = {
  "am"   : "are",
  "was"  : "were",
  "i"    : "you",
  "i'd"  : "you would",
  "i've"  : "you have",
```

```
  "i'll"  : "you will",
  "my"  : "your",
  "are"  : "am",
  "you've": "I have",
  "you'll": "I will",
  "your"  : "my",
  "yours"  : "mine",
  "you"  : "me",
  "me"  : "you"
}

#-----------------------------------------------------------------------
# gPats, the main response table.  Each element of the list is a
#  two-element list; the first is a regexp, and the second is a
#  list of possible responses, with group-macros labelled as
#  %1, %2, etc.
#-----------------------------------------------------------------------

standard = [
          (concept_types.HasA, ["Is that because it has %1?"]),
          (concept_types.IsA, ["Do you have need of other %1?"]),
     ]

wdy = [
    (concept_types.HasProperty, ["Why do you want me to %1 that?"]),
    (concept_types.IsA, ["Do you think a %1 would help you?"]),
    ]

wci = [
    (concept_types.HasProperty, ["Have you tried other ways of %1?"]),
    (concept_types.IsA, ["Is there another %1 you can't do?"]),
    ]

ic = [
    (concept_types.HasProperty, ["Have you tried other ways of %1?"]),
    (concept_types.IsA, ["Is there another %1 you can't do?"]),
    ]
ia = [
    (concept_types.HasProperty, ["Are you %1?"]),
    (concept_types.IsA, ["Would you also describe yourself as %1?"]),
    ]
im = [
    (concept_types.HasProperty, ["So you're %1?"]),
```

```
    (concept_types.IsA, ["Do you %1?"]),
    ]
ay = [
    (concept_types.HasProperty, ["Is it because I %1?"]),
    (concept_types.IsA, ["Would you say I %1?"]),
    ]
wt = [
    (concept_types.HasProperty, ["Do you ask because %1?"]),
    (concept_types.IsA, ["Why do you ask about %1?"]),
    ]
hw = [
    (concept_types.HasProperty, ["Do you want to %1?"]),
    (concept_types.IsA, ["Do you think you could %1?"]),
    ]
bc = [
    (concept_types.HasProperty, ["Have you tried %1?"]),
    (concept_types.IsA, ["Are you sure? What about %1?"]),
    ]
gPats = [
  [r'I need (.*)',
          standard,
          [
          "Why do you need %1?",
          "Would it really help you to get %1?",
          "Are you sure you need %1?"
          ]
  ],

  [r'Why don\'?t you ([^\?]*)\??',
  wdy,
  [  "Do you really think I don't %1?",
          "Perhaps eventually I will %1.",
          "Do you really want me to %1?"]],

  [r'Why can\'?t I ([^\?]*)\??',
   wci,
  [  "Do you think you should be able to %1?",
          "If you could %1, what would you do?",
          "I don't know -- why can't you %1?",
          "Have you really tried?"]],

  [r'I can\'?t (.*)',
    ic,
```

```
[  "How do you know you can't %1?",
        "Perhaps you could %1 if you tried.",
        "What would it take for you to %1?"]],

[r'I am (.*)',
 ia,
[  "Did you come to me because you are %1?",
        "How long have you been %1?",
        "How do you feel about being %1?"]],

[r'I\'?m (.*)',
 im,
[  "How does being %1 make you feel?",
        "Do you enjoy being %1?",
        "Why do you tell me you're %1?",
        "Why do you think you're %1?"]],

[r'Are you ([^\?]*)\??',
 ay,
[  "Why does it matter whether I am %1?",
        "Would you prefer it if I were not %1?",
        "Perhaps you believe I am %1.",
        "I may be %1 -- what do you think?"]],

[r'What (.*)',
 wt,
[  "Why do you ask?",
        "How would an answer to that help you?",
        "What do you think?"]],

[r'How (.*)',
 hw,
[  "How do you suppose?",
        "Perhaps you can answer your own question.",
        "What is it you're really asking?"]],

[r'Because (.*)',
 bc,
[  "Is that the real reason?",
        "What other reasons come to mind?",
        "Does that reason apply to anything else?",
        "If %1, what else must be true?"]],
```

```
[r'(.*) sorry (.*)',
 standard,
[  "There are many times when no apology is needed.",
        "What feelings do you have when you apologize?"]],

[r'Hello(.*)',
 standard,
[  "Hello... I'm glad you could drop by today.",
        "Hi there... how are you today?",
        "Hello, how are you feeling today?"]],

[r'I think (.*)',
 standard,
[  "Do you doubt %1?",
        "Do you really think so?",
        "But you're not sure %1?"]],

[r'(.*) friend (.*)',
standard,
[  "Tell me more about your friends.",
        "When you think of a friend, what comes to mind?",
        "Why don't you tell me about a childhood friend?"]],

[r'Yes',
standard,
[  "You seem quite sure.",
        "OK, but can you elaborate a bit?"]],

[r'(.*) computer(.*)',
standard,
[  "Are you really talking about me?",
        "Does it seem strange to talk to a computer?",
        "How do computers make you feel?",
        "Do you feel threatened by computers?"]],

[r'Is it (.*)',
standard,
[  "Do you think it is %1?",
        "Perhaps it's %1 -- what do you think?",
        "If it were %1, what would you do?",
        "It could well be that %1."]],

[r'It is (.*)',
```

```
standard,
[  "You seem very certain.",
        "If I told you that it probably isn't %1, what would you feel?"]],

[r'Can you ([^\?]*)\??',
standard,
[  "What makes you think I can't %1?",
        "If I could %1, then what?",
        "Why do you ask if I can %1?"]],

[r'Can I ([^\?]*)\??',
standard,
[  "Perhaps you don't want to %1.",
        "Do you want to be able to %1?",
        "If you could %1, would you?"]],

[r'You are (.*)',
standard,
[  "Why do you think I am %1?",
        "Does it please you to think that I'm %1?",
        "Perhaps you would like me to be %1.",
        "Perhaps you're really talking about yourself?"]],

[r'You\'?re (.*)',
standard,
[  "Why do you say I am %1?",
        "Why do you think I am %1?",
        "Are we talking about you, or me?"]],

[r'I don\'?t (.*)',
standard,
[  "Don't you really %1?",
        "Why don't you %1?",
        "Do you want to %1?"]],

[r'I feel (.*)',
standard,
[  "Good, tell me more about these feelings.",
        "Do you often feel %1?",
        "When do you usually feel %1?",
        "When you feel %1, what do you do?"]],

[r'I have (.*)',
```

```
standard,
[  "Why do you tell me that you've %1?",
        "Have you really %1?",
        "Now that you have %1, what will you do next?"]],

[r'I would (.*)',
standard,
[  "Could you explain why you would %1?",
        "Why would you %1?",
        "Who else knows that you would %1?"]],

[r'Is there (.*)',
standard,
[  "Do you think there is %1?",
        "It's likely that there is %1.",
        "Would you like there to be %1?"]],

[r'My (.*)',
standard,
[  "I see, your %1.",
        "Why do you say that your %1?",
        "When your %1, how do you feel?"]],

[r'You (.*)',
standard,
[  "We should be discussing you, not me.",
        "Why do you say that about me?",
        "Why do you care whether I %1?"]],

[r'Why (.*)',
standard,
[  "Why don't you tell me the reason why %1?",
        "Why do you think %1?" ]],

[r'I want (.*)',
standard,
[  "What would it mean to you if you got %1?",
        "Why do you want %1?",
        "What would you do if you got %1?",
        "If you got %1, then what would you do?"]],

[r'(.*) mother(.*)',
standard,
```

```
[  "Tell me more about your mother.",
        "What was your relationship with your mother like?",
        "How do you feel about your mother?",
        "How does this relate to your feelings today?",
        "Good family relations are important."]],

[r'(.*) father(.*)',
standard,
[  "Tell me more about your father.",
        "How did your father make you feel?",
        "How do you feel about your father?",
        "Does your relationship with your father relate to your feelings today?",
        "Do you have trouble showing affection with your family?"]],

[r'(.*) child(.*)',
standard,
[  "Did you have close friends as a child?",
        "What is your favorite childhood memory?",
        "Do you remember any dreams or nightmares from childhood?",
        "Did the other children sometimes tease you?",
        "How do you think your childhood experiences relate to your feelings today?"]],

[r'(.*)\?',
standard,
[  "Why do you ask that?",
        "Please consider whether you can answer your own question.",
        "Perhaps the answer lies within yourself?",
        "Why don't you tell me?"]],

[r'quit',
standard,
[  "Thank you for talking with me.",
        "Good-bye.",
        "Thank you, that will be $150.  Have a good day!"]],

[r'(.*)',
standard,
[  "Please tell me more.",
        "Let's change focus a bit... Tell me about your family.",
        "Can you elaborate on that?",
        "Why do you say that %1?",
        "I see.",
        "Very interesting.",
```

```
            "%1.",
            "I see.  And what does that tell you?",
            "How does that make you feel?",
            "How do you feel when you say that?"]]
    ]

#---------------------------------------------------------------------
#  command_interface
#---------------------------------------------------------------------
def command_interface():
    print "Therapist\n---------"
    print "Talk to the program by typing in plain English, using normal upper-"
    print 'and lower-case letters and punctuation.  Enter "quit" when done.'
    print '='*72
    print "Hello.  How are you feeling today?"
    s = ""

    log = logger.logger("conversation")

    therapist = eliza();
    while s != "quit":
            try: s = raw_input(">")
            except EOFError:
            s = "quit"
            print log.log(s)
            while s[-1] in "!.": s = s[:-1]
            log.log(">>> " + s)
            print log.log(therapist.respond_conceptnet(s))


if __name__ == "__main__":
    command_interface()
```

conceptnet.py

```
import json, urllib, util
#/s/contributor/globalmind
SEARCH_BASE =
'http://conceptnet5.media.mit.edu/data/5.1/search?sources=/s/contributor/omcs'

def get_results(word, type):
    url = SEARCH_BASE + "&limit=100&start=/c/en/" + word
```

```
    result = json.load(urllib.urlopen(url))
    returnlist = []
    edges = json.dumps(result['edges'])
    edges2 = json.loads(edges)
    for s in edges2:
            if type in s['nodes']:
                    tmp  = json.dumps(s['surfaceText'])
                    subtmp = tmp[tmp.find("[[",tmp.find("]]")+2)+2:tmp.find("]]",tmp.find("]]")+2)]
                    potres = strip_useless_words(subtmp.lower())
                    if count_words(potres)<=2:
                            returnlist.append(potres)
    url = SEARCH_BASE + "&limit=100&end=/c/en/" + word
    result = json.load(urllib.urlopen(url))
    edges = json.dumps(result['edges'])
    edges2 = json.loads(edges)
    for s in edges2:
            if type in s['nodes']:
                    tmp  = json.dumps(s['surfaceText'])
                    subtmp = tmp[tmp.find("[[")+2:tmp.find("]]")]
                    potres = strip_useless_words(subtmp.lower())
                    if count_words(potres)<=2:
                            returnlist.append(potres)
    return filter_duplicates(returnlist)

def filter_duplicates(seq):
            seen = set()
            seen_add = seen.add
            return [ x for x in seq if x not in seen and not seen_add(x)]

useless_words = [
                    "an",
                    "a",
                    "have",
                    "is",
                    "like",
                    ]

def get_important_word(words):
            return util.get_first(word for word in words.split(' ') if not word in useless_words)

def strip_useless_words(text):
    return " ".join([x for x in text.split(" ") if x not in useless_words])
```

```
def count_words(text):
    return len(text.split(" "))
```

util.py

```
import random

def get_first(iterable):
        for i, x in enumerate(iterable):
        return x
        return None

def get_random(input_list, limit = None):
        limit = limit or len(input_list) - 1
        return input_list[random.randint(0, min(len(input_list) - 1, limit))]
```

Version 2

eliza-conceptnet.py

```
#-----------------------------------------------------------------------
#  Based on eliza.py
#
#  a cheezy little Eliza knock-off by Joe Strout <joe@strout.net>
#  with some updates by Jeff Epler <jepler@inetnebr.com>
#  hacked into a module and updated by Jez Higgins <jez@jezuk.co.uk>
#  last revised: 28 February 2005
#-----------------------------------------------------------------------

import conceptnet
import string
import re
import random
import util
import logger
import nlp

class concept_types:
    IsA =  "/r/IsA"
    HasA = "/r/HasA"
    HasProperty = "/r/HasProperty"
```

```
class forms:
        GrammaticalNumber = 0xFF
        Singular = 0
        Plural = 1

        Article = 0xFF00
        Definite = 0x0000
        Indefinite = 0x0100
        Partive = 0x0200
        Negative = 0x0300
        ZeroArticle = 0x0400

class replacement_action:
        KeepCurrent = 1
        MakePlural = 2
        MakeSingular = 3

class search_type:
   StartToEnd = ["start"]
   EndToStard = ["end"]
   Both = ["start", "end"]

def make_response(replacement_words, response, rep_action):
        if rep_action == replacement_action.MakeSingular:
        replacement_words = nlp.singularize_sentence(replacement_words)
        if rep_action == replacement_action.MakePlural:
         replacement_words = nlp.pluralize_sentence(replacement_words)
        return response.replace('%1', replacement_words)

class eliza:
  def __init__(self):
        self.keys = map(lambda x: re.compile(x[0], re.IGNORECASE), gPats)
        self.values = map(lambda x: x[2],gPats)
        self.responses = map(lambda x: (re.compile(x[0], re.IGNORECASE), x[1]), gPats)

  def generate_conceptnet_response(self, words, responses):
        word = nlp.lemmatize(nlp.get_important_word(words))

        util.debug("this is the word it finds interesting: " + word)

        results, response = util.get_first(res for res in ((conceptnet.get_results(word, r[0],
nlp.get_wordtype(word), r[2]), r[1]) for r in responses) if len(res[0]) > 0) or (None, None)
```

```
        if results:
        return make_response(util.get_random(results, 10), *util.get_random(response))

        return None

#---------------------------------------------------------------------
# translate: take a string, replace any words found in dict.keys()
#  with the corresponding dict.values()
#---------------------------------------------------------------------
def translate(self,str,dict):
        words = string.split(string.lower(str))
        keys = dict.keys();
        for i in range(0,len(words)):
        if words[i] in keys:
        words[i] = dict[words[i]]
        return string.join(words)

#---------------------------------------------------------------------
#  respond: take a string, a set of regexps, and a corresponding
#       set of response lists; find a match, and return a randomly
#       chosen response from the corresponding list.
#---------------------------------------------------------------------
def respond(self,str):
        # find a match among keys
        for i in range(0,len(self.keys)):
        match = self.keys[i].match(str)
        if match:
        # found a match ... stuff with corresponding value
        # chosen randomly from among the available options
        resp = random.choice(self.values[i])
        # we've got a response... stuff in reflected text where indicated
        pos = string.find(resp,'%')
        while pos > -1:
        num = string.atoi(resp[pos+1:pos+2])
        resp = resp[:pos] + \
        self.translate(match.group(num),gReflections) + \
        resp[pos+2:]
        pos = string.find(resp,'%')
        # fix munged punctuation at the end
        if resp[-2:] == '?.': resp = resp[:-2] + '.'
        if resp[-2:] == '??': resp = resp[:-2] + '?'
        return resp
```

```
  def respond_conceptnet(self, str):
        match, conceptnet_responses = util.get_first(((r[0].match(str), random.sample(r[1],
len(r[1]))) for r in self.responses if r[0].match(str))) or (None, None)

        if not match:
        util.debug("No match for sentence: " + str)
        return

        if len(match.groups()) == 1:
        return self.generate_conceptnet_response(match.group(1), conceptnet_responses) or
self.respond(str)
        else: return self.respond(str)

#-----------------------------------------------------------------------
# gReflections, a translation table used to convert things you say
#         into things the computer says back, e.g. "I am" --> "you are"
#-----------------------------------------------------------------------
gReflections = {
  "am"   : "are",
  "was"  : "were",
  "i"      : "you",
  "i'd"  : "you would",
  "i've"  : "you have",
  "i'll"  : "you will",
  "my"  : "your",
  "are"  : "am",
  "you've": "I have",
  "you'll": "I will",
  "your"  : "my",
  "yours"  : "mine",
  "you"  : "me",
  "me"  : "you"
}

#-----------------------------------------------------------------------
# gPats, the main response table.  Each element of the list is a
#  two-element list; the first is a regexp, and the second is a
#  list of possible responses, with group-macros labelled as
#  %1, %2, etc.
#-----------------------------------------------------------------------

standard = [
        (concept_types.HasA, [("Is that because of its %1?", replacement_action.KeepCurrent)],
```

```
search_type.Both),
        (concept_types.IsA, [("What about other %1?", replacement_action.MakePlural)],
search_type.StartToEnd),
        (concept_types.IsA,  [("Does that statement hold for other %1?",
replacement_action.MakePlural)], search_type.StartToEnd)
    ]

need = [
        (concept_types.HasA, [("Is that because of its %1?", replacement_action.KeepCurrent)],
search_type.Both),
        (concept_types.IsA, [("Do you have need of other %1?",
replacement_action.MakePlural)], search_type.StartToEnd),
        (concept_types.IsA,  [("Would you like other %1?", replacement_action.MakePlural)],
search_type.StartToEnd)
    ]

wdy = [
  (concept_types.HasProperty, [("Why do you want me to %1 that?",
replacement_action.KeepCurrent)], search_type.Both),
  (concept_types.IsA, [("Do you think a %1 would help you?",
replacement_action.KeepCurrent)], search_type.Both),
  ]

wci = [
  (concept_types.HasProperty, [("Have you tried other ways of %1?",
replacement_action.KeepCurrent)], search_type.Both),
  (concept_types.IsA, [("Is there another %1 you can't do?", replacement_action.KeepCurrent)],
search_type.Both),
  ]

ic = [
  (concept_types.HasProperty, [("Have you tried other ways of %1?",
replacement_action.KeepCurrent)], search_type.Both),
  (concept_types.IsA, [("Is there another %1 you can't do?", replacement_action.KeepCurrent)],
search_type.Both),
  ]
#ia = [
#   (concept_types.HasProperty, [("Are you %1?", replacement_action.KeepCurrent)],
search_type.Both),
#   (concept_types.IsA, [("Would you also describe yourself as %1?",
replacement_action.KeepCurrent)], search_type.Both),
#   ]
#im = [
```

```
#    (concept_types.HasProperty, [("So you're %1?", replacement_action.KeepCurrent)],
search_type.Both),
#    (concept_types.IsA, [("Do you %1?", replacement_action.KeepCurrent)], search_type.Both),
#    ]
ay = [
   (concept_types.HasProperty, [("Is it because I %1?", replacement_action.KeepCurrent)],
search_type.Both),
   (concept_types.IsA, [("Would you say I %1?", replacement_action.KeepCurrent)],
search_type.Both),
   ]
wt = [
   (concept_types.HasProperty, [("Do you ask because %1?",
replacement_action.KeepCurrent)], search_type.Both),
   (concept_types.IsA, [("Why do you ask about %1?", replacement_action.KeepCurrent)],
search_type.Both),
   ]
hw = [
   (concept_types.HasProperty, [("Do you want to %1?", replacement_action.KeepCurrent)],
search_type.Both),
   (concept_types.IsA, [("Do you think you could %1?", replacement_action.KeepCurrent)],
search_type.Both),
   ]
bc = [
   (concept_types.HasProperty, [("Have you tried %1?", replacement_action.KeepCurrent)],
search_type.Both),
   (concept_types.IsA, [("Are you sure? What about %1?", replacement_action.KeepCurrent)],
search_type.Both),
   ]
gPats = [
  [r'I need (.*)',
          need,
          [
          "Why do you need %1?",
          "Would it really help you to get %1?",
          "Are you sure you need %1?"
          ]
  ],

  [r'Why don\'?t you ([^\?]*)\??',
  wdy,
  [  "Do you really think I don't %1?",
          "Perhaps eventually I will %1.",
          "Do you really want me to %1?"]],
```

```
[r'Why can\'?t I ([^\?]*)\??',
 wci,
[  "Do you think you should be able to %1?",
        "If you could %1, what would you do?",
        "I don't know -- why can't you %1?",
        "Have you really tried?"]],

[r'I can\'?t (.*)',
  ic,
[  "How do you know you can't %1?",
        "Perhaps you could %1 if you tried.",
        "What would it take for you to %1?"]],

[r'I am (.*)',
 standard,
[  "Did you come to me because you are %1?",
        "How long have you been %1?",
        "How do you feel about being %1?"]],

[r'I\'?m (.*)',
 standard,
[  "How does being %1 make you feel?",
        "Do you enjoy being %1?",
        "Why do you tell me you're %1?",
        "Why do you think you're %1?"]],

[r'Are you ([^\?]*)\??',
 ay,
[  "Why does it matter whether I am %1?",
        "Would you prefer it if I were not %1?",
        "Perhaps you believe I am %1.",
        "I may be %1 -- what do you think?"]],

[r'What (.*)',
 wt,
[  "Why do you ask?",
        "How would an answer to that help you?",
        "What do you think?"]],

[r'How (.*)',
 hw,
[  "How do you suppose?",
```

```
        "Perhaps you can answer your own question.",
        "What is it you're really asking?"]],

[r'Because (.*)',
 bc,
[  "Is that the real reason?",
        "What other reasons come to mind?",
        "Does that reason apply to anything else?",
        "If %1, what else must be true?"]],

[r'(.*) sorry (.*)',
 [ ],
[  "There are many times when no apology is needed.",
        "What feelings do you have when you apologize?"]],

[r'Hello(.*)',
 standard,
[  "Hello... I'm glad you could drop by today.",
        "Hi there... how are you today?",
        "Hello, how are you feeling today?"]],

[r'I think (.*)',
 standard,
[  "Do you doubt %1?",
        "Do you really think so?",
        "But you're not sure %1?"]],

[r'(.*) friend (.*)',
standard,
[  "Tell me more about your friends.",
        "When you think of a friend, what comes to mind?",
        "Why don't you tell me about a childhood friend?"]],

[r'Yes',
standard,
[  "You seem quite sure.",
        "OK, but can you elaborate a bit?"]],

[r'(.*) computer(.*)',
standard,
[  "Are you really talking about me?",
        "Does it seem strange to talk to a computer?",
        "How do computers make you feel?",
```

```
        "Do you feel threatened by computers?"]],

[r'Is it (.*)',
standard,
[  "Do you think it is %1?",
        "Perhaps it's %1 -- what do you think?",
        "If it were %1, what would you do?",
        "It could well be that %1."]],

[r'It is (.*)',
standard,
[  "You seem very certain.",
        "If I told you that it probably isn't %1, what would you feel?"]],

[r'Can you ([^\?]*)\??',
standard,
[  "What makes you think I can't %1?",
        "If I could %1, then what?",
        "Why do you ask if I can %1?"]],

[r'Can I ([^\?]*)\??',
standard,
[  "Perhaps you don't want to %1.",
        "Do you want to be able to %1?",
        "If you could %1, would you?"]],

[r'You are (.*)',
standard,
[  "Why do you think I am %1?",
        "Does it please you to think that I'm %1?",
        "Perhaps you would like me to be %1.",
        "Perhaps you're really talking about yourself?"]],

[r'You\'?re (.*)',
standard,
[  "Why do you say I am %1?",
        "Why do you think I am %1?",
        "Are we talking about you, or me?"]],

[r'I don\'?t (.*)',
standard,
[  "Don't you really %1?",
        "Why don't you %1?",
```

```
        "Do you want to %1?"]],

[r'I feel (.*)',
standard,
[  "Good, tell me more about these feelings.",
        "Do you often feel %1?",
        "When do you usually feel %1?",
        "When you feel %1, what do you do?"]],

[r'I have (.*)',
standard,
[  "Why do you tell me that you've %1?",
        "Have you really %1?",
        "Now that you have %1, what will you do next?"]],

[r'I would (.*)',
standard,
[  "Could you explain why you would %1?",
        "Why would you %1?",
        "Who else knows that you would %1?"]],

[r'Is there (.*)',
standard,
[  "Do you think there is %1?",
        "It's likely that there is %1.",
        "Would you like there to be %1?"]],

[r'My (.*)',
standard,
[  "I see, your %1.",
        "Why do you say that your %1?",
        "When your %1, how do you feel?"]],

[r'You (.*)',
standard,
[  "We should be discussing you, not me.",
        "Why do you say that about me?",
        "Why do you care whether I %1?"]],

[r'Why (.*)',
standard,
[  "Why don't you tell me the reason why %1?",
        "Why do you think %1?" ]],
```

```
[r'I want (.*)',
standard,
[  "What would it mean to you if you got %1?",
        "Why do you want %1?",
        "What would you do if you got %1?",
        "If you got %1, then what would you do?"]],

[r'(.*) mother(.*)',
standard,
[  "Tell me more about your mother.",
        "What was your relationship with your mother like?",
        "How do you feel about your mother?",
        "How does this relate to your feelings today?",
        "Good family relations are important."]],

[r'(.*) father(.*)',
standard,
[  "Tell me more about your father.",
        "How did your father make you feel?",
        "How do you feel about your father?",
        "Does your relationship with your father relate to your feelings today?",
        "Do you have trouble showing affection with your family?"]],

[r'(.*) child(.*)',
standard,
[  "Did you have close friends as a child?",
        "What is your favorite childhood memory?",
        "Do you remember any dreams or nightmares from childhood?",
        "Did the other children sometimes tease you?",
        "How do you think your childhood experiences relate to your feelings today?"]],

[r'(.*)\?',
standard,
[  "Why do you ask that?",
        "Please consider whether you can answer your own question.",
        "Perhaps the answer lies within yourself?",
        "Why don't you tell me?"]],

[r'quit',
standard,
[  "Thank you for talking with me.",
        "Good-bye.",
```

```
        "Thank you, that will be $150.  Have a good day!"]],

  [r'(.*)',
  standard,
  [  "Please tell me more.",
        "Let's change focus a bit... Tell me about your family.",
        "Can you elaborate on that?",
        "Why do you say that %1?",
        "I see.",
        "Very interesting.",
        "%1.",
        "I see.  And what does that tell you?",
        "How does that make you feel?",
        "How do you feel when you say that?"]]
  ]

#---------------------------------------------------------------------
#  command_interface
#---------------------------------------------------------------------
def command_interface():
  print "Therapist\n---------"
  print "Talk to the program by typing in plain English, using normal upper-"
  print 'and lower-case letters and punctuation.  Enter "quit" when done.'
  print '='*72
  print "Hello.  What is on your mind today?"
  s = ""
  log = logger.logger("conversation")

  therapist = eliza();
  while s != "quit":
        try: s = raw_input(">")
        except EOFError:
        s = "quit"
        print log.log(s)
        while s[-1] in "!.": s = s[:-1]
        log.log(">>> " + s)
        print log.log(therapist.respond_conceptnet(s))


if __name__ == "__main__":
  command_interface()
```

conceptnet.py

```
import json, urllib, util, nlp
#/s/contributor/globalmind omcs
SEARCH_BASE =
'http://conceptnet5.media.mit.edu/data/5.1/search?sources=/s/contributor/omcs'

def get_results(word, type, wordtype = 'NN', searches = ['start', 'end']):
   returnlist = []
   for term in searches:
         url = SEARCH_BASE + "&limit=100&"+term+"=/c/en/" + word
         result = json.load(urllib.urlopen(url))
         edges = json.loads(json.dumps(result['edges']))
         util.debug("these are all the things it found on conceptnet:")
         for s in edges:
               if type in s['nodes'] and "/c/en/" + word in s['nodes']:
                     tmp  = json.dumps(s['surfaceText'])
                     potres =
util.strip_useless_words(tmp[tmp.find("[[",tmp.find("]]")+2)+2:tmp.find("]]",tmp.find("]]")+2)].lower(
))
                     util.debug(potres)
                     if util.count_words(potres)<=2:
                           if nlp.is_wordtype(potres.split(' ')[len(potres.split(' '))-1], wordtype):
                                 if not(word in potres) :
                                       returnlist.append(potres)
   util.debug("this is what it returned to the bot:")
   util.debug(util.filter_duplicates(returnlist))
   return util.filter_duplicates(returnlist)
```

nlp.py

```
import nltk
import util

class word_class:
         Noun = 'NN'
         Verb = 'V'
         Adjective = 'JJ'

def plural(word):
         if word.endswith('y'):
```

```
        return word+'s' if is_vowel(word[-2]) else word[:-1] + 'ies'
        elif word[-1] in 'sx' or word[-2:] in ['sh', 'ch']:
        return word + 'es'
        elif word.endswith('an'):
        return word[:-2] + 'en'
        else:
        return word + 's'

def get_wordtype(word):
        [(word, wordtype)] = nltk.pos_tag(nltk.word_tokenize(word))
        return wordtype

def get_important_word(sentence):
        if get_last_of_wordtype(sentence, word_class.Noun) != None:
        util.debug("there's a noun!:"+get_last_of_wordtype(sentence, word_class.Noun))
        return get_last_of_wordtype(sentence, word_class.Noun)
        elif get_last_of_wordtype(sentence, word_class.Adjective) != None:
        util.debug("there's an adjective!:"+get_last_of_wordtype(sentence,
word_class.Adjective))
        return get_last_of_wordtype(sentence, word_class.Adjective)
        elif get_last_of_wordtype(sentence, word_class.Verb) != None:
        util.debug("there's a verb!:" + get_last_of_wordtype(sentence, word_class.Verb))
        return get_last_of_wordtype(sentence, word_class.Verb)
        else:
        util.debug("there's something else!")
        return util.get_first(word for word in sentence.split(' ') if not word in util.useless_words)


def get_last_of_wordtype(sentence, wordtype):
        tagged_words = nltk.pos_tag(nltk.word_tokenize(sentence))
        util.debug(tagged_words)
        if util.get_last(tagged_words, lambda val: val[1].startswith(wordtype)) != None:
        return util.get_last(tagged_words, lambda val: val[1].startswith(wordtype))[0]
        return None

def is_wordtype(word, des_wordtype):
        [(word, wordtype)] = nltk.pos_tag(nltk.word_tokenize(word))
        return True if wordtype.startswith(des_wordtype) else False

def get_pos(sentence, pos):
        tagged_words = nltk.pos_tag(nltk.word_tokenize(sentence))
        return util.get_last(tagged_words, lambda val: val[1] == pos)
```

```
def lemmatize(noun):
	return nltk.WordNetLemmatizer().lemmatize(noun)

def add_article(noun):
	if is_vowel(noun[0]):
	return 'an ' + noun
	return 'a ' + noun

def is_vowel(char):
	return char in 'aeiuyo'

def singularize_sentence(sentence):
	util.debug("it singularizes")
	plural_noun = get_pos(sentence, 'NNS')

	if(plural_noun == None):
	return sentence

	return sentence.replace(plural_noun[0], add_article(lemmatize(plural_noun[0])))

def pluralize_sentence(sentence):
	util.debug("it pluralizes")
	singular_noun = get_pos(sentence, 'NN')

	if(singular_noun == None):
	return sentence

	return sentence.replace(singular_noun[0], plural(singular_noun[0]))
```

util.py

```
import random

DEBUG = True

def debug(str):
	if DEBUG:
	print str

def get_first(iterable, func = lambda val: True):
	for x in iterable:
	if func(x):
```

```
        return x
        return None

def get_last(iterable, func = lambda val: True):
        for x in reversed(iterable):
        if func(x):
        return x
        return None

def get_random(input_list, limit = None):
        limit = limit or len(input_list) - 1
        return input_list[random.randint(0, min(len(input_list) - 1, limit))]

def filter_duplicates(seq):
        seen = set()
        seen_add = seen.add
        return [ x for x in seq if x not in seen and not seen_add(x)]

useless_words = [
                "an",
                "a",
                "have",
                "is",
                "like",
                ]

def strip_useless_words(text):
        return " ".join([x for x in text.split(" ") if x not in useless_words])

def count_words(text):
        return len(text.split(" "))
```