Assignment 1a: Functional Chatterbot

CS109L - Autumn 2014

Turn in Deadline: Thursday October 30th at 11:59 PM
Redo Deadline: Thursday December 4th at 11:59 PM

1 Overview

One of the most famous programs in the history of artificial intelligence is Joseph Weizenbaum's program ELIZA. Written in the mid sixties, it emulates (or parodies) a Rogerian psychotherapist who tries to make a patient talk about his or her deepest feelings.[1] ELIZA's fame is due in part to the remarkable observation that many people found her so friendly and understanding. In fact, some said she was the first therapist they had met who really knew how to listen. Today, in homage to Weizenbaum's program, the tendency to unconsciously assume computer behaviors are analogous to human behaviors is known as the ELIZA effect.

Over the years, ELIZA has had many successors. The type of program she represents is now known generally as a *chatterbot*. A chatterbot is a program that attempts to simulate typed conversation with the aim of (at least temporarily) fooling a human into thinking they are talking to another person.

For your first assignment, you will be creating a chatterbot written in `R`. The goal of the assignment is to have you practice clean, concise, and effective functional programming. The assignment will also give you exposure to some of `R`'s important data structures and functions and (for those of you who choose to use them) practice using `R`'s built-in debugging and unit testing frameworks.

The chatterbot you will be implementing is defined by a list of pattern-based rules. Each rule consists of two parts: a pattern and a list of templates. For example, the pattern **I need *** plus the templates **Why do you need *** and **Would it really help you to get *** and **Are you sure you need *** comprises a complete rule.

Input from the patient (the person communicating with the chatterbot) is matched sequentially against each of the chatterbot's rules. When the input successfully matches a rule's pattern, one of the rule's templates is given at random as a response. Words in the input that match a wildcard from the pattern are bound to wildcards in the template response. In the rule shown above (and in the assignment) we let asterisk characters delineate wildcards. So the phrase **I need octopus tentacles** when successfully matched against the pattern **I need *** from the rule above might return **Why do you need octopus tentacles**.

2 Pattern Matching

As you can see, the basic machinery of a chatterbot consists of pattern matching. Your first task for this assignment will be to write a function to determine if (and how) a target string matches a pattern and a function to substitute pattern wildcard matches into template responses.

[1] http://en.wikipedia.org/wiki/Person-centered_therapy

To generalize our algorithms, will be using generic R lists instead of strings. Since a string can be thought of as a list of characters or words, our generalized approach will be sufficient in the implementation of a chatterbot that only performs pattern matching on strings.

The function `PatternMatch(wildcard, pattern, target)` tries to match two lists: `pattern` and `target`. The list `pattern` may contain elements equal to wildcard, but the list `target` may not. We say `pattern` *equals* `target` if each and every element in `pattern` matches a set of corresponding elements from `target`, where elements from `pattern` and `target` are taken in sequence.[2] A non-wildcard element from `pattern` matches a single element in target if and only if the elements are identical. A wildcard element can match an arbitrarily long sub-list of `target`.[3] If the two lists are equal, the function returns the sub-list of `target` that matches the first occurrence of `wildcard` in `pattern`. If the lists are not equal, the result is `NULL`.

The function `PatternSubstitute(wildcard, template, sub)` replaces each occurrence of the element `wildcard` in the list `template` with the element `sub`.

Of the two functions, the first is the more difficult to implement, but it comprises the majority of the work done by a chatterbot. Feel free to implement the functions in whichever order you prefer, though I recommend you start with the latter. The only requirement is that each function contains at least one mapping function and absolutely no `for` loops. Your functions may also be recursive.

To get you started, here are some implementation hints:

- The function `identical(x, y, ...)` tells you if two R objects are exactly the same. Use it to test whether elements of `pattern` and `target` match or are wildcards.
- Use the function `is.null(x)` to test whether an object is `NULL`.
- For the pattern matching, there are three base cases, each of which should be fairly straightforward:
  1. both lists are empty
  2. `pattern` is empty but `target` is not
  3. `pattern` is not empty but `target` is

  For the case of two non-empty lists, the first element of `pattern` can either be `wildcard` or not. Each possibility should inform how to implement the rest of the function.

To aid in debugging, here is some sample output from my own solution:

```
> PatternMatch("x", list(2, "*", "x", "+", 3), list(2, "*", "7", "+", 3))
[[1]]
[1] "7"

> PatternMatch("*", list("frodo"), list("gandalf"))
NULL

> PatternMatch(2, list(1, 3:5), list(1, 3:5))
list()

> PatternMatch("*", list("*", "and", "*"), list("yoda", "and", "vader"))
[[1]]
[1] "yoda"
```

[2]So to pair elements of `pattern` and `target`, we pop an element from `pattern` and any number of elements from `target`.
[3]This is the same as saying that a wildcard can match any number of elements from `target`.

```
> PatternMatch("fish", list(1, "fish", 2, "fish"), list(1, -1, -1, 2))
[[1]]
[1] -1

[[2]]
[1] -1

> PatternSubstitute("*", list("to", "*", "or not to", "*"), "be")
[[1]]
[1] "to"

[[2]]
[1] "be"

[[3]]
[1] "or not to"

[[4]]
[1] "be"
```

Make sure to test these functions extensively, as they are the key to the chatterbot solution.

3 Pattern Transformation

A *pattern transformation* takes a target and a rule consisting of a pattern and templates and transforms the target into a list of responses. To *apply* a rule to a target means to see if the target is *equal* to the rule's pattern and then, if it is, to substitute the result of the pattern match into the rule's list of templates.

Recall that a chatterbot is defined internally as a list of rules, where each rule is itself a list with two elements. The first is a pattern, and the second is a list of templates. The file `init_chatterbot.R` is not only responsible for constructing the chatterbot we will use, but also provides utility functions for extracting the components of a chatterbot rule: `GetPattern(rule)` and `GetTemplates(rule)`.

Write the function `PatternTransform(target, rule, aux = NULL, wildcard = '*')` which applies `rule` to `target`. The parameter `aux` is a function that, if non-`NULL`, is applied to the result of the pattern match before the substitution is made. You will later see how this is useful. The function should return a list of transformations, or `NULL` if target does not match rule's pattern.

This function should be fairly simple to write and will leverage the functions you wrote in the previous part of the assignment. Once again, the only requirement for this part of the assignment is that you use a mapping function and no `for` loops.

4 Chatterbot Implementation

The main function of the chatterbot is an interactive loop that reads a line of input and responds by writing a line of output. You may start the chatterbot in an interactive `R` session by sourcing `chatterbot.R` and calling `RunChatterbot()`. By default, this requires the `init_chatterbot.R` file to be in the same directory as `chatterbot.R`. I encourage you to look at the function, although it is not strictly necessary to complete the assignment. The function reads input from the user and sequentially applies each of the chatterbot's rules to the input. When a successful match is made, a transformation is chosen at random and printed to the console.

We now have only one thing left to do to complete our chatterbot implementation. One of the keys to the success of ELIZA was a simple technique known as *reflection*. Reflecting a phrase means that you replace each occurrence of a first person word or expression with the corresponding second person word or expression and vice versa.

Write the function `Reflect(target)` which replaces each word in target with the corresponding word in the global map `reflections`, which has been instantiated for you in `init_chatterbot.R`.[4] For example:

```
> Reflect(list("I", "am", "never", "going", "to", "eat", "you"))
[[1]]
[1] "you"

[[2]]
[1] "are"

[[3]]
[1] "never"

[[4]]
[1] "going"

[[5]]
[1] "to"

[[6]]
[1] "eat"

[[7]]
[1] "me"
```

Notice now that the call to `PatternTransform()` in `RunChatterbot()` sets the `aux` parameter to `Reflect` so that the chatterbot's responses use all the correct pronouns.

5 Submission

1. Zip up `chatterbot.R` and `init_chatterbot.R` as well as any extra `.R` or `README` files into `chatterbot.zip`
2. Note: If you have done any extensions (see below), please submit your extended chatterbot implementation in separate files suffixed with `_ext` (so you may have `chatterbot_ext.R` and `init_chatterbot_ext.R`). If you have extended the chatterbot, please also include your non-extended version, and a text file describing what you did for the extension called `extension_description.txt`.
3. Email `chatterbot.zip` to cs109l.submissions@gmail.com (note that this is a separate submission email address) with the subject:

[4] A map in `R` is, you guessed it, a list with named elements. You can access the value associated with key *foo* either by `map$foo` or `map['foo']`. This is an $O(1)$ lookup!

- “Assignment 1a Turn In [Insert SUNET ID Here]”
- “Assignment 1a Redo [Insert SUNET ID Here]”

The body may be blank as long as the .zip file is attached. You may turn the assignment in as many times before the deadline and I will only grade the most recent assignment.
As an example, if I were to submit assignment 1a for the redo deadline, I would send my .zip file with the subject:
Assignment 1a Redo hgkshin.

6 Extension Ideas

There are lots of fancy things you can do to improve your chatterbot. The sky is the limit! I've included only a couple ideas for you below. The best and funniest chatterbots I come across during grading may be shown in class, with your permission. If you do a really stellar job you will get my undying admiration.

Chatterbot Rules: Play around with the chatterbot rules. Modify the existing rules or write a few of your own. Better yet, write an automated translation from normal responses to “themed” responses.[5]

Reductions: As a more interesting extension, consider the problem that sometimes chatterbot mimicking is too rigid. This is an issue as it can destroy the illusion of a listening person. If, for example, you say **I am very very tired** you may get the response **How long have you been very very tired** instead of **How long have you been tired**. A general approach that adds a lot of flexibility is to reduce the input by removing some syntactic variants of questions that are essentially the same. Here are some suggested reductions:

1. **please *** reduced to *****
2. **can you *** reduced to *****
3. **could you *** reduced to *****
4. **tell me if you are *** reduced to **are you ***
5. **tell me who * is** reduced to **who is ***
6. **tell me what * is** reduced to **what is ***
7. **do you know who * is** reduced to **who is ***
8. **do you know what * is** reduced to **what is ***
9. **are you very *** reduced to **are you ***
10. **I am very *** reduced to **I am ***
11. **hi *** reduced to **hello**

[5]Some themes from years past were *Lolcat chatterbot* and *cruel chatterbot*.