## AnswerBot

Less compressed and better explanations + demos can be found here: makurell.github.io/projects/answerbot/.

AnswerBot is, put simply, a search engine. You input a question, it parses it using the NLP techniques of Part-Of-Speech tagging and dependency parsing and then searches through Wikipedia, making use of semantic similarity calculations, to gather information to answer the input. The NLP functionalities just mentioned are provided by the Python library, SpaCy.

## Question Parsing

What needs to be extracted from the input is terms, categorised into being either 'detail' or 'keywords'.

When does a 'detail' stop being just additional information and start being a 'keyword', though? It isn't as binary as you might first think - and resolving terms to be strictly either may result in a less complete representation of the question and so in inaccuracies. For this reason, I decided to take a fuzzy approach and arrange the terms into a linear hierarchal list (aka: a spectrum) of relative 'detail'-ness (dependency/importance) instead.

| **Natural Language** | **Abstract Hierarchy** |
|---|---|
| Obama's age | `["Obama", "age"]` |
| Obama's dad's age | `["Obama", "dad", "age"]` |
| the biggest animal ever seen in Europe | `["Europe", "animal", "biggest"]` |

Internally, this structure shall be represented by a list of terms, where 'keyword'-like terms emerge at the left and the 'detail'-like emerge at the right.

## Parsing

The *`question`* is split into *queries* and the queries are parsed into *terms*, which are arranged into the hierarchy as described. View code.

The parsing works using a function, in which an input Token has all its dependants (children) traversed through, and for each child, it is decided (by considering the dependency type + POS) whether to prepend or append the child to the list (relative to the input token) and whether to omit certain terms. The function is recursively called for each of the children as well - the termination point being: when the input token has no children.

The recursion is initiated with the root token of a certain query (which is a Span object).

Note on terminology: I call a group of *terms* a *group*. And a list of these *groups* is a *grouping* (but also may be called a *variation* since they can be thought of as being a variant of the hierarchal structure).

## Variations

Some terms being grouped together may give better results, and so every way of grouping the terms is considered: Everything is grouped together into a list, then that list is split at every possible position. The spaces in-between items can be thought of as *split positions*. Assuming 1 means to split and 0 to not, the ways of splitting, then, is the binary pattern up to $2^{n-1}-1$ where $n$ is the length of the list. View code. More info.

Each way of ordering the different groups of terms is considered too, using the standard-library *itertool*'s *permutation* function. View code.

## Candidate Searching

Initially, the groupings are all run through the Wikipedia search API to get some initial candidate pages. A relevancy metric is calculated for each, using their titles, and those under a threshold are discarded: $s(p_t, g_0)$ where $s(..,..)$ is SpaCy's semantic similarity function, and $p_t$ is the page's title. NB: $g_0$ refers to the 0-th group of the grouping (the first). After the elimination, the contents for each page are downloaded through the API.

## Page Ranking

The remaining candidates are ranked, this time with their full contents taken into consideration, with the metric (for a given page $p$): $\frac{\sum_i^l s(p_c,g_i)}{l} + s(p_t, g_0)$ – where $p_c$ is the page's content, and $l$ is the length of the grouping.

## Data Searching

Now we want to find sentences, given a list of input sentences, that maximise relevancy to the groupings. The list of *groups* in every *grouping* is iterated over, and all the sentences in the given input sentences are ranked by their relevancy to the *group*. All but the top (certain number of) sentences are then discarded. The remaining sentences is the list of sentences that is operated on in the next iteration (the scope). In this fashion, by eliminating less relevant sentences for each group, the most relevant sentences for each *grouping* are selected.

The metric: for a given *group* $a$ and sentence $x$: $\sum_i^l s(x,\ a_i) + \frac{1}{2}\left(\sum_i^l \left(\frac{\sum_j^k s(a_i, p(x)_j)}{k}\right)\right)$ where $p(..)$ is the parsing algorithm (the same as the one used on input questions), that returns a list of keywords of length $k$.

## Analysis

A set of questions were run through the program and I noted and classified the program's behaviour and execution time. All data available here. E.g: failed on "What does goat mean" - gave info about goats not "greatest of all time".

The main issue of the program is only giving relevant information without answering the question. (See below).

| Class | Info |
|---|---|
| Top Page + Item | Is the first 'page' in the results and is in the preview items (top 3 items) |
| Top Item | Is not first 'page' in the results but is in the preview items (top 3 items) |
| Top Page | Not a top item but is the first page in the results |
| Buried | Neither first page nor top item but answer can be found within one of the pages |
| Relevant | Question not answered, but relevant information is given |
| Irrelevant | Question not answered and irrelevant information given |
| No Items | No search results (no items message shown) |
| Error | Exception occurs |

| Type | Info |
|---|---|
| Fact | Answerable with a specific value, theoretically should be same independent of source |
| Attribute | Searching for the attribute x of y (e.g age of y, population of y) |
| Search | General search as opposed to question |
| Info | Answerable with information, not necessarily expecting a 'value' response |
| Other | Invalid/Improperly formed/Out of scope (asking for the time, etc) |

- 61% of the questions were answered and 86% of the time, relevant information was given in some form.
- A large proportion of questions that were answered *Top Page + Item* were *fact* type questions.
- A high proportion of *attribute* questions only resulted in relevant information but no answer.
- Amount of *Top Page + Item* and *Relevant* results were equal => Just as likely that program gives only relevant info as it is to give an immediate answer, right at the top of the results.
- Of the times the question is answered, it is answered as *Top Page + Item* most of the time.
- *Fact* and *Info* type questions have the highest answer rates.
- The type of questions that are *unanswered* is quite evenly split – no one type of question has a greater tendency than the others to be unanswered.
- The peak amount of times, the program takes 0-10s, and most of the time, it takes 0-30s.

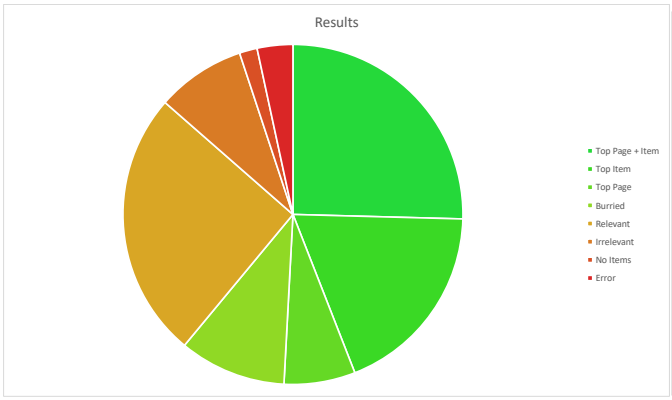
## Reflections

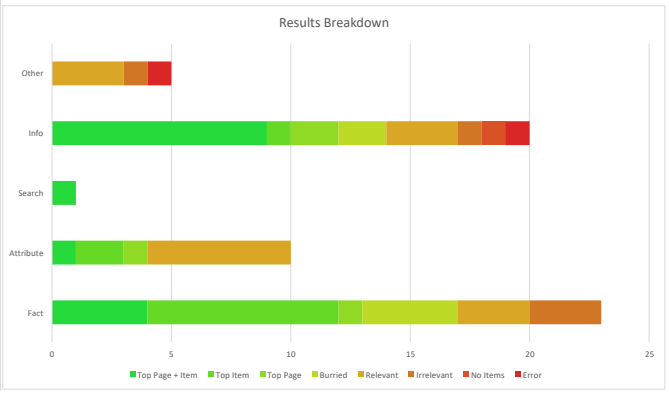
Because the key potential limiting factors of the program are: the NLP functions (handled by SpaCy), parsing and ordering of words, candidate scope (Wikipedia), candidate selection and elimination and sentence selection, the program could be improved by improving any one of these factors such as by expanding the scope, making a more complex parsing algorithm, refining the metrics used, etc.

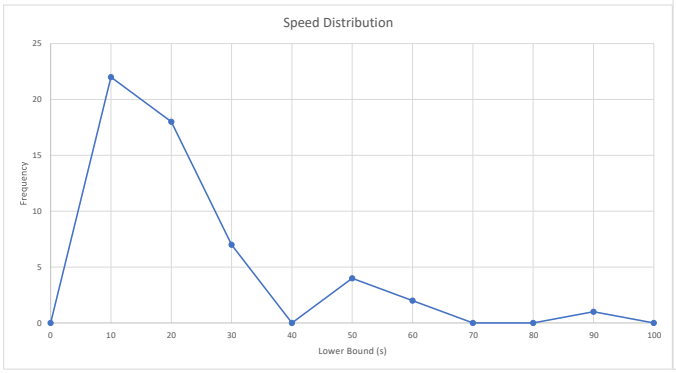
The program could also be improved by taking different approaches to the goal entirely:

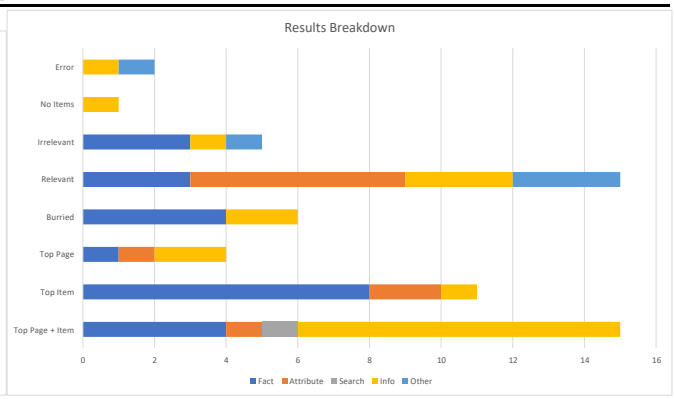
| Approach | Reason |
|---|---|
| Keywords parsing + ordering done directly by Machine Learning approaches. | 3rd party dependency for core functionality removed (so can have greater control of accuracy) + approach is less error prone and more direct. |
| Implement an extensive set of known question types, with set parsing and answering schemes for each. | Accuracy + reliability of interpretation + answering known question types would greatly increase. |
| Parse an expected answer type (number, date, day, etc) and weight answers accordingly. | Answer accuracy would be increased as less likely/inapplicable answers would be penalised more. |

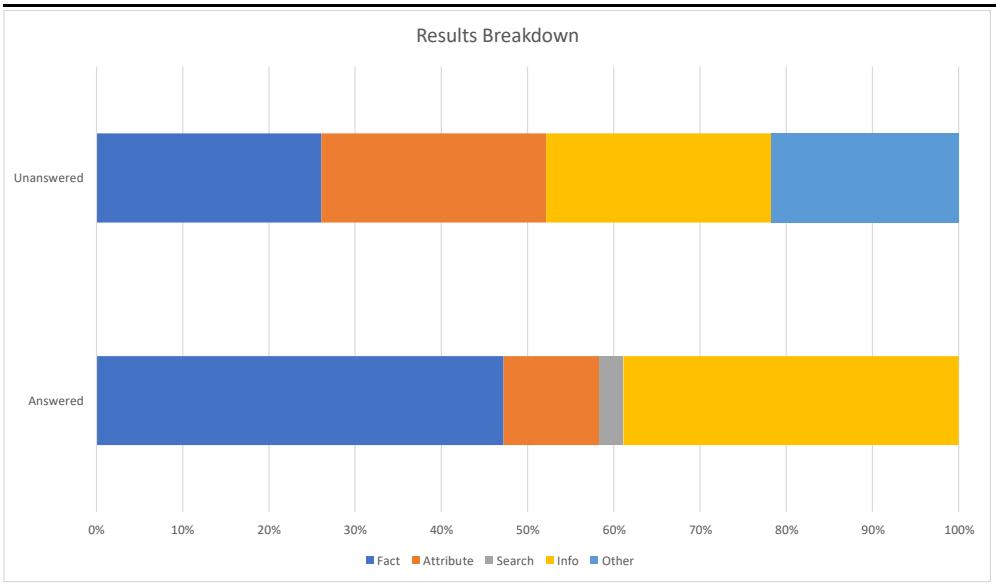
In terms of the speed of the program, which at the moment is pretty slow, it can be improved through indexing and caching because that would mean avoiding each document to be downloaded and parsed upon each query.

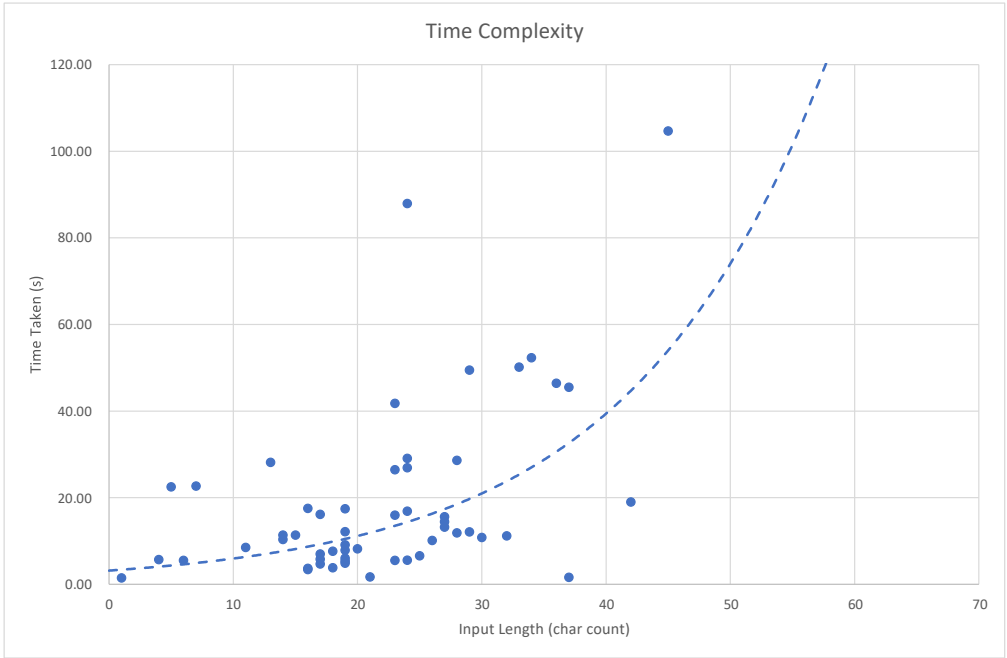

Results
Top Page + Item
Top Item
Top Page
Burried
Relevant
Irrelevant
No Items
Error


Results Breakdown
Other
Info
Search
Attribute
Fact
0
5
10
15
20
25
Top Page + Item
Top Item
Top Page
Burried
Relevant
Irrelevant
No Items
Error


Speed Distribution
Frequency
Lower Bound (s)
0
5
10
15
20
25
0
10
20
30
40
50
60
70
80
90
100


Results Breakdown
Error
No Items
Irrelevant
Relevant
Burried
Top Page
Top Item
Top Page + Item
0
2
4
6
8
10
12
14
16
Fact
Attribute
Search
Info
Other


Results Breakdown
Unanswered
Answered
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
100%
Fact
Attribute
Search
Info
Other


Time Complexity
Time Taken (s)
Input Length (char count)
0.00
20.00
40.00
60.00
80.00
100.00
120.00
0
10
20
30
40
50
60
70