Challenges in Implementing an Infobot

Joseph A. Boyle

October 25, 2017

1 Introduction

In this project, we create a program capable of taking in user inputs in the form of natural sentences, and generating suggestions of where to eat dinner. These type of programs, commonly called InfoBots, generally consist of either a neural network or algorithmic approach to bridge the connections between user input and program output. Our approach, described in section 3, involves utilizing keyword extraction, document similarity, and document weighing to generate suggestions, and a state machine to determine which type of discussion we are currently having with the user.

Developing such a system is a fairly involved task, as it requires several distinct constructs:

- The ability to derive meaning from a sentence.
- A source of data for the InfoBot to actually build suggestions from (ie: restaurants, their menus, etc).
- Some way to decide which restaurant is better than the other, based on previous user input.

We address the first point in section 2, and the other two in section 3.

1.1 Operation Overview

The implemented bot functions by having a back-and-forth conversation with the user as follows:

Bot: Hello, what would you like to eat?!
User: Hello, I would like to eat either chicken or burritos.
Bot: I have found a restaurant, ..., here is a general overview of it: .... Would you like to eat here?
User: No.
Bot: I have found a restaurant, (the next restaurant) ..., here is a general overview of it: .... Would you like to eat here?
User: Actually, I would like pizza.
Bot: I have found a restaurant, ..., here is a general overview of it: .... Would you like to eat here?
User: Can I see the menu?
Bot: Here's the menu: .... Would you like to eat here?
User: Yes I would!
Bot: Okay, here are the contact information: ...

2 Natural Language Processing

We define a word as any English word, and a sentence as any string of one or more words, optionally separated by one or more punctuation marks (., ,, !, ?, ...). A word in this sense necessarily has a part of speech, such as: noun, pronoun, adjective, verb, adverb, etc. Given a sentence, we can derive which part of speech each word via various algorithms explored in the below sections.

Since nouns are words describing people, places, actions, things, and ideas, they generally carry the weight of a sentence in terms of describing the what it is taking about. In the sentence "I want Thai food", "I want" is irrelevant to what the sentence is talking about – the nouns "Thai" and "food". Given an arbitrarily long sentence, we can remove the words that don't convey what the actual subject matter is – what the user is trying to eat.

2.1 Dictionaries and Switches

Perhaps the most primitive way of doing this is to create a long list of words which are defined as nouns, and then for a given sentence s, fetch all words in the sentence which appear in the dictionary.

This approach has five downfalls:

- The dictionary will be incredibly large. Many estimates indicate that there are approximately 100,000 nouns[1]. Each word contains about 5 letters[2], thus requiring $(5 + 1) * 100000 = 600000$kB to store these words in the best case.
- Speed is extremely slow. With 100,000[1] nouns in the English language, a ten word sentence will yield an upwards of a million comparisons.
- Relating to point one, this list has to actually be assembled, which is difficult to do by hand in a short timeframe.
- Pronouns and noun phrases aren't captured. Consider "The Original Picken Chicken" or "Loui City Pizza" (two restaurants in New Brunswick): a simple dictionary may get "Chicken", "City", or "Pizza", but will miss out on the other words which are needed to describe a specific place.
- Humans are typically bad at spelling. Consider phrases like "I want Carribean food", where "Carribean" won't be recognized, as the correct spelling is "Caribbean".

2.2 Machine learning

As with most recent computer science projects, there is a large push in exploring the uses of machine learning. We define machine learning as any program which utilizes neural networks to train on a fixed dataset. During training, the network assigns "weights" to nodes called neurons, such that when given a similar input it produces the same output. This is done via weights instead of, say, a case table (ie: $a \implies 1$, $b \implies 2$, ..., etc), to afford the system the ability to be able to use the data previously evaluated to essentially make assumptions when given a unique input.

The large downfall of neural networks and machine learning in general is that they require large datasets to train on. That is, in just the same way that humans need to see a large number of examples to understand

a pattern, a neutral network does, too. It is trivially easy to implement the neural network – the "brain" – but quite hard to show it enough examples such that it understands what it is doing.

In the context of our application, this amounts to essentially creating a test set of several million sentences and their respective keywords, and feeding them into a neural network. This alone is a fairly difficult task, especially in a short timespan.

2.3 Natural Language Processing Techniques

Natural Language Processing (NLP) is a fairly large and blooming subfield of Computer Science, which attempts to utilize dictionaries, machine learning ideas, and other algorithmic approaches to derive meaning from a given text.

Primarily, we are interested in Part of Speech tagging, which is the assignment of a part of speech to each word in a sentence. Once all are assigned, we can simply pick out the nouns that we are seeking. Unfortunately, this process is somewhat inaccurate and actually quite difficult to implement correctly[3]. For this reason, it is quite difficult to correctly implement these algorithms efficiently in a short timespan; much like machine learning algorithms, getting these things right are the subject matter of entire doctoral studies.

3 Implementation

The implementation of the InfoBot roughly involves taking in some query from the user, extracting the nouns, referred to as keywords, and then performing an analysis of which restaurants most closely conform to those keywords. As the user manipulates their queries and rejects restaurants, we negatively weight restaurants that the user does not like. This gives the effect of elevating potentially good restaurants, until the user finds one that they ultimately go to.

Below we outline several of the factors of our algorithm, namely where the data for the restaurants came from (3.1), and the remainder discussing how queries are mapped to documents.

3.1 Data Sources

A number of sources for restaurant information exist, most notably Yelp, Google Places, and EatStreet. Yelp offers far more reviews than actual menu content, and Google Places tends to offer location data but no menus or reviews. EatStreet, however, offers a full set of location data and a menu. Despite lacking reviews, being able to pull every bit of data needed to actually know what a restaurant sells. Every data source has fairly strict API limits – you are only allowed so many pulls per day/month. This is unfortunately the largest downside in using a source that doesn't combine all of the data you may need: to access Yelp reviews, for example, we would need to execute multiple queries for every single restaurant in the data set, which would be fairly expensive.

We pulled all of the restaurants in New Brunswick, and stored them for offline usage, such that the bot could be run many times without exhausting the API limits. This also afford flexibility in being able to tag up the data sources as we learn more about the user and what they may enjoy or not enjoy about a particular restaurant.

3.2 Keyword extraction

Given a lack of experience in Natural Language Processing, it was decided to choose a library which could handle Noun Extraction. For this project, that library was. This library is fairly good at extracting noun phrases, which then could be used to create a keyword string.

In some events, the noun extraction simply fails. This could be due to either the user not entering nouns, or the bot expecting "acceptance" or "rejection" answers. For example, after suggesting a restaurant, the robot looks for one of four events:

- The user "Accepting" the suggestion, at which point we stop.
- The user "Rejecting" the suggestion, at which point we negatively weight the restaurant and find the next suggestion.
- The user asks for a menu, at which point we show the menu and then ask for more input.
- The user says something else, at which point we assume it's a new query.

The first three points are handled by dictionaries. In general, users will say "yes", "yeah", "ok", or other similar terms to "accept" a result, and similar things can be said for rejecting a result or asking for a menu. When all three of these events fail, we assume the user is simply trying to give us a query. If we cannot extract keywords from this query, we use the entire query – due to the nature of the document similarity algorithm discussed next, having a slightly more verbose query isn't the end of the world, save for a few possibly erroneous samples.

3.3 Document Similarity

We consider each restaurant to be a "document", in which its contents are the name, location, full menu, and types of food. Thus, when we compare a query and a restaurant, we see how much in common they have.

We make this computation via the cosine similarity of a given document D and the query Q, where td-idf(A, B) is term frequency of A and B multiplied by the inverse document frequency of A and B:

$$\frac{\text{td-idf}(D,Q) \cdot \text{td-idf}(Q,Q)}{||\text{td-idf}(D,Q)|| * ||\text{td-idf}(Q,Q)||} \tag{1}$$

We compute the cosine similarity instead of using the td-idf directly, as it allows for a simple numeric comparison, versus comparing each and every field, which is much more computationally expensive and less intuitive.

3.3.1 Term Frequency

The term frequency is how often each word in the query Q occurs in the document D. That is, given a document *"The cat in the hat"*, and a query *"the cat"*, the term frequencies are as follows: $\{$"*the*" $\implies$ 2,"*cat*" $\implies$ 1$\}$.

This helps give an insightful measure into how much of the document deals with the same words as the query. What we can't generalize, though, is comparing two term frequencies from two different documents of differing lengths. Consider D_1 = *"The cat in the hat"*, and D_2 = *"To be or not to be? That is the question. ... also, I like cats!"*. Certainly, a longer document may contain a higher frequency by virtue of its length. Thus, we divide each term frequency by the number of words in the document. In the original example, then, the term frequencies become: $\{"the" \implies \frac{2}{5}, "cat" \implies \frac{1}{5}\}$.

3.3.2 Inverse Document Frequency

Given a large number of documents, some words may be more common than others. Surely, *pizza* or *chicken* are very common across many restaurants, but *escargot* appears very rarely. Thus, if the query is *"I want chicken or escargot"*, much more weight should be applied to restaurants that have escargot, as otherwise those with chicken would overshadow it.

Thus, we define the inverse document frequency for a word W as follows:

$$log(\frac{\text{Total number of documents}}{\text{Number documents with W in it}}) \tag{2}$$

Being a logarithmic relationship means that for every ten documents a word appears in, it will be weighted one time lower than those with less appearances than it.

3.3.3 Term Frequency - Inverse Document Frequency

We now combine these two factors. Given our example in 3.2.1, and assuming we have 100 documents, 90 of which contain the word "the", and 15 of which contain the word "cat", the word "the" has an inverse document frequency of $log(\frac{100}{90}) = 0.0458$, and "cat" has an inverse document frequency of $log(\frac{100}{15}) = 0.824$. Thus, we get a final similarity vector of: $\{"the" \implies \frac{2}{5} * 0.0458, "cat" \implies \frac{1}{5} * 0.824\} \implies$ $\{"the" \implies 0.01832, "cat" \implies 0.1648\}$.

Thus, we now have the ability to compare two documents based on similarity to the document, without regard to the document length.

3.3.4 Document Weight

As the user interacts with the program, we assign various weights. Originally, every restaurant has a weight of "1", or neutral. For every interaction that the user shows signs of disinterest in the restaurant, we lower this by one, to a minimum of -20. A restaurant with a "weight" of -20 indicates a restaurant that the user would like to go to under no circumstances, and a restaurant with a "weight" of 20 indicates a restaurant that a user would go to in a heartbeat.

We compute a "logarithmic weight" to scale the similarity score with as such:

$$ln(e + (2 * (weight - 1) * e)) \tag{3}$$

That is, for every step below 1 that we take, the restaurants drops roughly two times below equally similar restaurants of a favorability one step higher. This grows quite quickly, such that a restaurant with a

bad "weight" will only appear when if it has a significantly higher similarity to the query than all other restaurants.

4 Conclusions and Discussion

Fortunately, there exists a large amount of research in NLP, and an even larger amount of data already accumulated for nearly any dataset to be assembled. If the bot needed the ability to do sentiment analysis, there exists a Google API for it. If the bot needed to offer information on reviews in it's internal documents (perhaps to negatively weight poorly reviewed restaurants), there exists a Yelp API.

4.1 Data Overload

Truly, the largest setback in the development of the bot is data overload. In the early developments of chatbots, the datasets were much smaller and less easy to come by, by virtue of data collection not being as easy with less processing power. It is incredibly difficult, despite all of the data available to us, to create a reliable dataset which covers all of the following favorable points:

- Relevant: many data sources don't contain much that's useful by themselves.
- Cheap: most APIs either require large fees for creating an API key, or limit the number of pulls you can make to an absurdly small number.
- Fast: many APIs limit how quickly you can access data (Google Places, for for example, allows 10 queries/second).

As the saying commonly goes: you can be fast, cheap, or reliable; pick two. Much of the data accumulation of this project came down to stitching together information, rather than pulling from one source, which carries with it certain risks, specifically of poorly matched data (ie: Restaurant A accidentally gets linked to Restaurant B's reviews by accident, due to similar identifiers.

4.2 Qualified Relationships

One of the largest shortcomings of this bot is that it fails to understand quantifiers of keywords. For example, consider *"I want burritos without beef."*, which yields the keywords *"burritos"* and *"beef"*. Our algorithm will now search for restaurants that are similar to the query of *"burritos beef"*, as previously described. There is no negative weight on the word *beef*, though, as the user intended.

This flaw in the algorithm also affects "and" versus "or" relationships. Consider the differences between the queries *"I want beef and chicken"* and *"I want beef or chicken"*. The first implies that the user only wants restaurants that sell both, while the second implies that the user wants restaurants with one or the other, or both.

This flaw exists in the system for two reasons. First, it is fairly difficult to allow fully quantified results with such a small number of restaurants in our area – we limit the results to just those in the immediate vicinity of New Brunswick, which doesn't offer a very huge variety. Second, it is fairly difficult to correctly interpret the meaning of a complex sentence. Consider the query *"I would like beef but not chicken or chicken but*

not rice". It is difficult for humans to interpret this kind of sentence correctly, let alone a machine, and this assumes that the user input their query as they actually intended – human languages are ripe with lack of expressiveness due to colloquialisms.

4.3 Final Thoughts

Developing bots is a downright difficult process, requiring the developer to address a multitude of problems: what type of data we expect from the user, what types of results the user expects, what types of data we need to fulfill those user expectations, how to process user input, how to organize data flows, etc. As with most software projects, the largest downfall is almost always scope – generally, things that seem easy to implement in the mind are much harder to actually develop – and that scope grows quickly.

It is for this reason that we've seen an abundance of weak AI[4]. It is difficult to work with unimaginable inputs, but rather to focus on solving smaller problems. In developing a bot which has to work with humans, it becomes clear that the UNIX philosophy – write small programs that other programs can utilize, which in turn can be utilized by larger programs, and so on – is useful in the development of a larger AI. That is, it is probably quite easier to develop an AI that knows to refer to one of many other AIs in its toolbelt to solve individual problems, rather than have a large AI developed that can do it all. This, of course, carries with it problems such as standardizing inputs and outputs, which is still a significant problem.

References

[1] "How Many Words Are There in the English Language." *Oxford Dictionaries — English*, Oxford Dictionaries, en.oxforddictionaries.com/explore/how-many-words-are-there-in-the-english-language.

[2] V. V. Bochkarev, A.V. Shevlyakova, V.D. Solovyev, "Average word length dynamics as indicator of cultural changes in society"

[3] Daniel Jurafsky James H. Martin, "Speech and Language Processing."

[4] Jeff Kerns — Feb 15, 2017. "What's the Difference Between Weak and Strong AI?" *Machine Design*, 15 Feb. 2017, www.machinedesign.com/robotics/what-s-difference-between-weak-and-strong-ai.