An Experimental Application of Unsupervised Modeling to Chatbots

Christopher Varga and Nicholas Marble
University of Colorado at Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, CO 80918

Abstract

Chatbots are computer programs designed to emulate an intelligent conversation that is indistinguishable to what a human could provide. They are used for a multitude of functions from technical assistance to personal companionships. A large variety of these chatbots use large databases with collected hard coded answers or attempt to learn proper answers through some sort of supervised machine learning. These approaches can sometimes create reasonable but unhuman-like responses. We propose to instead use actual human responses derived from Twitter to simulate a more human grammar.

Introduction

Chatbots are used daily around the world for a multitude of tasks. A chatbot is defined by many as a form of artificial intelligence that has the ability to mimic a human-like conversation. This simple idea can easily be implemented to create technical support identities or intelligent personal assistants, and perhaps even for personal companionship. But, to implement a design that can truly be indistinguishable or equivalent to a human is no trivial task.

One of the first known implementations of a chatbot was the program ELIZA. ELIZA was written in MAD-Slip for the IBM 7094. Its name was chosen to emphasize that it may be incrementally improved by its users (Weizenbaum 1966).

Human: Men are all alike.
ELIZA: In what way?
Human: They're always bugging us about something.
ELIZA: Can you think of a special example?
Human: Well, my boyfriend made me come here.
ELIZA: Your boyfriend made you come here?
Human: He says I'm depressed much of the time.
ELIZA: I'm sorry to hear that you are depressed much of the time.
Human: It's true. I'm unhappy.
ELIZA: Do you think coming here will help you not to be unhappy?
Human: I need some help, that much seems certain.

Figure 1: An example of a conversation with ELIZA.

ELIZA used keyword mapping, where it searched the input for a word and associated rules with it. Like in the example, the keyword depressed triggers a response of Im sorry to hear that you are depressed. While this may work well in certain instances, if the user was to ask a question, this technique would not work as well. To move away from these hard-coded responses you need to implement a type of machine learning environment. Machine learning will allow the program to learn new words or sentences without the need to hard-code these responses. An example of using a machine learning technique can be found in A.L.I.C.E., Artificial Linguistic Internet Computer Entity.

ALICE was created by Dr. Richard S. Wallace in 2003. While learning, ALICE creates an Artificial Intelligence Mark-Up Language which holds in general three different categories generated by a pattern consisting of words, spaces and wild-card symbols (Wallace 2003). These categories are Atomic, Default and Recursive. A template is generated as a response for each category. The Atomic category holds all patterns that do not include the wild card, while the Default category holds all patterns that do have the wildcard symbols. Finally, the Recursive will hold categories that may be too large to accurately deduce a meaning, and hence need to be shortened into smaller ones. For example, Hello * would be categorized in the Default section and a templated response can be generated as Hi there *.

We will be using this type of categorizing with modification that these templated responses will be pulled from the Twitter API to emulate more real human responses. Using this technique will allow for real-time categorization without the restriction of a permanent, static database. These responses will also reflect the currently used phrases and words. The idea is that while a more ridged response or Template may result in a higher acceptance rate from the user, a dynamically generated response will generate a more human like response that makes it less distinguishable as a computer program.

Related Work

Similar work with was done by Ritter et al. (Ritter et al. 2010). However, Ritter et al. focused more on the tagging of the Twitter conversations with the approach of unsupervised conversation models, where the discovery of acts amounts to clustering utterances with similar conversational roles. The goal of this was to eliminate the collection and tagging process and allow the learning algorithm to categorize how people converse in a new medium.

Banko and Etzioni instead attempted to use entire websites collected as a corpus for use with ALICE. At the beginning of each learning task, ALICE receives an unexplored concept $c(n)$ as input, instantiates the set of extraction patters (e.g such as) with name of $c(n)$, (e.g. fruit such as), applies them over the input corpus, and uses the assessment model to add high-probability instances of $c(n)$ to its theory (Banko 2007). This is beneficial in the fact that it was almost 80% accurate but is not representative of normal speech behavior. Our own approach is discussed below.

Implementation

We begin by receiving a sentence from a user through a command line interface. In our original strategy, we planned to classify the tweets and the input into one of eight broad categories, namely: Status, Question to Followers, Reference Broadcast, Question, Reaction, Comment, Answer, and Response (Ritter et al. 2010). The goal behind this strategy was to have the chatbot produce more accurate responses by responding to Questions with Answers, for example. However, during the implementation we noticed that strictly enforcing this strategy actually caused our chatbot to become less realistic in some scenarios. This is because, in a real conversation, humans often respond to a question with a question. By restricting the chatbot's responses to particular broad categories, we made certain realistic responses impossible from the outset, and the chatbot was therefore too rigid. Therefore, instead, we now focus primarily on finding a tweet in the same narrow category, not what broad category the tweet is in.

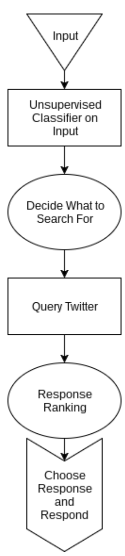

Figure 2: Logic flowchart for our chatbot.

We maintain a continuously updating local database of tweets to use for chatbot responses. The goal of using Twitter tweets as a live corpus is to simulate a more relaxed and realistic conversation, such as what one would find on the Internet.

We tried grouping the tweets and the input into similar groups in two different ways. First, we tried grouping the tweets and the user input into a cluster by using the k-means clustering module in the Pattern python library (Smedt and Daelemans 2012). Once the input was placed into a cluster, we then used our RankReponse function to identify and respond with the most similar tweet to the input. The idea behind this approach was to insure that the response tweet was as relevant to the input as possible. However, the results of the basic k-means approach was not effective. The algorithm took an increasingly long time as we increased the number of tweets it could operate on, taking upwards of an hour for only 1000 tweets. This slow time was somewhat mitigated by using a smaller value for k in the algorithm and specifying fewer iterations. However, yielding a relevant response from the chatbot using k-means still took too long to be feasible.

The inaccuracy and slow speed of the k-means strategy in finding relevant tweets for user input prompted us to take a look at a different strategy, which is now our main approach, namely, latent semantic analysis (LSA). This approach treats each tweet as a bag-of-words, where we do not care about the *order* of the words in a tweet, only their *frequency*. Since some words in the tweets such as "the" or "a" have a much higher frequency, LSA eliminates this background noise by assigning a weight to each word using the $tfidf$ formula (or *term frequency inverse document frequency*)

$$\text{tfidf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$$

where t is the term in question, d is the document, and D is the set of all documents. The idf formula, in turn is

$$\text{idf}(t, D) = \log \frac{N}{n_t}$$

where $N = |D|$ (the number of documents) and

$$n_t = |\{d \in D : t \in d\}|$$

The weighted frequencies of the terms in each tweet are then represented as a term-document matrix where each row represents a unique word and each column represents a document (in our case a tweet). In order to group the tweets into narrow categories, we then use the Pattern python library to perform singular value decomposition (SVD) on our term-document matrix in order to create a concept space of tweets and terms that are related to similar concepts. We include the user input into our term-document matrix so that it can also be grouped into a narrow concept category, and then rank the relevance of those tweets using our RankResponse algorithm, as we did for the k-means approach. The tweet that is ranked the most relevant is returned as a response from the chatbot.

```
User:      do you like trump?
Chatbot:   did KellyAnne marry into the Trump family and I just
           missed it?  Yeah, didn't think so…

           (u'trump', 0.09642409227494203)
```

Figure 3: Example of most probable response from 'trump' category

The logic of our RankReponse algorithm is as follows. First, we use the bag-of-words approach and vectorize the user input and tweets according to word frequency. For example, the following tweet A ="It is a nice day is it not' and input B="I think it is a nice day" would be vectorized as follows:

$c =$ [i think it is a nice day not]
$a = [0, 0, 2, 2, 0, 1, 1, 1]$
$b = [1, 1, 1, 1, 1, 1, 1, 0]$

We then calculate the cosine similarity $\frac{ab}{|a||b|}$ between the vectorized input and each tweet. The tweets are ranked according to cosine similarity, and the tweet with the highest similarity is selected as a response. If none of the tweets is similar, that is, if the cosine similarity of all the tweets is zero, then the chatbot returns "I'm sorry, I do not understand."

RankResponse($tweets, input$)

```
R = []                              ▷ Initialize our list of responses
for tweet in tweets do
    r = CosineSimilarity(tweet, input)
    InsertSorted(r, tweet, R)
end for
return R[0]                         ▷ The most similar result is returned
```

Interestingly, since our corpus is based on Twitter, which is known for slang and typos, the chatbot tends to provide more accurate responses when the user input also contains slang and typos, since there is a higher similarity between the user input and the tweets. An example of an informal conversation utilizing slang from our chatbot is below:

```
User:      hi there, i'm chris.
Chatbot:   i'm back & i love it
User:      glad you are happy
Chatbot:   where are you now that i need you
User:      i am home, you?
Chatbot:   i miss you, i miss you, i miss you..
User:      where r u?
Chatbot:   GBC Disaster Relief Training.
           Why r u not trained? What r u waiting on?
```

Figure 4: Sample conversation with our chatbot.

Evaluation Methods

We evaluate the effectiveness of our chatbot both qualitatively and quantitatively. First, to evaluate our chatbot qualitatively, we ask a specific question, and then determine whether the response from the chatbot seemed human-like. This evaluation method obviously necessarily requires a degree of subjectivity, but nonetheless allows us to judge the relative effectiveness of the chatbot in holding a conversation. In this area, the chatbot performed surprisingly well. Depending on the inputs given, the responses seemed quite relevant. For example, any input including a politician such as Trump receives a remarkably realistic and emotional response such as what one would find on the internet. Since our Chatbot does not keep track of previous questions, the responses can sometimes seem out of context and vague. However, the vagueness of many of the response tweets also helps to keep the conversation open-ended feeling. This is considered a positive by us, since we are aiming for the chatbot to seem less scripted and more natural.

Second, in order to quantitatively evaluate the effectiveness of our chatbot, we originally planned to adopt a method inspired by Ritter (Ritter et al. 2010). This method attempts to objectively rate the conversational accuracy of the chatbot in terms of its ability to replicate an actual twitter conversation. In other words, we would use a real tweet as input to the chatbot. Then, we expect the response given by the chatbot to be the actual tweet given by a human in a real world conversation.

However, we noticed that an easier way to quantitatively measure our chatbot would be to simply measure the number of responses that chatbot made, since it is possible that the chatbot could fail to make a response (when the similarity of all the tweets is zero). We thus rate the overall effectiveness of the chatbot by calculating the percentage of responses it makes. We consider our chatbot to be a success if it's response rate is near 80%. Our results are below.

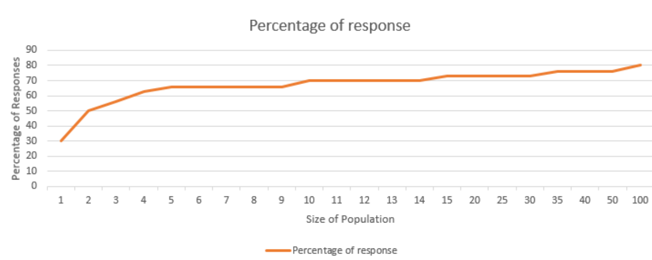

Figure 5: Experimental results for our chatbot

We noticed that querying a larger number of tweets resulted in a higher response rate. We attribute this to the fact that, with a larger corpus to choose from, it is more likely that the chatbot will be able to find a similar tweet within the same narrow category as the input. The highest response rate we could achieve was around 73% on average with a population of 100 tweets, with a peak of 80% for 100 tweets.

Future Explorations

A key advantage to our chatbot approach is that it will more closely mimic real human conversations by including slang and other less formal language available in Twitter messages. However, an obvious downside to our approach is that using real verbatim tweets means that the chatbot's responses will be less personalized. For example, currently, our chatbot cannot refer to its conversation partner by a specific name unless, by coincidence, a real tweet happens to contain that name.

Therefore, a future area of exploration for our chatbot could include modifying its response so that it is not an exact verbatim tweet. For example, if we know that the chatbot is conversing with someone named Bob, we could inject that name in front of the tweet in specific contexts. In order to accomplish this, we would need to ensure that the name injection resulted in a coherent tweet. This would involve more closely analyzing the parts of speech of the response tweet in order to make accurate injections specific to the context of the conversation and sentence structure, and keeping track of the history of past conversations.

References

[1] Weizenbaum, J. (1966). ELIZA-A computer program for the study of natural language communication between man and machine. Communications of the ACM, vol 9 issue 1 (pp. 36-45).

[2] Wallace, R. (2003). The elements of AIML style. ALICE AI Foundation.

[3] Ritter, A., Cherry, C., & Dolan, B. (2010, June). Unsupervised modeling of twitter conversations. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics* (pp. 172-180). Association for Computational Linguistics.

[4] Banko M., Etzioni O., (2007). Strategies for Lifelong Knowledge Extraction from the Web. Turing Center, University of Washington.

[5] Smedt, T. D., & Daelemans, W. (2012). Pattern for python. Journal of Machine Learning Research, 13(Jun), 2063-2067.