Impersonating Myself on the Internet

Abraham Shultz
University of Massachusetts Lowell
1 University Ave, Lowell, MA
ashultz@cs.uml.edu

ABSTRACT
This paper describes the implementation of a computer program that responds to chat messages sent by users of an internet chat client. The program uses a statistical model of previous chat messages sent and received by the author to determine possible meanings of incoming messages and select an appropriate response. Two heuristics are also used to allow the program to respond in cases where the meaning of a message is unclear.

Author Keywords
chat, chatbot, dialog, Bayesian, TF-IDF, log processing, python, AIM, instant messenger

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation: Miscellaneous—Optional sub-category

INTRODUCTION
I have attempted to build chat program that behaves sufficiently like me on instant messaging chat that my friends are, at least temporarily, fooled into thinking that they are talking to me. The program does not start conversations, but will respond when a user sends it a message.

The basis of the chatbot is a Bayesian statistical model of my previous chat conversations. It uses prior probabilities based on Term Frequency-Inverse Document Frequency (TF-IDF) metrics to attempt to determine the most important word of a sentence, and then responds with a randomly-chosen sentence that shares that most important word. S

PROJECT DESCRIPTION
In order to respond reasonably to a given input, the program must first determine what the sentence is "about", and then select a response that is related to the incoming sentence. So that it can always respond to an incoming message, my program uses two heuristics and a statistical model of sentences that I have received from and sent to other people over the last two years. The model provides most of the appearance of understanding text, while the heuristics serve to save processing time and ensure that the chatbot always has a response.

The first heuristic is a simple matching test applied to each sentence as soon as it arrives. If the other party to the conversation sends a sentence that is identical to one that occurs in my chat logs, the will send whatever I originally sent as a response. This is adequate for dealing with expected call-response pairs in communication between humans, such as "Good night" - "Good night" or "I love you" - "I love you too".

Assuming there is no exact match, the program attempts to determine which word in the incoming sentence is the most relevant to the meaning of the sentence. This word is the "keyword" of the sentence. For each word in the sentence, the program assigns it a score that is the probability that that word is the keyword.

The probability that a word is the keyword is:

\[
P(class_{foo}|"foo") = \frac{P("foo"|class_{foo}) \times P(class_{foo})}{P("foo")}
\]

That is, the probability of a sentence being in a class called "class_{foo}" given an observation of the word "foo" is the probability of observing the word "foo" in a sentence from class_{foo}, multiplied by the probability of a sentence being in class_{foo}, and all divided by the probability of observing the word "foo". This is a very naive implementation of Bayes' theorem to classification.

P("foo") is easily calculated from the chat logs. First, the logs are split into messages people have sent to me ("their messages"), and messages I have sent to other people ("my messages"). Their messages are broken down into a word count, enumerating each unique word in every sentence. Words such as "the" tend to have very high counts, while words like "obfuscate" or "leonine" are used more rarely. The probability of observing a word is:

\[
P("foo") = \frac{\text{count of "foo"}}{\text{total number of words}}
\]

P("foo"|class_{foo}) is calculated exactly like P("foo"), but the probability is only calculated over the sentences which have been decided to be in class_{foo}, rather than all sentences.
The formula for calculating the probability of a sentence belonging to class \( \text{foo} \) is:

\[
P(\text{class}_\text{foo}) = \frac{\text{sentences in class}_\text{foo}}{\text{total sentences}}
\]

To separate the sentences into classes, all of the chat conversations are run through a TF-IDF scoring program. TF-IDF is a simple metric that calculates a score for each unique word in the conversation. The score is:

\[
\frac{\text{count of word}}{\text{words in chat}} \times \log \frac{\text{count of other conversations}}{\text{count of conversations with word}}
\]

For each word, starting with the highest scoring word, all sentences which contain that word are tagged as belonging to the class named after that word. Assuming "spades" was the highest scored word in a conversation, the sentence "The only card I need is the ace of spades" would be tagged as belonging to \( \text{class}_{\text{spades}} \). If the TF-IDF tagger runs out of words with a score greater than 0 before it runs out of sentences to tag, the remaining sentences are tagged as having no clear keyword.

All of these probabilities are calculated \textit{a priori} from my old chat logs. I have used a chat program configured to log chats for just over two years, and it is rare for me to not speak with anyone over IM for a day. As a result, I have saved up a considerable number of recorded chat conversations on all sorts of topics, with a wide variety of people. Some of the conversations are with friends, some with co-workers, and all of them are tagged with time and date information, as well as the identities of participants.

Given the calculated probabilities, the chatbot determines the probability that a sentence is about a particular word in that sentence, by assigning a probability for each word in the sentence that it is the word that that sentence is about. \( P(\text{class}_\text{foo} | \text{"foo"}) \) is calculated for each word in the sentence and for each possible class, keeping the highest score for each word. The word with the highest overall score is assumed to be the keyword for that sentence.

This phase of the process takes a considerable amount of computer time, as every word in the sentence must be compared with every class detected by the TF-IDF algorithm. It is likely possible to reduce this time, by using some of the methods described in the “Discussion” section of this paper. However, the period taken to select a response is roughly that required for a human typist to enter their response, and so the lag seems “realistic” to another human. If the program could respond too quickly, it would be apparent that it was not a human using a keyboard, but a program.

Once the keyword is determined, it is used to look up all of my sentences which the TF-IDF tagger determined to have that keyword. Assuming there exist any such sentences, the chatbot selects one at random and sends it as a reply. However, it is possible that there is no keyword in the sentence, or that no sentence I have said is tagged with the keyword of the sentence. This is where the second heuristic is used to select the response.

In sentences where the most important word is implied, such as "Let's do that", or “active listening” responses, such as "Hmmm” or “Oh?”, the subject is too vague to respond to correctly. If "that" were an allowed subject, the computer's idea of the subject of the sentence could change radically over a single utterance, resulting in incoherent conversation.

When the chatbot determines that there is no matching keyword among my sentences for the keyword found in the incoming sentence, it selects from the subset of my sentences that did not have a keyword. Since these tend to be vague, short sentences or onomatopoeia, they are usually a sufficient, if unclear, response.

To handle the actual communication with the human conversationalist, the program interfaces with Pidgin, a multi-protocol IM client, over D-Bus. D-Bus is a desktop message-passing inter-process communication API. The chatbot registers a callback function to be called when a message arrives. When a message arrives, the callback is called with the incoming message as a parameter. That message is used as described above to generate the response. If the program were run with different chat log data, it would generate different responses, so it may be used to emulate anyone with whom I have chatted enough to build up a large log.

**ANALYSIS AND RESULTS**

The goal of the script was to produce a chatbot that could imitate me on IM with sufficient fidelity that it would fool someone who was speaking to it. For very short conversations, on the order of two to four exchanges, this goal was met. However, the longer the conversation went on, the more likely the chatbot was to do something unusual.

Frequently, it would tip its hand by being inappropriately affectionate. The majority of my chat conversations are with my girlfriend, and we will express affection by messages such as "*hug". Since that string occurs frequently across many conversations, it was assigned a very low TF-IDF score. As a consequence, the “active listening” response set contains a high proportion of affectionate messages. If any human conversationalist sent a message that the chatbot did not understand, it was likely to hug them.

Another way that the chatbot would fail was frequent repetition. It only had a few responses tagged as being about “pie”, so mentioning pie several times would quickly result in a repeat of a previous message.

The final way that the chatbot would indicate that it was not really me is by coming up with responses that did share a common word with the sentence it was responding to, but were not related, such as responding with a sentence about waiting for the bus when asked to “wait for a minute”. Both sentences were about waiting, but they had nothing to do with each other, and so the overall effect was incoherence.

Overall, it does not appear that a simple mix of TF-IDF and Bayesian statistics are sufficient to produce a convincing
conversationalist. The current implementation does appear to be a good basis on which a more robust conversational agent could be built. I expect that the largest improvements would be had by improving the conversation pre-processing, as discussed in a later section.

It is worth noting that none of the programming that went into developing this chatbot was aimed at producing a model of understanding, or of meaning. The chatbot does not attempt to emulate the processes of the human mind in comprehending the meaning of words. In fact, it has no attempt to determine meaning at all. Searle, in [4], draws a line between “Strong AI” and “Weak AI”. The strong AI position is to claim that thinking is formal symbol manipulation, and so is directly amenable to implementation in a computer program. Weak AI is the stance that, while some computer models of thought may be useful to the study of the mind, the do not constitute minds themselves.

My program does not, nor does it make any pretense to, actually understand sentences. It does attempt to convey the appearance of understanding sentences. If it were to somehow be perfected, such that it appeared to all onlookers to understand sentences, either it would have to actually understand sentences (the strong AI view), or it would be a “good enough” model of sentence understanding (the weak AI view), but either way, it would be indistinguishable from something that does actually understand sentences.

A subgoal of the program was to be amusing, which the chatbot managed to do quite well. Users frequently conversed with it for quite a few more lines after recognizing that its responses were frequently nonsense, just to try to get it to say amusing things.

DISCUSSION
The project actually has far more code that is related to processing the chat logs than it does code that interfaces with the chat client or generates the response messages. A big part of the processing is getting the sentences into a format that matches across all conversations. This generally involved lowercasing the message, and in some cases splitting it at periods to isolate sentences.

Most of the ways that the script could be extended to improve its ability to accurately pick out the keyword of a sentence are based on adding more preprocessing steps to the log file handling, rather than altering the mathematical basis of the classifier.

Preprocessing could take care of three major problems in the way the program classifies sentences. The first problem is that the program responds only to exact words. As a result, it will respond to the word “pie” with a different set of responses than it would use to respond to the word “pies”, despite the fact that one word is just the plural of the other.

Reducing a derived form of a word to its root form, such as “cat” from “cats” and “swim” from “swimming” is a common problem in natural language processing. The process is called stemming, and usually consists of eliminating prefixes and suffixes to reach a root word. In [1], some of the approaches to stemming and potential pitfalls are discussed. While this information is sufficient to allow me to develop my own stemming algorithm, there are already software libraries available to stem words. Adding stemming to the chatbot script would not be a matter of integrating one of these libraries. This would cause the chatbot to seem to understand general meanings, rather than reacting to specific words, by collapsing all derivations to a single word.

Another approach to generalization would be using a thesaurus or similar database to look up synonyms or related words. If the statistical model failed to come up with any sentences related to baking, it could widen the search to cooking-related words, such as “oven”, or baking-related words such as “bread” or “flour”. WordNet is one such database [2]. It contains words linked by synonymy, like a thesaurus, but also by semantic relationships and hierarchical structures. For example, WordNet maintains links indicating the “is a” relationship between “desk” and “furniture”, but that relationship is not reciprocal, the way it would be with synonyms.

The second problem is that the program only works on the word level, and even then, it only works with words that are explicitly in the sentence. This causes the chatbot to always respond to a sentence with a sentence that includes the keyword. Any mention of pie will elicit a sentence about pie which contains the exact word “pie”, but the program is ignorant of the distinction between sweet potato pie and pecan pie.

This problem can likely be reduced or eliminated by approaches to tagging that will improve the tagger’s ability to distinguish important phrases (rather than operating at the word level). Amazon.com generates collections of “Statistically Improbable Phrases” or SIPs, which are phrases that occur in one work, but are rare in other works. This is effectively a generalization of TF-IDF to the multi-word phrase level, and is sometimes useful for determining what concepts a book on Amazon.com is about. It also sometimes captures elements of an author’s style, which is useful for a chat program that is intended to imitate a specific person.

The third problem is that the statistical model generates many spurious classes that are not useful for conversation. These classes tend to arise because sentences with an implied subject allow common words like “these” to get selected as keywords. In a short sentence with many common words, the highest scored word may not beat the others by much, and so the sentence will get classified into one of the common classes. Since these classes are more based on keywords that were only selected by narrow margins than well-defined keywords, they have a lot of not very closely related sentences. The resulting response tends to appear to not be related to the input sentence.

This problem could be eliminated by removing many of the words that the TF-IDF scorer should not regard as being pos-
sible keywords. *Hapax legomena* (words that occur exactly once in the entire corpus) are not likely to be something important and more likely to be something that was briefly brought up and then ignored, or a misspelling of another word. On the other end of the spectrum are words that occur all of the time, but are never the subject of sentences, such as “the”, “of”, “and”, and so on. It would be possible to build a blacklist of words that should not be considered as possible keywords by examining the word counts and eliminating some proportion of the most and least common words. These improvements would stop the program from working with useless keywords, such as “these”, or misspellings like “robotise”. This would also reduce execution time by eliminating many common words from consideration.

Even in typed data, human communication is very free-form. Users do not stick to perfect punctuation, so sometimes ellipses extend for more than three periods, or terminal punctuation is left off of a message. Because the users are not talking, but are using chat to emulate conversation, they tend to use onomatopoeic utterances, such as “Hmmm” or “Eh?”. Since these are not really words, they don’t have any standard spelling or rules for use in the English language.

Some form of spelling correction or rejection of words that are not recognized as proper English words would improve the class generation. As it currently stands, it is possible to get into a humming match with the chatbot, because “Hmmm” occurred often enough (with varying terminal punctuation) to be allocated to class_hmmm. As a result, it will respond to “Hmmm”, with some selection from “Hmmm?”, “Hmmm…”, “Hmmm.”, and so forth.

This problem is a symptom of a larger problem with the chatbot, which is that it does not maintain any context while it converses with the human user. Each new sentence is a totally new input, and is never related to any previous sentence. In order to appear to have a consistent idea of the subject under discussion from one utterance to the next, the chatbot should maintain a pointer to the subject of the conversation.

The simplest way to do this would be to simply have one subject recorded, and use that to select replies when there was no clear subject in the incoming message. That way, the chatbot would appear to “elaborate on”, or at least re-mention, the topic under discussion. When the user replied with a sentence that did have a clear keyword, the topic would be updated, to allow the chatbot to follow the thread of the conversation.

This does not capture a common element of human conversation, which is the tendency to return to a previously discussed topic. To capture this ability, the chatbot would have to be extended to have a stack of topics, ordered by probability of mention. When a new sentence arrived, every word would be scored as usual, by the Bayesian scorer, but instead of only the most likely word being used, the scores of each word would be combined with weighted scores of the previous topics. The weighting for each previous topic’s probability would go down as more sentences passed without it being mentioned, and go up when it was mentioned. The chatbot would use these weighted probabilities to select the most likely topic to converse about. Assuming the weights were set well, the chatbot could appear to return to previous topics, or be “reminded” of subjects that have related keywords.

CONCLUSION

Overall, the program worked well, in the sense that it frequently responded to human conversationalists with sentences that were at least tangentially related to words they had mentioned. However, it did not convince anyone that they were talking to a human, at least beyond the opening few lines of the conversation.

Most of the open areas for improvement are in cleaning the input data set to encourage the TF-IDF scorer to favor useful keywords and avoid forming large classes around pronouns or common misspellings. Adding stemming and the ability to use synonyms would permit generalization, while adding the ability to work with longer phrases would permit more specific responses. Determining an appropriate balance between these two properties is likely an AI project itself.

Another major project would be the generation of novel responses, as opposed to merely replaying my old IM responses. I had intended to use a a Markov chain of order 3 to select words based on statistical analysis of the appropriate sentences, but the output was sufficiently incoherent that it would not convince anyone that a sane or sober human had generated it.

ACKNOWLEDGMENTS

The work described in this paper was conducted as part of a Fall 2011 Artificial Intelligence course, taught in the Computer Science department of the University of Massachusetts Lowell by Prof. Fred Martin.

I’d like to thank all my friends whose chat logs went into developing the data set for this program. I’d especially like to thank my girlfriend, for teaching my chatbot to love.

REFERENCES


