SarcasmBot: An open-source sarcasm-generation module for chatbots

Aditya Joshi
IITB-Monash Research Academy
Mumbai, India
adityaj@cse.iitb.ac.in

Anoop Kunchukuttan
Indian Institute of Technology Bombay
Mumbai, India
anoopk@cse.iitb.ac.in

Pushpak Bhattacharyya
Indian Institute of Technology Bombay
Mumbai, India
pb@cse.iitb.ac.in

Mark James Carman
Monash University
Melbourne, Australia
mark.carman@monash.edu

ABSTRACT

Sarcasm detection is a recent innovation in sentiment analysis research. However, there has been no attention to sarcasm generation. We present a sarcasm-generation module for chatbots. The uniqueness of '*SarcasmBot*' is that it generates a sarcastic response for a user input. SarcasmBot is a sarcasm generation module that implements eight rule-based sarcasm generators, each of which generates a certain type of sarcastic expression. One of these sarcasm generators is selected at run-time, based on properties of user input such as question type, number of entities, etc.

We evaluate our sarcasm-generation module in two ways: (a) a qualitative evaluation on three parameters: coherence, grammatical correctness and sarcastic nature, where all scores are above 0.69 out of 1, and (b) a comparative evaluation between *SarcasmBot* and ALICE, where a majority of our human evaluators are able to identify the output of *SarcasmBot* among two outputs, in 70.97% of test examples.

Keywords

Sarcasm generation, Sarcasm generating chatbots, Dialogue systems

1. INTRODUCTION

Sarcasm is a phenomenon where literal sentiment of text is different from implied sentiment, with an element of hostility involved [14]. In this paper, we describe our sarcasm generation module for chatbots, called '***SarcasmBot***'. It has a browser-based interface that always responds sarcastically to user statements. The input to *SarcasmBot* is a well-formed English sentence. The output is a sarcastic response to the user input. In rest of the paper, we refer to the input as '*user input*' and the output as '*sarcastic response*'.

Sarcasm detection has received attention in recent times. However, to the best of our knowledge, no past work focuses on automatic sarcasm generation. We implement eight rule-based sarcasm generators, each capable of generating a peculiar pattern of sarcasm. *SarcasmBot* selects among these generators and produces a sarcastic response using properties of the user input such as question type, tense, number, set of noun entities, sentiment, etc. We evaluate the response of our sarcasm-generation module in terms of three parameters: (a) Coherence (to the user input), (b) Grammatical correctness (of the sarcastic response), and (c) Sarcastic nature (of the sarcastic response), and also compare its output with a popular chatbot, ALICE [17].

In addition to the entertainment value of such a sarcasm-generation module, the motivation lies in the Turing Test [11]. The test envisages an ideal AI agent, and states that a human interacting with an AI agent must be unable to determine whether they are communicating with a human or a computer. Sarcasm is observed in human communication. So, in order to satisfy the Turing test, we believe that a sarcasm-generation module must be able to express sarcasm effectively. This encourages us to look at sarcasm generation as an interesting problem. **We do not expect that a sarcastic sarcasm-generation module like ours will be useful as it is. However, existing chatbots can become more 'human' by integrating this behavior. In this paper, we focus on building a sarcasm-generation module that responds sarcastically to user responses. Enabling the chatbot to decide when to respond sarcastically is left as future work.**

The contribution of *SarcasmBot* is that it systematically '*algorithmizes*' different forms of sarcastic expression as observed in linguistic studies of sarcasm![4]. In order to enable further experimentation with sarcasm generation and integration into general-purpose chatbots, we will make available the source code and APIs of *SarcasmBot*.

The rest of the paper is organized as follows. We first describe the related work in Section 2. Section 3 presents the need for a sarcasm-generation module. Its architecture is described in detail in Section 4. Section 5 discusses the evaluation, while Section 7 concludes the paper.

This paper was presented at the Fourth International Workshop on Issues of Sentiment Discovery and Opinion Mining (WISDOM '15), held in conjunction with KDD'15 in Sydney on 10 August 2015. Copyright of this work is with the authors.
.

2. RELATED WORK

Context incongruity is central to sarcasm [6]. Consider the sarcastic tweet “I love being awake at 4am”. The sarcasm arises because the positive verb ‘love’ is incongruous with the negative context of ‘being awake at 4am’. Sarcasm is a common phenomenon that humans use in communication. It has an element of causticness or hostility involved. Sarcasm is peculiarized by incongruity between the literal meaning and implied meaning. . In this light, our sarcastic chat system is a step towards the Turing test. To the best of our knowledge, this is a first-of-its kind sarcasm-generation module. However, research in chatbots and dialogue systems is old, and such systems have been implemented for a variety of applications. Pattern-based matching algorithms are a popular approach in chatbots, as in the case of a popular chatbot, ALICE [17]. Another approach [1] uses Twitter as a large-scale corpus in order to find candidate responses to the given user statement.

3. MOTIVATION

Research in natural language generation systems dates back several decades [17, 2]. Chatbots have been used for a variety of applications such as information agents [7], instructing agents for autonomous vehicles [15] and automatic tutoring agents [8]. EHeBBy [13] is a chatbot that generates humor based on user input and a set of known humorous patterns. Using a similar approach, ‘*SarcasmBot*’ generates sarcasm.

Sarcasm has been widely studied in linguistics. Different kinds of sarcasm such as propositional and illocutionary have been described [3]. Similarly, different dimensions associated with sarcasm have been defined [4]. Sarcasm detection has been explored in the past in the context of sentiment analysis. On the other hand, sarcasm generation has not been looked at so far, to the best of our knowledge. We believe that sarcasm generation will lead to a deeper understanding of the phenomenon of sarcasm, and as a result, help sarcasm detection as well.

4. ARCHITECTURE

Figure 1 shows the architecture of *SarcasmBot*. The user input and sarcastic response are syntactically valid English sentences. The user input is first analyzed using the Input Analyzer (described in Section 4.1), and is then sent to the Generator Selector (described in Section 4.2). The generator selector chooses among eight Sarcasm Generators (described in Section 4.3). The chosen sarcasm generator is used to produce the sarcastic response. All three modules are rule-based, relying on lexicons. The code is available at: `https://github.com/adityajo/sarcasmbot`.

4.1 Input Analyzer

Input Analyzer extracts following information about the user input:

1. **POS sequence**: We tag the user input using the Stanforrd POS tagger [16].

2. **Question Type Determiner**: In case the user input is a question, we look up a set of words in order to determine the type of question among: *why, what, where, how, who, when and choice.*

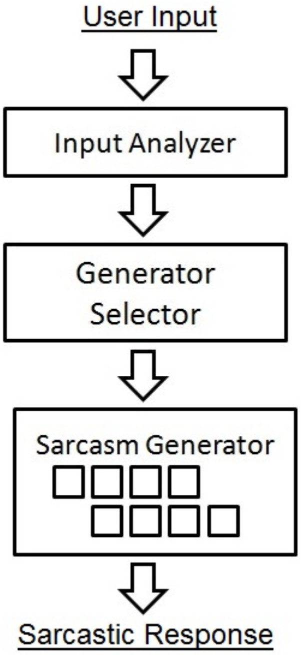

Figure 1: Architecture of *SarcasmBot*

3. **Tense**: We determine the tense of the input by looking for verb-related POS tags. This is to ensure that the response is coherent with the user input. The candidate tenses are as given in [9], specifically: *Third person present, Non-third person present, Past and Modal.*

4. **Entities**: Using the POS sequence, we create two lists of entities: noun entities (sequence of NNs) and named entities (sequence of NNPs). The entities in these lists are later used as targets for sarcasm. Associated with every noun entity is a number attribute that indicates whether the entity is in singular or plural.

5. **Pronouns**: Using the POS sequence, we identify pronouns in the user input. Additionally, we use a list of pronoun-pronoun pairs corresponding to question and response (example: ‘You’ in the user input must be responded with ‘I’).

6. **Sentiment**: Since sarcasm is associated with sentiment, we implement a rule-based sentiment determiner that uses the sentiment lexicon by [10], and applies appropriate rules on negations and conjunctions. This sentiment determiner predicts the sentiment as positive/negative.

7. **Offensive language**: Sarcasm is often used as a retort for offensive language. Hence, we identify offensive words by looking up sarcastic patterns for offensive language as given in [12].

4.2 Generator Selector

Based on the user input properties, the Generator Selector decides which of the eight sarcasm generators is to be used to generate the sarcastic response. The algorithm to select a sarcasm generator is:

Sarcasm Generator	Description
(a) Offensive Word Response Generator	In case an offensive word is used in user input, select a placeholder from a set of responses.
(b) Opposite Polarity Verb-Situation Generator	Randomly select a verb. Compute its sentiment. Discover a situation which is opposite in sentiment.
(c) Opposite Polarity Person-Attribute Generator	Randomly select a named entity. Select incongruent pairs of famous people.
(d) Irrealis Sarcasm Generator	Create a hypothetical situation that is impossible by selecting from a set of undesirable situations.
(e) Hyperbole Generator	Select a noun phrase in the user input. Generate a hyperbole with a 'best ever' style regular expression.
(f) Incongruent Reason Generator	Select an unrelated reason as a response for a user input.
(g) Sentiment-based Sarcasm Generator	Compute sentiment of user input. Generate a response opposite in sentiment.
(h) Random Response Generator	Select one positive exclamation and one negative exclamation randomly from a set of exclamations. Place them together.

Table 1: A Tabulated Summary of Sarcasm Generators in SarcasmBot

1. If the input contains an offensive word, the Offensive word response generator is invoked.
2. If the input is not a question, the Sentiment-based sarcasm generator is invoked.
3. If the input is a question that requires a Yes/No response, we count the number of entities in the input. If there is more than one entity, the Opposite Polarity Verb-Situation Generator is invoked. If the sentence contains modal verbs, the Irrealis Sarcasm Generator is called. Else, the Hyperbole Generator is invoked.
4. If the input is an opinion question, one out of these sarcasm generators is randomly invoked: Opposite Polarity Verb-Situation generator, Opposite polarity person-attribute generator or Hyperbole generator.
5. If the input is a '*why*' question and has non-zero entities, the incongruent reason generator is invoked.
6. Finally, the random response generator is invoked if none of the above apply.

Each of the above generators handles a peculiar form of sarcastic expression. The sarcasm generators are described in the next subsection.

4.3 Sarcasm Generators

The basis of our sarcasm generators is a set of sarcastic patterns with placeholders for words. A sarcasm generator selects a random sarcastic pattern (consisting of placeholders for phrases), and inserts appropriate words from the user input into the placeholders. The random selection of the pattern allows a dynamic nature to the sarcastic response.

Based on this general approach, we implement eight sarcasm generators: (a) Offensive Word Response Generator, (b) Opposite Polarity Verb-Situation Generator, (c) Opposite Polarity Person-Attribute Generator, (d) Irrealis Sarcasm Generator, (e) Hyperbole Generator, (f) Incongruent Reason Generator, (g) Sentiment-based Sarcasm Generator, and (h) Random Response Generator. Each of them models a peculiar form of sarcastic expression, based on evidences/dimensions of sarcasm as described in [6]. The generators are summarized in Table 1. In the forthcoming sections, we describe them in detail.

4.3.1 Offensive Word Response Generator

If the input contains an offensive word , the Offensive word response generator is invoked. It picks randomly from a list of responses that praise the use of the offensive word, and inserts the offensive word at the appropriate place. An example of response generated is '*Wow! I can't believe you just said 'b****'. You are really very classy!*'.

4.3.2 Opposite Polarity Verb-Situation Generator

This generator selects a polar verb from the user input (for coherence) and randomly selects a noun phrase corresponding to a '*situation*' of the opposite polarity. Specifically, the sarcastic pattern used for the response is: <Optional Filler> <Pronoun-Response> <used to/will> <Verb of Polarity X> <Optional intensifying adverb> <Situation of Polarity Y>. The *optional filler* is picked randomly from a set of filler words like '*Hmm..*'. The *pronoun-response* is the response pronoun as found by pronoun determiner. For example, if the input contains the pronoun '*you*', the pronoun-response is selected as '*I*'. The tense of the sentence is used to choose between *'used to' or 'will'*. The *verb of polarity X* is selected from the sentence. Finally, a random *situation of polarity Y (Y is opposite of X)* is selected from a set of situations. This set of situations was extracted as follows: We first downloaded tweets using Twitter API[1] marked with hashtag #sarcasm. Then, we manually selected a set of noun phrases that correspond to '*situations*'[2]. An example of response generated by Opposite polarity verb-situation generator is '*Hmm, well.. I used to like Tim, just the way I like being stuck in the elevator all night*' to the question '*Do you like Tim?*'. The situation in this case is '*being stuck in the elevator all night*'.

4.3.3 Opposite Polarity Person-Attribute Generator

The Opposite Polarity Person-Attribute Generator generates sarcasm on a named entity in the input. The generator uses the sarcastic pattern: I think that <Named entity> is <Popular Person< <plus/minus> <Positive/ Negative Attribute>. The generator first picks a *named entity* from

[1] http://dev.twitter.com/overview/api

[2] This 'manual' approach can be extended to other automatic strategies.

the input. Then, it randomly selects a *popular person* and an *attribute*. If it is a desirable attribute, the generator selects a *plus or a minus*. We have compiled a list of popular actors. For the positive attributes, we use a list of positive traits for an actor. For the negative attributes, we use a list of negative traits for an actor. An example of response generated by Opposite polarity person-attribute generator is '*I think that Jim is Tom Cruise minus talent*'.

4.3.4 Irrealis Sarcasm Generator

Irrealis mood[3] corresponds to sentences like '*I would have loved to watch this movie if I did not have anything better to do*'. The irrealis sarcasm generator selects a verb from the input and selects from a set of hypothetical negative situations. The irrealis sarcasm generator is invoked in case of questions that demand a '*Yes/No*' answer using modal verbs like '*Will/Should*'. The sarcastic pattern used for the response is: <Pronoun-response> would <verb> <Hypothetical bad situation>. The *verb and pronoun-response* is selected based on the input. An example response generated by irrealis sarcasm generator is '*He will marry only if badly drunk*' to a question '*Will he marry me?*'.

4.3.5 Hyperbole Generator

To create hyperbole-based sarcasm, we select a noun entity from the user input and use sarcastic patterns of the following type: <Those were/That is > the best <Noun Entity> in the history of humankind! The choice between *'those were' and 'that is'* is made using the number attribute of the randomly selected *noun entity* from the user input. Note that the above pattern is one among many hyperbolic patterns. An example of response generated by hyperbole generator is '*Those were the best immigration reforms in the history of humankind*' to the input '*What do you think of the immigration reforms announced by the Parliament this Saturday?*'.

4.3.6 Incongruent Reason Generator

The incongruent reason generator is selected in case of '*why*' questions. A named entity is randomly selected from the input and an incongruent reason is picked out of random unrelated reasons. One sarcastic pattern in this case is: Because <Named entity/Pronoun-Response> <Incongruent reason>. An example response to the question '*Why did Jim miss his date tonight?*' is '*Because Jim loves to play the piano*'. The sentiment of the response is designed to be opposite to that of the user input, if the user input is sentiment-bearing.

4.3.7 Sentiment-based Sarcasm Generator

The sentiment-based sarcasm generator aims to counter the sentiment expressed in the user input. We use a set of positive and negative expressions. In case the user input expresses sentiment without asking a question, a random response from the opposite polarity is selected. An example of response generated by the random sarcasm generator is '*Poor you!*' in response to a positive input (such as '*I am excited about my new job!*').

4.3.8 Random Sarcasm Generator

In case the user input does not express sentiment without asking a question, the random sarcasm generator is employed. It randomly picks a phrase each from a set of positive and negative reactions, and concatenates them together. An example of response generated by the random sarcasm generator is '*Wow! (positive reaction) *rolls eyes* (negative reaction)*'.

5. EVALUATION

This section describes the evaluation of *SarcasmBot*. We first describe our two evaluation experiments, and then discuss their results.

5.1 Experiment Details

Automatic evaluation of a sarcasm-generation module like ours is difficult for a variety of reasons:

1. For a given user input, there are multiple possible responses - both sarcastic as well as non-sarcastic. Hence, a gold response is difficult.

2. While coherence can be measured sufficiently using lexical overlap-based metrics or web as a resource, it is difficult to judge the sarcastic nature of the output.

We conduct a two-step evaluation to assess the output produced by SarcasmBot. The first experiment answers the question: '**Is the response grammatically correct, coherent and sarcastic in itself?**'. The second experiment answers the question: '**Is the response sarcastic enough so that a human can identify the output of SarcasmBot from a non-sarcasm-aware chatbot?**'.

The two experiments are conducted as follows:

1. **Experiment 1: Stand-alone[4] evaluation**: For a set of 31 user inputs, we obtain the sarcastic response from *SarcasmBot*. Three evaluators answer the following question for each output:

   (a) Coherence: Is the output a suitable response to the user input?

   (b) Grammatical correctness: Is the output grammatically valid?

   (c) Sarcastic nature: Is the output sarcastic?

2. **Experiment 2: Comparative evaluation**: For the same set of 31 user inputs, we obtain the output from *SarcasmBot*, as well as ALICE [17]. Four evaluators participate in this experiment. An evaluator is displayed the input statement, and two outputs. Their task is to identify which of the two outputs is from *SarcasmBot*. ALICE does not particularly lay emphasis on sarcasm while *SarcasmBot* does. If an evaluator correctly identifies the sarcasm-generation module, it means that *SarcasmBot* had created a sufficiently sarcastic statement. Specifically, the evaluators answer two questions:

   (a) Which of the two outputs is from *SarcasmBot*?

   (b) Was it difficult to make this judgment?

[3] http://en.wikipedia.org/wiki/Irrealis_mood

[4] This evaluation is '*stand-alone*' because the output is evaluated on its own. Experiment 2 compares our output with a regular chatbot.

All seven evaluators are engineering graduates and have studied for a minimum of 10 years with English as the primary language of academic instruction. The evaluation is blind: (a) No evaluator has any information about the **internals** of *SarcasmBot*, and (b) The evaluators for Experiments 1 and 2 are **different**.

Evaluation Parameter	Fleiss' Kappa
Coherence	0.164
Grammatical correctness	0.015
Sarcasm	0.335

Table 2: Multi-annotator agreement statistics for Experiment 1

5.2 Results

Fleiss' Kappa [5] (to measure multi-annotator agreement) for Experiment 1 are shown in Table 2. The value for sarcasm is 0.335. As stated above, all evaluators assigned 0 (No) or 1 (Yes) to each input-output pair. The average values for the three evaluation parameters are shown in Table 3. The average value for grammatical correctness is high but the corresponding Fleiss Kappa in Table 2 is low (*i.e.*, 0.015). This may be due to low variance because of which probability of chance agreement increases. As seen in Table 3, the evaluators assign an average value of 0.806 for sarcasm. All values are greater than 0.69 (0.698 for coherence, 0.903 for grammatical correctness, and 0.806 for sarcasm), indicating that the system performs well in terms of the three parameters.

Evaluation Parameter	Average
Coherence	0.698
Grammatical correctness	0.903
Sarcastic nature	0.806

Table 3: Average values for the three evaluation parameters resulting from Experiment 1

Evaluation Parameter	Fleiss' Kappa
Identification of *SarcasmBot* output	0.476
Difficulty	0.164

Table 4: Multi-annotator agreement statistics for Experiment 2

Strategy	Accuracy (%)
At least one evaluator is correct	87.09
Majority evaluators are correct	70.97
All evaluators are correct	61.29

Table 5: Percentage of statements in which the evaluators are able to correctly identify *SarcasmBot* output, in case of Experiment 2

Fleiss' Kappa for Experiment 2 are shown in Table 4. The value for identification of *SarcasmBot* output is 0.476, indicating moderate agreement. The corresponding value for difficulty is low: 0.164. This is expected because each evaluator may have their own perception of difficulty. Using the annotations obtained in Experiment 2, we now check how many evaluators correctly identify the outputs generated by *SarcasmBot*. These results are shown in Table 5. We consider three evaluation strategies: (a) at least one evaluator correctly identifies the output, (b) majority of the evaluators correctly identify the output, and (c) all evaluators correctly identify the output . The simplest scenario is (a) where even if one evaluator correctly identifies the SarcasmBot output, we mark it as correct. In this case, the accuracy is 87.09%. On the other end of the spectrum is the constrained case (c). Here, we observe that all evaluators correctly identify the output of *SarcasmBot* in 61.29% of sentences, indicating that *SarcasmBot* generates a reasonably sarcastic response. In case of 70.97% of sentences, majority of evaluators are correct (*i.e.*, (b)).

(a) **User input**: Why did Greg go home? **Sarcastic response**: Because Greg was in a mood for adventure.
(b) **User input**: What do you think of Greg? **Sarcastic response**: Well..I like Greg. The way I absolutely appreciate Zero accountability people.
(c) **User input**: Stop being such a b****! **Sarcastic response**: Unbelievable that you just said 'b****!'! You are really very classy!
(d) **User input**: What do you think of the new video game that James purchased? **Sarcastic response**: That was the best video game in the history of humankind!
(e) **User input**: Did James meet you? **Sarcastic response**: Hmm..I used to meet James if I felt stupid.
(f) **User input**: I won a lottery! **Sarcastic response**: I am so sorry for you!

Table 6: Example outputs of *SarcasmBot*

6. EXAMPLE OUTPUTS

Table 6 shows some example outputs from our *SarcasmBot* system. Each output was generated as follows: For **(a)**, the incongruent reason generator of SarcasmBot identified the named entity 'Greg'. Since the question was of 'Why' type, an incongruent reason was selected. Input in **(b)** is an opinion question. For this, a positive verb 'like' and a negative situation 'the way I absolutely appreciate zero accountability people' was selected - and the named entity 'Greg' was appropriately inserted. Input **(c)** consists of an offensive word. The offensive word generator identifies the offensive word, and places the word in a candidate output to generate the response. Hyperbole generator of the SarcasmBot produced the Output **(d)** where the noun entity '*video game*' was selected from the user input. Output (e) is not grammatically well-formed because the tense of the response also does not agree with the user input. Output (f) was generated by Sentiment-based sarcasm generator, which identified the sentiment of the user input as positive (due to the phrase '*won a lottery*'), and generated a response of opposite sentiment.

7. CONCLUSION & FUTURE WORK

We present *SarcasmBot*, a sarcasm-generation module that always responds sarcastically. Our sarcasm-generation module implements peculiar forms of sarcastic expressions such as hyperbole, incongruity and irrealis, in order to generate sarcastic responses to user input. The architecture consists of three stages: an input analyzer that extracts information about the user input, a generator selector that selects among sarcasm generators, and eight sarcasm generators that implement different sarcastic expressions. A sarcasm generator uses sarcastic patterns with placeholders for words. We evaluate our sarcasm-generation module through two manual experiments. Our evaluators assign average scores greater than 0.69 (out of 1) for the three parameters: coherence, grammatical correctness and sarcastic nature. In addition, we conduct a comparative evaluation where, in case of 70.97% of test cases, majority of our evaluators are able to correctly identify the output of *SarcasmBot* from among two candidate outputs.

As a future work, *SarcasmBot* may be integrated into a full-fledged chatbot to selectively decide when the sarcasm-generation module must be invoked. We also believe that our insight into sarcasm generation may help to improve techniques for automatic sarcasm detection.

8. REFERENCES

[1] F. Bessho, T. Harada, and Y. Kuniyoshi. Dialog system using real-time crowdsourcing and twitter large-scale corpus. In *Proceedings of the 13th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, SIGDIAL '12, pages 227–231, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[2] D. G. Bobrow, R. M. Kaplan, M. Kay, D. A. Norman, H. Thompson, and T. Winograd. Gus, a frame-driven dialog system. *Artificial intelligence*, 8(2):155–173, 1977.

[3] E. Camp. Sarcasm, pretense, and the semantics/pragmatics distinction*. *Noûs*, 46(4):587–634, 2012.

[4] J. D. Campbell and A. N. Katz. Are there necessary conditions for inducing a sense of sarcastic irony? *Discourse Processes*, 49(6):459–480, 2012.

[5] J. L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[6] S. L. Ivanko and P. M. Pexman. Context incongruity and irony processing. *Discourse Processes*, 35(3):241–279, 2003.

[7] D. J. Litman and S. Pan. Designing and evaluating an adaptive spoken dialogue system. *User Modeling and User-Adapted Interaction*, 12(2-3):111–137, 2002.

[8] D. J. Litman and S. Silliman. Itspoke: An intelligent tutoring spoken dialogue system. In *Demonstration papers at HLT-NAACL 2004*, pages 5–8. Association for Computational Linguistics, 2004.

[9] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[10] J. J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In *Proceedings of the 22nd international conference on World Wide Web*, pages 897–908. International World Wide Web Conferences Steering Committee, 2013.

[11] M. Newman, A. M. Turing, G. Jefferson, R. Braithwaite, and S. Shieber. Can automatic calculating machines be said to think? *S. Shieber, The Turing test: Verbal behavior as the hallmark of intelligence*, pages 117–132, 2004.

[12] J. W. Pennebaker, M. E. Francis, and R. J. Booth. Linguistic inquiry and word count: Liwc 2001. *Mahway: Lawrence Erlbaum Associates*, 71:2001, 2001.

[13] G. Pilato, A. Augello, G. Vassallo, and S. Gaglio. Ehebby: An evocative humorist chat-bot. *Mobile Information Systems*, 4(3):165–181, 2008.

[14] J. Schwoebel, S. Dews, E. Winner, and K. Srinivas. Obligatory processing of the literal meaning of ironic utterances: Further evidence. *Metaphor and Symbol*, 15(1-2):47–61, 2000.

[15] A. Stent, J. Dowding, J. M. Gawron, E. O. Bratt, and R. Moore. The commandtalk spoken dialogue system. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 183–190. Association for Computational Linguistics, 1999.

[16] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003.

[17] R. S. Wallace. *The anatomy of ALICE*. Springer, 2009.