Knock Knock Joke Generator

Reagan Middlebrook

***Abstract*—A total knock knock joke generation solution would require sophisticated machine understanding of context and relationships. To make the project approachable, this generator focuses on one very simple category of knock knock joke. A working solution was found, but a better one can be created given more, larger, and better-targeted corpora.**

I. Theory

IN PURSUING my project I found it useful to divide knock knock jokes into categories and examine what solutions would be necessary to generate each type. In the descriptions below, “teller” is the person telling the joke, “respondent” is the person answering back, and “guest name” is the answer to “Who’s there?”.

- **Name Pun:** The punchline is based on the guest name sounding like words in a common phrase
  Example: “Anita” “Anita who?” “Anita borrow a pencil”
- **Misheard:** The punchline is based on the joke teller mishearing the respondent’s reply
  Example: “Ash” “Ash who?” “Bless you”
- **Literal:** The punchline is based on the new word that is created by adding an “ooh” sound to the guest name
  Example: “Who” “Who who?” “Is there an owl in here?”
- **Premise-Based:** The punchline relies on the premise that someone is knocking on an actual door
  Example: “Lettuce” “Lettuce who?” “Lettuce in, it’s cold out here”
- **Joke-Based:** The punchline relies on acknowledgment that the teller is telling a joke
  Example: “Wooden shoe” “Wooden shoe who?” “Wooden shoe like to hear another joke?”
- **Insulting:** The punchline relies on insulting the respondent or tricking them into appearing foolish
  Example: “Cow says” “Cow says who?” “No, a cow says ’Moo’“
- **Meta:** The punchline is based on playing with or disrupting the expected knock knock joke form.
  Example: “Interrupting pirate” “Interrupting pir-” “Avast!”

There is some overlap amongst categories, with the more advanced ones incorporating the Name Pun or Misheard structure, but thinking of them as separate allows us to examine the unique challenges that each would involve. Several categories could be ruled out right away as beyond the scope of this project for relying on additional knowledge of the world which was too complex to attempt. Meta relies on an understanding of knock knock joke form and expectations; Premise-Based relies on an understanding of situations that would lead a person to be knocking on a door (“Mikey”/“Mikey doesn’t fit in this lock”, “Olive”/”Olive next door”); Insulting relies on an understanding of what consitutes an insult or what types of knowledge are expected to be general and the lack of which are then embarrassing (“Europe”/”No, you’re a poo!”, “Broccoli”/”Broccoli doesn’t have a last name!”).

The categories with the most promise for an initial undertaking seemed to be Literal or Name Puns. The advantage of Name Pun is that it does not require any real understanding of what it’s saying, but it does rely on the ability to match a “fuzzy” pronunciation. In the example “Needle”/”Needle little money for the movies”, “Needle” doesn’t sound like “Need a”, it just sounds similar. The advantage of Literal is that it does not rely on pronunciation, beyond the appended “oo” sound, but to make sense the punchline does requre larger conceptual knowledge (“What are circumstances in which that word or noise would be made?”).

II. Implementation

I chose to attempt an implementation of the Literal category of knock knock joke using the Python Natural Language Toolkit. At its most basic level (finding a punchline, regardless of punchline quality), this category uses mechanics (identify what word is being sought in the punchline, find that word in a sentence or phrase of the corpora, return it as a punchline) that can then be expanded to apply to Name Puns later on once pronunciation data is incorporated.

Since most knock knock jokes involve simple, childish humor and are directed at an audience of children under 10, I gathered corpora from children’s literature to get texts with similar simple sentence structures. Because knock knock jokes are constructed as a conversation I also wanted as much conversational English as possible, so included as many corpora of chat logs and overheard conversations as I could find.

To implement an algorithm for the Literal category, I created a list of words to seek. This was comprised of the guest name with ”who”, ”oo”, ”ew”, ”ough”, “eau”, and ”ue” appended. I searched my corpora for concordances with these new words, but initial testing led to very few “hits” for the search words. I want my generator to accept user input for the guest name and create a joke in response, so this is not the ideal solution or at least should not be used in isolation. My next attempt was to add synonyms for the new words (if any existed) and then search the corpora for hits on the synonyms as well. This did lead to more results though there was no guarantee they would make sense in the context of a joke.

An improved algorithm switched from using Concordance Indexes (which returns the word itself along with X number of words to either side to try and establish context) and began tokenizing the corpora by sentence so I could pull out the entire sentence where the search word appeared. This worked but slowed the search down considerably and still does not always yield good results, especially when particularly "literary"/expository sentences are pulled from the corpora.

III. Improvements for the Future

I plan to continue working on this project and already have some new directions I want to pursue. The next immediate step is to improve the corpora that I'm using. Literary responses can be ruled out as the time taken to search them is greater than the odds they will yield an appropriate result. For example, in response to a search for "awry" (as a synonym of "askew"), my algorithm returned "If a chair was out of its place, or a table turned awry, or a tool put down where it should not be, she could not bear to see it for a minute, but put all things straight again, so that nobody was at a loss where to find anything, She was called Order." It is safe to say this will never be a punchline. I intend to create a script that will amend my literary texts to retain only the dialogue and none of the exposition. In addition to refining the existing texts, adding more dialogue-based corpora such as scripts and transcripts should help increase the likelihood of hits that fit the direct speech pattern of knock knock joke punchlines. The best responses seem to come from corpora comprised of real speech (such as the "overheard" corpus, as in the response to a search for "cashew"/"Nah, man, cashew nuts make you retarded". It's not a nice response, but it does fit the knock knock joke structure well.).

The more data that can be added, the greater my ability to refine results. If I'm only rarely getting a single hit, I don't want to rule it out for being poor quality. If I can get several or many hits I can take more factors into consideration (including sentence length [shorter answers, up to a point, are likelier to be better than long answers], whether a person is being spoken to vs being spoken about [the punchline should be a reply to the respondent], etc.).

Once the algorithm and corpora are producing satisfactory results for Literary jokes, I will proceed to the inclusion of phoneme data to begin including Name Puns as well. Once pronunciation libraries are incorporated, the search structure created for parsing data to find Literal punchlines can be easily modified to search based on sound rather than on literal string matching.

In the far future, given the reasonably successful implementation of those two categories, I wish to pull a large set of training knock knock joke data and attempt to apply evolution with artificial neural networks to see if it can be taught on its own how to recognize a better vs. a worse punchline (with the end goal of someday spontaneously generating its own punchlines rather than pulling them from other texts).