

Knock, Knock, Neo!

Spawning Knock-Knock Jokes

Rachit Nimavat

Toyota Technological Institute at Chicago

Chicago, IL - 60637

nimavat@ttic.edu

Abstract

Of the very little research done on computational humor, even little has been done on humor generation - that too is limited to fill-a-phrase-in-the-blank kind of jokes and none of them even generate a “free sentence”. In this work, we focus on the task of generating jokes of a simple fixed structure: Knock-Knock jokes. Our approach is the combination of find-a-word-play and generate-a-punchline routines. Our preliminary results suggest that more relaxed approach along these lines should be capable of generating sufficiently good quality jokes of similar classes.

1. Introduction

For generating jokes, the first step should be to actually understand them. Out of many theories proposed, one of the most influential in linguistics is Semantic Script Theory of Humor (SSTH) introduced by Victor Raskin [14]. It suggests that jokes have multiple parallel semantic scripts and the final and concluding portion (called punchline) forces the audience to abruptly switch to an unlikely one and re-interpret the entire joke text, producing humorous effect [4].

Generating jokes with their context is a hard problem, so instead we focus on jokes with a fixed rigid structure. The Knock-Knock jokes is a call and response type of joke which first appeared in a newspaper column back in 1934 [13]. In it, the “response line” contains a pun on the “call line”.

Punster: Knock-Knock!

Recipient: Who’s there?

Punster: X (Isabelle)

Recipient: X who?

Punster: Punchline (Isabelle working or should I keep knocking?)

In our task of Knock-Knock jokes, punchline is the line where the visitor introduces itself. Based upon the type of joke, it could either be a line containing “X” (Marry – Marry me) or wordplays on “X” (Heaven – Heaven seen you in ages) or “X who” (Tank – Thank you for responding). The other type of popular punchlines are valid response to the question “X who?” (Yuno who? – Aveda Kedavra) and anti-jokes, but we don’t consider them as far as this task is concerned.

2. Related Work

There is much work done on computational detection and comprehension of humor. For instance, Zhang & Lui use specially handcrafted syntactic and semantic features to recognize humor in tweets [8]. Kiddon & Brun detect “That’s what she said” jokes by re-posing it as metaphor identification problem [1]. These jokes are double meaning jokes with a hidden sexual interpretation. They design features to model euphuisms for sexually explicit nouns and sentence-structure’s similarity with ones from erotic domain. The only work done on Knock-Knock jokes is their detection by looking up in a bigram table after letter-substitutions to find wordplays [12].

As per best of our knowledge, out of few works done on humor generation none focusses on “free sentence generation”. For example, Labutov and Lipson tried to directly implement SSTH where they find two paths in semantic embedding of phrases which maximize overlap and minimize congruency [9]. Pattrovic & Matthews generate words “X,Y,Z” for “I like my X, like I like my Y, Z” jokes by ranking (Y,Z) tuples according to a

custom designed similarity metric [10]. An approach by Binsted & Ritchie uses handcrafted set of relationships between lexemes to produce “crossing riddles” like sheep X kangaroo = woolly jumper [11].

3. Methods: Master Algorithm

To simplify the problem even further, we assume that the seed X is given. Assuming that, the main algorithm that we have used can be divided into following segments:

- $W \leftarrow$ wordplays of X
- For each element in $W \cup \{X\}$, generate *punchlines* containing it
- Rank these punchlines according to the *score* function
- Output a weighted random element from this set of punchlines, with the output-probability of each element proportional to its score.

In the subsequent sections, we dwell upon each of the steps in our master algorithm.

4. Methods: Generating Wordplays

Wordplay is a literary technique in which the sense of a word is twisted to showcase wit. There are several classes of wordplays of which, one of the most popular is pun. Pun is a form of word or phrase, which suggests multiple meaning from it.

One of the most general form of puns is similar-sounding words or near-homophones, known as *homophonic* pun. Another and probably more restrictive subclass of puns is called *homographic* puns, which exploits differences of pronunciation in words. For example, in the popular line of Douglas Adam: "You can tune a guitar, but you can't tuna fish. Unless of course, you play bass.". Here, "tuna" is a homophonic pun on "tune a" and "bass" is a homographic pun.

Because of our models for generating punchlines, we cannot distinguish between different senses of the same word. So we limit ourselves to homophonic puns as far as this task is concerned. We note that some other forms of wordplays, like *spoonerism* (in which places of some consonants, vowels or morphemes are switched) and telling-character names are also doable in the same context, but we don't consider them in this task. Another interesting wordplay is *double entendre* or double-meaning. The problem of identifying it is discussed in detail by Kiddon & Brun [1].

For generating homophonic puns, we select top results from the following two approaches:

4.1. Phonetic Algorithms

Phonetic algorithms are used for indexing of words based upon their pronunciations, usually by complicated and hand-crafted rules. One can view them as "locality sensitive hashing scheme" optimized for spelling variations across names. We use off the shelf implementation of a state-of-the-art phonetic algorithm called double-metaphone. Since it ignores vowels by design, we modified it to consider vowels and group similar sounding vowels into the same bucket.

Given X , while searching for its wordplay, we look into the bucket of X and neighboring buckets that can be obtained by a single change in hash ($\text{tree}(\text{TRI}) \rightarrow \text{true}(\text{TRA})$), or by flipping the aspiration marker on a single consonant ($\text{tree}(\text{TRI}) \rightarrow \text{three}(\text{THRI})$).

4.2. Translating into phoneme codes

In this more sophisticated way, we directly produce phonemes instead of hashes. For instance, the double-metaphone algorithm produces same hash – PAT for the words "put" and "but". But their IPA phoneme codes are “b'ʌt” and “p'ʊt” respectively. For computing IPA representation, we use *espeak* utility [2]. Note that, using a more standard *CMUdict* [3] would not be sufficient in our case since we are interested in non-standard words utterances and names. Similar to the above approach, while generating wordplay of given X , we look into the neighboring buckets by a single change in the phoneme code and by flipping the aspiration marker.

Note that no approach is more powerful than another. For example, $\text{see}(s'i:)$ and $\text{she}(f'i:)$ can be identified using phoneme codes, but they map to "SI" and "X" respectively in double metaphone. Also, "doorbell"(TARPAL) and "dumb-bell"(TAMPAL) can be identified using previous approach, but they map to “d'ɔ:bəl” and “d'ʌmɛl” respectively in phonetic representation.

In both these approaches, we use unigrams and bigrams (after ignoring the white-spaces) of training data for training purposes. Also, a nice thing about this approaches is that X need not be present in the training examples.

5. Method: Generating Punchlines

Because of the rigid structure of our joke, the context is a single line where the visitor says "X". Any sentence containing the wordplays of X as the main part has high chances of being funny in the punchline, since it will probably violate the anticipation of house-owner that it would contain the introduction of the visitor. To see why it is likely to be funny, consider the sentence: "Let us in". It is a simple sentence with "Let us" as the main part. In itself it is not funny, but with the context "Lettuce. Lettuce who? Let us in" is funny.

Fix Y which could either be X or a wordplay of X. Since Y could be a collection of at most two words, say $Y=Y_1 Y_2$. We use the following approaches for generating punchlines containing Y:

5.1. Lifting punchline directly from memory

We have scrapped a collection of ten thousand one-liners from reddit.com and textfiles.com. Then we list all the sentences containing either Y_1 or Y_2 or both. We use the following heuristic to rank these sentences:

- Highest score is given if Y_1 and Y_2 are near the ends of sentence.
- Next higher score is given if Y_1 and Y_2 appear together, possibly anywhere in the sentence.
- Next higher score is given if Y_1 and Y_2 occur separately.
- A penalty is given if Y_1 and Y_2 occur separately and one of them is too common, that is have unigram frequency more than a threshold. This includes common words like is, of, not etc.; which we think don't contribute to funniness when they occur separately.

We consider this score in final ranking of the punchlines described in the next section. Some of the interesting sentences with high scores found in this approach are:

- Doctor – I'm a doctor, not a tagline
- Love – Love thy neighbor, but not in public

Some sentences which doesn't make funny punchlines, but still scored high in the model are:

- Doctor - Doctor told me to stop drinking, I use a straw
- Love – It costs to love

5.2. Markov Model

Markov model is a stochastic model used in modelling of systems with Markov assumption: that the next state of the system depends upon a fixed number of previous states. Here, we assume that sentences are a random sequence of words in which the current word depends upon at most three previous words.

To model this, we construct unigram, bigram and trigram tables which assigns probabilistic weight to the next word given single, two or three of the context words respectively. We then randomly pick a word from the resulting distribution. In this model, for simplification the weights are nothing but their frequencies. For instance, in the bigram table, $P(w|w_1 w_2) \propto \text{Weight}(w|w_1 w_2) = \text{Frequency}(w_1 w_2 w)$ in the training data. To counter the bias of the more detailed context in higher order tables, we amplify the weights of bigram and trigram tables by constant factors.

In general, Markov models grows sentence starting with the given seed Y. But in our task, the wordplays Y could appear anywhere in the sentence. So we generate a "forward sentence" starting from the seed and then a "backward sentence" ending in seed. And with the same probability, do this process in opposite order. To this end, we use two versions of the above mentioned tables: normal tables and reverse tables (just by reversing the training text). Thus, our model produces more general sentences than the ones formed by fixing positions of the words of Y.

To make the model better, we have incorporated some tweaks. We consider sentences of length at least 4 and at most 12. This helps us to filter out not-so-well-formed sentences. We amplify the weights of bigram and trigram tables more by a factor if the context words are "stop" words: like and, but, of etc. This helps us in maintaining somewhat more consistency between the segments of the sentence.

Some of the interesting sentences generated (which were not present in the training) by this model are:

- Police – The police men who are giving away someone else's cash
- Doorbell – I rang the doorbell but I guess you don't want me
- Door is – The door is jammed and I don't intend to hurt it

- Scold – It's cold war 3

Unlike the previous approach, it also produces many nonsensical sentences like:

- Police – Excellent police she delivered you go to him
- It's cold – It's cold we done something

Though we believe that using more data and longer context, these nonsensical sentences can more or less be dealt with.

5.3. Char-RNN

The principal issue with the above two approaches is that many of the sentence they produce do not utilize the rigid context present in Knock-Knock jokes: Of an *unknown* visitor knocking at the door saying it is X. So inspired by Andrej Karpathy's experiments with character-recurrent neural networks, we trained a model of char-RNN for this task [5].

We trained an LSTM with 2 hidden layers of 512 nodes each with dropout rate of 0.5 and truncated backpropagation through time of length 50 characters. The result produced by our model make many grammatical errors and the sentences digress wildly more often than not. Nevertheless, we believe that this may be due to less training data (about 4MB in total) and less training time (around 1 day on a CPU). For training data apart from the jokes scrapped from the internet, we used several humorous texts with contemporary English found on Project Gutenberg [6].

While sampling from the resulting char-RNN, we start with the seed of the form "Knock. Door is closed. Y who? Y ". Here our punchline is constrained to be starting with Y. Some of the more sensible (though not necessarily funny) punchlines produced by this model are:

- Knock. Door is closed. Doctor who? Doctor don't fuck your friends by 4 feet boobs.
- Knock. Door is closed. Let us who? Let us go away.
- Knock. Door is closed. It's cold who? It's cold, cool like curse of nude god.

But mostly the output is gibberish like:

- Knock. Door is closed. Doctor who? Doctor texts more way of electrical exfires one day and deposit to bad action.

6. Methods: Ranking Punchlines

Apart from the inbuilt scores given when generating punchlines, we assign additional penalty proportional to its length and presence of “rare” words in it (to discourage spelling mistakes and slangy language). We also give additional scores for conversational words like “you”, “I” etc. We penalize single letter end-words to prevent “u.s”, “a.m” etc. from being understood as end of sentences. Finally, we output the weighted random punchline and polish its grammar using the “language_check” utility [7]. We note that the ranking procedure is rather primitive and much can be done to improve it.

7. Conclusions

Our approach of generating wordplays finds most of the homophonic puns found in human generated Knock-Knock jokes. Our approach of modified Markov model performs better in terms of producing original punchlines. We believe that it can be improved by using bigger corpus of “conversational” one-liners, but we see no obvious solution of incorporating the context in such models. Seeing the recent success of @DeepDrumpf and general trend towards neural networks, it may be reasonable to hope more from char-RNN [15].

	Hu-man	5.1. Best	5.1.Ra ndom	5.2. Best	5.2.Ra ndom
Good Joke	67	60	27	60	27
Non-Joke	13	33	47	27	47
Gibberish	0	7	27	13	27

*A preliminary blind-evaluation by 5 friends [16]

8. Future Work

The SSTH implementation of [9] which tries to find two semantic concepts with high similarity but low congruency, might be helpful in finding good quality seeds “X” for our model. Instead of focusing on neighboring buckets for finding wordplays, it may be useful to explore more with gradually decreasing score and consider that score while ranking punchlines. The approach of generating punchlines may benefit a lot by borrowing machinery from dialogue systems. Also, as apparent from preliminary results here, this kind of approach may be useful in generating such restricted domain jokes from different classes.

Reference

- [1] Chloe Kiddon & Yuriy Brun. That's what she said: Double entendre identification. ACL (Short Papers) 2011: 89-94
- [2] Espeak utility for Linux. Stable release: 1.48.04 / April 6, 2014. (<http://espeak.sourceforge.net>)
- [3] Kevin Lenzo. CMU Dictionary version 0.7b. (<http://www.speech.cs.cmu.edu/cgi-bin/cmudict>)
- [4] Carrell, Amy. Raskin, Victor, ed. "Primer of Humor Research: Humor Research 8". Berlin, New York: Mouton de Gruyter: 2008: 303–332.
- [5] Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks 2015. (<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>)
- [6] Michael Hart. Project Gutenberg. (https://www.gutenberg.org/wiki/Humor_%28Bookshelf%29)
- [7] language_check (<https://pypi.python.org/pypi/language-check>) is a fork of language_tool utility by spirit (https://bitbucket.org/spirit/language_tool)
- [8] Renxian Zhang and Naishi Liu. Recognizing humor on twitter. Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. 2014: 889-898
- [9] Igor Labutov and Hod Lipson. Humor as circuits in semantic networks. 2012.
- [10] Sasa Petrovic & David Matthews. Unsupervised Joke Generation from Big Data. 2013.
- [11] Kin Binstead & Graeme Ritchie. An Implemented Model of Punning Riddles. 1994.
- [12] Julia Taylor & Lawrence Mazlack. Humorous Wordplay Recognition. 2004.
- [13] "Hee Haw News". In Rolfe Arrow Iowa edition. P4. Sep 10, 1934.
- [14] Victor Raskin. Semantic Mechanisms of Humor. 1985: 302
- [15] Brad Hayes. Twitter handle: @DeepDrumpf
- [16] A poll on google forms. (<http://bit.do/knock-knock>)