Puntective - A System and Framework for Pun and Wordplay Detection

Cody Hanson
University of Colorado Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, Colorado USA 80918
chanson@uccs.edu

Abstract

Puns and wordplay are easy for humans to comprehend, but difficult for computer programs. With the right algorithm, perhaps computers could mimic the way that humans can make connections between words, sounds, the current social setting, all while drawing from deep knowledge bases. This research proposes a set of heuristics, techniques, and enabling data structures for allowing a computer to assign a 'pun confidence score' to a piece of text, in a way that is extensible for future research.

Introduction

Written language is one of the crowning achievements of humans, and one of our most sophisticated and complex tools. An important part of this medium is that of humor and jokes. Humorous writing works to bring special joy to humans, and we are able to craft nuanced and smart jabs that require much thought to appreciate. What would it take to teach a computer the concept of 'humor'? How about just identifying its presence in written or spoken word? An especially complex form of humor is the 'Pun'. A pun is often referred to as a 'play on words' , and while easy for most humans to recognize, present a challenge to computers and the field of natural language processing,

In this paper we propose a system of heuristics and processing that attempts to recognize whether a piece of text constitutes a pun. By using techniques of natural language processing and heuristics for finding common properties of puns we will assign a 'confidence score' which will attempt to quantify the likelihood that the piece of text is a pun. Equally interesting is the estimation by a computer that a piece of text *doesn't* have any humorous content. Ideally a good algorithm will be designed to handle both of these cases.

We also propose a technique for cataloguing results of various heuristics in a Graph data structure, that encodes relationships that illuminate humor in the edges and nodes. An algorithm will then be explained as to how to make use of this extensible graph structure.

Related Work

(Stamatatos, Fakotakis, and Kokkinakis 2000) discusses a technique for classifying the 'type' of publication that some text is from, depending on the word content and frequency. This approach was done with discriminant analysis, and a 'training' dataset. It was not limited to humorous text.

In (Kao, Levy, and Goodman), a computational model for humor is presented, dealing specifically with homophone puns. The authors limited themselves to this subset in order to narrow the problem scope. This work shows that it is possible to 'quantify' something like word play, and should serve as a great knowledge base for my own research. These authors used the 'Bag of Words' approach when doing their analysis.

(Ritchie 2005) presents a method for pun generation using computers, as well as examines the 'essence' and structure of different kinds of puns.

A chat bot for Yahoo Messenger is presented in (Augello et al.), and one of the components is the ability to detect whether the human the bot is chatting with has said something humorous or not, so that the avatar can change and respond appropriately. The techniques used to identify humor in this case were looking for alliteration, antinomy (a contradiction between two beliefs or conclusions that are in themselves reasonable; a paradox.), and 'adult slang' (presence of sexual words). These techniques were originally presented in (Mihalcea and Strapparava 2006). In both of these works, input was limited to 'one liners', short sentences that contain the entire joke. They also have a 'learning' element to their algorithm, which needs a training dataset, and uses some Bayesian probability techniques.

Proposed Methodology and Heuristics

The first step will be to read in the words of the text under examination. The first naive approach will to just count the frequency of each word in a 'bag of words' approach. This will be simple to implement initially, and won't require deep knowledge of natural language processing (NLP) algorithms that would be required to parse the sentence into a tree-like structure. Once the bag of words is constructed we can apply several proposed heuristics.

If the word frequency counting proves to not be powerful enough to capture the essence of the text, NLP algorithms such as those demonstrated by (Socher et al. 2013) will be researched in order to augment the approach. This would help to determine the meaning of a word, which in the english language is often determined by the context within a sentence.

Odd Hyphenations

A characteristic of puns that the authors have noticed is that they can sometimes contain 'odd' hyphenations, or more precisely, hyphenations of words that are not commonly seen.

Atheism is a non-prophet organization. A bicycle can't stand on its own because it is two-tired.

Figure 1: Puns with Odd Hyphenations

In Figure 1, 'non-prophet' (rather than 'non-profit') and 'two-tired' (rather than 'too-tired') are hyphenations which most humans would deem as atypical. My idea would be to look for hyphenated phrases in a dictionary, or other datastore of figures of speech, and if the phrase isn't found, then we can perhaps make the assumption that it is a form of wordplay.

Homophones

Puns often of homophones in them. A homophone is defined as being each of two or more words having the same pronunciation but different meanings, origins, or spelling, e.g., new and knew

What did the grape say when it got stepped on? Nothing - but it let out a little whine.

Figure 2: Pun with Homophones

In Figure 2, the wordplay using homophones is 'whine' which is a play on 'wine' which is related to 'grape'. By looking for homophones of 'whine', we would be led to 'wine', which we could then make a linkage to grapes and winemaking, using a dictionary or encyclopedia.

Dictionary Definitions

I used to be addicted to soap, but I'm clean now.

Figure 3: Pun with terms linked by dictionary definitions

In Figure 3, we can see some key words, 'addicted', 'clean' and 'soap' have the connections between them that constitute a pun. For this example, we would process 'addicted', and possibly find references to 'clean', and the dictionary definition of 'soap' could also have references to 'clean'. When we see that 'clean' has different definitions, with rather disparate meaning, we may be able to glean that there is a pun present.

One letter mutations

Did you see the movie about the hot dog? It was an Oscar Wiener.

Figure 4: Pun with relevant 1 letter mutation

In Figure 4 it can be seen that 'Wiener' is one letter removed from being the word 'Winner', which is an appropriate reference to movies and the Oscar's awards show. By looking for other words which have a single letter different, it can help us to make linkages to the other parts of the sentence, using our other techniques. This could be efficiently computed using regular expressions, and a simple list of english words.

Synthesized speech with dictionary search

The roundest knight at king Arthur's round table was Sir Cumference.

Figure 5: Pun with audible word play

In Figure 5, the word play of the phrase comes out when it is spoken. In this case, the Knight's name 'Sir Cumference' becomes a homophone with 'circumference'. A stretch goal for this research could be to process text with text to speech software, and then back again from the generated audio, back into text. If the transformed text differs from the input text, it could be indicative of audible word play. Of course this technique has its difficulties, and depends upon the quality of the text to speech, and speech to text synthesizers.

A Reuseable Graph Data Structure

An hypothesis that we propose is that a graph data structure could be the best way for a computer program to catalogue and make use of connections between words in humorous text. As humans automatically maintain a mapping of incongruities, relationships, and connections between parts of a sentence, a graph could be constructed to represent the state of the analysis that a computer makes against a piece of text.

The graph would have different types of nodes and edges, depending on what is being represented. For example, there would be a class of nodes that represent elements of the text under analysis, and there would also be nodes that represent 'commonalities' between parts of the text. An example of a commonality could be a related subject between two words, such as 'sports' between the elements 'ball' and 'score'. Another type of node could be 'parts of speech' information, which would allow different analytic modules to categorize parts of the sentence as verbs, adjectives, nouns, or as being part of a figure of speech. Adding this extra information could be required for more advanced humor detecting heuristics.

While in normal graphs, the edges don't necessarily carry much data of their own, the edges between nodes in this system are important because they define many different types of relationships between words. Some examples of different types of edges could be 'homophone' edges denoting that two nodes are homophones of each other, or a 'figure of speech' edge which denotes that several nodes belong to another 'figure of speech' node, denoting an idiomatic use of language. See Figure 6 for an example of a Pun Graph that has been decorated by various heuristics.

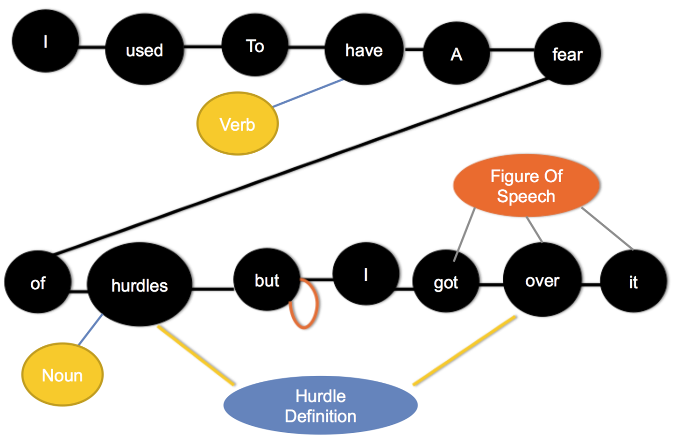

Figure 6: A Decorated Pun Graph

Generalized scoring

Because all edges and nodes inherit from the base classes that I will design, it makes the implementation of summing the contributions of all heuristics easy. To illustrate the general idea of how the system will use the graph, please refer to Algorithm 1. The list of heuristics to be ran could potentially be a dynamically generated set, and the number of rounds of analysis could be bound by a timer or other upper limit, to ensure that the computation happens within a timely manner, or perhaps continues longer to become more rigorous.

Motivation

By building an extensible and easy to use framework, the door will be opened for further research by other scholars to apply their new and novel heuristics to further decorate the graph. The graph data structure is also totally heuristic agnostic, and therefore doesn't limit what other researchers could try.

Tools and Datasets

The language of choice for implementation is Python, due to its rich set of libraries, object oriented design support, and ability to create rapid prototypes. It is also well regarded in the scientific community, and easy to learn to use, without the intricacies of learning how to use a C or Java compiler. There is also an powerful library for Python called the Natural Language Toolkit (Project 2014). This provides easy access to libraries that can be used for parsing sentences and building heuristics. It also provides an easy to use interface to Wordnet (Miller 1995).

```
Data: Input $text$
Data: Pun Graph $G$
Result: Pun Score for the text, cumulatively assigned
        by all heuristics
Initialize graph $G$ with the Input $text$;
Initialize $score = 0$;
while there are heuristics left to run do
    run heuristic on graph $G$;
    update graph $G$ with additional nodes and edges;
    next heuristic;
end
for $node \in G$ do
    $score = score +$ contribution of $node$;
end
for $edge \in G$ do
    $score = score +$ contribution of $edge$;
end
output $score$;
```

Algorithm 1: Algorithm for integrating the Pun Graph with arbitrary heuristics

The input data for my development tasks is a small set of puns found on the internet, as well as a set of sentences that are not considered humorous. Due to the small size of the dataset, it is primarily intended for use as development test cases, rather than an exhaustive study or benchmark of this system.

Current Progress

So far I have designed and outlined the classes for the Pun Graph data structure, and built the simple 'analysis engine' that will be used to decorate the graph. I have implemented one Pun heuristic which I will use for initial tests of the data structure, the 'homophone' heuristic. This heuristic posits that the presence of homophones makes the text more likely to be a pun.

I have designed two easy ways to run the program. One way is the 'automated batch input' method, which reads data from a set of files and automatically does the processing. The other method is an interactive prompt which allows the user to input arbitrary text to test the algorithms on.

Evaluation

In order to judge the effectiveness of the system, we will compare the results of the pun classification to that of a human expert. We will build a dataset that contains a mixture of text snippets that contain puns, and some that do not. Also, because not all puns are created equal, we will seek out puns of varying complexity (as judged by the number of 'connections' a human can make) to attempt to highlight limitations of our techniques. To ensure that the evaluation is fair, the texts in the dataset will be selected with as much randomness as possible, from a pool of candidates, and the algorithms will not be tailored in any way to a specific piece of text.

It will be important to track statistics on success rates, including false positives. False positives will indicate that the techniques are not refined enough, or are going down a path of incorrect analysis.

Speed of execution will not be a main goal or metric for this research. It is hopeful that search techniques will take at most on the order of a few minutes, but if the complexity of the analysis become of too high, those techniques will need to be rethought.

Another important piece of evaluation, although more subjective, will be to what degree the Pun Graph framework will be easily used and extended by other researchers. The ideal use case will be for the researcher to extend one of the base or derived classes for edges and nodes to specifically have the properties that they wish to explore (perhaps different scoring, or a new type of relationship).

Future Work

An interesting and valuable addition could be the rendering of the Pun Graph into an interactive image, where nodes and edges could be visualized, explored, and modified. The Graph could be output to a JSON format, which could be suitable for consumption by a graphing library such as D3 (Bostock 2014).

Another interesting technique could be to automatically transcribe vocalized words into text, and do real time analysis on the fly. This may or may not be feasible depending on processing latency and number of heuristics being applied.

Finally, a ‘training database’ of previously constructed pun graphs and their resulting scores could be useful in an advanced learning algorithm. By storing many samples of already processed humor, including confidence scores and relationships encoded in the Pun Graph, a learning agent could draw upon this data to more accurately predict if a new piece of text is humorous, and which other jokes and puns it is similar to.

Conclusion

This research topic takes on a difficult problem, teaching computers about humor. Due to the limited time available for the work, the authors will strive to make meaningful progress for a manageable subset of the problem. This includes not exhaustively searching for breakthrough heuristics, but instead focusing on building a framework that can be easily leveraged by other researchers.

References

Augello, A.; Saccone, G.; Gaglio, S.; and Pilato, G. Humorist bot: Bringing computational humour in a chat-bot system. In *CISIS*, 703–708.

Bostock, M. 2014. Data driven documents. `http://d3js.org/`.

Kao, J. T.; Levy, R.; and Goodman, N. D. The funny thing about incongruity: A computational model of humor in puns.

Mihalcea, R., and Strapparava, C. 2006. Learning to laugh (automatically): Computational models for humor recognition. *Computational Intelligence* 22(2):126–142.

Miller, G. A. 1995. Wordnet: A lexical database for english. *COMMUNICATIONS OF THE ACM* 38:39–41.

Project, N. 2014. Nltk.org. `http://www.nltk.org/`.

Ritchie, G. 2005. Computational mechanisms for pun generation. In *Proceedings of the 10th European Natural Language Generation Workshop*, 125–132.

Socher, R.; Bauer, J.; Manning, C. D.; and Ng, A. Y. 2013. Parsing With Compositional Vector Grammars. In *ACL*.

Stamatatos, E.; Fakotakis, N.; and Kokkinakis, G. 2000. Text genre detection using common word frequencies. In *Proceedings of the 18th Conference on Computational Linguistics - Volume 2*, COLING ’00, 808–814. Stroudsburg, PA, USA: Association for Computational Linguistics.