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, using a graph database 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.

Methodology and Heuristics

They analysis process must first populate the pun graph with the phrase under test. The phrase is read in, and split on whitespace, with punctuation marks such as commas and periods being stripped. Then 'word' nodes are created with 'NEXT' edges between them. This is the skeleton, and core of the graph.

Next a series of heuristics are applied, each having the pun graph at their disposal. In this implementation, the order of execution of heuristics is important, as the ones that come later have more nodes and edges to consider, and we

can build up complexity as we go. It makes sense to execute more elemental analyses first (such as labeling parts of speech), since they don't depend on anything from other heuristics.

The following sections describe some ideas for heuristics, some of which were implemented and tested in the results section.

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.

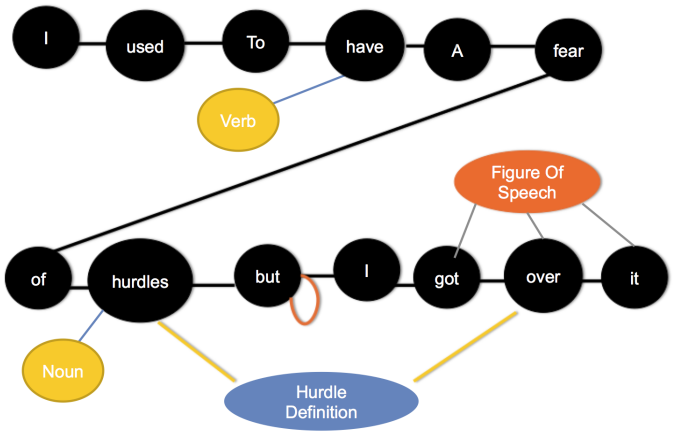

Figure 6: Concept - A Decorated Pun Graph

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

The idea of a graph to encode relationships between words and pieces of knowledge seems like an easy to grasp concept that has limitless potential for further research and extensibility. Any research could add their own types of data to a graph, and learn to take advantage of the encoded information in new and exciting ways.

Tools

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 2014b). This provides easy

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

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).

For the graph data structure, rather than build my own graph implementation, I found that using an existing product called Neo4J to be advantageous. Neo4J (Project 2014a) is a graph database that provides many benefits of a traditional SQL database, while being designed to encode data in nodes and edges, rather than tables. Neo4J provides a SQL-like query language called Cypher. This makes it easy to query for a subset of nodes and edges, and easily add to a graph. Neo4J is easily consumable via many programming languages, including Python.

Datasets

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.

The format of the data is JSON, so that it can be easily parsed by the analysis programs. JSON also makes it easy to add metadata to information, such as a list of tags that encode information about the phrase, as well as other properties that could be helpful. This is a more elegant technique than using a comma separated value file, and is less likely to degrade over time because of the use of key value pairs.

Results

The results of my work have two facets. The first being the degree that the system as it is currently implemented can classify phrases as Puns, and the second being the output of a framework that can be leveraged in future work and applying a new type of analysis to this problem. As I had

stated in midterm paper, the goal of the project wasn't to make the best classifier possible, but to explore this graph data structure and to see how some simple heuristics fit into it.

Qualitiative Results

One of the important accomplishments made was enabling the visualization of these pun graphs. By combining D3.js and the Graffeine project (Browne 2014), I was able to add a presentation layer above the Neo4J database that held the output graph of the heuristics applied by the analysis program. In Figure 7, a screenshot of the console can be seen. Another important capability of this graphical console is the ability to edit the nodes and edges of the graph. This would enable researchers and heuristic developers to manually edit and tweak the state of the graph to see how it affects their algorithms.

Because Neo4J can scale to handle large graphs, as the amount of analysis increases with more heuristics, the technique should scale with smart heuristic implementation. What is meant by smart implementation, is that heuristics leverage the query language of Neo4J to return only a subset of nodes and edges to process, instead of having to process the entire graph. This would not be easily possible for graphs that would be required to be held in-memory, as they could potentially exceed the working space of a machine, and don't have a rich built in query language.

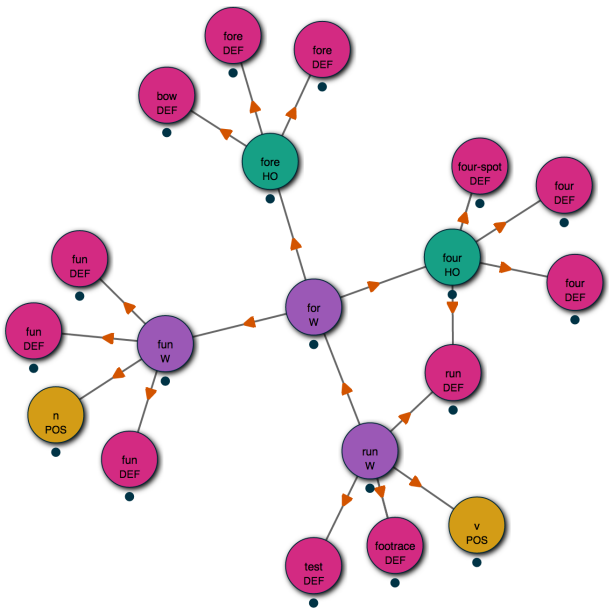

Figure 7: Screenshot of Puntective Implementation

Quantitative Results

In an attempt to quantify the results my current implementation obtained I ran the set of heuristics against my dataset, and averaged the scores across both positive and negative examples. For the purpose of this test, I assigned the value of a 'definition linkage' as 10 points, and the presence of a homophone is worth 5 points. We can see from the table of aggregate results that the heuristics I implemented are not quite enough to determine if something is a pun or not. This is due in part to their simplicity, and lack of depth. Going back to a qualitative approach for a moment, In some examples I tested, the 'definition-linkage' heuristic did connect words in the graph together that showed humor connections (if a human were building a graph for the same phrase).

'funniness score' for pun and non-pun examples.

Type of Phrase	Average	Median	Max	Min
Pun	80.3	65.0	185	15
Non-Pun	135.0	105.0	515	0

In this case, I didn't attempt to determine where the 'threshold' for the ascension from normal phrase to pun, since the techniques are still very unrefined, and the dataset is not large enough to be a representative sample of many kinds of phrases.

See Appendix A for detailed results.

Future Work

As can be seen by the inconclusive results of the simple heuristics that were implemented, additional and more complex heuristics will be required to be implemented to truly reveal the nature of humor in text, as recognized by humans. In order to be successful in this field, I think that computer science researches need to team up with those in the linguistics field, or be suitably well disciplined in both fields. Because words cannot often be looked at in isolation, advanced natural language processing techniques and algorithms that take into account the context of a word will be important to reliably classify phrases. Other heuristics that could be interesting to implement include more in depth definition comparisons, and 'figure of speech' analysis. 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.

The acquisition of a larger dataset of puns and non-puns will also be required for future work. If possible an enriched dataset that includes metadata, such as the 'setting' where the phrase was spoken, or other cultural cues which are important to the human understanding of humor.

Conclusion

This research topic took on a difficult problem, teaching computers about humor. While the problem may not have been solved by this attempt alone, the authors believe that using a graph topology to encode humor information has potential to be taken forward by future work. We also were able to show that there is some threshold of analysis that must be crossed in order to make these sophisticated classifications, and a few simple heuristics were not enough to be able to distinguish a pun from a non-pun. Due to the open source nature of the work I have done, other researchers will be able to pick up where I have left off.

References

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

Browne, J. 2014. D3 neo4j console. `https://github.com/julianbrowne/graffeine`.

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. 2014a. Neo4j graph database. `http://www.neo4j.org/`.

Project, N. 2014b. 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.

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.

Appendix A - Detailed test data and output

Input Dataset

```
{
  "phrases":[
    {"pun": true, "text":"Did you hear about the guy whose whole left side was cut off? He's
        all right now."},
    {"pun": true, "text":"I'm reading a book about anti-gravity. It's impossible to put down
        ."},
    {"pun": true, "text":"I'm glad I know sign language, it's pretty handy."},
    {"pun": true, "text":"Atheism is a non-prophet organization."},
    {"pun": true, "text":"Don't trust people that do acupuncture, they're back stabbers."},
    {"pun": true, "text":"A new type of broom came out, it is sweeping the nation."},
    {"pun": true, "text":"There was once a cross-eyed teacher who couldn't control his pupils
        ."},
    {"pun": true, "text":"The shoemaker did not deny his apprentice anything he needed. He
        gave his awl."},
    {"pun": true, "text":"The one who invented the door knocker got a No-bell prize."},
    {"pun": true, "text":"I relish the fact that you've mustard the strength to ketchup to me
        ."},
    {"pun": false, "text":"These pretzels are making me thirsty"},
    {"pun": false, "text":"What me worry?"},
    {"pun": false, "text":"To be, or not to be, that is the question."},
    {"pun": false, "text":"Do you guys want to go get some dinner? I am starving."},
    {"pun": false, "text":"Why did the chicken cross the road? To get to the other side."},
    {"pun": false, "text":"Ask not what your country can do for you, but what you can do for
        your country"},
    {"pun": false, "text":"That's one small step for man, one giant leap for mankind"},
    {"pun": false, "text":"Houston, we have a problem"},
    {"pun": false, "text":"The wagon is red, and I like to ride around in it."},
    {"pun": false, "text":"To infinity, and beyond"},
    {"pun": false, "text":"Have you called your grandmother lately? I bet she misses you."}
  ]
}
```

Test Output

```
Phrase (Pun?: True):185 points - Did you hear about the guy whose whole left side was cut off
    ? He's all right now.
Phrase (Pun?: True):64 points - I'm reading a book about anti-gravity. It's impossible to put
     down.
Phrase (Pun?: True):65 points - I'm glad I know sign language, it's pretty handy.
Phrase (Pun?: True):15 points - Atheism is a non-prophet organization.
Phrase (Pun?: True):115 points - Don't trust people that do acupuncture, they're back
    stabbers.
Phrase (Pun?: True):45 points - A new type of broom came out, it is sweeping the nation.
Phrase (Pun?: True):21 points - There was once a cross-eyed teacher who couldn't control his
    pupils.
Phrase (Pun?: True):65 points - The shoemaker did not deny his apprentice anything he needed.
     He gave his awl.
Phrase (Pun?: True):68 points - The one who invented the door knocker got a No-bell prize.
Phrase (Pun?: True):160 points - I relish the fact that you've mustard the strength to
    ketchup to me.
Phrase (Pun?: False):85 points - These pretzels are making me thirsty
Phrase (Pun?: False):20 points - What me worry?
Phrase (Pun?: False):160 points - To be, or not to be, that is the question.
Phrase (Pun?: False):155 points - Do you guys want to go get some dinner? I am starving.
Phrase (Pun?: False):105 points - Why did the chicken cross the road? To get to the other
    side.
Phrase (Pun?: False):515 points - Ask not what your country can do for you, but what you can
    do for your country
Phrase (Pun?: False):150 points - That's one small step for man, one giant leap for mankind
Phrase (Pun?: False):35 points - Houston, we have a problem
```

```
Phrase (Pun?: False):100 points - The wagon is red, and I like to ride around in it.
Phrase (Pun?: False):0 points - To infinity, and beyond
Phrase (Pun?: False):160 points - Have you called your grandmother lately? I bet she misses
    you.
Pun average:80.3
Pun median:65.0
Pun max:185
Pun min:15
Non-Pun average:135.0
Non-Pun median:105.0
Non-Pun max:515
Non-Pun min:0
```