Humorous Wordplay Recognition*

Julia M. Taylor
ECECS Department
University of Cincinnati
Cincinnati, OH, USA
tayloj8@email.uc.edu

Lawrence J. Mazlack
ECECS Department
University of Cincinnati
Cincinnati, OH, USA
mazlack@uc.edu

Abstract – *Computationally recognizing humor is an aspect of natural language understanding. Although there appears to be no complete computational model for recognizing verbal humor, it may be possible to recognize jokes based on statistical language recognition techniques. Computational humor recognition was investigated. A restricted set of all possible jokes that have wordplay as a component was considered. The limited domain of "Knock Knock" jokes was examined. A created wordplay generator produces an utterance that is similar in pronunciation to a given word, and the wordplay recognizer determines if the utterance is valid. Once a possible wordplay is discovered, a joke recognizer determines if a found wordplay transforms the text into a joke.*

Keywords: computational humor, jokes, statistical language recognition.

1 Introduction

Investigators from the ancient time of Aristotle and Plato to the present day have strived to discover and define humor origins. There are almost as many humor definitions as humor theories [5].

Humor is interesting to study not only because it is difficult to define, but also because sense of humor varies from person to person. The same person may find something funny one day, but not the next, depending on the person's mood, or what has happened to him or her recently. These factors, among many others, make humor recognition challenging.

Although most people are unaware of the complex steps involved in humor recognition, a computational humor recognizer has to consider all the steps in order to approach the same ability as a human being.

Natural language understanding focuses on verbal structure. A common form of humor is verbal humor. Verbal humor can involve reading and understanding texts. While understating the meaning of a text may be difficult for a computer, reading it is not.

One of the subclasses of verbal humor is the joke. Hetzron [2] defines a joke as "a short humorous piece of literature in which the funniness culminates in the final sentence." Most researchers agree that jokes can be broken into two parts: a setup and a punchline. The setup is the first part of the joke. It usually establishes certain expectations, and consists of most of the text. The punchline is a much shorter portion of the joke. It causes some form of conflict. The punchline can force another text interpretation, violate an expectation, or both [7]. Shorter texts are easier to analyze. As most jokes are relatively short, it may be possible to recognize them computationally.

Raskin's Semantic Theory of Humor [6] has strongly influenced the study of verbal humor, and jokes in particular. The theory is based on assumption that every joke is compatible with two scripts, and those two scripts oppose each other in some part of the text, usually in the punch line; therefore generating humorous effect.

Computational joke recognition or generation may be possible, but it is not easy. An "intelligent" joke recognizer requires world knowledge to "understand" most jokes. A joke recognizer and a joke generator require different natural language capabilities.

There have been very few attempts at computationally understanding humor. This may be partly due to the absence of a theory that can be expressed as an unambiguous computational algorithm. Similarly, there have only been a few computation humor generators; and, fewer still have been theory-based.

2 Wordplay Jokes

Wordplay jokes, or jokes involving verbal play, are a class of jokes that depend on words that are similar in sound, but are used in two different meanings. The difference between the two meanings creates a conflict or breaks expectation, and is humorous. The wordplay can be created between: two words with the same pronunciation and spelling, with two words with different

* **0-7803-8566-7/04/$20.00 © 2004 IEEE.**

spelling but the same pronunciation, and with two words with different spelling and similar pronunciation. For example, in $Joke_1$ the conflict is created because the word *toast* has two meanings, while the pronunciation and the spelling stay the same. In $Joke_2$ the wordplay is between words that sound nearly alike.

$Joke_1$: "Cliford: The Postmaster General will be making the *toast*.
Woody: Wow, imagine a person like that helping out in the kitchen!"

$Joke_2$: "Diane: I want to go to Tibet on our honeymoon.
Sam: Of course, we will go to bed."[1]

2.1 Knock Knock Jokes

A focused form of wordplay jokes is the Knock Knock joke. In Knock Knock jokes, wordplay produces humor. The structure of the Knock Knock joke provides pointers to the wordplay.

A typical Knock Knock (KK) joke is a dialog that uses wordplay in the punchline. Recognizing humor in a KK joke arises from recognizing the wordplay. A KK joke can be summarized using the following structure:

$Line_1$: "Knock, Knock"
$Line_2$: "Who's there?"
$Line_3$: any phrase
$Line_4$: $Line_3$ followed by "who?"
$Line_5$: One or several sentences containing one of the following:
- $Type_1$: $Line_3$
- $Type_2$: A wordplay on $Line_3$
- $Type_3$: A meaningful response to $Line_4$.

$Type_1$, $Type_2$, and $Type_3$ are types of KK jokes. $Joke_3$ is an example of $Type_1$, $Joke_4$ is an example of $Type_2$, and $Joke_5$ is an example of $Type_3$.

$Joke_3$: Knock, Knock
Who's there?
Water
Water who?
Water you doing tonight?

$Joke_4$: Knock, Knock
Who's there?
Ashley
Ashley who?
Actually, I don't know.

$Joke_5$: Knock, Knock
Who's there?
Tank
Tank who?
You are welcome.[2]

From theoretical points of view, KK jokes are jokes because $Line_3$ and $Line_5$ belong to different scripts that overlap in the phonetic representation of *water*, but also oppose each other.

3 N-Grams

To be able to recognize or generate jokes, a computer should be able to "process" word sequences. A tool for this activity is the N-gram, "one of the oldest and most broadly useful practical tools in language processing" [3]. An N-gram uses conditional probability to predict N^{th} word based on N-1 previous words. N-grams can be used to store word sequences for a joke generator or a recognizer.

N-grams are built from a large text corpus. As a text is processed, the probability of the next word N is calculated, taking into account end of sentences, if it occurs before the word N.

A bigram is an N-gram with N=2, a trigram is an N-gram with N=3, etc. A bigram model uses one previous word to predict the next word, and a trigram uses two previous words to predict the word.

4 Experimental Design

A further tightening of the focus was to attempt to recognize only $Type_1$ of KK jokes. The original phrase, in this case $Line_3$, is referred to as the *keyword*.

There are many ways of determining "sound alike" short utterances. This project computationally built "sounds like" utterances as needed.

The joke recognition process has four steps:

$Step_1$: joke format validation
$Step_2$: generation of wordplay sequences
$Step_3$: wordplay sequence validation
$Step_4$: punchline validation

Once $Step_1$ is completed, the wordplay generator generates utterances, similar in pronunciation to $Line_3$. $Step_3$ only checks if the wordplay makes sense without touching the rest of the punchline. It uses a bigram table for its validation. Only meaningful wordplays are passed to $Step_4$ from $Step_3$.

[1] $Joke_1$, $Joke_2$ are taken from TV show "Cheers"

[2] http://www.azkidsnet.com/JSknockjoke.htm

$Step_4$ checks if the wordplay makes sense in the punchline. If $Step_4$ fails, go back to $Step_3$ or $Step_2$, and continue the search for another meaningful wordplay.

It is possible that the first three steps return valid results, but $Step_4$ fails; in which case a text is not considered a joke by the Joke Recognizer.

The joke recognizer was trained on a number of jokes. It was then tested on a new set of jokes (twice the number of the training jokes). The jokes in the test set were previously "unseen" by the computer. This means that any joke, identical to a joke in the training jokes set, was not included in the test set.

4.1 Generation of Wordplay Sequences

Given a spoken utterance *A*, it is possible to find an utterance *B* that is similar in pronunciation by changing letters from *A* to form *B*. Sometimes, the corresponding utterances have different meanings. Sometimes, in some contexts, the differing meanings might be humorous if the words were interchanged.

A repetitive replacement process was used for the generation of wordplay sequences. For example, in $Joke_3$ if a letter *w* in a word *water* is replaced with *wh*, *e* is replaced with *a*, and *r* is replaced with *re*, the new utterance, *what are* sounds similar to *water*.

A table, containing combinations of letters that sound similar in some words, and their similarity value was used. The purpose of the table was to help computationally develop "sound alike" utterances that have different spellings. In this paper, the table will be referred to as the Similarity Table. Table 1 is an example of the Similarity Table. The Similarity Table was derived from a table developed by Frisch [1]. Frisch's table contained cross-referenced English consonant pairs along with a similarity of the pairs based on the natural classes model. Frisch's table was heuristically modified and extended to the Similarity Table by "translating" phonemes to letters, and adding pairs of vowels, close in sound. Other phonemes were translated to combinations of letters, and added to the table as needed to recognize wordplay from a set of training jokes.

The resulting Similarity Table shows the similarity of sounds between different letters or between letters and combination of letters. A heuristic metric indicating how closely they sound to each other was either taken from Frisch's table or assigned a value close to the average of Frisch's similarity values. The Similarity Table is a collection of heuristic satisficing values that might be refined through additional iteration.

Table 1: Subset of entries of the Similarity Table, showing sound similarity in words between different letters

a	e	0.23
e	a	0.23
e	o	0.23
k	sh	0.11
l	r	0.56
r	m	0.44
r	re	0.23
t	d	0.39
t	z	0.17
w	m	0.44
w	r	0.42
w	wh	0.23

When an utterance *A* is "read" by the wordplay generator, each letter in *A* is replaced with the corresponding replacement letter from the Similarity Table. Each new string is assigned its similarity with the original word *A*.

All new strings are inserted into a heap, ordered according to their string similarity value, greatest on top. The string similarity value was calculated using the following heuristic formula:

similarity of string = number of unchanged letters + sum of similarities of each replaced entry from the table

(Note, that the similarity values of letters are taken from the Similarity table. These individual letter values differ from the composite string similarity values.)

Once all possible one-letter replacement strings are found, the first step is complete. The next step is to remove the top element of the heap. This element has the highest similarity with the original word. If the removed element can be decomposed into a phrase that makes sense, this step is complete. If the element cannot be decomposed, each letter of its string, except for the letter that was replaced originally, is being replaced again. All newly constructed strings are inserted into the heap according to their similarity. Continue with the process until the top element can be decomposed into a meaningful phrase, or all elements are removed from the heap.

As an example, consider $Joke_3$. The joke fits a typical KK joke pattern. The next step is to generate utterances similar in pronunciation to *water*.

Table 2 shows some of the strings received after one-letter replacements of *water* in $Joke_3$. The second column

shows the similarity of the string in the first table with the original word *water*.

Table 2: Examples of strings received after replacing one letter from the word *water* and their similarity value to *water*

New String	String Similarity to w*ater*
watel	4.56
mater	4.44
watem	4.44
rater	4.42
wader	4.39
wator	4.23
whater	4.23
wazer	4.17

In this example, suppose, after all strings with one-letter replacements are inserted into the heap, the top element is *watel*, with the similarity value of 4.56. *Watel* cannot be decomposed into a meaningful utterance. This means that each letter of *watel,* except *l,* will be replaced again. The newly formed strings will be inserted into the heap, in the order of their similarity value. The letter *l* will not be replaced, as it is not the "original" letter from *water*. The string similarity of newly constructed strings will be most likely less than 4. This means that they will be placed below *wazer*. The next top string, *mater*, is removed. *Mater* is a word. However, it does not work in the sentence "*Mater* you doing." (See Sections 4.2 and 4.3 for further discussion.) Eventually, *whatar* will become the top string, at which point *r* will be replaced with *re* to produce *whatare*. *Whatare* can be decomposed into *what are* by inserting a space between *t* and *a*. The next step will be to check if *what are* is a valid word sequence.

Generated wordplays that were successfully recognized by the wordplay recognizer and their corresponding keywords are stored for the future use of the program. When the wordplay generator receives a new request, it first checks if wordplays have been previously found for the requested keyword. A new wordplay will be generated only if there is no wordplay match for the requested keyword, or the already found wordplays do not make sense in the new joke.

4.2 Wordplay Recognition

A wordplay sequence is generated by replacing letters in the keyword. The keyword is examined because: if there is a joke, based on wordplay, a phrase that the wordplay is based on will be found in $Line_3$. $Line_3$ is the keyword. A wordplay generator generates a string that is similar in pronunciation to the keyword. This string, however, may contain real words that do not make sense together. A wordplay recognizer determines if the output of the wordplay generator is meaningful.

A database with the bigram table was used to contain every discovered two-word sequence along with the number of their occurrences, also referred to as *count*. Any sequence of two words will be referred to as *word-pair*. Another table in the database, the trigram table, contains each three-word sequence, and the count.

The wordplay recognizer queries the bigram table. To construct the database, several focused large texts were used. The focus was at the core of the training process. Each selected text contained a wordplay on the keyword ($Line_3$) and two words from the punchline that follow the keyword from at least one joke from the set of training jokes. If more than one text containing a given wordplay was found, the text with the closest overall meaning to the punchline was selected. Arbitrary texts were not used, as they did not contain a desired combination of wordplay and part of punchline.

To construct the bigram, every pair of words occurring in the selected text was entered into the table. The concept of this wordplay recognizer is similar to an N-gram. For a wordplay recognizer, the bigram is used.

The output from the wordplay generator was used as input for the wordplay recognizer. An utterance produced by the wordplay generator is decomposed into a string of words. Each word, together with the following word, is checked against the database.

An N-gram determines for each string the probability of that string in relation to all other strings of the same length. As a text is examined, the probability of the next word is calculated. The wordplay recognizer keeps the number of occurrences of word sequence, which can be used to calculate the probability. A sequence of words is considered valid if there is at least one occurrence of the sequence anywhere in the text. For example, in $Joke_3$ *what are* is a valid combination if *are* occurs immediately after *what* somewhere in the text. The count and the probability are used if there is more than possible wordplay. In this case, the wordplay with the highest probability will be considered first.

4.3 Punchline Recognition

All sentences in a joke should make sense. A text with a valid wordplay is not a joke if the rest of the punchline does not make sense. For example, if the punchline of $Joke_3$ is replaced with "Water a text with valid wordplay," the resulting text is not a joke, even though the wordplay is valid. Therefore, there has to be a mechanism that can validate that the found wordplay is

"compatible" with the rest of the punchline and makes it a meaningful sentence.

Valid three word sequences were stored. This approach is described in [8]. The method was only partially successful in recognizing meaningful wordplay in the context of punchline.

If the wordplay recognizer found several wordplays that "produced" a joke, the wordplay resulting in the highest N-gram probability is used first.

An alternative approach would be to parse the punchline with the found wordplay. As the wordplay recognizer already determined that the wordplay is meaningful, checking the punchline for the correct grammatical structure may be enough for punchline validation. Preliminary results show that joke recognition can be increased by 30% from the numbers received in [8].

4.4 Punchline Generation

The wordplay generation and recognition algorithms used for the KK joke *recognizer*, can be used for a KK joke *generator*. Given a $Line_3$ of a KK joke, a joke generator can create a punchline by "reading" a sentence with a recognized wordplay, existing in the training text. This means that the program "reads" the training text until it discovers wordplay, received from the wordplay recognizer (see Section 4.2). It then copies the sentence with the wordplay from the training text into the punchline. The generated punchlines can be validated.

5 Results and Analysis

A set of 65 jokes from the "111 Knock Knock Jokes" website[3] and one joke from "The Original 365 Jokes, Puns & Riddles Calendar" was used as a training set. The Similarity Table, discussed in the Section 4.1, was modified with new entries until correct wordplay sequences could be generated for all 66 jokes. The training texts inserted into the bigram and trigram tables were chosen based on the punchlines of jokes from the training jokes set.

The program was run against a fresh test set of 130 KK jokes, and a set of 66 synthetic non-jokes with a structure similar to the KK jokes. The test jokes were taken from [4]. These jokes had the punchlines corresponding to any of the three KK joke types discussed earlier.

To test if the program finds the expected wordplay, each joke had an additional line, $Line_6$, added after $Line_5$. $Line_6$ is not a part of any joke. It only existed so that the wordplay found by the joke recognizer could be compared against the expected wordplay. $Line_6$ consists of the punchline with the expected wordplay instead of the punchline with $Line_3$. The expected wordplay was manually identified.

The jokes in the test set were previously "unseen" by the computer. This means that jokes in [4] that were identical to jokes in the training set were notconsidered.

Some jokes, however, were very similar to the jokes in the training set, but not identical. These jokes were included in the test set. As it turned out, to a human some jokes may look very similar to jokes in the training set, but treated as completely different jokes by the computer.

Out of the 130 jokes previously unseen by the computer, the program was not predicted to recognize eight jokes. These jokes were of $Type_3$ structure; and, therefore, were not meant to be recognized by the design.

The program was able to find wordplay in 85 jokes out of the 122 that it could have potentially recognized. In many cases, the found wordplay matched the expected wordplay.

The punchline generator (see Section 4.4) produced punchline sentences to 110 jokes taken from the test jokes set. All punchlines contained wordplay on $Line_3$.

The program was also run with 66 synthetic non-jokes. The only difference between jokes and non-jokes was the punchline. The non-jokes punchlines were intended to make sense with $Line_3$, but not with the wordplay of $Line_3$. The non-jokes were generated from the training joke set. The punchline in each joke was substituted with a meaningful sentence that starts with $Line_3$. If the keyword was a name, the rest of the sentence was taken from the texts in the training set. For example, $Joke_6$ became $Text_1$ by replacing "time for dinner" with "awoke in the middle of the night."

$Joke_6$: Knock, Knock
Who's there?
Justin
Justin who?
Justin time for dinner.

$Text_1$: Knock, Knock
Who's there?
Justin
Justin who?
Justin awoke in the middle if the night.

[3] http://www.azkidsnet.com/JSknockjoke.htm

A segment "awoke in the middle of the night" was taken from one of the training texts that was inserted into the bigram and trigram tables.

The program successfully recognized 62 non-jokes as such using N-grams for joke recognition.

6 Possible Extensions

The wordplay generator produced the expected wordplay in most jokes, but not all. A more sophisticated wordplay generator might improve the results. A better answer to letter substitution might be phoneme comparison and substitution. Using phonemes, the wordplay generator might be able to find more accurate matches.

An enhanced joke recognizer may be able to recognize jokes other than KK jokes. That is, if the new jokes are based on wordplay, and their structure can be defined.

7 Summary and Conclusion

Computational natural language has a long history. Areas of interest include: translation, understanding, database queries, text mining, summarization, indexing, and retrieval. There has been very limited success in achieving true computational understanding.

A focused area within computational natural language understanding is verbal humor. Some work has been achieved in computational humor generation. Little has been accomplished in understanding. There are many descriptive linguistic tools such as formal grammars. But, so far, there are no robust understanding tools and methodologies.

The KK joke recognizer is a first step towards the computational joke recognition. It is intended to recognize KK jokes that are based on wordplay. The recognizer's theoretical foundation is based on Raskin's Script-based Semantic Theory of Verbal Humor that states that each joke is compatible with two overlapping scripts that oppose each other. The $Line_3$ and the wordplay of $Line_3$ are the two scripts. The scripts overlap in pronunciation, but differ in meaning.

The joke recognition process can be summarized as:

$Step_1$: joke format validation
$Step_2$: generation of wordplay sequences
$Step_3$: wordplay sequence validation
$Step_4$: punchline validation

The success of KK joke recognizer heavily depends on the appropriate letter-pairs choice for the Similarity Table and on the training text selection.

The KK joke recognizer "learns" from the previously recognized wordplays when it considers the next joke. Unless the needed (keyword, wordplay) pair is an exact match with an already found pair, the previously found wordplays will not be used for the joke.

The joke recognizer was trained on 66 KK jokes; and tested on 130 KK jokes and 66 non-jokes with a structure similar to KK jokes.

The program successfully found and recognized wordplay in most jokes. It also successfully recognized texts that are not jokes, but have the format of a KK joke. It was only partially successful in recognizing most punchlines in jokes using N-gram. The initial punchline recognition results using a parser look more promising.

In conclusion, the method was reasonably successful in recognizing wordplay. However, it was less successful in recognizing when an utterance using the wordplay might be valid.

References

[1] S. Frisch, Similarity And Frequency In Phonology. Doctoral dissertation, Northwestern University, 1996

[2] R. Hetzron, "On The Structure Of Punchlines." HUMOR: International Journal of Humor Research, 4:1, 1991

[3] D. Jurafsky and J. Martin, Speech and Language Processing, Prentice-Hall, New Jersey, 2000

[4] A. Kostick, C. Foxgrover and M. Pellowski, 3650 Jokes, Puns & Riddles, Black Dog & Leventhal Publishers, New York, 1999

[5] R. Latta, The Basic Humor Process, Mouton de Gruyter, Berlin, 1999

[6] V. Raskin, The Semantic Mechanisms Of Humour, Reidel, Dordrecht, 1985

[7] G. D. Ritchie, "Describing Verbally Expressed Humour", Proceedings of AISB Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science, Birmingham, 2000

[8] J.M.Taylor and L.J. Mazlack, "Computationally Recognizing Wordplay In Jokes", Proceedings of Cognitive Science Conference, Chicago, 2004