Computationally Recognizing Wordplay in Jokes

Julia M. Taylor (tayloj8@email.uc.edu)
Lawrence J. Mazlack (mazlack@uc.edu)
Electrical & Computer Engineering and Computer Science Department
University of Cincinnati

Abstract

In artificial intelligence, researchers have begun to look at approaches for computational humor. Although there appears to be no complete computational model for recognizing verbally expressed humor, it may be possible to recognize jokes based on statistical language recognition techniques. This is an investigation into computational humor recognition. It considers a restricted set of all possible jokes that have wordplay as a component and examines the limited domain of "Knock Knock" jokes. The method uses Raskin's theory of humor for its theoretical foundation. The original phrase and the complimentary wordplay have two different scripts that overlap in the setup of the joke. The algorithm deployed learns statistical patterns of text in N-grams and provides a heuristic focus for a location of where wordplay may or may not occur. It uses a wordplay generator to produce 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.

Introduction

Thinkers from the ancient time of Aristotle and Plato to the present day have strived to discover and define the origins of humor. Most commonly, early definitions of humor relied on laughter: what makes people laugh is humorous. Recent works on humor separate laughter and make it its own distinct category of response. Today there are almost as many definitions of humor as theories of humor; as in many cases, definitions are derived from theories (Latta, 1999). Some researchers say that not only is there no definition that covers all aspects of humor, but also humor is impossible to define (Attardo, 1994).

Humor is an interesting subject 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 these steps in order to approach the same ability as a human being.

A common form of humor is verbal, or "verbally expressed, humor" (Ritchie 2000). Verbally expressed humor involves 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 verbally expressed humor is the joke. Hetzron (1991) 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, usually consisting of most of the text, which establishes certain expectations. The punchline is a much shorter portion of the joke, and it causes some form of conflict. It can force another interpretation on the text, violate an expectation, or both (Ritchie, 1998). As most jokes are relatively short, it may be possible to recognize them computationally.

Computational recognition of jokes may be possible, but it is not easy. An "intelligent" joke recognizer requires world knowledge to "understand" most jokes.

Theories of Humor

Raskin's (1985) *Semantic Theory of Verbal Humor* has strongly influenced the study of verbally expressed humor. 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.

Another approach is Suls' (1972) two-stage model, which is based on false expectation. The following algorithm is used to process a joke using two-stage model (Ritchie, 1999):

- As a text is read, make predictions
- While no conflict with prediction, keep going
- If input conflicts with prediction:
  - If not ending – PUZZLEMENT
  - If is ending, try to resolve:
    - No rules found – PUZZLEMENT
    - Cognitive rules found –HUMOR

There have been attempts at joke generation (Attardo, 1996; Binsted, 1996; Lessard and Levison, 1992; McDonough, 2001; McKay, 2002; Stock and Strapparava, 2002) and pun recognizers (Takizawa, et al. 1996; Yokogawa, 2002) for Japanese. However, there do not appear to be any theory based computational humor efforts. This may be partly due to the absence of a theory that can be expressed as an unambiguous computational algorithm. In the cases of Raskin and Suls, the first does not offer any formal algorithm, and the second does not specify what a cognitive rule is, leaving one of the major steps open to interpretation.

Wordplay Jokes

Wordplay jokes, or jokes involving verbal play, are a class of jokes depending 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 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 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]

Sometimes it takes world knowledge to recognize which word is subject to wordplay. For example, in $Joke_2$, there is a wordplay between "Tibet" and "to bed." However, to understand the joke, the wordplay by itself is not enough, a world knowledge is required to "link" honeymoon with "Tibet" and "to bed."

A focused form of wordplay jokes is the Knock Knock joke. In Knock Knock jokes, wordplay is what leads to the 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 $Line_3$.

$Joke_3$ 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, both Raskin's (1985) and Suls' (1972) approaches can explain why $Joke_3$ is a joke. Following Raskin's approach, the two belong to different scripts that overlap in the phonetic representation of "water," but also oppose each other. Following Suls' approach, "what are" conflicts with the prediction. In this approach, a cognitive rule can be described as a function that finds a phrase that is similar in sound to the word "water," and that fits correctly in beginning of the final sentence's structure. This phrase is "what are" for $Joke_3$.

N-grams

A joke generator has to have an ability to construct meaningful sentences, while a joke recognizer has to recognize them. While joke generation involves limited world knowledge, joke recognition requires a much more extensive world knowledge.

To be able to recognize or generate jokes, a computer should be able to "process" sequences of words. A tool for this activity is the N-gram, "one of the oldest and most broadly useful practical tools in language processing" (Jurafsky and Martin, 2000). An N-gram is a model that uses conditional probability to predict N^{th} word based on N-1 previous words. N-grams can be used to store sequences of words for a joke generator or a recognizer.

N-grams are typically constructed from statistics obtained from a large corpus of text using the co-occurrences of words in the corpus to determine word sequence probabilities (Brown, 2001). 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.

"The probabilities in a statistical model like an N-gram come from the corpus it is trained on. This training corpus needs to be carefully designed. If the training corpus is too specific to the task or domain, the probabilities may be too narrow and not generalize well to new sentences. If the training corpus is too general, the probabilities may not do a sufficient job of reflecting the task or domain" (Jurafsky and Martin, 2000).

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

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. The only feasible method for this project was computationally building up "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
$Step_4$: last sentence 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 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

If the wordplay is not in the end of the punchline, $Step_4$ takes the last two words of the wordplay, and checks if they make sense with the first two words of text following the wordplay in the punchline, using two trigram sequences. If the wordplay occurs in the end of the sentence, the last two words before the wordplay and the first two words of the wordplay are used for joke validation. 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 punchline recognizer is designed so that it does not have to validate the grammatical structure of the punchline. Moreover, it is assumed that the $Line_5$ is meaningful when the expected wordplay is found, if it is a joke; and, that $Line_5$ is meaningful as is, if the text is not a joke. In other words, a human expert should be able to either find a wordplay so that the last sentence makes sense, or conclude that the last sentence is meaningful without any wordplay. It is assumed that the last sentence is not a combination of words without any meaning.

The joke recognizer is to be trained on a number of jokes; and, tested on jokes, twice the number of training jokes. The jokes in the test set are previously "unseen" by the computer. This means that any joke, identical to the joke in the set of training jokes, is not included in the test set.

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 is used for generation of wordplay sequences. Suppose, a letter a_1 from A is replaced with b_1 to form B. 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 Similarity Table is to help computationally develop "sound alike" utterances that have different spellings. In this paper, this 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 (1996). 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 that are close in sound. Other phonemes, translated to combinations of letters, were added to the table as needed to recognize wordplay from a set of training jokes.

The resulting Similarity Table approximately 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 should be taken as a collection of heuristic satisficing values that might be refined through additional iteration.

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

a	e	0.23
e	a	0.23
e	o	0.23
en	e	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 words are inserted into a heap, ordered according to their similarity value, greatest on top. When only one letter in a word is replaced, its similarity value is being taken from the Similarity Table. The similarity value of the strings is 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 values differ from the similarity values of strings.

Once all possible one-letter replacement strings are found, and inserted into the heap, according to the string similarity, 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 this element can be decomposed into an utterance that makes sense, this step is complete. If the element cannot be decomposed, each letter of the 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.

Consider $Joke_3$ as example. 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.

Suppose, the top element of the heap 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 for 'l' will be replace 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 not the 'original' letter from 'water.' The string similarity of newly constructed strings will be most likely less than 4. (The only way a similarity of a newly constructed string is greater than 4 is if the similarity of the replaced letter is above 0.44, which is unlikely.) 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 on *Wordplay Recognition* and *Joke Recognition* for further discussion.) The process continues until 'whater' is the top string. The replacement of 'e' in 'whater' with 'a' will result in 'whatar'. 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.

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 'Water'
watel	4.56
mater	4.44
watem	4.44
rater	4.42
wader	4.39
wather	4.32
watar	4.23
wator	4.23
whater	4.23
wazer	4.17

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. The new wordplays 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.

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. The joke recognizer, discussed in section on *Joke Recognition*, queries the trigram 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 table, 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 model 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. 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.

For example, in $Joke_3$ 'what are' is a valid combination if 'are' occurs immediately after 'what' somewhere in the text.

Joke Recognition

A text with 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 still 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.

A concept similar to a trigram was used to validate the last sentence. All three-word sequences are stored in the trigram table.

The same training set was used for both the wordplay and joke recognizers. The difference between the wordplay recognizer and joke recognizer was that the wordplay recognizer used pairs of words for its validation while the joke recognizer used three words at a time. As the training text was read, the newly read word and the two following words were inserted into the trigram table. If the newly read combination was in the table already, the count was incremented.

As the wordplay recognizer had already determined that the wordplay sequences existed, there was no reason to re-validate the wordplay.

To check if wordplay makes sense in the punchline, the last two words of the wordplay, $\mathbf{w_{wp1}}$ and $\mathbf{w_{wp2}}$, are used, for the wordplay that is at least two words long. If the punchline is valid, the sequence of $\mathbf{w_{wp1}}$, $\mathbf{w_{wp2}}$, and the first word of the remainder of the sentence, $\mathbf{w_s}$, should be found in the training text. If the sequence $<\mathbf{w_{wp1}}\ \mathbf{w_{wp2}}\ \mathbf{w_s}>$ occurs in the trigram table, this combination is found in the training set, and the three words together make sense. If the sequence is not in the table, either the training set is not accurate, or the wordplay does not make sense in the punchline. In either case, the computer does not recognize the joke. If the previous check was successful, or if the wordplay has only one word, the last check can be performed. The last step involves the last word of the word play, $\mathbf{w_{wp}}$, and the first two words of the remainder of the sentence, $\mathbf{w_{s1}}$ and $\mathbf{w_{s2}}$. If the sequence $<\mathbf{w_{wp}}\ \mathbf{w_{s1}}\ \mathbf{w_{s2}}>$ occurs in the trigram table, the punchline is valid, and the wordplay fits with the rest of the final sentences.

If the wordplay recognizer found several wordplays that "produced" a joke, the wordplay resulting in the highest trigram sequence probability was used.

Results and Analysis

A set of 65 jokes from the "111 Knock Knock Jokes" website[3] and one joke taken from "The Original 365 Jokes, Puns & Riddles Calendar" (Kostick, et al., 1998) was used as a training set. The Similarity Table, discussed in the Section on *Generation of Wordplay Sequences*, 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 set of training jokes.

The program was run against a test set of 130 KK jokes, and a set of 65 non-jokes that have a similar structure to the KK jokes.

The test jokes were taken from "3650 Jokes, Puns & Riddles" (Kostick, et al. 1998). These jokes had the punchlines corresponding to any of the three KK joke structures 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 jokes in the test set were previously "unseen" by the computer. This means that if the book contained a joke, identical to the joke in the set of training jokes, this joke was not included in the test set.

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 they were not the same. As it turned out, some jokes to a human may look very similar to jokes in the training set, but treated as completely different jokes by the computer.

Out of 130 previously unseen jokes the program was not expected to recognize eight jokes. These jokes were not expected to be recognized because the program is not expected to recognize their structure.

The program was able to find wordplay in 85 jokes, but recognized only seventeen jokes as such out of 122 that it could potentially recognize. Twelve of these jokes have the punchlines that matched the expected punchlines. Two jokes have meaningful punchlines that were not expected. Three jokes were identified as jokes by the computer, but their punchlines do not make sense to the investigator.

Some of the jokes with found wordplay were not recognized as jokes because the database did not contain the needed sequences. When a wordplay was found, but the needed sequences were not in the database, the program did not recognize the jokes as jokes.

In many cases, the found wordplay matched the intended wordplay. This suggests that the rate of successful joke recognition would be much higher if the database contained all the needed word sequences.

The program was also run with 65 non-jokes. The only difference between jokes and non-jokes was the punchline. The punchlines of non-jokes 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.

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.

Possible Extensions

The results suggest that most jokes were not recognized either because the texts entered did not contain the necessary information for the jokes to work; or because N-grams are not suitable for true "understanding" of text. One of the simpler experiments may be to test to see if more jokes are recognized if the databases contain more sequences. This would require inserting a much larger text into the trigram table. A larger text may contain more word sequences, which would mean more data for N-grams to recognize some jokes.

It is possible that no matter how large the inserted texts are, the simple N-grams will not be able to "understand" jokes. The simple N-grams were used to understand or to analyze the punchline. Most jokes were not recognized due to failures in sentence understanding. A more sophisticated tool for analyzing a sentence may be needed to improve the

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

joke recognizer. Some of the options for the sentence analyzer are an N-gram with stemming or a sentence parser.

A simple parser that can recognize, for example, nouns and verbs; and analyze the sentence based on parts of speech, rather than exact spelling, may significantly improve the results. On the other hand, giving N-grams the stemming ability would make them treat, for example, "color" and "colors" as one entity, which may significantly help too.

The wordplay generator produced the desired wordplay in most jokes, but not all. After the steps are taken to improve the sentence understander, the next improvement should be a more sophisticated wordplay generator. The existing wordplay generator is unable to find wordplay that is based word longer than six characters, and requires more that three substitutions. A better answer to letter substitution is phoneme comparison and substitution. Using phonemes, the wordplay generator will be able to find matches that are more accurate.

The joke recognizer may be able to recognize jokes other than KK jokes, if the new jokes are based on wordplay, and their structure can be defined. However, it is unclear if recognizing jokes with other structures will be successful with N-grams.

Summary and Conclusion

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

A focused area within natural language is verbally expressed humor. Some work has been achieved in computational generation of humor. Little has been accomplished in understanding. There are many linguistic descriptive tools such as formal grammars. But, so far, there are not robust understanding tools and methodologies.

The KK joke recognizer is the first step towards computational recognition of jokes. 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 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
$Step_4$: last sentence validation

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

The KK joke recognizer "learns" from the previously recognized wordplays when it considers the next joke. Unfortunately, unless the needed (keyword, wordplay) pair is an exact match with one of the found (keyword, wordplay) pairs, the previously found wordplays will not be used for the joke. Moreover, if one of the previously recognized jokes contains (keyword, wordplay) pair that is needed for the new joke, but the two words that follow or precede the keyword in the punchline differ, the new joke may not be recognized regardless of how close the new joke and the previously recognized jokes are.

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 of the jokes. It also successfully recognized texts that are not jokes, but have the format of a KK joke. It was not successful in recognizing most punchlines in jokes. The failure to recognize punchline is due to the limited size of texts used to build the trigram table of the N-gram database.

While the program checks the format of the first four lines of a joke, it assumes that all jokes that are entered have a grammatically correct punchline, or at least that the punchline is meaningful. It is unable to discard jokes with a poorly formed punchline. It may recognize a joke with a poorly formed punchline as a meaningful joke because it only checks two words in the punchline that follow $Line_3$.

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

References

Attardo, S. (1994) *Linguistic Theories of Humor*. Berlin: Mouton de Gruyter

Binsted, K. (1996) *Machine Humour: An Implemented Model Of Puns*. Doctoral dissertation, University of Edinburgh

Frisch, S. (1996) *Similarity And Frequency In Phonology*. Doctoral dissertation, Northwestern University

Hetzron, R. (1991) On The Structure Of Punchlines. *HUMOR: International Journal of Humor Research,* 4:1

Jurafsky, D., & Martin, J. (2000) *Speech and Language Processing*, New Jersey: Prentice-Hall

Kostick, A., Foxgrover, C., & Pellowski, M. (1998) *3650 Jokes, Puns & Riddles*. New York: Black Dog & Leventhal Publishers

Latta, R. (1999) *The Basic Humor Process*. Berlin: Mouton de Gruyter

Lessard, G., & Levison, M. (1992) Computational Modelling Of Linguistic Humour: Tom Swifties. *ALLC/ACH Joint Annual Conference*, Oxford

McDonough, C. (2001) *Mnemonic String Generator: Software To Aid Memory Of Random Passwords*. CERIAS Technical report, West Lafayette, IN

McKay, J. (2002) Generation Of Idiom-based Witticisms To Aid Second Language Learning. *Proceedings of Twente Workshop on Language Technology 20*, University of Twente

Raskin, V. (1985) *The Semantic Mechanisms Of Humour*, Dordrecht: Reidel

Ritchie, G. (1999) Developing The Incongruity-Resolution Theory. *Proceedings of AISB 99 Symposium on Creative Language: Humour and Stories*, Edinburgh

Ritchie, G. (2000) Describing Verbally Expressed Humour. *Proceedings of AISB Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science*, Birmingham

Stock, O., & Strapparava, C. (2002) Humorous Agent For Humorous Acronyms: The HAHAcronym Project. *Proceedings of Twente Workshop on Language Technology 20*, University of Twente

Suls, J. (1972) A Two-Stage Model For The Appreciation Of Jokes And Cartoons: An Information-Processing Analysis. In J. H. Goldstein and P. E. McGhee (Eds.) *The Psychology Of Humor* NY: Academic Press

Takizawa, O., Yanagida, M., Ito, A., & Isahara, H. (1996) On Computational Processing Of Rhetorical Expressions - Puns, Ironies And Tautologies. *Proceedings of Twente Workshop on Language Technology 12*, University of Twente

Yokogawa, T. (2002) Japanese Pun Analyzer Using Articulation Similarities. *Proceedings of FUZZ-IEEE*, Honolulu