Towards a General Framework for Humor Generation from Rated Examples

Thomas Winters and **Vincent Nys** and **Danny De Schreye**
Computer Science Department
KU Leuven
Leuven, Belgium
{firstname}.{lastname}@cs.kuleuven.be

Abstract

Many computer systems are becoming increasingly tailored to their users, customizing and optimizing their experience. However, most conversational agents do not follow this trend when it comes to humorous interactions. Instead, they employ pre-written answers regardless of whether the user liked previous similar interactions. While there already exist several computational humor systems that can successfully generate jokes, their joke generation models, parameters or even both are often fixed. In this paper, we propose GOOFER, a general framework for computational humor that learns joke structures and parameterizations from rated example jokes. This framework uses metrical schemas, a new notion we introduce, which are a generalization of several types of other schemas. This new type of schema makes regular schemas compatible with machine learning techniques. We also propose a strategy for identifying useful humor metrics based on humor theory, which can be used as features for the machine learning algorithm. The GOOFER framework uses these novel concepts to construct a pipeline with new components around previous generators. Using a mapping to our previous work on analogy jokes, we show that this framework cannot only generate this type of jokes well, but also find the importance of specific humor metrics for template values. This indicates that it is on the right track towards joke generation systems that can automatically learn new templates and schemas from rated examples. This work thus forms a stepping stone towards creating programs with a sense of humor that is adaptable to the user.

Introduction

Generating jokes is one of the research tasks in the field of computational humor. In this field, there are three distinct kinds of computational tasks, being the generation, detecting and analysis of humorous artefacts (Binsted et al. 2006; Ritchie 2002). The ideal computational humor system would be able to perform all three mentioned task categories on all types of humor. However, most computational humor systems tend to focus on a single task, performed on a specific type of humor (Ritchie 2001). Joke generation systems also tend to follow a fixed-rule set, and are thus unable to nudge their jokes towards the preferences of a certain user. Existing systems also exist in isolation from each other. Until now, no attempt has been made to generalize over the existing research in the field in order to create systems that generalize previous research and as such are capable of doing more types of tasks on more types of humor. This paper focuses on the first and the last task categories: generating and analysis of humor. This allows the generator to steer towards certain jokes based on analysis of rated jokes (e.g. of a certain user). In previous work, we explored how to perform this task on analogy jokes using a system called GAG (*Generalized Analogy Generator*) (Winters, Nys, and De Schreye 2018). In this work, we extend that system to a general framework called GOOFER (*Generator Of One-liners From Examples with Ratings*). This framework could be used for many types of short jokes and is able to generalize and improve several previous humor generators.

The effectiveness of a joke generation system depends on the quality of the jokes it produces. However, this quality depends on many factors, e.g. word choice (Stock and Strapparava 2003), word order (narrative) (Raskin 1985), amount of incongruity (Kao, Levy, and Goodman 2016; Ritchie 2002) and cultural understanding (Petrović and Matthews 2013). Some promising factors have not yet been studied in much detail, such as the individual and temporal dependencies, by which we mean that joke quality also depends on the observer and the time when the joke was observed. It goes without saying that to increase the effectiveness of a joke generation system, the jokes should be tailored to an individual user's preferences. Seeing the disagreement between users in the evaluations of jokes from previous research (Goldberg et al. 2001; Petrović and Matthews 2013; Stock and Strapparava 2003; Winters, Nys, and De Schreye 2018), it is necessary to account for individual user preferences when generating jokes. However, most systems that are able to adapt jokes to user preference, are not responsible for the generation of these jokes, but are mere humor recommendation engines (Goldberg et al. 2001). We assume the reason for this absence is that most of the existing humor generators utilize some form of rule set or assumptions about their type of joke, or have a model trained to mimic the textual input without considering the quality. This implies that they cannot update the content of their jokes without human intervention. Even worse, this also implies that most are completely incapable of automatically learning new types of humor than the type of humor they were designed for, nor can they update their rules based on rating for previously generated jokes. This has some serious impli-

cations for both the individual and temporal dimensions of the generator's joke quality.

Having the capability of learning humor from joke examples along with their perceived quality can help joke generators nudge their generative space towards more interesting generations. Using this mechanism, it can account for the temporal and individual factors of humor appreciation, by learning new types of jokes from popular platforms and learn to adapt to users by feeding its own generated jokes with ratings back into its training data.

Background

Humor Theory

The incongruity-resolution theory (IR) is probably the most widely accepted theory, and is also most relevant to computational humor research (Krikmann 2006). It states that humor originates from noticing an incongruity in incompatible views, and resolving this incompatibility (Binsted et al. 2006). This is in line with the argument that humor stems from a sudden mental bisociation. (Koestler 1964). A "bisociation" describes the two different viewpoints an act of creativity tends to have, and manifests itself as a jump between two self-consistent but incompatible frames of reference (Ritchie 2001; Krikmann 2006). Most humor theorists seem to agree on the idea of humor being the combination of two such frames, since most jokes can be distilled to a set-up and a punchline (Ritchie 2001).

Ritchie argued that several formal humor theories were often far from being implementable enough for generative purposes, and created his own formal theory, extending the incongruity-resolution theory. He uses the notion of surprise disambiguation, which states that two different interpretations for the set-up of a joke must exist, an obvious interpretation (e.g. *"the fish are in an aquarium"* for the joke in Figure 1) and a hidden interpretation (*"the fish are in a military vehicle"*) that only becomes apparent through the punchline, which forces the hidden interpretation as the only remaining possible interpretation (Ritchie 1999). Hearing the punchline will thus cause an inconsistency with the interpretation assumed when hearing the setup. This inconsistency starts a mental search process to the hidden interpretation, which should be the only interpretation compatible with both the set-up and the punchline. Finding this cognitive rule to explain the mismatch causes laughter to ensue (Ritchie 1999).

Ritchie proposed several properties to identify the relationships specified in his theory, which creates more concrete measurements for joke generation and detection than previous theories (Ritchie 1999; 2002), which we visualized in Figure 1.

- OBVIOUSNESS: The first interpretation should be very obvious (e.g. *fish are usually in aquariums*), otherwise observers hearing the hidden interpretation will not experience an incongruity at the end of the joke, due to the joke being fully compatible with this interpretation. This property thus quantifies how obvious the initial interpretation of the set-up is.
- CONFLICT: There should be a conflict between the punchline and the obvious interpretation, otherwise the search process to the cognitive rule explaining the mismatch will not start (e.g. *you can not drive an aquarium*).
- COMPATIBILITY: The meaning of the punchline should be compatible with the second, hidden interpretation, otherwise the search for a cognitive rule will not stop, ensuing in puzzlement (e.g.*"drive tank" is compatible with "in a military vehicle" interpretation*).
- COMPARISON: There should be a contrasting relationship between the two possible interpretations of the setup, as there would be no bisociation otherwise (e.g. *if the hidden interpretation was "in the sea", it would be less funny*).
- INAPPROPRIATENESS: The second interpretation should be inherently odd, inappropriate or taboo, as this will make the interpretation less obvious, and more humorous to the observer (e.g. *fish usually do not drive vehicles*).

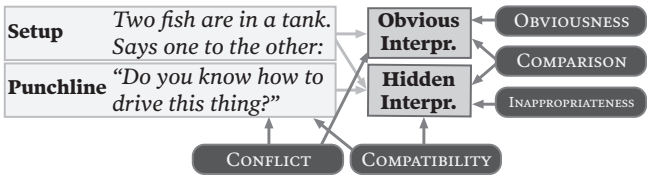

Figure 1: Visualisation of the IR theory

Templates & Schemas

There are several approaches to text generation, such as templates, grammars, Markov chains and recurrent neural networks. Templates are probably one of the most simplistic and naive methods, but they are a powerful tool mostly employed in macros, user interfaces and chat bots (Pilato et al. 2008). They are also extensively used in computational humor projects (Binsted and Ritchie 1994; Manurung et al. 2008; Venour 1999; Lessard and Levison 1992; Raskin and Attardo 1994; Winters and Mathewson 2019).

A template, in the meaning we intend, can be defined as a text with slots or variables. These variables are filled in later by another data source. It also allows data sources to work with different templates. In this work, we call the values to be filled into a particular template "template values".

Schemas are often used as the data source for templates in computational humor (Binsted and Ritchie 1994; Manurung et al. 2008; Venour 1999). They are for example used in the first computational joke production engine, JAPE, as well as its successor STANDUP (Manurung et al. 2008). These systems generate punning riddles in a question-answer format. In these works, schemas are defined as the structure defining the relationships between key words in a joke (Binsted and Ritchie 1994).

In STANDUP, schemas consist of five parts (Manurung et al. 2008):

- **Header:** the variables and the name of the template, e.g. `newelon2(NP, A, B, HomB).`
- **Lexical preconditions:** the syntactic, phonetic, structural or semantic constraints on the

variables, e.g. `nouncompound(NP,A,B), homophone(B,HomB), noun(HomB).`

- **Question specification:** the templates to match the assigned variables to, along with some lexical constraints for their template values, e.g. *"What do you call a `[SynHomB]` with a `[MerA]`?"* with constraints like `shareproperties(NP, HomB).`
- **Answer specification:** Similar to the question specification, e.g. "A `[A] [HomB].`" with constraints like `phrase(A,HomB)`
- **Keywords:** used to define equivalence between jokes, e.g. `[NP, HomB].`

Such a schema can then generate jokes like Joke 1.

JOKE 1:
What do you call a shout with a window?
A computer scream. (Manurung et al. 2008)

Unsupervised Analogy Generator

Petrovic & Matthews have created a model for generating analogy jokes using the *"I like my X like I like my Y, Z"* template, which we will call the unsupervised analogy generator (Petrović and Matthews 2013). They argue their program is the first fully unsupervised humor generation system, as they did not rely on a hard-coded schema approach, but on relations used in a minimization model (although one could also argue that due to specifying this model, it is only partially unsupervised). Their model encodes five relations about the X, Y and Z in these analogy jokes. It fixes the template values such that every template value is a single word, more specifically that X and Y are both nouns and that Z is an adjective. The system requires X to be defined by the user, and uses n-grams to choose Y and Z in such a way that Z is an adjective usable for both X and Y. The relational assumptions used in the model are that the joke is better the more frequent the attribute is used to describe both nouns, the less common the attribute is, the more ambiguous the attribute is and the more dissimilar the two nouns are (Petrović and Matthews 2013). These assumptions are all shown to be implementable as a metric resulting in a number. In order to rank how funny a joke is, the program minimizes the product of these five metrics. This research thus does not use machine learning techniques on training data to generate jokes.

Evaluators considered jokes created by the unsupervised analogy joke generator funny 16% of the time. Human-produced jokes using the same template were considered to be funny in 33% of the time. (Petrović and Matthews 2013). A joke generated by this system can be seen in Joke 2.

JOKE 2:
I like my relationships like I like my source, open (Petrović and Matthews 2013)

Joke Template Extraction

T-PEG (*Template-Based Pun Extractor and Generator*) is a system created for the extraction of templates, aimed at pun templates (Hong and Ong 2009). It generates punning riddles similar to jokes created by JAPE and STANDUP. In order to find a template, the system receives a single punning riddle, for which it replaces some words with variables. The template extraction algorithm is capable of detecting several types of variables, even hidden schema variables. However, they noted that the system heavily relied on linguistic relationships between these template values. In the author's evaluation, 69.2% of the found templates were actually usable for joke generation (Hong and Ong 2009).

Other researchers tested T-PEG by clustering several similar STANDUP-generated jokes based on structural similarity (Agustini and Manurung 2012). Their system extracts templates using T-PEG, and employs agglomerative clustering on these templates using a single majority rule using a semantic similarity evaluation function. They tested this system by automatically verifying whether the templates and schemas used in STANDUP generated jokes were correctly found, and found that it had an overall precision of 61% (Agustini and Manurung 2012).

GOOFER

GOOFER (*"Generator of One-Liners From Examples with Ratings"*) is a novel theoretical computational humor framework for generating short jokes based on rated example jokes. In this section, we generalize the notion of schemas, create a theoretically founded set of metrics for humor purposes, and describe the flow and components of this novel framework.

Schema Generalization

Constraint-based Schemas As mentioned earlier, templates and schemas are an often used approach in computational humor. Schemas usually use lexical relations on the single word template values, such as synonymy and homonymy. We call this type *constraint-based schemas*. Generating template values given a seed is straightforward in constraint-based schemas: once a template variable is filled in, the possibilities for the other words are limited to those that are in the strictly defined relations with the already filled in template values. The limited search space of constraint-based approaches has two effects: it has the benefit of being efficient when generating, but their generative space might appear small compared to generators using other approaches. As illustrated earlier, this type of schemas can be written in a ProLog-like notation, which reveals that these schemas can both generate as well as check jokes. This notation only works for constraint-based schemas, and a different notation for schemas is required in order to incorporate metrics instead of strict relations.

Schemas Generalization Recognizing components of joke generation systems as templates and/or schemas, even if they do not use these explicitly, is a useful exercise to come up with new ways of modeling similar systems. As such, Venour (1999) showed how a Tom Swifty joke generator (Lessard and Levison 1992) was implicitly using templates and schemas. We show how to extend his approach even further, and map other systems that do not use

constraint-based approaches onto a more general type of schema that would allow the use of machine learning algorithms. In order to achieve this, we introduce the notion of a metrical schema.

Metrical Schema We define a metrical schema as having the following components, inspired by the definition of a schema of the STANDUP generator (Manurung et al. 2008):

- **Header:** the name of the schema, as well as the variables used in this schema.
- **Generator:** a generator to propose candidate template values for jokes.
- **Features:** the features used and which variables they are used on. They map template values to numbers.
- **Aggregator:** the way to aggregate the features and to choose the output jokes, e.g. all feature values must be above a certain value or the sum of certain feature values is higher than the sum of other features.
- **Template:** the template with slots for the variables.
- **Keywords:** the most relevant variables, used for calculating equivalence between schema outcomes, and to avoid producing similar jokes which might bore the user due to lack of surprise.

Proof of Generalization In order to show that a metrical schema is a generalization of the constraint-based schema, we have to show that a theoretical mapping from the latter to the former exists. First, the header and the keywords are transferable. Second, multiple templates (such as in STANDUP) can be mapped to a single template through concatenation. Third, the constraints can be mapped to functions that output 0 if the given constraint is violated and 1 if it is satisfied. These functions form the features of the new metrical schema. Fourth, since the schema only allows the assignment of variables that make all the functions map to 1, the aggregator function is defined as a function that only returns *true* if all features return 1. The only attribute left to define now is the generator, as this came for free using the ProLog-like constraints. Since this is a theoretical mapping, we ignore efficiency. We can then define the generator as generating all possible combinations of assignments to the variables using all words of the constraint-based generator's lexicon. This concludes the mapping from a constraint-based schema to a metrical schema, showing that it is a strictly more general notation.

Example of Mapping With this new notion of a metrical schema, we can map the previously discussed analogy generator (Petrović and Matthews 2013) to the template and schema approach. As discussed earlier, this system uses metrics to calculate five metric values from a joke using the *"I like my X like I like my Y, Z"* template. It applies minimization of the product of several values (e.g. dissimilarity and ambiguity of certain template values) over the space of possible jokes. This could thus not have been represented using constraint-based schema, as there is no notion of minimization nor feature values in this type of schemas. Their system only generates *"I like my X like I like my Y, Z"* jokes, and has one model to generate this, implying it only uses one template with only one schema. It is relatively straightforward to map their model to a metrical schema, as can be seen on Figure 2. This mapping shows that the new metrical schema notion is also generalizing joke generators that did not explicitly use (constraint-based) schemas.

Header: `pm_analogy_model(X,Y,Z)`
Metrics: `relatedness(X,Z)`, `relatedness(Y,Z)`, `dissimilarity(X,Z)`, `ambiguity(Z)`, `uncommonness(Z)`
Aggregator: Product of features is below threshold t.
Generator:

1. Take X from input.
2. Generate Z from X as a possible adjective used with X with Google Ngrams.
3. Generate Y from Z as a possible noun used after Z with Google Ngrams.

Template: `I like my <X> like I like my <Y>, <Z>.`
Keywords: `[X,Y,Z]`

Figure 2: Our mapping of the unsupervised analogy generator (Petrović and Matthews 2013) to a metrical schema.

Classification and Regression Schemas This new generalization allows for the use of machine learning by choosing a machine learning algorithm for the aggregator component. A classification algorithm can learn which feature values are correlated with what discrete rating (e.g. from the mode score of a collection of ratings, or the score of one specific person). In a similar fashion, a regression algorithm can estimate a non-discrete rating (e.g. from the average rating). By training to distinguish good jokes from bad jokes, and only allow jokes with an estimated score above a certain threshold, these algorithms act as the aggregator of the metrical schema. They also need to be accompanied by a more naive generator as the template values generator of the metrical scheme. The classification algorithms thus judge whether any of the candidates generated by the template values generator have a score exceeding a certain threshold in order to be considered "good". We call this type of metrical schema a *classification schema* or a *regression schema* depending on the used algorithm.

Metric Set Identification

Now that we have defined schemas such that they are capable of using classification and regression algorithms, we need to define the metrics usable for calculating features from template values. These metrics should be metrics that make sense for a joke judging algorithm. As discussed earlier, Ritchie's incongruity-resolution theory identifies five properties necessary for verbal humor (Ritchie 1999). We can use these properties to identify and validate a set of potential metrics to identify humor, similar to what we did in GAG (Winters, Nys, and De Schreye 2018).

For OBVIOUSNESS, metrics need to measure how obvious an interpretation is. This can be approximated using association or semantic distances in a lexicon: words that are not far from each other semantically, are probably linked to the same and common (since a lexicon contains it) interpretation. Another good measuring function is word frequency using 1-grams. If the word frequency is high, chances are that this word is a common word, where people associate this word with one specific, obvious meaning more easily.

For CONFLICT, we need the first interpretation to conflict with the punchline. This can also be approximated using association or semantic distance, as larger distances correlate with higher conflict. Previous research used n-grams for this (Petrović and Matthews 2013), because when particular words of the punchline are used more with specific other words of the setup, the meaning of this combination of words suddenly becomes more important than all other used words linked to the first interpretation, increasing the conflict with these words.

For COMPATIBILITY, the metrics should measure how compatible the hidden interpretation is with the set-up, which causes the search process to stop searching for another cognitive rule. Again, n-grams can approximate this, as more frequent particular words indicate higher compatibility. The number of meanings a word has is another interesting metric related to its ambiguity (Petrović and Matthews 2013). The surprise disambiguation confirms this, as the set-up is supposed to be ambiguous, with one obvious meaning. Previous research also used metrics determining how similar words sound (Manurung et al. 2008; Binsted and Ritchie 1994; Valitutti et al. 2013; Venour 1999), as homonyms can link the first and the second interpretation, but ensure that only the second interpretation is appropriate in a context.

For COMPARISON, the metrics should measure how much contrast the two possible interpretations have. Previous research did this by analyzing the domains of the words (Raskin 1985) using *WordNet Domains* (Magnini and Cavaglià 2000; Stock and Strapparava 2003), although *WordNet* itself could also be used to find dissimilarity using semantic distance. Adjective vector differences have also been successfully used in computational humor research for approximating this property (Kiddon and Brun 2011; Petrović and Matthews 2013). Calculating this value is done by looking up the frequency of the adjectives used for a noun, and calculating a value based on its difference that describes how different the contexts are that certain words occur in.

For INAPPROPRIATENESS, the metrics should approximate how odd, inappropriate, taboo and/or absurd the second interpretation is. Adjective sexiness and noun sexiness are used in DEViaNT innuendo detection system to calculate how likely it is that a word is an innuendo-related word (Kiddon and Brun 2011). It compares the frequency of the word in a sexual corpus with a non-sexual corpus for adjectives, and adjective vector differences with body parts for nouns. Inappropriateness can also be approximated using the unigram frequency in a balanced corpus, as it is related to its unpredictability, and is thus capable of identifying odd words (Petrović and Matthews 2013).

The identified metrics form a foundation for the knowledge base of our generic framework and are used by the machine learning algorithms to extract features from given jokes. This metric set is not exhaustive, and some metrics complement each other. However, it shows how to cover as much as possible with a small number of metrics from a humor theory point of view, by only selecting a metric from each category and making sure all five dimensions are covered. In GOOFER, this metrics knowledge base is extensible, allowing it to improve the performance for particular type of joke and for testing new humor theories.

Framework Flow

The GOOFER framework uses the previously introduced classification and regression schemas and metric set to learn generating jokes based on a corpus of human-rated jokes. It first extracts the templates from the given jokes and transforms the dataset to the template values with their ratings for each template. This transformed dataset is used to learn classification schemas for each discovered template. A generator then proposes a large number of template values. The classification schema picks the template values that it considers best, based on its learned humor knowledge. These template values are then inserted into their template to create a set of output jokes.

Components

Human evaluation The example jokes have to be rated for the classifier to distinguish good jokes from bad jokes. This is the only place where humans are involved in the algorithm, apart from the user of the system delivering seed words and possibly input jokes.

Template extractor Jokes often have a template, and as such, can be clustered to discover these templates (Hong and Ong 2009; Agustini and Manurung 2012). The template extractor component detects templates for the template store, and extracts the template values for the metric-based rater. It might also detect additional information about the template, such as the part of speech of certain template values.

Template store The templates found by the template extractor are stored in the template store, along with their own trained template values classifier, for its metrical schema.

Metric-based rater Template values need feature values in order to be able to be processed by machine learning algorithms. The metric-based rater achieves this task using the identified metrics set, and using every metric on all (combinations of) template values that fit its prerequisites, thus mapping template values to a set of numbers.

Values generator The metrical schema requires that a generator proposes candidate template values for jokes. This component can thus be any other template-using joke generation system. It both receives a seed and some additional instructions from the related template (e.g. part of speech) about how to generate these values. This second type of information decreases the size of the generative space, but

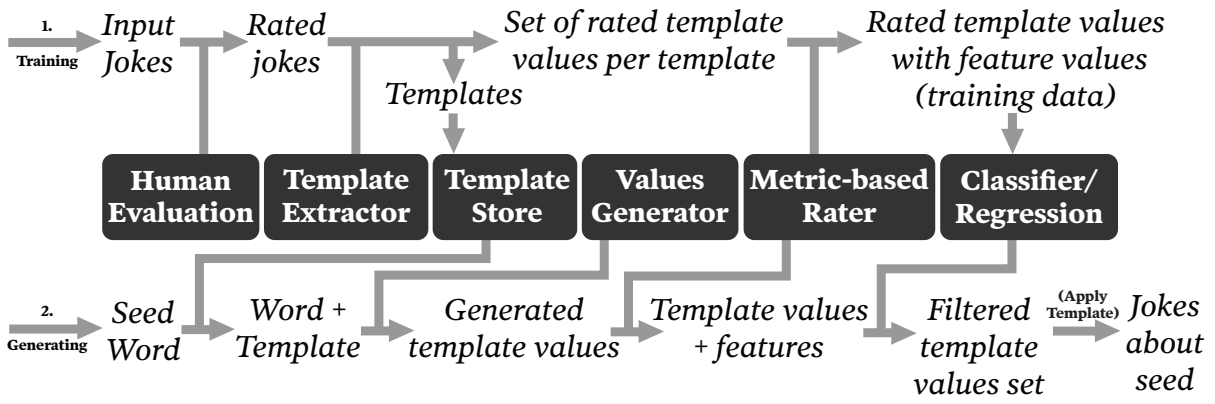

Figure 3: GOOFER framework for joke generation from examples, generalized from GAG (Winters, Nys, and De Schreye 2018).

should increase the score the classifier assigns to the generated values on average.

The generator could use any kind of text generation technique. Since we made the assumption of only having single word template values, an interesting way is using n-grams to generate related words, similar to the unsupervised analogy generator (Petrović and Matthews 2013).

As mentioned earlier, the values generator could also be another computational humor system that employs a template and schema approach, or can be shown to use an equivalent system, as we did earlier. Using such a generator in the GOOFER framework would extend an existing system by allowing it to learn from previous generations.

Classifier After the template values are converted to feature values, they receive scores from the algorithm of the classification or regression schema, which differs from template to template. A wide range of classification and regression algorithms are applicable here. In our experiments, we found RandomForests to work well for both classification as well as regression purposes (Winters, Nys, and De Schreye 2018). Additionally, when a decision tree approach is used and the knowledge base is composed of understandable humor theory metrics, the resulting decision trees might be understandable for humans. This is due to decision trees stating which attributes cause the largest impurity decreases, which can help humans learn what metrics contribute most to the funniness of specific type of joke.

Regression algorithms are useful because they are capable of dealing with real numbers. This means that they can use the average score a joke gets as the score. This is in contrast with classification algorithm that only support a certain number of classes, for which we use the mode or the rating of a single user. This can also be a disadvantage, as the perceived funniness can be entirely different between people, meaning that the average rating might often be close to the middle of the rating range.

One important factor to account for when choosing a classifier or regression algorithm, is the ability to deal with noisy metrics used in GOOFER. The framework deploys a large number of metrics on all possible combinations of the template values, meaning many features might be noisy. The chosen algorithm should thus be resistant to this noise. Since Random Forests ignore parts of the features and of the data set, this might be one of the reasons why it performs so well in our previous experiments (Winters, Nys, and De Schreye 2018).

Code

Several components of this framework have been implemented for the evaluation. The code is available on `https://github.com/TWinters/goofer`.

Evaluation

Generalized Analogy Generator

In order to show that systems implementing the theoretical specification of the GOOFER framework work and to evaluate its results, it requires an implementation. We already implemented a subset of its components in our GAG (*Generalized Analogy Generator*) system (Winters, Nys, and De Schreye 2018), covering most GOOFER framework steps and components, apart from the template extraction step due to only having a single template (being *"I like my X like I like my Y, Z"*). The reason for this is that it simplified data collection, and because previous research on template extraction for jokes had already been successfully done (Hong and Ong 2009; Agustini and Manurung 2012).

The second simplification is the template values generator, which uses n-grams in a similar fashion as the unsupervised analogy generator (Petrović and Matthews 2013). Since the analogy joke template tends to use nouns for X and Y and adjectives as Z, we built our template values generator accordingly[1]. As noted earlier, the POS information of template values can be found by the template extraction algorithms, implying this simplification adds no extra information. The third simplification is that the human evaluation component is also responsible for generating the input jokes

[1]This is the most obvious kind of content for this template. In reality however, we noted that a significant amount of people diverge from these word types when creating this type of joke themselves, for example by naming a relation to another noun as Z.

used in the training data, as we build a platform to collect both. This ensured that the format of the joke is following the supported analogy joke template. This does not violate any of the assumptions of the GOOFER framework, since the input jokes were defined as coming from any source. The GAG system only used a select number of the metrics we proposed, but covered all of the five necessary properties. The GAG system thus generalized the unsupervised analogy generator using this framework.

To evaluate the GAG system, 203 different volunteers helped us collect a data set[2] of 524 jokes (of which 100 generated) and 9452 ratings to these jokes in total, about half of which were for evaluation. In this evaluation of GAG, we found that the best generation model was the classifier schema, with 11.41% jokes having four or more stars on five, whereas humans with similar constraints achieved this rating 21.08% of the time (Winters, Nys, and De Schreye 2018). This is a similar ratio of funniness between generated and human created jokes as the unsupervised analogy generator (Petrović and Matthews 2013), thus successfully generalising a previous system using GOOFER.

Metrics Importance Analysis

As mentioned earlier, GOOFER can find the importance of each metric for each position when using for example decision tree algorithms. The importance of each attribute for the training data using the random forest algorithm in the GAG system is given in Table 1. The importance value is based on how well a metric helps decreasing the entropy of the values in the random trees, ranging from 0 (no decrease) to 1.

Importance	Applied metric
0.67	relative_sexual_freq_2
0.67	relative_frequency_2_1
0.62	adj_vector_similarity_0_1
0.59	relative_sexual_freq_0
0.56	sexual freq_0
0.53	word_senses_2
0.52	relative_frequency_2_0
0.52	sexual freq_2
0.50	word_senses_0
0.50	relative_sexual_freq_1
0.50	frequency_0
0.45	frequency_2
0.43	sexual freq_1
0.36	frequency_1
0.26	word_senses_1

Table 1: The attribute importance of every metric to a certain template value position according to the regression version of the Random Forest algorithm on the average score of the training data.

The found attribute importance seems to be conforming to the model used in the unsupervised analogy generator (Petrović and Matthews 2013), as their metrics roughly map to the 2nd, 3rd, 6th, 7th and 12th most important metrics. The fact that frequency_2 is such a low scoring feature might be because of its similarity to the the top scoring feature, indicating a possible oversight for the INAPPROPRIATENESS property in that research. The found importance ranking is a good sign for GAG and GOOFER, since it seems to somewhat be able to learn these earlier found assumptions about this type of joke. The GAG system effectively generalizes the system created by Petrovic, as it eliminates the need to explicitly model the minimization function and it is capable of detecting even more possible schemas. This shows that our GOOFER framework can effectively make a more generic version of existing research.

[2] https://github.com/twinters/jokejudger-data

Future Work

Generalizing Templates and Template Extraction

In this paper, we chose to work with templates that are a list of fixed strings. One issue with this is that they represent fixed sentences that do not allow small (grammatical) variations. One possible way of solving this is by defining templates using grammars with variables instead of strings and template slots alternating each other. The template extraction methods would then need to be updated, e.g. require new distance measures for grammar trees in order to find these grammar tree templates. Ideally, it would have grammar trees that could represent minimal generalizations of template values in different levels, e.g. same word, same stem, same POS, any word etc. This would increase training data per template, as the template extractor would merge training data of the similar templates.

Sentence Generation

The GOOFER framework assumes for simplicity that template values are single words. Both the discussed metrics and the generated template values have been focused for single word template values. In order to successfully generate jokes using multiple words as template values, the template value generator has to be updated to be capable of proposing such larger parts of sentences.

Probabilistic Logic Schemas

We already showed how using certain machine learning algorithms such as decision tree learners can give indication of the importance of certain attributes, giving some insight into what constitutes a good joke. We also discussed how previous work used ProLog-like notations for representing schemas. Using probabilistic logic (like ProbLog (De Raedt, Kimmig, and Toivonen 2007)) might lead to probabilistic schemas, which would combine the benefits of both the metrical schema introduced in this paper, and upgrade constraint-based schemas to the probabilistic paradigm. The rules induced by such a framework would be more human comprehensible than the algorithms we used in this paper, and could be a useful tool to create human-curated versions, leading to co-creation of joke generator specifications between humans and machines.

Conclusion

In this research, we created a computer program that is capable of learning to generate humor based on human-rated examples. We achieved this by extending and generalizing computational humor concepts such as schemas. We also used humor theory and other computational humor research in order to argue, identify and evaluate a knowledge base of humorous metrics. These findings are used in the GOOFER framework, which is capable of learning humor from rated examples. It achieves this by finding correlations between metrics applied onto template values used in templates that make a joke humorous, and indicate the most important metrics for a particular set of jokes. This framework shows how machine learning algorithms can alleviate humans from the elaborate task of crafting schemas for humor generation by hand. We thus hope that this framework can be a stepping stone towards creating a more adaptive computational sense of humor.

Acknowledgements

This work was partially supported by Research foundation - Flanders (project G.0428.15).

References

Agustini, T., and Manurung, R. 2012. Automatic evaluation of punning riddle template extraction. In *Proceedings of the Third International Conference on Computational Creativity*, 134–139.

Binsted, K., and Ritchie, G. 1994. An implemented model of punning riddles. *CoRR* abs/cmp-lg/9406022.

Binsted, K.; Bergen, B.; Coulson, S.; Nijholt, A.; Stock, O.; Strapparava, C.; Ritchie, G.; Manurung, R.; Pain, H.; Waller, A.; and O'Mara, D. 2006. Computational humor. *IEEE Intelligent Systems* 21(2):59–69.

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007)*, 2462–2467.

Goldberg, K.; Roeder, T.; Gupta, D.; and Perkins, C. 2001. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval* 4(2):133–151.

Hong, B. A., and Ong, E. 2009. Automatically extracting word relationships as templates for pun generation. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, CALC '09, 24–31. Stroudsburg, PA, USA: Association for Computational Linguistics.

Kao, J. T.; Levy, R.; and Goodman, N. D. 2016. A computational model of linguistic humor in puns. *Cognitive Science* 40(5):1270–1285.

Kiddon, C., and Brun, Y. 2011. That's what she said: Double entendre identification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, 89–94.

Koestler, A. 1964. *The Act of Creation*. An Arkana book : psychology/psychiatry. Arkana.

Krikmann, A. 2006. Contemporary linguistic theories of humour. *Folklore: Electronic Journal of Folklore* 33:27–58.

Lessard, G., and Levison, M. 1992. Computational modelling of linguistic humour, tom swifties. *ALLClACIi92 Conference Abstracts* 175–178.

Magnini, B., and Cavaglià, G. 2000. Integrating subject field codes into wordnet. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC-2000)*. Athens, Greece: European Language Resources Association (ELRA). ACL Anthology Identifier: L00-1167.

Manurung, R.; Ritchie, G.; Pain, H.; Waller, A.; Mara, D.; and Black, R. 2008. The construction of a pun generator for language skills development. *Applied Artificial Intelligence* 22(9):841–869.

Petrović, S., and Matthews, D. 2013. Unsupervised joke generation from big data. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 228–232. Sofia, Bulgaria: Association for Computational Linguistics.

Pilato, G.; Augello, A.; Vassallo, G.; and Gaglio, S. 2008. Ehebby: An evocative humorist chat-bot. *Mobile Information Systems* 4(3):165–181.

Raskin, V., and Attardo, S. 1994. Non-literalness and non-bona-fide in language: An approach to formal and computational treatments of humor. *Marcelo Dascal, Pragmatics & Cognition 2:1* 31–69.

Raskin, V. 1985. *Semantic Mechanisms of Humor*. Studies in Linguistics and Philosophy. D. Reidel, 1 edition.

Ritchie, G. 1999. Developing the incongruity-resolution theory. *Informatics Report Series*.

Ritchie, G. 2001. Current directions in computational humour. *Artificial Intelligence Review* 16(2):119–135.

Ritchie, G. 2002. The structure of forced reinterpretation jokes. *Proceedings of April Fools' Day Workshop on Computational Humour (TWLT 20)* 47–56.

Stock, O., and Strapparava, C. 2003. Hahacronym: Humorous agents for humorous acronyms. *Humor-International Journal of Humor Research* 16(3):297–314.

Valitutti, A.; Toivonen, H.; Doucet, A.; and Toivanen, J. M. 2013. "let everything turn well in your wife": Generation of adult humor using lexical constraints. In *ACL (2)*, 243–248. The Association for Computer Linguistics.

Venour, C. 1999. The computational generation of a class of puns. Master's thesis, Queen's University, Kingston, Ontario.

Winters, T., and Mathewson, K. W. 2019. Automatically generating engaging presentation slide decks. In Ekárt, A.; Liapis, A.; and Castro Pena, M. L., eds., *Computational Intelligence in Music, Sound, Art and Design*, 127–141. Cham: Springer International Publishing.

Winters, T.; Nys, V.; and De Schreye, D. 2018. Automatic joke generation: Learning humor from examples. In *Distributed, Ambient and Pervasive Interactions: Technologies and Contexts*, volume 10922 LNCS, 360–377. Streitz, Norbert.