Automatic Joke Generation: Learning Humor from Examples

Thomas Winters, Vincent Nys, and Daniel De Schreye

KU Leuven, Belgium,
`info@thomaswinters.be`,
`vincent.nys@cs.kuleuven.be`,
`danny.deschreye@cs.kuleuven.be`

Abstract. Computational humor systems often employ explicit rules encoding assumptions about what constitutes a funny joke. This paper explores how a program can teach itself to generate jokes based on a corpus of rated example jokes. We implement a system called *"Generalized Analogy Generator"* (Gag) capable of generating jokes using the *"I like my X like I like my Y, Z"* template. We use established humor theory and extend computational humor concepts to allow the system to learn the structures of the given jokes and estimate how funny people might find specific instantiations of joke structures. We also implement a platform for the collection of jokes and their ratings, which are used for the training data and evaluation of the system. Since Gag uses generalized components and learns its own schemas, this program successfully generalizes the most well-known analogy generator in the computational humor field.

Keywords: computational humor, joke generation, analogy generation, machine learning, crowdsourcing

1 Introduction

Humor has always played an important part in human communication. Laughing at one another's jokes and remarks often shows understanding and empathy. Humorous communication in human-machine interactions can strengthen the relationship between humans and virtual assistants. Large companies such as Google and Apple are already hiring comedy writers to make their virtual assistants funnier for this purpose [1][2]. The responses of their virtual assistants are thus written beforehand, rather than generated by the program itself.

Computational humor is a field of artificial intelligence linking humor with computational artifacts. Many advancements have been made over the years in this field in terms of linguistic theories as well as in computer programs capable of generating or detecting specific types of humor. These programs are, for example, capable of creating punning riddles [3][4][5], generating analogies [6], detecting innuendos and one-liners [7][8] and more [9][10][11][12][13][14][15]. However, there has been little research on making the generated humor adapt to the user or adapt over time. Most humor generation research focuses on making jokes based on fixed rule sets, which allow the efficient creation of millions of jokes or the discovery of jokes according to the researcher's assumptions. Such programs do not learn how humorous certain users might find certain jokes nor do they improve themselves over time. This paper, together with previous research, shows significant disagreement between different users among the evaluations of jokes [16][6][9]. The only humorous systems, to the best of our knowledge, that adapt the jokes they present to their user, do not generate said jokes themselves. The use of Eigentaste in the Jester system, for example, merely functions as a humor recommendation system [16]. An integrated humor generator that is capable of generating jokes adapted to the user might thus outperform a generator that does not possess this capability.

This paper explores a joke generation algorithm that learns from a set of human-rated example jokes. We analyze and extend concepts and results from existing computational humor research and use them to specify a more generic and extensible solution. Since our method allows a joke generation algorithm to adapt its output to user evaluation ratings, it can be used to make virtual assistants funnier to their users.

We use analogy jokes as the type of humor we focus on in this paper. More specifically, our algorithm learns how funny certain users might find certain instantiations of analogy jokes using a fixed template, *"I like my X like I like my Y, Z"*. It is also capable of measuring the importance of each metric for measuring the funniness of a joke. This system allows us to examine whether a computer program can learn the relations between words that establish humorous interpretation when used in a particular template. We argue why certain metrics work well to estimate the funniness of a joke by using an existing humor theory. Our analysis of this theory allows us to extend the set of metrics used by a previous analogy generator. Our method differs from previous approaches in that rather than having predefined assumptions about metric values in a rigid model, it learns the optimal combination of the metric values, which allows it to adapt to user preferences. We call the created computational humor system the *"Generalized Analogy Generator"*, or GAG for short.

Since GAG requires a dataset of rated jokes to learn from, we created a platform called *"JokeJudger.com"*. Users were asked to submit analogy jokes and evaluate jokes from other users using a Likert scale. These jokes were then used to train and evaluate GAG in two separate phases. We queried 203 volunteers and received 9034 ratings for 524 jokes, 100 of which were generated by our algorithm. We discuss the implementation, philosophy and lessons learned when developing JokeJudger.

We implemented several different versions of GAG by changing its joke rating algorithm. More specifically, we compare the classifier and regression versions of the Random Forest algorithm by comparing user evaluations of their generated jokes. Our evaluation phase on JokeJudger leads to several conclusions. Firstly, we found that using classification outperformed regression in terms of perceived joke quality. Secondly, compared to the jokes submitted by JokeJudger volunteers, the jokes created by GAG are perceived as funny half as often. This performance is equally good as existing research on analogy joke generation. However, since our method is more generic, it can easily be extended to work with other types of humor, or even to systems that automatically learn several types of humor. The extensibility, the performance and the adaptability to user evaluations make our method more valuable in the context of human-computer interaction

2 Background

This paper extends several existing computational humor generators by reusing and generalizing their components. We discuss some necessary concepts in more detail in this section.

2.1 Templates & Schemas

There are several approaches when it comes to text generation, such as using templates, grammars, Markov chains using n-grams and more recently recurrent neural networks trained on characters [17][18]. 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 [19]. They are also extensively used in computational humor projects [3][4][10][20][21][22][5].

A template, in the meaning we intend, can be defined as a text with variables, also called slots. These slots are filled in later by another data source. In this work, we call the instantiated values to be filled into the slots of a particular template "template values". The slots for these template values tend to have their own variable name each, so that the data source can easily fill these in. It also allows data sources to work with different templates.

Schemas are often used as the data source for templates in computational humor [3][4][10][22][5]. They are used in the first computational joke production engine, JAPE [3]. In this system, schemas are defined as the structure defining the relationships between slots of the template [3]. They are responsible for generating the template values, often using a lexicon to find template values having a certain relationship (e.g. synonymy) with other template values.

2.2 Petrović & Matthews Analogy Generator

Petrović and Matthews created a model for generating analogy jokes using the *"I like my X like I like my Y, Z"* template [6]. They state that 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. Their model encodes five relations about the X, Y and Z *"I like my X like I like my Y, Z"* jokes. It constrains every template value to be a single word, more specifically it assumes that X and Y are both nouns and that Z is an adjective. The system requires X to be defined by the user. The system generates further values using Google Ngrams by choosing Y and Z such that Z is an adjective usable for both X and Y. The relational assumptions used in the model are that the joke is funnier the more often 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 [6]. These assumptions are all shown to be implementable by a metric mapping the words to numbers between zero and one. For most of these metrics, they relied on Google Ngrams to find the appropriate values. The research also uses WordNet to look up the number of senses of a word [23]. In order to rank how funny a joke is, the program minimizes the product of these five relations. Unlike our approach, this research thus does not use machine learning techniques on training data to generate jokes, but rather to find the optimal choice of metrics for the model.

This system is quite successful, as its generations are considered funny 16% of the time, while human-produced jokes using the same template are considered to be funny in 33% of the time [6].

2.3 Ritchie's Incongruity-Resolution Theory

Over the years, a multitude of theories about humor and laughter have emerged [24]. One of them is the incongruity-resolution theory, which states that humor arises from the resolution of a conflict between the assumed interpretation and the interpretation caused by the punchline. However, most humor theories are often far from implementable. Ritchie, one of the JAPE [3] and STANDUP [4][25] researchers, created his own formal theory extending the incongruity-resolution theory [26]. Earlier incongruity-resolution theories treat ambiguity between two interpretations as the source of humor. He argues that the problem with this assumption is the lack of distinction in whether ambiguity causes humor or confusion [26]. Surprise disambiguation is an element often used in incongruity-resolution theories, stating that two different interpretations for the set-up of a joke must exist. The first interpretation is the most obvious one, whereas the second is the hidden meaning. We can demonstrate this using the following joke: *'Two fish are in a tank. One turns to the other and says: "Do you know how to drive one of these things?"'*. In the example, the first interpretation is that the two fish are in an aquarium, while the second interpretation is that they are in a military vehicle. The audience should only become aware of the second meaning through the punchline, which forces the second, hidden meaning as the only remaining possible interpretation [26].

Ritchie proposed several properties to identify the relationships in a more formal way [26]. The first property is *"Obviousness"*, that expresses how obvious the first interpretation is compared to the second interpretation. E.g. since fish are usually in water, it seems obvious that the interpretation of the word *"tank"* is an aquarium. The second property is *"Conflict"*, which entails that the punchline conflicts with the first interpretation. Since you can't drive an aquarium, this conflict prompts the brain to look for other possible interpretations. The next property he proposes is the *"Compatibility"* property, expressing that the punchline is compatible with the second, hidden interpretation. This allows listeners to stop searching for other interpretations when they find the fitting, second interpretation, namely that of a military vehicle, as it is compatible with the set-up and punchline. The *"Comparison"* property requires that the two possible interpretations are different. The final property is *"Inappropriateness"*, stating that the second interpretation should be inherently odd, inappropriate or taboo [26]. E.g. two goldfish in a military tank is an inherently odd interpretation. We use this theory to identify metrics for our system.

3 Generalized Analogy Generator

We implemented a system called *"Generalized Analogy Generator"* (Gag for short). This system is capable of generating jokes using the *"I like my X like I like my Y, Z"* template. It differs from previous computational humor systems in that rather than building explicit models or schemas, it learns how to instantiate the template values from rated joke examples.

3.1 System Flow

The flow of the Gag system can be seen in Figure 1. We built a platform called *"JokeJudger"* that helps volunteers rate and create *"I like my X like I like my Y, Z"* jokes. These jokes are then processed by the *"Template Processor"*, which ensures that there is only one word for each template value. These words and their interrelations are then assigned feature values using metrics, so that they can be used as training data for the classifier component. This classifier component learns to recognize how funny a joke is. In the second stage of the program, a user can submit a seed word. The program then generates candidate jokes by filling in the template values randomly using n-grams. These candidate jokes are then assigned values as well, so that the classifier component can classify them based on their potential funniness. Only the jokes with the highest potential funniness are then displayed to the user.

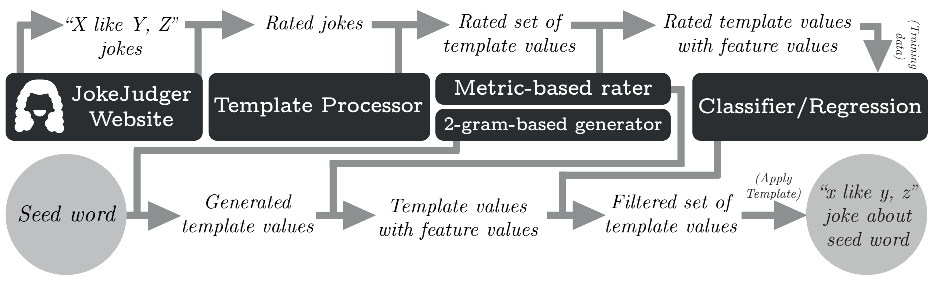
"X like Y, Z" jokes
Rated jokes
Rated set of template values
Rated template values with feature values
(Training data)
JokeJudger Website
Template Processor
Metric-based rater
2-gram-based generator
Classifier/Regression
Seed word
Generated template values
Template values with feature values
Filtered set of template values
(Apply Template)
"x like y, z" joke about seed word

Fig. 1. A schematic overview of the Generalized Analogy Generator GAG.

3.2 JokeJudger

We implemented a data collection web application called JokeJudger[1] as the source of our training data. The web application allows users to create *"I like my X like I like my Y, Z"* jokes and rate jokes created by other users. We invested significant effort into developing this platform. We employ several tactics in order to make the site as user-friendly an make users revisit the site.

Another possibility to handle the data collection and rating that is often used in research [6][8] is scraping popular sites equipped with like-based systems containing jokes, such as Twitter[2] and Reddit[3]. The problem with these like-based systems are that they do not necessarily always reflect how well the joke is received. This is due to sites not exposing all jokes equally, as well as having a significant bias towards a specific audience. Visitors of a specific website as well as followers of a comedian on Twitter tend to have a correlated taste of humor. The amount of exposure for a tweet on Twitter is correlated to several other factors, such as the number of followers of the posting user as well as to the number of followers of any user retweeting or liking the tweet (both these actions make the tweet show up in other users' timelines) and even the hashtags used. Higher exposure means that the tweet has a larger amount of people that can like the tweet. The audience size of a tweet is however not freely available except for the user posting this tweet and is thus hard to factor out. Most other platforms for humorous content have similar problems, since exposing top-scoring jokes is to their own advantage, even though this might leave equally good jokes remain at the bottom.

We acknowledged the need for a platform that presents jokes independently of the quality perceived by other users as a requirement to create our system. The most similar platform to our platform is The New Yorker's platform for voting captions for their image of the week.[4] The captions collected by this platform have also been studied for computational humor purposes [27][15]. There are however large differences between their platform and our provided platform. One such difference is that JokeJudger users are able to get help creating their joke if necessary, and that they are rewarded with several insightful analyses about their created jokes later on. JokeJudger also uses a Likert scale of size five, instead of three, adding more precision. It is also the first time to our knowledge of that an open platform was created to acquire jokes of a specific format and of which the source code is released for reuse.[5]

During the first phase, the training data collection phase, the platform collected 336 jokes and 4828 ratings by 106 users. We calculate the agreement for a joke as the percentage of people who submitted a rating equal to the mode, i.e. the most commonly occurring rating, of all ratings for this joke, or the average of these percentages if there are multiple modes. The average agreement for

[1] `http://jokejudger.com` and `https://jokejudger.herokuapp.com`
[2] `https://twitter.com`
[3] `https://reddit.com`
[4] `http://www.newyorker.com/cartoons/vote`
[5] `https://github.com/TWinters/JokeJudger`

all the human-created jokes is 41.36%. Alternatively, if we define agreement as picking one of the modes (instead of averaging the agreement percentages of the modes), the average agreement about the ratings in the dataset is 48.61%. The dataset thus exposes the large disagreement between raters about the quality of a joke. This is one of the typical characteristics of humor, which makes it hard for programs to identify the quality of a joke.

At the end of the evaluation phase, we collected 524 jokes (100 of which are generated by GAG), 9034 ratings, 418 markings (*"Too offensive"* or *"I don't understand"*) coming from 203 users. The agreement on the jokes was of a largely similar magnitude, being 41.87% for the first definition and 48.52% using the second definition. The collected datasets have been made available online.[6]

3.3 Template Processor

The *"Template Processor"* component is responsible for transforming a joke to a set of template values by removing the template of the joke. Our algorithm assumes that there is only one type of joke (namely using the *"I like my X like I like my Y, Z"* template), but this component can be extended to allow for other templates. This could be achieved by using template extraction algorithms from other computational humor research [22][5]. The metrics used by this system also assume that there is only one word per template value. However, JokeJudger users were allowed to use as many words for each template value as they wanted, as to not limit their creative freedom even more. The template processor is responsible for handling these multi-word template values.

We propose several solutions for being able to use single-word metrics to evaluate template values that consist of multiple words. In order to convert this problem to a smaller problem, we could find out what part of speech is used for a certain template slot. In our case, we can assume that X and Y tend to be nouns, while Z tends to be an adjective, as previous research on this type of joke has made similar assumptions [6]. The given multi-word template value is then searched for words of this part of speech. If there is only one word of this part of speech, the problem is solved. If there is no word of this part of speech, we could look for the next likely part of speech or alternatively just ignore the data point. If there are multiple candidates of this part of speech, a first solution is that the metric is used on every possibility, after which the maximum, minimum, average or other aggregation function is used on these feature values. Another solution when dealing with multiple values, is to create all combinations of all candidate words for the template values and replace the original template values with this set. Jokes such as *"I like my coffee like I like my war, gruesome and cold"* are transformed to the two jokes *"I like my coffee like I like my war, gruesome"* and *"I like my coffee like I like my war, cold"*. This is the solution we picked for our implementation, as it is the most general solution, since it does not require any knowledge about the metrics. This operation transforms our collected JokeJudger training data to 528 sets of single-word template values.

[6] `https://github.com/TWinters/JokeJudger-Data`

3.4 Used Metrics

In order to use classification algorithms on the template values, we need a set of feature values per joke. These feature values can be calculated from the template values using metrics on one or several of these jokes. These chosen metrics should make sense for a joke judging algorithm. Ritchie's incongruity-resolution theory, as discussed in Section 2.3, identifies five properties necessary for verbal humor [26]. We made sure to cover all humor theory properties by implementing at least one metric per property.

The chosen metrics are:

- **Word Frequency:** We used the 1-gram model of the *English One Million* dataset from Google Ngrams[7] to calculate the frequency of any word. These datasets are provided as zipped comma separated value files, making them hard to search through. In order to query these datasets more easily, we created a Java program[8] to extract these files to a MySQL database for efficient querying. This metric can be used to estimate the *"Obviousness"* property of words.
- **Frequency of word combinations (2-grams):** The frequency of word combinations can be estimated using 2-grams. We use Google Ngrams and our created program to load these n-grams and calculate the frequency of adjective-noun combinations on all compatible template values. Since we only require adjective-noun combinations, we combine part of speech (POS) evidence of WordNet and Stanford Log-Linear POS tagger[9] to only store possible adjective-noun pairs. The frequency of word combinations is correlated with the *"Conflict"* property, as it indicates how often or how rarely certain words are usually used with each other, and thus how many interpretations might conflict. It also helps measuring the *"Compatibility"* property, as it finds how well words fit the second interpretation.
- **Number of meanings metrics:** The number of meanings metric is calculated using the number of definitions provided by WordNet for a particular word [28]. This metric helps measuring the *"Compatibility"* property, as words with more meanings are more likely to be used in different contexts, and thus usable to find the interpretation implied by the punchline [6].
- **Adjective Vector Similarity:** The similarity in the list of adjectives used with a noun can be found by using 2-grams of Google Ngrams, which can be loaded into the database using our program. In order to calculate the adjective vector, the metric finds all adjectives and their frequencies that they are used before these nouns in sentences, by dividing their number of appearances by the sum of these counts, over all the adjectives used with the noun. The metric then sums all products of these frequencies of the adjectives used in both vectors. This is similar to the *noun dissimilarity* factor used in Petrović's Analogy Generator, although they use a different calculation [6].

[7] `https://storage.googleapis.com/books/ngrams/books/datasetsv2.html`
[8] `https://github.com/TWinters/google-ngrams-to-mysql`
[9] `https://nlp.stanford.edu/software/tagger.shtml`

This metric helps measuring the *"Comparison"* property, as it can discover how dissimilar certain words are and thus gives a measure how dissimilar interpretations are.

– **Word sexiness:** The word sexiness is calculated by comparing the frequency of a word in a corpus of a sexual domain to a normal domain. We already discussed how to calculate the word frequency in a normal corpus above. In order to calculate this value, we use a similar, but simplified, approach as used in the DEViaNT system [7]. This research used the texts from `textfiles.com` under the erotica category[10] as their sexual corpus. In order to scrape all these files, we created a Java tool that downloads all documents of this site from any given category.[11] We added both the frequency in sexual corpora as this frequency divided by the Google N-gram word frequency. This metric is used to measure how present the *"Inappropriateness"* property in a joke is. This property is not explicitly accounted for by any metric used in Petrović's analogy generator [6].

The chosen metrics are very similar to the metrics chosen in previously existing analogy generator research, as their metrics have been shown to be successful [6]. However, key differences are that in our model, the metrics are applied to every template value or template value combinations that fits the (POS) prerequisite. This not only results in having fifteen features instead of five, but also in a more general model, as we do not make any assumption about the importance of certain values for specific template values over others. Another difference is that our metric set includes the word frequency in a sexual corpus, as there was not metric explicitly covering the *"Inappropriateness"* property of Ritchie's incongruity-resolution theory [26]. We also implemented the metrics slightly differently, as the metrics are used for classification and not for minimization, and therefore have less strict requirements.

3.5 Template Values Generator

The generation of candidate template values is done using 2-grams. The 2-grams used in our system are the same Google Ngrams used in the *Frequency of word combinations* metric. Since we know that jokes using the *"I like my X like I like my Y, Z"* template mostly use nouns for X and Y and adjectives as Z, we built our template values generator accordingly.[12] Note that the information of the POS of the words of a template could also be found algorithmically, and assumptions about the POS could be discovered by the algorithm itself when dealing with other templates. The template values generator finds nouns for X and Y and an adjective that could be used to the left of both words according to

[10] `http://textfiles.com/sex/EROTICA`

[11] `https://github.com/TWinters/TextFilesComScraper`

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

the n-grams. If seed words are given, these can be assigned to X if one is given, or to X and Y if two seed words are given. For example, a user might enter *"coffee"* for X, for which the generator will try out several possibilities for Z (e.g. *"strong"*) and generate possible Y for Z (e.g. *"men"*). A similar generator is also deployed in Petrović's system [6], making it easier to compare the differences in the evaluation of our approaches. Using 2-grams as a generator also simplifies the way the metrics work on the template values, as most of the metrics used are designed to work on single words, and thus do not need to be processed by the *Template Processor* during this phase.

3.6 Classification & Regression

We implemented several different versions for the machine learning component of GAG. There are two types of approaches we used, being the use of classification and the use of regression. The difference between these two types is that regression involves predicting a continuous value and classification identifies group membership to a label. A regression algorithm thus allow us to predict the average rating a joke might get. Using a classification algorithm on the data set, the algorithm can predict the most popular Likert-scale rating for a joke, since this defines five possible classes. Using a classifier thus gives the system the same five star option a human has. We use WEKA in our implementation for both options.

Classifying the Mode Rating. When using classification algorithms, we use the mode rating received for each joke. The distribution of the modes of the jokes can be seen in the *"total"* column of Table 1. The most frequent mode is the two star rating, with 174 instances. The ZeroR algorithm is a classifier that classifies every given instance to the most frequent class, in this case the two star rating. This naive algorithm thus already classifies 32.95% of the training data correctly.

We compared several classification algorithms on their number of correctly classified instances using 10-fold cross-validation. Random Forest [29] was the best scoring classifier for this task, classifying $\frac{325}{528} = 61.55\%$ of the training instances correctly. Since we use a general approach in this system, we do not encode assumptions about which metrics are important for which template values. Our system thus uses a multitude of humor metrics applied to every compatible template value. Using all features together in a single model can result in overfitting noisy data due to having too many features. It thus makes sense that Random Forest, which uses multiple subsets of the features, works relatively well for the GAG system.

A weakness of using classification on the mode rating is that the classification algorithm does not explicitly know about the ordering of the labels. For example, it does not know that score 4 and 5 are closer to each other than score 2 and 5.

Comparing the performance of the Random Tree algorithm (61.55%) to the agreement between users (41.36%, as explained in Section 3.2), shows that the

Table 1. The confusion matrix of the Random Tree classifier on mode ratings of the training data.

Classified as →	1 star	2 stars	3 stars	4 stars	5 stars	*total*
1 star	30	16	10	1	0	*57*
2 stars	1	114	46	13	0	*174*
3 stars	1	40	119	10	0	*170*
4 stars	1	26	24	57	2	*110*
5 stars	2	3	6	1	5	*17*

algorithm picks the most frequent rating more often than the human evaluators in the training data do.

We can see that the recall is rather low for the five star rating class, as it is very often misclassified ($\frac{2+3+6+1}{2+3+6+1+5} = 70.5\%$ of the time). This might be due to a lack of jokes with a five star rating or five star jokes being intrinsically very unpredictable. These misclassifications cause the GAG system to miss a lot of good jokes. This is something we can live with, as the system generates potentially millions of candidate jokes. More worrisome is the fact that two star ratings are often misclassified as four stars.

The precision of a classification algorithm determines the chance that a classification to a certain class is correct. This is a more important value than recall, as the system should pick high quality jokes from the randomly generated set of jokes. This value is especially important for high rating labels such as 4 and 5 stars, since it is more important that jokes rated as such are great jokes than that we might miss good jokes because they got a lower label. The confusion matrix (Figure 1) shows us that no joke classified as having a mode of five stars by the algorithm has a score lower than four. However, the subset of jokes with a mode rating of five is rather small and thus does not allow for large variety. We thus use both the four and five star rating to filter the set of generated jokes. According to the confusion matrix, filtering out four and five star classifications gives us a precision of $\frac{57+2+1+5}{1+13+10+57+2+1+5} = \frac{65}{89} = 73\%$ jokes that actually have four or five star mode rating in the training data.

Regression on the Average Rating. Regression algorithms can be used on the average rating of each joke in the dataset in order to predict the quality of the joke. We used the RRSE (*Root Relative Squared Error*) value to compare several regression algorithms of the WEKA toolkit. The regression version of the algorithm using the Random Forest algorithm scored the highest as well, with a RRSE of 84.04% and a RMSE of 43.25%. We assume the reason that the Random Forest algorithm performs best is the same as the reason it performs well for classification, being that it creates trees that use subsets of the features and data.

Another important advantage the Random Forest algorithm brings, is that it is capable of finding the importance of each attribute by checking its average impurity decrease in the decision trees. This means that the algorithm is capable of stating the importance of each metric for each position. A humor theorist can thus use this system with a Random Forest algorithm to validate and discover assumptions about a certain collection of rated jokes. We noticed that the attribute importance found for all attributes were in line with previous research on analogy generators. Most of the metrics used on specific template values which were most similar to metrics used in Petrović's model [6] had a relatively high importance. This is a good sign for our algorithm, since it seems to be able to learn these earlier found assumptions about this type of joke.

4 Evaluation

We generated 100 jokes using the Gag system. Half of them were generated using the classification algorithm, the other half using the regression algorithm. We generated these jokes based on values for X present in the training data, similar to how Petrović created his evaluation data [6]. Since classifying all possible jokes with a fixed X generated by 2-grams (roughly $\sim$10.000.000 jokes) requires a lot of processing power, we added a random sampler in the generation process. The generator randomly samples adjectives used with X in a uniformly distributed way as candidates for Z and then randomly samples candidates for Y, also uniformly distributed. Since we do not want the output jokes to be similar, we added the option to Gag to eliminate jokes that are too similar to higher scoring jokes from its output. We do this by only allowing one template value to be equal between jokes. This is similar to how Standup filters its jokes, using the *"keywords"* attribute in its schemas to calculate its similarity [4]. The number of jokes having a mode of 5 is significantly smaller than the others, which limits the variety of the output of jokes with five stars when using the classification algorithm. To solve this, we used Gag to output generated jokes with a score with a score of 4 or higher. We then randomly sample a quantity of jokes proportional to the frequency of X in the training data.

These 100 jokes were then uploaded to JokeJudger, interspersed with human-created jokes. Since JokeJudger presents jokes with the lowest number of ratings first, we made sure 67 new human-created jokes were added. This ensured that the first raters did not only see the jokes generated by Gag. This decreased the chance that the evaluating users knew that they were rating generated jokes as well as giving us a more reliable control group of human-created jokes.

During the evaluation phase, 4424 ratings and markings were given to jokes on JokeJudger. The 50 jokes from the classification algorithm variant received 745 ratings, whereas the 50 regression algorithms jokes received 721 ratings. The 424 human-created jokes on JokeJudger received the remaining 2958 ratings. The platform was marketed in several ways, including posts on Facebook, Twitter, Reddit and by a newsletter to members of the JokeJudger platform during the training data collection phase.

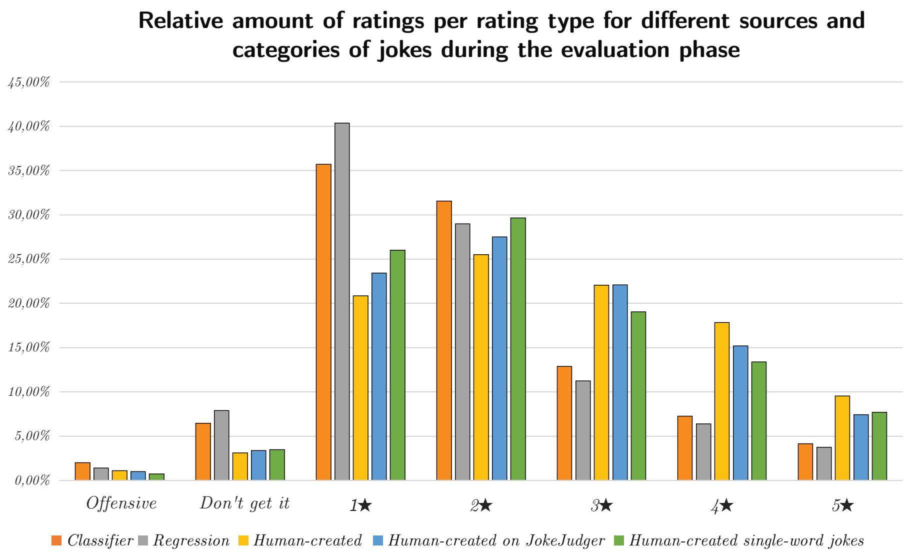

Fig. 2. The resulting evaluations of the JokeJudger and GAG jokes at the end of the evaluation phase.

4.1 Comparing Classification with Regression

We can see on Figure 2 that the jokes generated by the classification algorithm are more positively received than jokes generated using the regression version of GAG. The regression jokes received 5% more 1 star ratings, as well as 1.5% more *"I don't understand"* markings. The classification jokes receive more ratings than the regression jokes for every rating category larger than or equal to two stars. Since three stars mean "Okay", 24.3% of the classifier jokes were seen as average jokes or better, compared to 21.36% of the regression jokes.

One possible explanation for the higher quality jokes created by the classification algorithm is that both four and five star rated jokes were allowed to be sampled, due to the substantially smaller number of five star rated jokes in the training data. Five star jokes also had a low precision when using the Random Forest algorithm, probably due to the unpredictability of truly good quality jokes. For regression however, these high-scoring, difficult-to-predict jokes might affect the continuous scores in a way that is more difficult to account for.

4.2 Comparing Generated Jokes with Human Created Jokes

We can see that human-generated jokes in general receive higher scores than the jokes generated by GAG. Human-created jokes are considered to be *"quite good"* and *"great"* jokes 27.38% of the time, whereas our classification algorithm only receives these types of ratings 11.41% of time. One possible reason for the

Table 2. The percentage of ratings higher than or equal to four stars out of five for each category of Figure 2.

Source	*4+ ratings*
GAG (Classifier)	11.41%
GAG (Regression)	10.12%
Human (All)	27.38%
Human (JokeJudger)	22.61%
Human (Single words)	21.08%

lower quality of generated jokes is that they all have a similar shape, since they only use single-word template values. It might be that jokes using only single words as template values in *"I like my X like I like my Y, Z"* jokes are generally perceived as worse jokes. We can see evidence for this claim by looking at the *"Human-created single-word jokes"* statistics. These jokes are only perceived as better than *"quite good"* 21.08% of the time. This means that the classification algorithm, with 11.41% jokes with four or more stars, performs more than half as well as human generated jokes using the same single-word constraint.

Another possibility is that by executing our multi-word aggregation (see Section 3.3), we transformed the dataset too much and made the generator learn from low quality jokes and still perceive them as high quality. For example, these types of jokes sometimes have several adjectives describing both nouns as Z, where the last one is often the funniest. An example of this is the joke *"I like my women like I like my art: pretty and silent"*. Although both receive the same ratings in our transformed dataset, the perceived funniness of the *"silent"* punchline might not be similar to the the *"pretty"* punchline.

One thing to note is that computer generated jokes receive more *"too offensive"* markings. This might be because jokes during the training data phase were flagged manually to be hidden unless the user disabled the offensiveness filter on his profile. During the evaluation phase, this page was left untouched for evaluation integrity, meaning that the generated jokes, unlike the old human-created jokes, were unfiltered.

4.3 Comparison with Existing Analogy Generator

In the evaluation of his analogy generator, Petrović used a Likert scale of size 3: *"funny"*, *"somewhat funny"* and *"not funny"* [6]. The human generated jokes (collected from Twitter) in his data got 33.1% funny ratings, his model 16.3% and his baseline, which generated random jokes with his 2-gram generator and maximizes only adjective vector difference between X and Y, 3.7% funny ratings.

We can map our Likert scale of size 5 to his scale by saying that 1-2 maps to *"not funny"*, 3 to *"somewhat funny"* and 4-5 to *"funny"*. Using this new classification, human generated jokes are 27.38% of the time perceived as funny, while our classification version of the generator is perceived as funny 11.41% of the time. Both the percentage of the funniness of the human generated jokes as well as the percentage of our generator is lower. There are several possible reasons to explain this. A first reason might be that Petrović's research only used five different raters. These raters might have been more inclined to rate jokes as funny than the large, diverse population on JokeJudger. A second reason might be that the quality of jokes submitted to JokeJudger is lower due to the barrier being lower. The barrier might be lower because users are able to submit jokes in an anonymous way. It might also be that jokes found on platforms like Twitter, are generally more funny due to the higher threshold for posting, as well as better jokes receiving more exposure, as we discussed in 3.2. We can see evidence for this claim in the fact that human-created jokes on JokeJudger were perceived as funny 27.38% of the time, whereas when we filter out our initial dataset collected on such platforms, the average funniness of human-created jokes on JokeJudger drops to 22.61%. This means that, using this constraint, the Gag system with its 11.41% funniness is again more than half the time as funny as human-created jokes. This is similar to the existing research, where the generated jokes were also perceived as funny half as often as human-created jokes.

5 Discussion

5.1 Extensions

There are several possible extensions of Gag for future research. We summarize and discuss several of them in this section.

Template Extraction. In this paper, we assumed that all example jokes were using the *"I like my X like I like my Y, Z"* template. We believe that adding a component that clusters jokes per template would enable this system to work for several types of jokes. Template clustering and extraction on jokes has already been successfully executed by other researchers [22][5]. However, working with multiple templates requires a large dataset of rated jokes that includes similar jokes. One way of overcoming this issue is allowing templates to have small (syntactical) differences. This could possibly be done by encoding them as grammars with variables instead of the usual approach of alternating fixed strings and template slots. Additionally, templates using similar schemas could also be clustered together. However, this is more difficult than detecting similar templates, as it requires to detect similarities between jokes employing different narrative styles. This would increase the amount of training data per template if done correctly, as the training data that would be found for each of the similar templates can then get merged.

In this paper, we used five different types of metrics to measure fifteen different relations between the template values of a joke. We believe that increasing the metric set might be useful when dealing with more types of humor. This larger set would allow to measure the incongruity-resolution theory properties in a different way, which might increase the accuracy of the classifier and regression component. Other computational humor generators and detectors might be useful sources of new metrics [7][11][6][4][14][8][9].

Multi-word Template Value Generation. The presented system assumes for simplicity that template values are single words. Both the discussed metrics and the generated template values have been created for single word template values. In Section 3.3, we already proposed several methods for dealing with multi-word template values using single-words metrics. Another solution would be to introduce more complex metrics, preferably still capable of measuring all properties proposed in the incongruity-resolution theory [26]. In order to successfully generate jokes using multiple words as template values, the template values generator also has to be updated to be capable of proposing such multi-word template values.

Integration in Human-Machine Interface. It would be interesting to integrate GAG, or a more generalized version allowing for more templates, in a human-machine interface. Instead of ratings using a Likert scale, the interface could try to estimate, using existing sentiment analysis techniques, how well each joke was received based on the users reaction and use this as training data for the generation of future jokes. However, such a system would need to distinguish several types of users based on their joke evaluations. As we noted in Section 3.2, the rates of agreement on the rating of the jokes were rather low. It is thus reasonable to assume that the evaluation of humor for all types of users is not predictable using only one general model. To maximize the perceived funniness for each user, it might be interesting to cluster users based on how they perceived previous jokes, and train a model for each cluster.

5.2 Conclusion

In this research, we created a computer program that is capable of learning to generate humor based on human-rated examples. We used humor theory and extended other computational humor research in order to argue, identify and evaluate a set of humorous metrics and other components. These findings are used in the GAG system, which is capable of learning to generate analogy jokes from rated examples. This system shows how machine learning algorithms, more specifically classification and regression algorithms, can alleviate humans from the elaborate task of crafting schemas for humor generation by hand. These algorithms find correlations between metrics applied onto template values used in templates that make a joke humorous. We also showed how the system can be used to help explain in a human-comprehensible way what the most important metrics for a particular set of jokes are. The code of this program is made available online.[13]

We created and published a reusable platform called JokeJudger[14] to collect and rate jokes. We argued why such a platform was necessary for GAG to function optimally. With this platform, we created a dataset of human-rated *"I like my X like I like my Y, Z"* jokes. Such a dataset is useful as it provides ratings

[13] `https://github.com/TWinters/Goofer`
[14] `https://github.com/TWinters/JokeJudger`

that have significantly less exposure bias on their scores than jokes from other sources.

We evaluated the system by comparing jokes generated using classification and using regression with each other. We concluded that the version of GAG using a classification algorithm outperforms the version using a regression algorithm. We also showed that the performance in terms of relative rate of funniness is comparable to existing research, even though we used a more generic approach. We also compared the performance of the jokes generated by GAG with jokes created by our volunteers having similar one-word template value constraints. This led to the conclusion that the jokes generated by GAG are perceived as funny half as often as human-created jokes.

References

1. Heate, B.: Google is looking to creative writers and comedians to help humanize assistant. `https://techcrunch.com/2016/10/10/google-laughsistant/` (2016)
2. Zwaag, G.V.D.: Apple zoekt een grappenmaker voor siri: heb jij genoeg humor om te helpen? `https://www.iculture.nl/nieuws/siri-vacature-grappen-bedenken/` (2016)
3. Binsted, K., Ritchie, G.: An implemented model of punning riddles. CoRR **abs/cmp-lg/9406022** (1994)
4. Manurung, R., Ritchie, G., Pain, H., Waller, A., Mara, D., Black, R.: The construction of a pun generator for language skills development. Applied Artificial Intelligence **22**(9) (2008) 841–869
5. Agustini, T., Manurung, R.: Automatic evaluation of punning riddle template extraction. In: ICCC. (2012) 134–139
6. Petrović, S., Matthews, D.: Unsupervised joke generation from big data. In: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Sofia, Bulgaria, Association for Computational Linguistics (August 2013) 228–232
7. Kiddon, C., Brun, Y.: That's what she said: Double entendre identification. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL), Portland, OR, USA (June 2011) 89–94 `http://dl.acm.org/citation.cfm?id=2002756`ACM ID: 2002756.
8. Mihalcea, R., Strapparava, C.: Making computers laugh: Investigations in automatic humor recognition. In: HLT/EMNLP 2005, Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference, 6-8 October 2005, Vancouver, British Columbia, Canada. (2005) 531–538
9. Stock, O., Strapparava, C.: Hahacronym: Humorous agents for humorous acronyms. Humor-International Journal of Humor Research **16**(3) (2003) 297–314
10. Venour, C.: The computational generation of a class of puns. Master's thesis, Queen's University, Kingston, Ontario (1999)
11. Valitutti, A., Toivonen, H., Doucet, A., Toivanen, J.M.: "let everything turn well in your wife": Generation of adult humor using lexical constraints. In: ACL (2), The Association for Computer Linguistics (2013) 243–248
12. Chandrasekaran, A., Parikh, D., Bansal, M.: Punny captions: Witty wordplay in image descriptions. CoRR **abs/1704.08224** (2017)

13. Justin McKay, B.M.: Generation of idiom-based witticism to aid second language learning. Proceedings of April Fools' Day Workshop on Computational Humour (TWLT 20) (April 2002) 77–87
14. Taylor, J.: Computational Recognition of Humor in a Focused Domain. University of Cincinnati (2004)
15. Shahaf, D., Horvitz, E., Mankoff, R.: Inside jokes: Identifying humorous cartoon captions. In: KDD, ACM Press (2015) 1065–1074
16. Goldberg, K., Roeder, T., Gupta, D., Perkins, C.: Eigentaste: A constant time collaborative filtering algorithm. Information Retrieval **4**(2) (July 2001) 133–151
17. Karpathy, A., Johnson, J., Li, F.: Visualizing and understanding recurrent networks. CoRR **abs/1506.02078** (2015)
18. Karpathy, A.: The unreasonable effectiveness of recurrent neural networks. `https://karpathy.github.io/2015/05/21/rnn-effectiveness/` (2015)
19. Pilato, G., Augello, A., Vassallo, G., Gaglio, S.: EHeBby: An evocative humorist chat-bot. Mobile Information Systems **4**(3) (2008) 165–181
20. Lessard, G., Levison, M.: Computational modelling of linguistic humour, tom swifties. ALLClACIi92 Conference Abstracts (1992) 175–178
21. Raskin, V., Attardo, S.: Non-literalness and non-bona-fide in language: An approach to formal and computational treatments of humor. Pragmatics and Cognition **2**(1) (1994) 31–69
22. Hong, B.A., Ong, E.: Automatically extracting word relationships as templates for pun generation. In: Proceedings of the Workshop on Computational Approaches to Linguistic Creativity. CALC '09, Stroudsburg, PA, USA, Association for Computational Linguistics (2009) 24–31
23. Miller, G.A.: Wordnet: A lexical database for english. Commun. ACM **38**(11) (November 1995) 39–41
24. Krikmann, A.: Contemporary linguistic theories of humour. Folklore: Electronic Journal of Folklore (33) 27–58
25. Waller, A., Black, R., Mara, D.A., Pain, H., Ritchie, G., Manurung, R.: Evaluating the standup pun generating software with children with cerebral palsy. ACM Transactions on Accessible Computing (TACCESS) **1**(3) 1–27
26. Ritchie, G.: Developing the incongruity-resolution theory. In: AISB Symposium on Creative Language: Stories and Humour, Edinburgh, UK (April 1999) 78–85
27. Shahaf, D., Horvitz, E., Mankoff, R.: Inside jokes: Identifying humorous cartoon captions. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '15, New York, NY, USA, ACM (2015) 1065–1074
28. Fellbaum, C.: Wordnet: An Electronic Lexical Database. Christiane Fellbaum. Volume 69. University of Chicago Press (jul 1999)
29. Breiman, L.: Random forests. Mach. Learn. **45**(1) (October 2001) 5–32