Filling the Blanks (*hint*: plural noun) for Mad Libs® Humor

Nabil Hossain*, **John Krumm**†, **Lucy Vanderwende**†, **Eric Horvitz**† and **Henry Kautz***

*Department of Computer Science
University of Rochester
{nhossain,kautz}@cs.rochester.edu

†Microsoft Research AI
Redmond, Washington
{jckrumm,lucyv,horvitz}@microsoft.com

Abstract

Computerized generation of humor is a notoriously difficult AI problem. We develop an algorithm called *Libitum* that helps humans generate humor in a Mad Lib®, which is a popular fill-in-the-blank game. The algorithm is based on a machine learned classifier that determines whether a potential fill-in word is funny in the context of the Mad Lib story. We use Amazon Mechanical Turk to create ground truth data and to judge humor for our classifier to mimic, and we make this data freely available. Our testing shows that Libitum successfully aids humans in filling in Mad Libs that are usually judged funnier than those filled in by humans with no computerized help. We go on to analyze why some words are better than others at making a Mad Lib funny.

1 Introduction

As technologists attempt to build more natural human-computer interfaces, the inclusion of computer-generated humor becomes more important for creating personable interactions. However, computational humor remains a long-standing challenge in AI. Despite decades devoted to theories and algorithms for humor, the best computerized humor is still mediocre compared to humans. Humor requires creativity, sophistication of language, world knowledge, empathy and cognitive mechanisms, which are extremely difficult to model theoretically. A more modest goal for computational humor is to build machines that help humans create humor rather than replace them.

We develop and test an algorithm, called Libitum[1], for a computer-aided approach to humor generation. Its aim is to help a human player fill in the blanks of a Mad Lib® story to make it funny. The algorithm generates candidate words, and its core component is a machine-trained classifier that can assess whether a potential fill-in-the-blank word is funny, based on several features, including the blank's surrounding context. We trained the classifier on Mad Lib stories that were filled in and judged by humans. We note that in our work, we give players, both human and computer, access to the full Mad Lib story, including the sentences surrounding the blanks. In regular Mad Libs, players do not see the surrounding sentences. Figure 1 shows a sentence from a typical Mad Lib, completed by a human player.

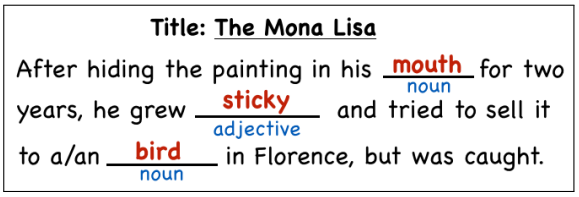

Figure 1: Example of a Mad Lib sentence. The original words for the blanks, in order, were "apartment", "impatient" and "gallery".

The work presented here makes three contributions. The first is the creation of a challenging benchmark for humor generation, a vital aspect of human communication which has received relatively little attention in the NLP community. This benchmark, based on Mad Libs, (i) is challenging, but doable by both humans and machines, (ii) provides quantitative results so that progress can be measured, and (iii) cannot be gamed by trivial strategies, such as filling in random words. The benchmark dataset is annotated and judged us-

[1]Libitum is Latin for "pleased". Mad Lib is a humorous variation of the Latin ad lib, where lib is short for Libitum.

Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 638–647
Copenhagen, Denmark, September 7–11, 2017.

ing Amazon Mechanical Turk, with several steps taken to reduce effects of human variation in humor taste. Our second contribution is that we create and demonstrate a computer-aided humor algorithm that, in most cases, allows humans to generate funnier Mad Lib stories than they can on their own without computer assistance. The third contribution is an analysis of our test data and algorithm that helps to quantitatively explain what makes our results humorous.

Our work goes beyond the modeling and generation of language to simply convey information. Instead, we are trying to create a pleasurable feeling using humor. This is analogous to the Story Cloze Test (Mostafazadeh et al., 2017), where the task is to choose a *satisfying* ending to a story.

We note the previous work called "Visual Madlibs" (Yu et al., 2015). Although its title implies similarity to our work, it is about an image data set augmented with fill-in-the-blank questions, such as "This place is a park."

2 Related Work

Developing a general humor recognizer or generator is hard, and some researchers consider it an AI-complete problem (Stock and Strapparava, 2003), *i.e.*, "solving" computational humor is as difficult as making computers as intelligent as people.

While our goal is generating humor, there is a growing literature of projects for recognizing humor, mostly aimed at specific types of jokes.

Taylor and Mazlack's (2004) work on knock-knock jokes is based on Raskin's (2012) "semantic theory of verbal humor", which says that humor comes from script opposition. Here, the joke posits two different interpretations that oppose each other, resulting in humor. They analyze the two key parts of knock-knock jokes to discover the wordplay that results in the humorous opposition.

Mihalcea and Strapparava (2006) developed a classifier to recognize one-liner jokes by looking for alliteration, antonymy, and adult slang. Compared to different classes of non-jokes, they achieved an accuracy of between 79% and 97%.

Davidov *et al.* (2010) decompose potential sarcastic sentences into specialized parts of speech and match them against sentence patterns that they discovered in their corpus of sarcasm. Like us, they used Mechanical Turk to find ground truth.

Kiddon and Brun (2011) present a classifier that recognizes double entendres that become funny after adding "That's what she said" at the end.

Other humor recognition efforts do not start with constraints on the type of humor they recognize. For instance, Zhang and Liu's (2014) system recognized humorous tweets based on 50 linguistic features from humor theory, achieving a classification accuracy of 82%. Raz (2012) shows how to classify humorous tweets into one of 12 joke categories. Betero and Fung (2016) take on the challenging task of recognizing humor from TV sitcoms, using a neural network. They use the sitcom's laugh track to identify ground truth.

Our work is aimed at generating humor rather than recognizing it. As with previous work on humor recognition, humor generation has been largely limited to specific types of jokes. The three examples below, along with ours, show that automatically generating humor still relies on specializing around a subgenre of jokes with customized approaches that have yet to yield a general method for generating humor.

Binsted *et al.*'s Joke Analysis and Production Engine (JAPE) produced punning riddles, such as:

Question: What do you call a weird market?

Answer: A bizarre bazaar. (Binsted et al., 1997)

JAPE worked by discovering candidate ambiguities in either spelling (*e.g.*, cereal *vs.* serial) or word sense (e.g. river bank vs. savings bank). Evaluated by children, JAPEs jokes were funnier than non-jokes, but not as funny as human-generated jokes.

Petrovic and Matthews (2013) created an algorithm to generate jokes such as "I like my coffee like I like my war, cold," filling in the three blanks. They encoded four assumptions about what makes a joke funny, using discrete probability tables learned from a large corpus of regular text data along with part-of-speech tagging and an estimate of different possible senses of the words. 16% of their automatically generated jokes were considered funny, compared to 33% when the same type of jokes were generated by people.

HAHAcronyn, from Stock and Strapparava (2003) attempted to take existing acronyms (e.g. DMSO for Defense Modeling and Simulation Office) and make an alternate, funny version (e.g. Defense Meat-eating and Salivation Office). Their algorithm keeps part of the acronym and then looks for what to change in the remainder, with goals of different semantics, rhymes, antonyms, and extreme-sounding adverbs.

Besides the novelty of looking at Mad Libs, our work is different in that it seeks to generate humor for *longer* passages than the acronyms, one-liners, and short riddles of previous work.

In addition to algorithmic work, there is a long history of research into general theories of humor (O'Shannon, 2012; Weems, 2014; Wilkins and Eisenbraun, 2009). One of the main thrusts is incongruity theory, which says that a joke is funny when it has a surprise that violates the conventional expectation. According to the Benign Violation Theory (Raskin, 2008), the unexpected must logically follow from the set up and must not be offensive to the reader, otherwise the reader is left confused and the joke is not funny. Similarly, the Sematic Script Theory of Humor (SSTH) says that a joke emerges when it can be interpreted according to two different, generic scripts, one of which is less obvious (Attardo and Raskin, 1991). Labutov and Lipson make a first step at exploiting the SSTH theory to automatically generate two-line jokes (Labutov and Lipson, 2012).

In general, it is difficult to apply these theories directly to humor recognition and generation, however, because they require a high degree of common sense understanding of the world. Because of this, most successful algorithmic work on humor is limited to using relatively shallow linguistic rules on specific types of jokes. This is also true of our work, which concentrates on filling the blanks in Mad Libs, described next.

3 Mad Libs®

Invented in 1953 by Roger Price and Leonard Stern (2008a), Mad Libs is a fill-in-the blank game intended to be humorous. A Mad Lib consists of a story of several sentences and a title. Some of the words are replaced with blanks, each of which has a **hint** belonging to a certain **hint type**, such as a part of speech. Players fill in each blank with a word that agrees with the hint. A player can see only the story's title and the list of blanks with hints. The resulting filled-in Mad Lib is usually funny, because players fill in the blanks with no knowledge of the story (except for its title). The humor comes from the nonsensical filled-in words in the context of a sensible, coherent story. Figure 1 shows part of a filled-in Mad Lib created from a story describing the theft of the Mona Lisa.

Filling in Mad Libs is a novel challenge for automatic humor generation. The title and words surrounding the blanks in a Mad Lib provide a contextual scaffolding that an algorithm can exploit to choose appropriate words for the blanks that make the resulting story humorous.

In order to incorporate such context, our rules for playing Mad Libs differ from the original ones: both our algorithmic and human players are allowed to look at the story as a guide to filling in the blanks. This makes the problem much richer, because players can take advantage of the story's text in choosing which words to fill in. Without looking at the story, our algorithm would be reduced to one that chooses only *a priori* funny words.

3.1 Fun Libs Generation

Mad Libs are copyrighted, and therefore it is difficult to release a data set by using stories from Mad Libs books. Instead we studied original Mad Libs to develop our own dataset, which we call **Fun Libs**. This data set, including filled-in words and funniness assessments from Mechanical Turkers, is available online.[2]

Designing Mad Lib-like stories requires skill, because the Mad Lib context is usually designed in a way to help generate humor. To create our own stories, we first examined 50 Mad Libs from one of the many Mad Libs books (Price and Stern, 2008b). We found that the mean number of blanks, observed words and sentences per Mad Lib were, respectively, 16.0 ($\sigma = 2.25$), 114.84 ($\sigma = 20.58$) and 9.04 ($\sigma = 2.38$). There were 14 unique hint types.

One of our main goals is to build a system that can create meaningful, diverse, and funny stories which apply to a broad audience. However, in pilot tests with human players, we found that six of the original hint types restricted the variety of humor that can be generated by filling in their blanks, by: (i) not affording a variety of possibilities to fill in (hint types: color, silly word, exclamation) and subtlety in humor generation (number), and (ii) generally requiring the audience to have knowledge of cultural references and specifics (person name, place). Hence, we discarded them, leaving eight hint types in our Fun Libs dataset, as shown in Table 1. Some of them have variants such as plurality for nouns and tenses for verbs.

Next, we created our dataset of 50 Fun Libs using simple Wikipedia articles because, similar to

[2]Fun Libs dataset: https://www.microsoft.com/en-us/download/details.aspx?id=55593

Hint Type	Mad Libs	Fun Libs
Noun	7.06	7.00
Adjective	4.06	3.12
Verb	1.22	3.10
Part of the Body	0.98	0.46
Adverb	0.44	0.38
Type of Food	0.22	0.20
Animal	0.20	0.36
Type of Liquid	0.18	0.16
All Blanks	**16.0***	**14.78**

Table 1: Mean hint types per story in the two datasets. *The mean number of blanks in the Mad Libs dataset was computed based on 14 hint types.

Mad Libs, these articles have a title and text. Creating the stories involved finding a Wikipedia article and picking sentences which have potential to generate humor (with very minimal edits such as reducing verbosity), and then replacing some words with blanks in a way such that the overall blank, sentence, word and hint type distributions are similar to their respective distributions in the 50 original Mad Libs. Table 1 shows the means of the hint type distributions for the two datasets. We randomly sampled 40 Fun Libs for training and kept the remaining 10 for evaluation.

4 Data Annotation

With a set of Mad Lib-like stories in place, our next task was to objectively create filled-in stories to use as the basis for the remainder of our analysis. We used Amazon Mechanical Turk workers, bootstrapping from an initial set of filled-in stories, to then finding qualified turker judges, and then finding qualified turker players to fill in the blanks. Our goal was to create a labeled dataset with filled-in blanks and a **funniness grade** for each filled-in word. We selected turkers who are native English speakers (from USA, Canada, Great Britain and Australia), have a HIT approval rate above 97%, and have completed at least 10,000 HITs. We now describe how we further selected qualified turkers and the methods we applied to obtain better quality for the labeled data.

4.1 Judge Selection

To grade how funny a filled-in story is, we needed turker judges who were unbiased referees of general humor. We selected judges before players, because we used the judges to find qualified players. Finding good judges is challenging, because humor is subjective. We launched a qualification task to select qualified judges, with clear instructions on what makes a desirable judge. This task involved giving turkers a set of seven filled-in stories to be graded for funniness. Out of these seven, three were direct excerpts from Wikipedia articles with some words marked as filled-in blanks. Except for designating some words as filled-in, these stories were unaltered from Wikipedia, and therefore were not funny. The remaining four stories were filled in by humans to make them funny. Turkers were allowed to select, for each story, a grade from {0,1,2,3} (a scale which we used throughout our work) described as follows:

0 - Not funny
1 - Somewhat funny
2 - Moderately funny
3 - Funny

We marked the ground-truth grade of the Wikipedia excerpts as 0, and we used volunteers from our research group to decide the ground-truth grade of the other four stories between one and three. We used 60 candidate turkers to do this qualification task, out of which 27 successfully assigned 0 to all the Wikipedia excerpts and 1-3 to the other four stories. We selected all of them as qualified judges. 16 others graded all the Wikipedia excerpts as 0 but considered 1 of the other four stories to be "Not funny". We sorted these turkers using the total Euclidean distance of their grades from the ground truth, and chose the top 13 among them to get 40 judges in total. To make sure that turkers read each story, we also asked a question that could be answered correctly only by understanding the story's context. During our initial tests, we removed two judges who were repeatedly failing to answer these questions correctly and took very little time to grade. Therefore, our final judge pool included 38 judges. This selection process was designed to find judges that were careful, consistent, and representative in objectively judging the funniness of a filled-in story.

4.2 Player Selection

Players are the people who fill in the blanks in a story. Our goal was to find turker players who were good at creating funny stories by filling in blanks. There was no overlap between the qualified judges and the qualified players. To select qualified players, we created two extra fill-in-the-blank stories to be used for player selection alone. We gave both stories to 50 candidate turkers to fill

in. We instructed candidate turkers to avoid using slang words, sexual references or bathroom humor as these are crude, easy techniques for generating humor, and do not require creativity. Further, we required turkers to fill in each blank with exactly one word (using alphabetic characters only) that can be found in a US English dictionary, that is grammatically correct, and that is not colloquial.

To lessen the impact of story contexts and to have a variety in the humor generated, another task was launched with two new stories and 30 new candidate turkers. For each filled-in story, we graded the funniness of each story on our 0-3 scale using 10 qualified judges (described above) to mitigate the effects of variations in humor taste. We ranked the potential players and selected the highest ranked as qualified players. A total of 25 qualified players were obtained from the two batches.

For both the judge and player selection phase, we launched several small pilot tests and used turkers' feedback to design a better data labeling phase, which we cover next.

4.3 Labeling Fun Libs

Our goal in labeling filled-in stories was to assess the funniness of the overall filled-in story, the funniness contribution of each filled-in word, and other aspects of the humor. For each of the 40 stories in the training set, we used 5 players to fill the blanks, giving us a total of 200 filled-in stories. The players also self-graded the humor in their completed stories.

Each of these stories was graded by 9 judges to represent an audience rather than an individual and to reduce effects of different humor tastes. Judges answered the following:

- Which of the filled-in words caused humor.
- Funniness of the story (integer scale of 0-3).
- How coherent the story is, with the filled-in words (integer scale of 0-3).
- To what extent the filled-in words caused the story to deviate from its original topic as suggested by the title (integer scale of 0-3).
- Whether the humor generation technique of **incongruity**[3] was applied by the player.
- A verification quiz, which can be answered using the context of the story, to help ensure that judges read the filled-in story carefully.

[3]Incongruity theory of humor says that a joke is funny when it has a surprise, often at the end, that violates the conventional expectation, often set up at the start (Weems, 2014).

We asked judges about coherence because we expected incongruity would play a significant role in the humor of Mad Libs due to their nature.

Judges and players each received 60 U.S. cents per HIT. We also announced bonuses for the top 10 judges selected based on other judges' agreements with them and the top 10 players based on how funny their filled-in stories were, as graded by the judges. The total Mechanical Turk cost, including the bonuses, was approximately US$2000.

5 Computer-aided Humor

In this section, we will discuss our machine learning approach to generating humorous Fun Libs. First we trained a random forest classifier (Breiman, 2001) to predict whether a filled-in word is funny using features from the story and the title. Then our technique for generating complete, funny Fun Libs involves, for each blank:

1. Use a language model to generate words
2. Keep the top 20 funny candidate words as ranked by our classifier
3. Use humans to decide which candidate to fill in the blank with

5.1 Language Model

To supply reasonable words for each blank, we used the Microsoft Web Language Models API (Wang et al., 2010), trained using Web Page titles and available as a Web service. To generate candidate words for a blank, we use the API's word completion tool to get a list of all possible candidate words using context windows up to four previous words. Next, for each word we computed the **joint probability score**, using the API's joint probability tool to get a probability estimate of how well each candidate fits into the containing sentence. Then we ranked candidates using this score. We expect the words with high scores to be the more fitting (less humorous) words for the context, while those with low scores are more likely to be the incongruous words that may generate surprise or humor when used in the sentence.

5.1.1 Candidate Refinement

Since Web page titles for our language model are not forced to be grammatically correct, the generated words are not all appropriate. Thus we applied the following constraints to get a cleaner candidate list:

- The candidate must be in WordNet (Fellbaum, 1998) or a US English dictionary.
- For a blank that is a sub-category of nouns (*e.g.*, "animal"), the candidate must have the same sub-category as its WordNet hypernym.
- The candidate must have a part of speech tag that agrees with the hint.
- The candidate must not be a word with non-alphabetic characters or a stop word.
- The candidates for nouns and sub-categories have to fit the context in terms of plurality .
- Candidate must not be a slang, adult or bathroom-related word (filtered using a list of 4,012 words), because such words would be too crude, producing shallow, easy humor.

5.2 Features

To predict whether a filled-in word is funny or not, our classifier uses the following ten features extracted from the **(word, story)** pair:

1. Hint type.
2. Length of the word.
3. Language model's joint probability for the containing sentence with the word filled-in.
4. The word's relative position, in terms of joint probability, in the ranked candidate list generated by the language model: a value in the interval [0,1], with 0 implying the candidate has the highest joint probability in the list.
5. Minimum, maximum and average letter trigram probabilities using letter bigram and letter trigram counts from the Google Web Trillion Word Corpus (Brants and Franz, 2006). The purpose of these features is to capture the *phonetic funniness* of words (*e.g.*, "whacking" instead of "fighting").
6. The candidate's similarity to the three contexts — title, overall story, and the containing sentence. This is the cosine similarity of the candidate's word embedding vector with the average word embedding vector for each context. The vectors were computed using GloVe vectors (Pennington et al., 2014) trained with 840 billion tokens from Web data.

5.3 Classification

The full training dataset includes 40 labeled Fun Libs, each filled in by 5 different players, each of which was graded by 9 different judges. We split the data randomly by story titles, keeping 30 for training and 10 in a validation set. This ensures that the classifier does not see the validation stories' contexts during training. Thus, our training set consisted of features and funniness labels for filled-in words from 150 stories. Further, we assigned labels using a **vast majority** vote, *i.e.*, a filled-in word having, out of nine judges' votes:

- six or more "funny votes" is **funny**
- three or fewer funny votes is **not funny**

Otherwise the word was discarded. As a result, the final dataset includes 1939 instances of filled-in words: 1449 for training and 490 for validation.

We experimented with several machine learning models, including linear regression, to predict the number of positive votes for each word. Among these, the random forest classifier worked best. We performed 10-fold cross validation to train this classifier, optimizing based on the F1 score. The results are shown in Table 2. On the validation data, the classifier has an F1 score of 0.756 and an overall classification accuracy of 69.5%. While these quality measures leave room for improvement, they show that the classifier is clearly biased toward choosing funnier words.

We show results from three baseline classifiers in Table 3. Among these, the "Chance" classifiers always predict the most frequent class found in the training set, and "Chance Hint" predicts the most frequent class for each hint type. The other baseline is a Linear SVM classifier trained using the three most important features of the trained random forest as shown in Figure 2. The SVM's learned weights for the similarity features between the word and the containing sentence, the entire story, and the title, respectively, were -0.449,

	Train	**Validation**
Precision (Funny)	0.712	0.715
Recall (Funny)	0.856	0.801
F1 score (Funny)	0.778	0.756
Accuracy	**0.727**	**0.695**

Table 2: Filled-in word classification results.

Baseline	**Train**	**Validation**
Chance	0.571	0.543
Chance Hint	0.595	0.591
Linear SVM (3 feat.)	0.606	0.600

Table 3: Baseline classification accuracies.

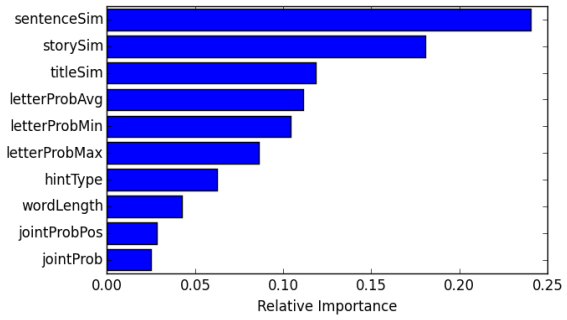

Figure 2: Feature importance for our classifier.

-0.883 and 1.344, showing that funnier filled-in words are similar to the title but not to the body of the story. This suggests that incongruity is playing a significant role in generating humor.

6 Evaluation and Discussion

In this section, we define three approaches for generating humor in Mad Libs, and then we compare and analyze the results we obtain from them. These approaches are:

1. **FreeText**: players fill in the blanks without any restrictions.
2. **LMC** (Language Model + Multiple Choice): for each blank, players choose a word from up to 20 candidate words generated by the language model and sorted by their joint probability score[4].
3. **Libitum**: Similar to the LMC method, except here we rank all the words up to the top 500 words generated by the language model using our classifier, keeping up to the top 20 "humorous" candidates[4].

These methods are designed to study the outcome of humor generated by humans only vs. humans with machine help. The purpose of the LMC model is to study whether the language model alone is a good aid to humans when creating humor, and the benefit of adding machine learning.

For evaluation, we used our ten test Fun Libs, and for each of our three approaches, each of the ten stories was completed by three players. Figure 3 shows the mean grade for these 30 stories. In the figure, the titles are sorted left to right based on the maximum mean story grade among the titles in the Libitum approach. The stories per title are also sorted left to right in descending mean grade. In only one story (ID = 21), the Libitum model receives a lower mean grade than the LMC model, suggesting that adding the machine learning to the language model helps generate significantly more humor than the language model alone. The FreeText model is fairly consistent in generating more humor than the LMC model, which beats the FreeText model in only 7 out of 30 instances. However, the Libitum approach frequently outperforms the FreeText (human only) approach, and it achieves significant gain in generating humor in the stories with the titles "Valentine's Day" and "Cats".

Interestingly, the two stories that received the highest mean grade ("Batman" and "Ducks") are from the FreeText format. This suggests that given more freedom, humans are capable of generating a stronger degree of humor than when they are restricted to a limited number of choices. Excerpts from the best "Batman" story are shown in Figure 4. Here, the strategy employed by the FreeText player is to consistently portray Batman as an obese person obsessed with eating, exploiting the superiority theory of humor[5] (Mulder and Nijholt, 2002). This is remarkable, because it shows how skilled humans are at finding and relating multiple coherent concepts, achieving meaningful and steady humor via continuity, something which is very difficult for machines to do. Much of the humor generated by the Libitum approach here is via incongruity — the filled-in words are quite humorous mainly because they fit their contexts but do not match the expectation of the reader (*e.g.*, Batman is an inefficient superhero). At times, some of the filled-in words in the Libitum approach coherently generate humor, for instance, in the last sentence when Batman is described as wearing a veil to fight acne.

Since each human has a bias towards his/her own understanding of humor, we also studied how the stories appealed to the judges individually by counting the total number of judges for each grade in the three approaches. Table 4 shows the results, where the difference between the mean fun-

[4]In rare cases, for a blank, the language model was only able to generate less than 20 candidate words that pass the candidate refinement step. For such cases, the LMC candidate list was expanded to at least 10 words by adding words randomly from all possible words in WordNet that fit the blank's hint type, if necessary. For Libitum, WordNet was used to randomly add words fitting the blank's hint type to make a list of 50 words that pass the candidate refinement phase. These words were sorted using Libitum and used to ensure the final candidate list had at least 10 words.

[5]Superiority humor comes from the misfortunes or shortcomings of others

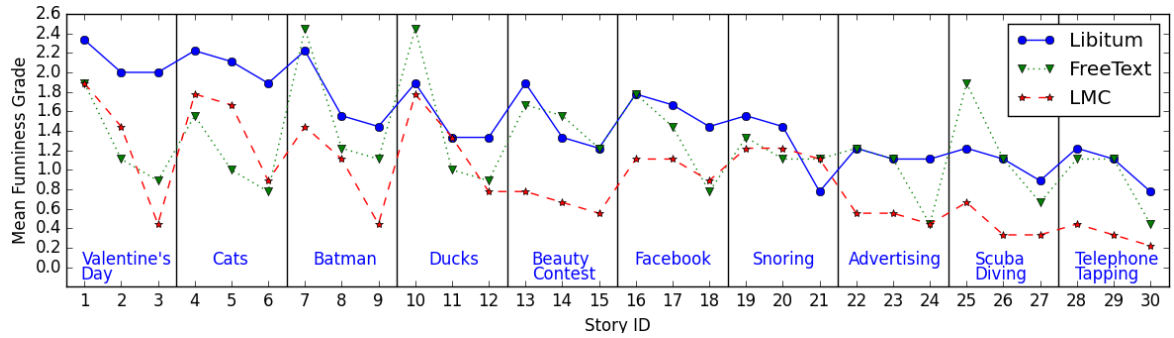

Figure 3: Mean funniness grade for the 30 test stories and their titles. The maximum mean grade for a story in the Libitum format was used to order the titles. For each title, the three stories are sorted in descending mean grade.

Grade	FreeText	LMC	Libitum
0	65	102	37
1	97	100	103
2	84	56	86
3	24	12	44
mean	**1.25**	**0.92**	**1.51**

Table 4: Funniness grades for the 30 stories for the three humor generation formats.

niness grades for each pair of approaches is statistically significant with $p < 0.005$ when a 2-sample t-test was performed. Using Krippendorff's Alpha (Krippendorff, 1970), we also found positive agreements between judges for these and training set ratings.

As expected, the LMC model is the poorest in terms of generating humor. Further, for each non-zero grade, the Libitum model received more votes than the FreeText model. A possible reason is that the judges and players have different perceptions of humor. In the FreeText approach, the common technique employed by players was to use words that are coherent, belonging to a specific topic or domain, and to guide the story towards one conclusion (*e.g.*, the Batman story in Table 4). When this technique worked well, the humor generated was very strong. However, the players have their own biases towards what is humorous, and having more freedom in the FreeText format allowed them to explore their own concept of humor, which could be too narrow to appeal to a broader audience. The Libitum approach, by restricting the players, prevented them from inserting words that they themselves thought were funny, but were not actually funny to people in general.

Table 5 shows passages from stories containing filled-in words that received 9 funniness votes (the maximum possible) from the judges. The story ID and the algorithm used are also provided. Here, in the "Ducks" story, the FreeText player chose to generate humor by developing a steady mockery by satirizing ducks as politicians, whereas the LMC player chose the incongruity approach. The "Beauty Contest" story shows the outstanding skills of humans in generating humor when two blanks are directly connected to each other (*i.e.*, "brawler" and "deadly"). For the same segment, Libitum was also able to aid the players in generating a very funny word, however, the coherence between the blanks does not appear strong. With the computer aided approaches, it is quite difficult to suggest candidates for pairs of (or more) blanks such that the choices are coherent.

6.1 Correlations

Table 6 shows correlations between different ratings by judges (coherence, topic change and incongruity) and the stories' funniness grades. Incongruity had the strongest positive, and statistically significant (with $p < 0.001$), correlation with the graded humor of a story. Coherence also appears very important for generating humor in all the datasets except LMC, where it is difficult to generate coherent words using a language model only. Libitum is likely aiding players in achieving funniness by providing coherent words. In LMC, most of the words generating humor are probably random, incongruous words since the change of topic strongly positively correlates with increasing the humor but coherence does not. Lastly, the player's self grade of humor has no significant relationship with the judges' grade of humor, suggesting that each person has his/her own biases about what is humorous.

Alg.	ID	Text
FText	10	Most ducks are republican (adjective) birds, they can be found in both saltwater and fresh **vodka** (liquid). Ducks are omnivorous, eating **talkative** (adjective) plants . . . People commonly feed ducks in ponds stale **jokes** (noun), thinking that the ducks will like to have something to **ridicule** (verb)
LMC	10	Most ducks are evil (adjective) birds, they can be found in both saltwater and fresh **coffee** (liquid) . . . Some ducks are not evil (adjective), and they are bred and kept by **jesus** (noun).
FText	14	A beauty contest is a/an elaborate (adj) contest to decide which **brawler** (n) is the most **deadly** (adj)
Libtm	15	observational organism **flammable**
Libtm	5	Cats are the most **barbaric** (adjective) pets in the world. They were probably first kept because they ate **humans** (animal). Later cats were bullied (verb [past]) because they are **corrupt** (adjective) . . . Cats are active carnivores, meaning they hunt **online** (adjective) prey.

Table 5: Excerpts from test set stories, showing filled-in words receiving 9 (out of 9) funniness votes (boldfaced) from our judges. The story ID and the approach provided can be used to trace the mean funniness grade of the story in Figure 3.

Batman is a fictional character and one of the most famous / **obese** / sexiest / **inefficient** (adjective) superheroes . . . Batman began in comic books and he was later used / liposuctioned / arrested / imprisoned (verb [past]) in several movies, TV programs, and books. Batman lives in the fictional / edible / holy / abyssal (adjective) city of Gotham. . . . Batman's origin story is that as a/an young / obnoxious / adult / algebraic (adjective) child, Bruce Wayne saw a robber murder / eat / play / prank (verb) his parents after the family left a/an theater / sauna / trail / bag (noun). Bruce decided that he did not want that kind of violence / meal / luck / poem (noun) to happen to anyone else. He dedicated his life to protect / devouring / pronounce / demolish (verb) Gotham City. Wayne learned many different ways to fight / nibble / spell / crochet (verb) as he grew up. As an adult, he wore a/an costume / prosthesis / wig / **veil** (noun) to protect his identity / belly / head / jaw (noun) while fighting crime / gelatin / poverty / **acne** (noun) in Gotham.

Figure 4: Portions of the best "Batman" story, with the filled-in words ordered based on their sources as follows: Original / FreeText / LMC / Libitum. Each boldfaced word was rated "funny" by all 9 judges.

Assessment	Train	FText	LMC	Libtm
Coherence	**0.390**	**0.345**	0.023	**0.578**
Topic change	**0.266**	**0.281**	**0.467**	0.028
Self-grade	0.001	-0.054	0.024	0.151
Incongruity	**0.638**	**0.620**	**0.650**	**0.515**

Table 6: Correlation of different assessments of stories with their funniness grade. The boldfaced and underlined correlations are statistically significant, respectively, with $p < 0.001$ and $p < 0.05$.

6.2 Fully Automated System

We also tested an automated Mad Lib humor generation system, where we filled-in each test blank with the most funny candidate word from Libitum. The results were poor, with the stories having a mean funny grade of 0.80. This is expected, because without human support, Libitum cannot analyze complex word associations in order to come up with words that preserve meaning and coherence. This is evidenced by the statistically significant correlation ($p < 0.001$) of 0.456 between coherence and mean funniness grade scores from judges, suggesting that the judges mostly found those stories funny which were coherent.

7 Conclusion

Our goal was to create and explore a new benchmark for computational humor research. We created a copyright-free dataset that approximately matches the statistical characteristics of Mad Libs. Then we vetted a pool of Mechanical Turk judges and players to create ground truth data. We developed an algorithm called Libitum that generates and classifies potential blank-filling words as either funny or not in the context of a Mad Lib. For each blank, Libitum supplied a list of potentially funny words from which a human could choose. As judged by humans, the Libitum-aided words easily worked better than words from a simple language model and were usually better than even words generated by human players who could fill in the blanks in whichever way they liked.

Our three contributions, the benchmark, Libitum, and the analysis of what makes it funny, advance the state of the art in computer humor by demonstrating a successful computer aided humor technique and quantitatively analyzing what makes for funny fill-in-the-blank words for Mad Libs. These analyses show that coherent stories have tremendous potential in making a Mad Lib humorous, a promising direction for future work.

References

Salvatore Attardo and Victor Raskin. 1991. Script theory revis (it) ed: Joke similarity and joke representation model. *Humor-International Journal of Humor Research*, 4(3-4):293–348.

Dario Bertero and Pascale Fung. 2016. A long short-term memory framework for predicting humor in dialogues. In *Proceedings of NAACL-HLT*, pages 130–135.

Kim Binsted, Helen Pain, and Graeme D Ritchie. 1997. Children's evaluation of computer-generated punning riddles. *Pragmatics & Cognition*, 5(2):305–354.

Thorsten Brants and Alex Franz. 2006. Web 1t 5-gram version 1. Linguistic Data Consortium, Philadelphia.

Leo Breiman. 2001. Random forests. *Machine learning*, 45(1):5–32.

Dmitry Davidov, Oren Tsur, and Ari Rappoport. 2010. Semi-supervised recognition of sarcastic sentences in twitter and amazon. In *Proceedings of the fourteenth conference on computational natural language learning*, pages 107–116. Association for Computational Linguistics.

Christiane Fellbaum. 1998. *WordNet*. Wiley Online Library.

Chloe Kiddon and Yuriy Brun. 2011. That's what she said: double entendre identification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 89–94. Association for Computational Linguistics.

Klaus Krippendorff. 1970. Estimating the reliability, systematic error and random error of interval data. *Educational and Psychological Measurement*, 30(1):61–70.

Igor Labutov and Hod Lipson. 2012. Humor as circuits in semantic networks. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 150–155. Association for Computational Linguistics.

Rada Mihalcea and Carlo Strapparava. 2006. Learning to laugh (automatically): Computational models for humor recognition. *Computational Intelligence*, 22(2):126–142.

Nasrin Mostafazadeh, Michael Roth, Annie Louis, Nathanael Chambers, and James F Allen. 2017. Lsdsem 2017 shared task: The story cloze test. *LSDSem 2017*, page 46.

Matthijs P Mulder and Antinus Nijholt. 2002. Humour research: State of art. Technical report, University of Twente, Centre for Telematics and Information Technology.

Dan O'Shannon. 2012. *What are You Laughing At?: A Comprehensive Guide to the Comedic Event*. A&C Black.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

Sasa Petrovic and David Matthews. 2013. Unsupervised joke generation from big data. In *ACL (2)*, pages 228–232.

Roger Price and Leonard Stern. 2008a. *50 Years of Mad Libs*. Penguin Group.

Roger Price and Leonard Stern. 2008b. *Best of Mad Libs*. Penguin Group.

Victor Raskin. 2008. *The primer of humor research*, volume 8. Walter de Gruyter.

Victor Raskin. 2012. *Semantic mechanisms of humor*, volume 24. Springer Science & Business Media.

Yishay Raz. 2012. Automatic humor classification on twitter. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Student Research Workshop*, pages 66–70. Association for Computational Linguistics.

Oliviero Stock and Carlo Strapparava. 2003. Hahacronym: Humorous agents for humorous acronyms. *Humor*, 16(3):297–314.

Julia M Taylor and Lawrence J Mazlack. 2004. Computationally recognizing wordplay in jokes. In *Proceedings of the Cognitive Science Society*, volume 26.

Kuansan Wang, Christopher Thrasher, Evelyne Viegas, Xiaolong Li, and Bo-june Paul Hsu. 2010. An overview of microsoft web n-gram corpus and applications. In *Proceedings of the NAACL HLT 2010 Demonstration Session*, pages 45–48. Association for Computational Linguistics.

Scott Weems. 2014. *Ha!: The science of when we laugh and why*. Basic Books.

Julia Wilkins and Amy Janel Eisenbraun. 2009. Humor theories and the physiological benefits of laughter. *Holistic nursing practice*, 23(6):349–354.

Licheng Yu, Eunbyung Park, Alexander C Berg, and Tamara L Berg. 2015. Visual madlibs: Fill in the blank description generation and question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2461–2469.

Renxian Zhang and Naishi Liu. 2014. Recognizing humor on twitter. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 889–898. ACM.