LAFF-O-TRON: LAUGH PREDICTION IN TED TALKS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Andrew Acosta

September 2016

COMMITTEE MEMBERSHIP

TITLE:	Laff-O-Tron: Laugh Prediction in TED Talks
AUTHOR:	Andrew Acosta
DATE SUBMITTED:	September 2016
COMMITTEE CHAIR:	Foaad Khosmood, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Clark Turner, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Franz Kurfess, Ph.D. Professor of Computer Science

ABSTRACT

Laff-O-Tron: Laugh Prediction in TED Talks

Andrew Acosta

Did you hear where the thesis found its ancestors? They were in the "parent-thesis"! This joke, whether you laughed at it or not, contains a fascinating and mysterious quality: humor. Humor is something so incredibly human that if you squint, the two words can even look the same. As such, humor is not often considered something that computers can understand. But, that doesn't mean we won't try to teach it to them.

In this thesis, we propose the system Laff-O-Tron to attempt to predict when the audience of a public speech would laugh by looking only at the text of the speech. To do this, we create a corpus of over 1700 TED Talks retrieved from the TED website. We then adapted various techniques used by researchers to identify humor in text. We also investigated features that were specific to our public speaking environment. Using supervised learning, we try to classify if a chunk of text would cause the audience to laugh or not based on these features. We examine the effects of each feature, classifier, and size of the text chunk provided. On a balanced data set, we are able to accurately predict laughter with up to 75% accuracy in our best conditions. Medium level conditions prove to be around 70% accuracy; while our worst conditions result in 66% accuracy.

Computers with humor recognition capabilities would be useful in the fields of human computer interaction and communications. Humor can make a computer easier to interact with and function as a tool to check if humor was properly used in an advertisement or speech.

ACKNOWLEDGMENTS

Thanks to:

- My parents and brother for being with me for the past 5 years of school.
- Dr. Foaad for guiding me through this process, especially since I had 0 knowledge of this stuff a year ago.
- Garrett, Raymond, Anna, and my other friends who I bounced ideas off of and who helped keep me sane.
- Etienne and Jarrel for supplying jokes that actually managed to make it in.
- TED for providing amazing talks and great transcripts. I learned a lot more from this project because of them.
- Loading Ready Run for inspiring the name and the hours of procrastination entertainment.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

"... Pleasure has probably been the main goal all along. But I hesitate to admit it, because computer scientists want to maintain their image as hard-working individuals who deserve high salaries. Sooner or later society will realize that certain kinds of hard work are in fact admirable even though they are more fun than just about anything else. (Knuth 1993)" [17]

Computers are being used for amazing things. There are computers that can verify faces with sight, computers that can determine music by hearing it, and computers that react to touch. But, with all of these advancements, programmers have yet to provide computers with one of the most important, essential, and human sense: a sense of humor. This is a sense so complicated that, while we have hearing aids and glasses, nothing fixes a terrible sense of humor. But maybe modern technology can help.

Much work has been done in the field of computational humor, both in understanding and generating humor. Although there has been success in humor identification, it has always been directed at understanding humor in specific kinds of jokes or environments. For example, understanding if a sentence can be made into a "That's What She Said" joke [9] is great, but their technique only works for that specific kind of joke. Detecting humor in Yelp reviews [22] or on Twitter [35] are major advancements in computational humor beyond the realm of jokes, but contain environment specific humor. After all, who speaks in 140 characters bursts and says "LOL" in real life?

Laff-O-Tron attempts to widen the scope of humor recognition by attempting to

predict laughter in TED talks; a large environment that has gone unexplored.

1.1 Laugh Prediction

Laugh prediction is the ability to, after a line is said, predict how likely it is that it will be followed by audience laughter. In a sense, this is trying to identify whether something was a punchline of a funny joke, without needing to discover what the joke is or why it is funny. This can be done by analyzing paragraph structure, word choice, and humorous stylistics [34].

1.2 Applications of Laugh Prediction

Laughter is an essential part of being human, and humor has been ingrained into our species no matter the culture. On a grand scale, an AI that is able to laugh at a correct time would be a major step towards creating a more human like AI. However, let's look at more immediate uses for laugh prediction as well.

When it comes to researching humor, laugh prediction will allow new perspectives on joke structure and design. This can provide new insight on how to better tell jokes. With some fine tuning, a user might be able to practice their joke delivery style on specific speech environments; such as wedding ceremonies or political speeches.

Laugh prediction could be used to test if something will merit a laugh in fields like advertising. This is important, as the repercussions from poorly made jokes can be massive when it comes to major advertisements. However, this risk comes with high reward, as humorous commercials tend to be memorable.

Lastly, laugh prediction will be able to improve human computer interaction when it comes to communication. It will make the computer appear more human, as it will appear to have a sense of humor. Adding humor into human computer interaction

has been proven to improve people's opinions of it [20] [6]. Imagine a world where you can tell Siri or Cortana a joke, and it would be able to laugh (or groan) accordingly.

1.3 Laff-O-Tron

This work outlines an application, referred to as Laff-O-Tron, which attempts to predict when there will be laughter in a text script. Laff-O-Tron uses machine learning and natural language processing (NLP) to analyze the script for a public speech, and make predictions where an audience would laugh based off of the training scripts. For this thesis, Laff-O-Tron was trained using transcripts from TED Talks and was successfully able to predict laughter. We believe that with a different training corpus, Laff-O-Tron can be used to predict laughter or even applause in other environments such as political speeches.

1.4 Why TED Talks?

In order to train Laff-O-Tron, we needed data where the laughter had already been identified. Rather than manually adding laughter to scripts we already had, we decided to find new scripts that already had laughter marked down. After a thorough investigation, we found that TED Talk transcripts had audience laughter already marked, and decided to use them as our data. This design choice greatly impacts how Laff-O-Tron works, as the environment has an incredible impact on humor. The kind of humor that is prevalent in bars, for example, is different than the humor found at a wedding. Thus, Laff-O-Tron is designed to predict laughter in a live talk environment.

1.5 Contribution

"[I]t is almost impossible to design a general algorithm that can classify all the different types of humor, since even human[s] cannot perfectly classify all of them" [34]. Because of this, work in the field of computational humor has had to focus on specific environments and types of humor. The goal of the Laff-O-Tron project was to show that it is possible for a computer system to be able to predict laughter in an environment fundamentally different than the previously studied environments. This system is built to detect humor in a public talk environment which, unlike previously studied environments, tends to focus more on humorous anecdotes and observations [33].

Chapter 2

BACKGROUND

"If you fail to prepare, you prepare to fail. (Benjamin Franklin)"

Laugh prediction is a topic that falls under the humor recognition part of computational humor. Humor recognition has employed multiple algorithms and methods in order to determine what is humorous and what is not. This is a complicated problem as one man's laugh can be another man's groan. Despite this, people have had success working on systems from detecting one-liners [17] to finding humorous Yelp comments [22]. These achievements are possible thanks to the fields Computational Linguistics and Machine Learning.

Since the goal of Laff-O-Tron is to predict when the audience would laugh in a TED talk, let's first look at TED talks.

2.1 TED Talks

"TED is a nonprofit devoted to spreading ideas, usually in the form of short, powerful talks (18 minutes or less)" [12]. The name TED stands for the original three fields the talks would cover: technology, entertainment, and design. Since its start, however, it has expanded to cover areas of study beyond those original three. With speakers from around the globe with many different fields and backgrounds, TED has covered all kinds of topics from government corruption to why 4am has such a bad reputation. However, TED did not start out this popular, having grown over many years.

"TED was born in 1984 out of Richard Saul Wurman's observation of a powerful convergence among three fields: technology, entertainment and design" [12]. The first TED event included things like Benoit Mandelbrot showing how to map coastlines

using his developing theory on fractal geometry. Unfortunately, the event lost money which caused it to be put on the shelf. In 1990, Wurman and his new partner Harry Marks, tried again and had more success. As TED became more popular and its speaker roster grew, it gained the attention of a private nonprofit foundation known as the Sapling Foundation. The Sapling Foundation saw it as a way to spread great ideas, and acquired it. It has since grown as a non-profit and has evolved to cover more than just talks at global conferences. TEDx, for example, allows independent organizers to create TED-like events in their own communities while TED-ED aims to amplify the ideas of teachers and students worldwide [12].

TED talks come in over 100 languages from all over the globe [12]. However, for this project we focused only on the transcripts of official TED talks and those only in English. This means no TEDx or TED-ED talks. Because TED occasionally has talks where bands play music, some of the talks pulled for this project are musical performances as well. Unfortunately, I am not allowed to post any of the transcripts I used for Laff-O-Tron in their entirety as they are under copyright. However, I have provided a web scraper that can be used to grab transcripts, as well as have pickled data of talks already broken up with features extracted.

2.2 Machine Learning

"Machine Learning is a natural outgrowth of the intersection of Computer Science and Statistics" [19]

Machine learning tries to answer the question: "How can we build computer systems that automatically improve with experience" [19]. Algorithms attempt to accurately predict certain characteristics of data by analyzing past sample data and generating models from it.

We chose to implement machine learning for Laff-O-Tron since it is an effective

method for solving problems that are too complex to manually design algorithms for [19]. Trying to make an algorithm for humor seems impossible to get right. Machine learning algorithms provided a way for us to get accurate enough results without needing to have a perfect algorithm. Once we decided to utilize machine learning, we chose to use a Python library called Scikit-Learn, a powerful tool for the machine learning field, for analyzing the TED Talk data used to train Laff-O-Tron [3].

2.2.1 Supervised Learning

Supervised learning is a machine learning method which trains the AI on data that has **already been labeled**. Pre-labeling the data is where the supervised part comes in, as it involves telling the computer what groups already exist instead of having the computer make the groups itself. For Laff-O-Tron, the data used are chunks of TED Talks that we already labeled as positive or negative. The label was determined by the presence of laughter during or immediately after the chunk was said.

Alternatively to supervised learning, there is the unsupervised learning method. For unsupervised learning, the AI generates models with the initial unlabeled data, and then begins labeling the data itself. This method can be more complex, but it also can be more powerful than its supervised sibling.

Both of these methods hope to create a system that can label new data correctly, as shown in Figure 2.1. However, because supervised learning is more often used for classification, we chose to use that over unsupervised learning [19].

2.2.2 Classification Algorithms and Models

There are a variety of algorithms used to create a classification model, or classifier. "The choice of which specific learning algorithm we should use is a critical step " [10]. Each model has its own strengths and weaknesses. Some algorithms may be more

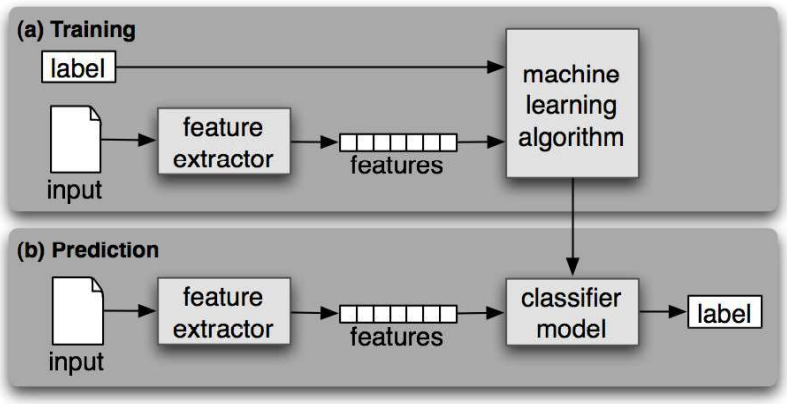

Figure 2.1: Supervised Classification Visualization [5].

powerful but the time they take to run is unfeasible, while others may perform better with different kinds of data. Because of this, multiple models were tried each with differing results.

A machine learning classifier is a classification model built off an algorithm. For example, a Naive Bayes classifier is a classification model that is built from the Naive Bayes algorithm.

Naive Bayes - A simple Naive Bayesian network is made up of directed acyclic graphs with only one parent [10]. Using this, we can make a classification model based on **probability** and **estimation**. We will call the model Naive Bayes from this point forward.

The Naive Bayes algorithm is one of the simplest classification algorithms, and has seen a lot of use. It trains the model by going through feature sets with matching classifications and finding the probability of each feature being associated with each classification. For testing, it goes through a feature set and chooses the most probable classification based on the presence or absence of the features. It is called naive because it assumes that all features are independent of other features [5]. This is almost always wrong, leading to Naive Bayes having less accuracy than other more

complicated methods [10]. However, it has a major advantage in being quick to train [10].

The Naive Bayes equation is as follows:

$$classify(f_1, ..., f_n) = max(C = c) \prod_{i=1}^{n} p(F_i = f_i | C = c) \tag{2.1}$$

In this equation, the feature set of n features being passed in is represented as f_1 to f_n. Thus, f_i is an individual feature, such as the word "psychic" or "pineapple". The class label, such as "murder mystery", is represented as "c".

Let's look at an example where we are trying to figure out the genre of an article based off of its words using Naive Bayes. The feature set is the words in the article, and the possible genres are the classifications. To determine the genre of the article, the Naive Bayes classifier will go through each feature (word) in its feature set. For each feature, it will attempt to see which genres that feature is most associated with and increase those genres probabilities. It will also look at which words are significant but are not in the text, and adjust the probability accordingly. Figure 2.2 shows an example where the words "dark" and "football" appear in the text. "Dark" is often associated with the "Murder Mystery" genre, but it can appear in the other genres [5]. So it increases the probability of "Murder Mystery" slightly by moving the dot closer to it. But "football" is almost always used in the "Sports" genre, so the probability of "Sports" greatly increases.

Maximum Entropy - Another statistical based learning algorithm is Maximum Entropy. It is a machine learning method for classification that uses a model that is similar to the one used by Naive Bayes. However, the Maximum Entropy algorithm uses search techniques to find a set of parameters that will maximize the performance of the classifier, instead of using probabilities to set the model's parameters. To accomplish this, Maximum Entropy chooses the model parameters using iterative optimization techniques, which initializes the model's parameters to random values

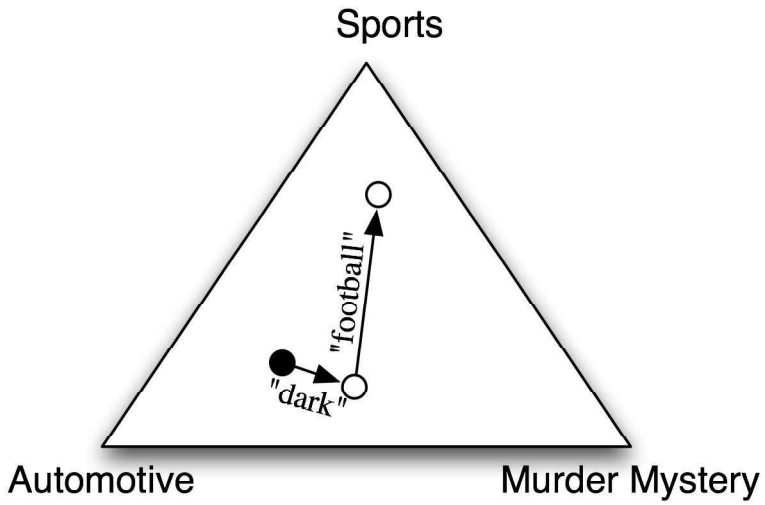

Figure 2.2: An abstract illustration of the procedure used by the Naive Bayes classifier to choose the topic for a document [5].

and then repeatedly refines them to bring them closer to the optimal solution [5].

Maximum entropy is based on the "over-riding principle that when nothing is known, the distribution should be as uniform as possible, that is, have maximal entropy" [10]. For example, consider a classification problem with four classes, and that we know that the word "professor" has a 40% of being in a document with the **faculty** label, and 20% for each of the other classes. Then, if we get a document without "professor" we would guess the uniform distribution of 25% for each class. This model is the perfect example of a maximum entropy model for known constraints and results in a probability distribution across each class. In this example with one constraint, calculating the distribution is easy. However, this gets much harder as more constraints are added [21].

Although more powerful than Naive Bayes on average, Maximum Entropy can take a very long time to run due to its iterative approach. Users should keep this in mind when their training sets, number of features, and number of labels are large [5].

Decision Tree - Decision Trees are another machine learning model that can be

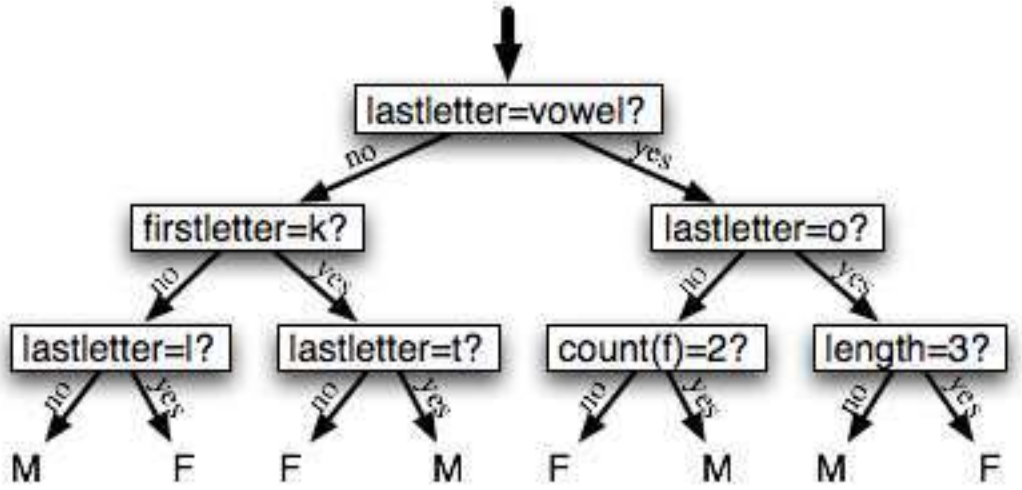

Figure 2.3: Decision Tree model for the name gender task [5].

.

used for classification. Like Naive Bayes, Decision Trees need to use supervised learning. This means that you must train it with pre-classified feature sets [5]. Unlike Naive Bayes, Decision Tree classifiers are **logical** classifiers, and not based on probabilities.

The tree contains decision nodes and leaf nodes. Figure 2.3 shows a decision tree that is used to classify a name as either a male name or a female name. The decision nodes check the features being passed into the tree. In Figure 2.3, these would be nodes like "first letter = k" and "last letter = vowel". Based off if the decision node is true or false, the tree will go down to the next decision node or leaf node. The leaf nodes are what classify the object. In Figure 2.3, this can be seen as the "m" for male name and "f" for female name at the bottom of the tree [5].

Decision trees have the advantage of being relatively simple to understand and are well suited for any problem with many hierarchical organized decisions. However, they do have some disadvantages. Because the decision tree splits the training data to create the nodes, the amount of data used to train some of the lower nodes becomes quite small. This means that the nodes on the bottom might be trained with such a

small portion of the training data that they overfit the training set [5]. Overfitting a training set means that the decision tree learns "patterns that reflect idiosyncrasies of the training set rather than linguistically significant patterns in the underlying problem" [5]. In other words, the decision tree will learn quirks of the training data that don't exist outside of it. Overfitting can be minimized by pruning away the decision nodes that don't help improve the accuracy of the tree.

Boosting - *"Boosting solves hard machine-learning problems by forming a very smart committee of grossly incompetent but carefully selected members" [28].*

Boosting is an approach to machine learning based on the idea that a strong and highly accurate predictor can be created from multiple weak and inaccurate predictors. One of the requirements of boosting is that the weak predictors are at least slightly better than random guessing. Boosting algorithms then run through the training data with the predictors, noting what each predictor misses and adjusting how much it weighs each predictor's decision on certain features. For example, if predictor A is really good at predicting data that has Z features, than A's classification would have more weight when predicting data with Z features. This way, when running the algorithm on new data, it attempts to play on the strengths of each predictor to improve their total value.

The equation for the final predictor is as follows:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x)) \tag{2.2}$$

In this equation, H(x) is the final result of the predictor, which is a class label. For Laff-O-Tron, this is either "Laugh Positive" or "Laugh Negative". T is the number of rounds the boosting algorithm ran, and therefore the number of weak predictors made. The weak predictor is represented as h_t with α_t showing its weight. This weak predictor has a weight between 0 and 1. The "x" represents the vector of features. We used a variety of features, so these could be a mix of numbers or words such as

"4" and "ever".

We chose to use boosting, specifically an algorithm called AdaBoosting, to improve the results of our weaker classifiers. By varying which classifier we boosted and how many of them we trained we were able to optimize the results of one of our classifiers. Although the results across all the classifiers varied, we did have success as some of our highest results came from one of the boosted algorithms.

Other works that used boosting include identifying humor in tweets, which used Gradient Boosted Regression Trees [35].

Random Forest - The Random Forest algorithm creates a classifier in a similar method to AdaBoosting. Using decision trees as the weak classifier, it grows a forest of them each trained on different subsets of data. To make a classification, each tree has a vote, and the class with the most votes is picked. Because so many trees are made, Random Forest classifiers don't have the same problem with overfitting that Decision Tree classifiers do. This means the trees grow as large as need be, and are unpruned. The results are errors rates that compare favorably to AdaBoost, the boosting algorithm we used for this project [32].

The high accuracy and resistance to overfitting that come with Random Forests have made this model very popular among classifiers. Because of this, we choose to use it in this project.

Support Vector Machines - A Support Vector Machines (SVM) is another tool for machine learning. An SVM is a classification model that is "defined by a separating hyperplane" [2]. This means that it will take pre-labeled data and then generate a hyperplane to separate the data by label. This hyperplane attempts to separate the data in a way to maximize the distance from the hyperplane to the nearest training data points. This can be seen in Figure 2.4. The training data comes in the form of vectors, giving it the name Support Vector Machine, which are put into a higher

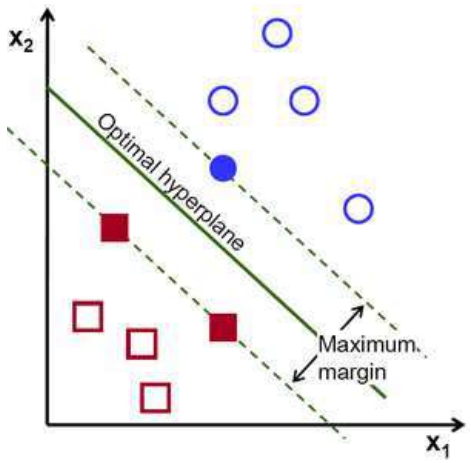

Figure 2.4: The optimal hyperplane separating the blue circles from the red squares [2].

dimensional space by a kernel function. By putting it into a higher dimensional space, the SVM becomes able to separate the data easier.

There are multiple kinds SVMs that vary based on how the kernel function works with the most common being: linear, polynomial, radial basis function, and sigmoid. We chose to use linear.

2.2.3 Scikit-Learn

Scikit-Learn is a Python library filled with tools for data mining and data analysis. It is open source and commercially useable. Scikit-Learn has 6 sets of tools: Classification, Regression, Clustering, Dimensionality Reduction, Model Selection, and Preprocessing [3]. In our case, however, we mostly just used the classification tools. This is where we got the code for AdaBoost, Random Forest, and our Support Vector Machine.

2.3 Computational Linguistics

"Human knowledge is expressed in language. So computational linguistics is very important."Mark Steedman, ACL Presidential Address (2007) [29]

Computational linguistics is an interdisciplinary field concerned with understanding written and spoken language from a computational perspective and building artifacts that can process and produce language [29]. Our method for laugh prediction in particular is concerned with parsing and understanding text, as well as identifying patterns in it. To do so, we need to find a corpus of data to work with. This corpus will have what we use for training and testing. Then we need to process the data, which was done using tools such as the Natural Language Toolkit (NLTK) and Word2Vec. After processing the data, we can attempt to find patterns in it. These patterns will hopefully correspond to what makes the audience laugh.

2.3.1 Corpus

TED Corpus - On TED's website, ted.com, users can watch thousands of TED Talks, as well as other media such as TEDx and the TED Radio Hour. Many of these talks come with interactive transcripts which were transcribed by TED. Users can listen to the talk and follow along with the script making it easier to understand. We created our corpus from these interactive transcripts.

To obtain these transcripts, we created a web scraper which went to TED's website and grabbed the web pages which had transcripts for them. We then passed the website to a parser which grabbed the name of the talk, the speakers, the tags it had (funny, persuasive, etc.), and the text of the transcription. This meant that the talk was split up into paragraphs by the transcriber, and there were sound effects throughout the talk. This included things like applause, music, and laughter. This

was the reason we picked TED talks in the first place, as having laughter already identified allows us to use it to train and test Laff-O-Tron.

We stored each talk in its own text file and kept a file containing the metadata of all of the talks. For each talk we keep track of the text, talk name, speaker names, tags, word count, laugh count, and the location of the first laugh. In total, we have 1787 talks of varying lengths and laugh quantities. They cover a wide variety of subjects from a variety of speakers. In a separate file, we stored information about the laughs across all of the files. This included the number without laughs, the distribution of laughs per file, and data on how long until the first laugh occurred.

The proportion of positive to negative samples varied based off how we broke up the talks by sentence or by paragraph. By sentence there are 3236 positives and 98774 negatives. By paragraph there are 1509 positives and 18087 negatives.

Brown Corpus - "The Brown Corpus was the first million-word electronic corpus of English, created in 1961 at Brown University. This corpus contains text from 500 sources, and the sources have been categorized by genre, such as news, editorial, and so on" [5]. This corpus, pulled from NLTK, was used to collect word frequencies across multiple genres. This was used in order to approximate what words could be considered rare, compared to words that could be considered very common. Unfortunately, the corpus is from 1961 so it is fairly out of date, but we believed it would still be useful for word frequencies.

2.3.2 NLTK

We used the Natural Language Toolkit (NLTK) python library for POS tagging, tokenizing strings, collecting word frequencies, removing stopwords, stemming words, creating N-grams, and accessing the Brown corpus. We also took advantage of its built-in classifiers: Naive Bayes, Decision Trees, Maximum Entropy, and a wrapper

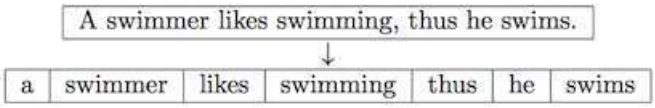

Figure 2.5: Example of tokenizing a sentence [25].

for classifiers from Scikit-learn. NLTK is an open source library that includes "extensive software, data, and documentation" related to the field of Natural Language Processing (NLP) [5]. Originally created back in 2001 at the University of Pennsylvania, it has developed into an easy to use library that allows users to dive into NLP without much experience. NLTK works with Python 2 and 3, and can easily be downloaded from their website: http://www.nltk.org/ [5].

Word Frequency and Hapaxes - Before we start parsing the talk, we can grab the frequency of each word in it. This was done with the help of an NLTK function. Every word was converted to lower case so that the function would see "The" and "the" as the same word. This allowed us to obtain a distribution of how often each word was said by count and by frequency.

Using these word frequencies, we were able to discover which words were common, rare, or only appear once. Words that only appear once are known as hapaxes [5].

Tokenization - To parse text, we need to break it up into tokens using a method called tokenizing.

Tokenizing by words can be thought of as splitting the text into individual words and ignoring all white space. Punctuation is handled differently, based on what the programmer wants. An example can be seen in Figure 2.5. As you can see, there is a token for each word in the sentence and the spaces are ignored. The tokenizer we used treats punctuation as their own words, while the tokenizer in in Figure 2.5 chooses to ignore them. One can also tokenize by characters instead of words.

Stopwords - Stopwords are "high-frequency words like **the**, **to** and **also** that we sometimes want to filter out of a document before further processing" [5]. There is not usually much we can learn from stopwords that can allow us to distinguish between different kinds of texts [5]. It is for this reason that we tend to remove or ignore them when creating word tokens.

Normalizing Text - Often we want to normalize text so that small differences on the same word won't hurt our classifier. This includes things like case collapsing: which turns uppercase letters to lower case letters so that the difference between "The" and "the" can be ignored. "Often we want to go further than this, and strip off any affixes, a task known as stemming" [5]. This allows us to ignore the difference between "running" and "run". It should be noted that stemming words will affect your results from calculating word frequencies and Parts of Speech (POS) tagging.

Ngrams - Once we tokenize the text, we can break the tokens up into ngrams in order to analyze it. Ngrams are just collections of N tokens put together to be looked at. An example can be seen in Figure 2.6. Compared to the tokenization picture, you can see that the unigram (1-gram) chunk is just the sentence tokenized. In the bigram (2-gram) chunk, each bigram is a pair of words that are next to each other. As you can see, the same word shows up in 2 bigrams: one with the previous word and one with the next word. The trigrams (3-grams) chunk shows how each trigram is a triple of words that works like the bigrams above it. As mentioned above, ngrams are based on the tokens provided, meaning that the way we tokenize text determines the ngrams that we analyze.

Parts of Speech - Beyond parsing, we looked at what percent of the words in each paragraph are nouns, verbs, and adjectives. We were able to do this using a Parts of Speech (POS) tagger. A POS tagger looks at a word and tries to decide what kind of word it is. For example, it might guess that "homework" is a noun, and "dreading"

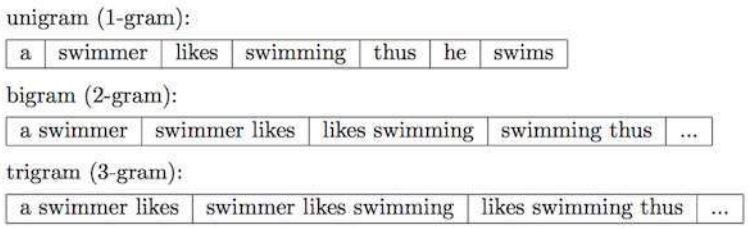

Figure 2.6: An example of breaking up a sentence into ngrams [25].

is a verb. With this, all we had to do was put our tokens through the tagger, and it will tag each token with its respective POS.

2.3.3 Word2vec

"Word2vec is a two-layer neural net that processes text. Its input is a text corpus and its output is a set of vectors: feature vectors for words in that corpus" [4]. Word2vec can be used in Java and Python. Although it is a powerful tool with lots of uses, we only used it for calculating the cosine similarities between words.

Cosine Similarities between Words - Cosine similarity is a method of identifying how similar two words are and is supported by Word2vec. The output ranges from 0 (no similarity) to 1 (total similarity). An example of this can be seen in Figure 2.7, which shows the top 9 closest matches to the word Sweden. As one can see, the nations of Scandinavia and several wealthy, northern European, Germanic countries are among the top nine [4].

2.3.4 TextBlob

"TextBlob is a Python (2 and 3) library for processing textual data" [13]. TextBlob provides an easy to use API that allows users to handle NLP tasks such as noun phrase extraction, sentiment analysis, translation, and more. It plays nicely with

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

Figure 2.7: Example of the top 9 words most similar to "Sweden" using the cosine distance in Word2vec [4].

NLTK, which makes it even more valuable [13]. For this project, TextBlob was used for its sentiment analysis functionality.

Sentiment Analysis - Sentiment analysis attempts to identify the sentiment of a statement. The idea is that the words are "associated with positive or negative sentiments and such measurements reflect the emotion expressed by the writer" [34]. TextBlob has a built in sentiment analysis tool which provides two metrics: polarity and subjectivity.

2.3.5 Stanford Parser

The Stanford parser is a natural language parser built in Java with a few Python ports. "A natural language parser is a program that works out the grammatical structure of sentences, for instance, which groups of words go together (as "phrases") and which words are the subject or object of a verb" [14]. An example of the results of parsing the phrase "Fruit flies like a banana" can be seen in Figure 2.8.

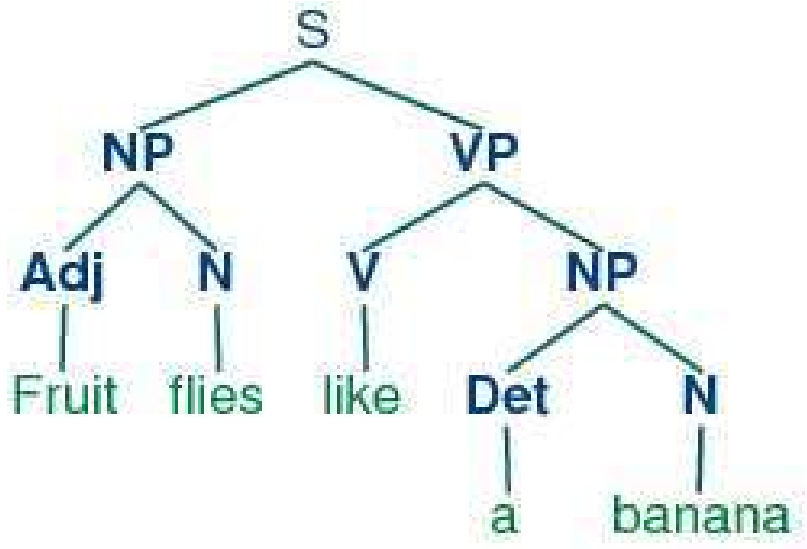

Figure 2.8: A POS tree created from the phrase "Fruit flies like a banana". This is similar to what the Stanford Parser would return [5].

This parser is regularly updated although the ports may not be. We attempted to use a port of the Stanford parser found here github.com/dasmith/stanford-corenlp-python to find out how many ways a sentence could be parsed. For more information on the Stanford parser and all of its features, please visit their website at http://nlp.stanford.edu.

2.3.6 Named Entity Recognition

"The goal of a named entity recognition (NER) system is to identify all textual mentions of the named entities. Named entities are definite noun phrases that refer to specific types of individuals, such as organizations, persons, dates, and so on" [5]. This is useful for normalizing text so that patterns can be caught easier. For example, "Tim Cook" and "Steve Jobs" would both be classified under "Person" and can be replaced with "Person" in the text. A problem would occur, however, when trying to then tag "Apple" as "Company" or "Delicious Fruit!"

NER systems can also help normalize things such as numerical values. For example, "5%" and "25%" would both be classified as "PERCENT".

2.4 Computational Humor

"Computational humor is a branch of computational linguistics and artificial intelligence which uses computers in humor research. It is not to be confused with computer humor (i.e., jokes about computers, programmers, users, and computing)." (Wikipedia)

Computational humor is not a relatively new area, with the first dedicated conference organized in 1996 [8]. This conference was last held again in 2012.

"Work on computational humor can be divided into two classes: humor recognition and humor generation" [23]. Humor generation is the more commonly seen of the two. One of the earliest implementations of humor generation dates back to 1992 with a pun generator [11]. Since then, not only have more pun generators been made, but other flavors of joke generators have been created as well. Here is an example of a pun generated by the software STANDUP, designed to teach children about the English language.

"What do you call a nauseous tree?"

"A sick-amore!" [27]

Joke generators have stuck to jokes with pre-built structures. Instead of generating whole sentences, they just fill in the blanks. This is still very impressive, as it means that the AI understands humor enough to utilize these structures in a humorous way.

Although there has been much work on humor recognition, it is often seen as the harder of the two since humor generation gets the luxury of using the structures previously mentioned [26]. To make up for this, machine learning classification models

have been used to find the differences between humorous and non-humorous text. Unfortunately, this has led to humor recognition focusing on specific structures of humor. For example, one-liner detection is humor recognition that works only on the one-liner structure. This means that jokes that do not utilize this structure will go over this system's metaphorical head. The same is true with a method designed to tell if something is a knock knock joke. Although these are still very impressive, it is important to note that they focus on what the differences between a humorous and non-humorous text are, and not why the text itself is humorous.

2.5 Humor Theory

Humor theory is an age old field that has been traditionally divided into three branches. These are not all of the branches, but only the basic traditional branches. "Most of the humor theories ever proposed are actually mixed theories, and many contemporary researchers believe that humor in its totality is too huge and multiform a phenomenon to be incorporated into a single integrated theory" [18].

2.5.1 Theories of Incongruity, Inconsistency, or Contradiction

Incongruity can be defined as the state of being out of place. Thus, these theories focus more on some objective or characteristics of a humorous text or act. This objective or characteristic seems out of place or unexpected; it seems to contradict the whole. Because of this, this branch of humor theory is more on a cognitive level.

This theory assumes that every humorous act involves two frames of reference that are mutually incompatible, but share a common part. This common part allows someone to shift from one frame to another. This causes the recipient to follow the first frame of reference, maintaining that interpretation until the shift to the second frame occurs and the interpretation fails. This is when some cognitive work occurs

allowing the interpretation to shift to the second frame of reference. It is believed then that "the renewal of understanding is attended by the emotion of surprise and satisfaction, causing the reaction of laughter" [18]. Below is an example that follows this branch of humor theory:

An English bishop received the following note from the vicar of a village in his diocese: "Milord, I regret to inform you of my wife's death. Can you possibly send me a substitute for the weekend" [18]?

2.5.2 Theories of Superiority, Disparagement, or Criticism

This theory addresses the alleged aggressive characteristic of laughter. This is "humor [that] is said to be pointed against some person or group, typically on political, ethnic or gender grounds." Although this branch of humor may seem to be more hostile and destructive, this does not mean that this branch of humor is objectively bad. Insulting others using the cooperative principle of joke-telling can be used to for a social intimacy between all involved. This can be seen when a group of coworkers poke fun at a dictatorial boss. It can also be used to criticize someone or the state of something. The example below is designed to criticize lawyers:

"Criminal lawyer is a redundancy" [17].

2.5.3 Theories of Release, Relief, or Relaxation

This branch of theories focuses more on the psychological effect of humor on the recipient. These effects are generally thought of as a way to convert stress or socially taboo aggressive impulses into something that is more socially accepted. This can be seen when someone is angry or stressed out about a subject, so they make jokes to make them feel better.

For example, someone just dumped might say: "If I ever need a heart transplant, I'd want my ex's. It's never been used."

2.6 Tools and Corpora Used by Other Researchers

The tools and corpora we have not used, but were used by other researchers can be found below in Table 2.1 and Table 2.2.

Table 2.1: This is a table of the corpora used by other researchers, but not us. The table contains the name, the users of the corpora, and a description of it.

Name	Used By	Description
Yelp Data Set Challenge database	Humor Detection in Yelp Reviews	A collection of 1.6 million Yelp reviews by 366,000 users for 61,000 businesses.
Pun of the Day	Humor Anchor Extraction	A public data set constructed from http://www.punoftheday.com/ containing various puns.
One-Liner Data set	One-liner detection, Humor Anchor Extraction	Created by Mihalcea and Strapparava for one-liner detection, this corpus holds one-liners gathered from various Internet sites.
Ap News, New York Times, Yahoo! Answer, and Proverbs	Humor Anchor Extraction	A Data set containing multiple headlines and titles published or found in AP News, New York Times, and Yahoo! Answer. Also has proverbs.
Reuters titles, Proverbs, British National Corpus	One-liner detection	This corpora contains titles of articles published in the Reuters newswire, proverbs collected online, and sentences extracted from the British National Corpus.
Knock Knock Jokes Corpus	Knock Knock Joke recognition	A corpus of 65 knock knock jokes from the "111 Knock Knock Jokes" website.
Erotic Corpus	TWSS	1.5 million sentences from *textfiles.com/sex/EROTICA*

Table 2.2: This is a table of the tools used by other researchers, but not us. The table contains the name, the users of the tool, and a description of it.

Name	Used By	Description
WordNet	Humor Anchor Extraction, One-liner detection, Characterizing Humour, Twitter recognition	WordNet is a large lexical database of English. It is a powerful tool which was used for gathering word meanings to approximate incongruity; find antonymy relations; find human centered words; and to extract words with sexual and political relations.
Subjectivity Lexicon (Wilson et al.,2005)	Humor Anchor Extraction	A word association resource used to gather the Polarity and Subjectivity of words for sentiment analysis
CMU Pronouncing Dictionary	Humor Anchor Extraction	The Carnegie Melon University open-source machine-readable pronunciation dictionary for North American English. It was used to extract phonetic features
Theano Library	Yelp Reviews	A numerical computation library. It was used to construct their learning models
WEKA machine learning package	TWSS	A data mining and machine learning library. It was used for their SVM classifier

Chapter 3

RELATED WORK

"Those who don't study the past will repeat its errors; those who do study it will find OTHER ways to err. (Bob Farrell)"

"Most existing studies on humor recognition are formulated as a binary classification problem and try to recognize jokes via a set of linguistic features" [34]. Other instances of humor recognition focus on a specific environment, such as Yelp reviews or Twitter [22] [35]. All of these are similar to Laff-O-Tron, but take place in different environments and thus must look at different things. So, let's take a look at the various humor recognition research from one-liner identification, to "That's what she said jokes", to investigating humor in Twitter posts.

3.1 One-Liners

Mihalcea and Strapparava were some of the first to attempt to use computational humor for humor detection. They created a detector for one liners such as "Take my advice, I don't use it anyway" [17]. The results were impressive, with accuracies of 84.82% and 96.95% when trying to identify humorous one liners from non-humorous proverbs and news article headlines respectively. This is particularly interesting because it is able to use the context held within the one line to identify if there is a joke [17].

To detect if something is funny, the system looks at humor-specific stylistic features and at the content of the statement [17]. They used a machine learning approach similar to the one we describe for Laff-O-Tron to attempt to detect humor from their content. Their results using content based analysis alone was incredibly high, but

when the humor-specific stylistics were added, the increase was statistically significant. So let's take a look at what stylistics they observed.

First, they decided to look at the stylistic feature of alliteration. This is because "one-liners often rely on the readers' awareness of attention-catching sounds, through linguistic phenomena such as alliteration, word repetition, and rhyme, which produce a comic effect even if the jokes are not necessarily meant to be read aloud" [17]. This is not a trend in just one-liners however, studies found that the "phonetic properties of jokes are at least as important as their content" [17]. Alliteration turned out to be the most useful indicator of humor of all the humor-specific stylistic features. Below are two examples of one-liners with their alliteration underlined:

"Veni, Vidi, Visa: I came, I saw, I did a little shopping" [17].

"Infants dont enjoy infancy like adults do adultery" [17].

They then looked at antonymy in the statement. This is because "humor often relies on some type of incongruity, opposition, or other forms of apparent contradiction" [17]. Unfortunately, accurate identification of all these properties is too difficult to accomplish. So Mihalcea and Strapparava decided to just look at the presence of antonyms in the statement. They also looked at indirect antonymy by checking if the synonyms of the words in the statement had antonyms in the statement. Unfortunately, the list of antonyms was not fully complete, which may have been why antonymy did not perform too well. Although it did help identify one-liners, it was nowhere near as useful as alliteration. However, it did do better than their next humor-specific stylistic feature [17]. Below are two examples of one-liners with their antonymy underlined:

"A clean desk is a sign of a cluttered desk drawer" [17].

"Always try to be modest and be proud of it!" [17]

Adult slang was the last stylistic feature they used. This checked to see if adult slang such as "sex" and "procreation" was present. To do this, they had their system detect sexual-oriented words. This was the least useful feature, but it did help when identifying one-liners. [17] A later experiment used a similar approach to detect when a "that's what she said" joke could be made. [9] Below are two examples of one-liners with their adult slang underlined:

"The sex was so good that even the neighbors had a cigarette" [17].

"Artificial Insemination: procreation without recreation" [17].

3.2 Knock Knock Joke Recognition

Of all the jokes an AI could recognize, one would expect a knock knock joke to be the easiest. There is a simple structure to look for, so it would make sense to end it there. Unfortunately, it is not that easy, as Taylor and Mazlack found out. They created a system that was designed to identify knock knock jokes [31]. To do this they had to look beyond the structure to see if what was said was really a joke or not. Their system needed to check if there was a funny punchline, or it would not be considered a joke. They tested their system with a corpus of real and fake knock knock jokes. The fake jokes had coherent last lines, but they were not actually punchlines. In this example, the first joke is a real knock knock joke, while the second is a fake:

"Joke 6: Knock, Knock

Who's there?

Justin

Justin who?

Justin time for dinner.

Text1: Knock, Knock

Who's there?

Justin

Justin who?

Justin awoke in the middle of the night" [31].

The system was able to recognize 69% of the jokes it was given as real. Furthermore, it was able to recognize 95% of the fake jokes as such. Although system still needs more work, it was a step in humor recognition.

3.3 Double Entendre Detector

In a similar vein to laugh prediction, Kiddon and Brun created a system for predicting if "That's what she said" should be said after a sentence. In 2011, they created a system called DEviaNT that is able to detect if a sentence has enough double entendres necessary to add "That's what she said" to the end of the sentence. They did so by looking at the words used and the structure of the sentence. This is because "That's What She Said Jokes" (TWSS) are "likely to contain nouns that are euphemisms for sexually explicit nouns and share common structure with sentences in the erotic domain" [9].

When looking at the words in the sentence, their system analyzed the sexiness of each word. This means that the presence of sexy words like "hot" or "wet" would

increase the likelihood of it being a TWSS. In order to do this, they trained an AI using their erotica corpus for erotic text, and the Brown corpus for non-erotic text. This allowed the AI to measure the sexiness of words. For detecting the structure of a TWSS, they looked at the amount of punctuation, pronouns, and subjects in the sentence.

Although this is focused on predicting TWSS, this has some connections to our work as well. The way they looked at structure in a TWSS is similar to how we looked at the structure of a segment to determine if the audience would laugh or not. Unfortunately, we chose not to measure the sexiness of each word, but instead how often that word was used in segments that caused laughter. Although two different measurements, there are probably many words that would rank highly in both of our systems.

3.4 Humor Recognition and Humor Anchor Extraction

"Although it is impossible to understand universal humor characteristics, one can still capture the possible latent structures behind humor" [34]

Highly influenced from previous humor recognition works, specifically Mihalcea and Strapparava's one-liner detection, Yang et. al. looked into humor recognition and extracting humor anchors. To identify humor, they looked at "several latent semantic structures behind humor" including "incongruity, ambiguity, phonetic style and personal affect" [34]. This is similar to what Mihalcea and Strapparava did in their analysis of one-liners. However, Yang et. al. took it a step further. They then tried "identifying anchors, or which words prompt humor in a sentence" believing it essential to understand humor in language [34].

A humor anchor is a semantic unit which enables humor in a sentence and appears in the form of words. For example, in "I am happy I know sign language; it is

pretty handy", the humor anchors would be "sign language" and "handy". Words such as "is" and "am" are unable to enable humor by themselves, and therefore are not humor anchors. This holds true for the words "sign" and "language" as well. However, the phrase "sign language" holds a different meaning than the individual words, and when combined with "handy", they are able to enable one of the latent structures previously discussed. In this case, the latent structure may be ambiguity or incongruity, as "handy" causes the person to think about hands, but means something else. It is important to know that the humor anchors are the *minimal complete* set of *meaningful* words that enable the humor. Therefore, if any of the humor anchor words were removed or replaced, then the humor would disappear [34].

To extract anchors, they trained a humor recognition classifier which they claimed provided "decent accuracy" for their environment of one-liners and puns [34]. Then, using parse trees, they generated a list of all *meaningful* words that could be humor anchors. Although this list was *complete*, they then had to shrink it to the smallest list possible to make it *minimal*. To do this, they use a Maximal Decrement method to find the smallest subset from the initial list of humor anchors that the humor recognition classifier still claims as humorous [34].

Their results found that the four latent structures previously discussed were effective at capturing humor. Their method for extracting anchors produced results significantly higher than random chance or by only looking at parts of speech. For example, in the joke "The one who **invented** the **door knocker** got a **No-bell** prize.", they were able to successfully detect the bold words as the humor anchors.

Although their results were positive, there is still a lot of work that could be done for humor anchor extraction. Both humor recognition and humor anchor extraction suffered from three things.

- Being unable to identify phrases such as "meteor shower" in "How does the

earth get clean? It takes a meteor shower". This prevents a connection between "clean", "earth", and "meteor shower".

- A lack of external knowledge. For example, "Veni, Vidi, Visa: I came, I saw, I did a little shopping" is unable to be seen as humorous without knowing the original sentence "Veni, Vidi, Vici: I came, I saw, I conquered".

- A lack of humor categorization can easily confuse a system by combining categories such as wordplay, irony, and sarcasm. Each of those categories is different, and may each need a different method.

3.5 Recognizing Humor on Twitter

Twitter is a fitting environment for trying to capture the humor of Web 2.0. With this in mind, Zhang and Liu attempted a project to recognize humor in Twitter posts, also known as tweets. There were three goals to this project. First, they systematically derived "humor related aspects and features from humor theories". Second, they wanted to identify humor-inducing characteristics on Twitter that distinguish between humorous tweets and non-humorous tweets, as well as humorous tweets and humorous non-tweets. Finally, they wanted to show how to harvest humor data from Twitter [35].

After looking at various theories on Twitter humor and humor in general, Zhang and Liu designated 5 categories of humor features.

- Phonetic Features: these features focus on the sounds the text makes. Similar to other related works, Zhang and Liu created features based off of alliteration and rhyme. This would help them find humor such as that in the joke "Some people come here to sit and think, I come here to shit and stink" [35].

- Morpho-syntactic Features: these features focus on the structure of the text to help identify incongruous structures. This focused on parts of speech patterns and parts of speech chains. The goal of these features was to identify humor in text with strange sentence structure such as "I came, I saw. I cleared the browser history" [35].

- Lexico-semantic Features: These features focus on semantic ambiguity as it often leads to incongruous sentences, which are often found in some humorous texts such as puns and punning. Features that focus on this include: looking at Twitter-specific symbols, lexical chains, and humor themes. The humor themes they focused on were sexual themes and political themes. Sexual themes were examined when looking at one-liners, however political themes were original to this project. The goal with these features were to try and identify humor such as that in "when nothing goes right... go left" [35].

- Pragmatic Features: these features hope to help capture if the text is intended to feel fun. Some pragmatic aspects are useful for identifying things like rhetorical questions such as "why can't I?" which can possibly trigger a feeling of fun. To capture this, they looked at deictics, discourse markers, and speech act cues. These features hoped to identify humor such as that in "If teachers have substitutes why can't I?" [35]

- Affective Features: these features focused on analyzing sentiment. This included looking at polarity, sensitivity, attention, pleasantness, and aptitude. These kinds of features were looked at in many of the other works discussed above.

Zhang and Liu trained a Gradient Boosted Regression Tree classifier, a powerful boosting method based on decision trees as base learners, using each category. Each classifier was trained with only one category so that the performance of each category

can be compared. They found that Lexico-semantic and Morpho-syntactic features proved to be the most powerful features, while Affective and Phonetic proved to be the weakest. Training with all the features created a classifier that was more powerful as a whole, but contained an excess of features. The best results came from a classifier trained with the top 34% of features [35].

3.6 Humor Detection in Yelp

To detect humor in Yelp reviews, Oliveria and Rodrigo decided to use something previously untested in the field of computational humor: deep learning. Using state-of-the-art deep learning, they showed that it is possible to use supervised learning to find what constitutes humor in the context of a Yelp review. To do this, they first use "shallow" learning methods, such as those used in Laff-O-Tron and the works discussed above. These are methods such as using Random Forests and Naive Bayes classifiers. With their "shallow" learning methods, they looked at word vector and bag-of-words features. Oliveria and Rodrigo then built deep feedforward networks on top of these features to measure how much of an effect basic feedforward nets would help in humor recognition. Finally, they used recurrent neural networks and convolutional neural networks to model the sequential nature of a review more accurately [22].

Oliveria and Rodrigo found that when using "shallow" learning methods, they got significantly better results with bag-of-words features than word vectors, showing that that these methods are not good when working in word vector space. However, when they built their deep feedforward networks, they noticed that the performance when using bag-of-words and word vectors surpassed the performance of bag-of-words alone on both "shallow" learning methods and their Deep Nets. This led them to believe that word vectors contain some meaning that deep learning is able to pick

up, but their traditional "shallow" methods cannot. Unfortunately, using recurrent neural networks, in particular GRUs, proved to be no more effective than their Deep Nets, leading them to believe that humor structures are too complex for this method. However, when using convolutional neural networks, they obtained the best performance of all their methods. Their results showed that deep learning can learn more expressive relationships between semantics and humor [22].

Chapter 4

SYSTEM OVERVIEW

Laff-O-Tron uses several ideas taken from previous attempts at laughter prediction. However, TED talks are a completely different environment than what previous works have focused on. This is because "most TED speakers do not tell jokes. Instead of jokes, they use humorous personal anecdotes and observations" [33]. Because of this, Laff-O-Tron looks at experimental talk specific features which other methods could not implement.

4.1 Sentence vs Paragraph Chunks

"It's not the size of the chunk that matters, but how you use it. (Jarrel)"

When parsing a talk into chunks, we had two options: paragraph chunking and sentence chunking. A chunk would be considered laugh positive if there is a laugh during or immediately after the chunk. See Figure 4.1 for an example.

In the results section, we will compare the results for both methods of chunking to see the strengths and weaknesses of each.

4.1.1 Paragraph Chunking

First, we could chunk the talk by the paragraphs designated by the transcriber. This implicitly provides Laff-O-Tron with the background knowledge of when the writer or transcriber feels a distinct section of the talk is done. This is realistic in scenarios where someone would be testing a script they wrote, which would usually be split into paragraphs already.

Figure 4.1: Example of positive and negative chunks. The second chunk is positive since it is immediately followed by laughter. This is indicated by the white arrow. The third chunk is positive since laughter is in the middle of it. The first chunk is negative as there is no laughter in the chunk or immediately after it.

4.1.2 Sentence Chunking

The second option is to chunk it by sentence. This means that each sentence is looked at individually, with a slight input from the previous sentence. This seems more versatile, but it lacks some of the context obtained when looking at a whole paragraph.

4.2 Cleaning Data

When creating training and testing data, some text needs to be removed. Laughter appears in text as "(Laughter)" and is removed along with any punctuation, such as hyphens, surrounding it. Applause, appearing in text as "(Applause)" is also removed, as it is an audience reaction that could unfairly influence predicting laughter. Finally, sound effects often appears in talks being written in parenthesis. These are removed before extracting features.

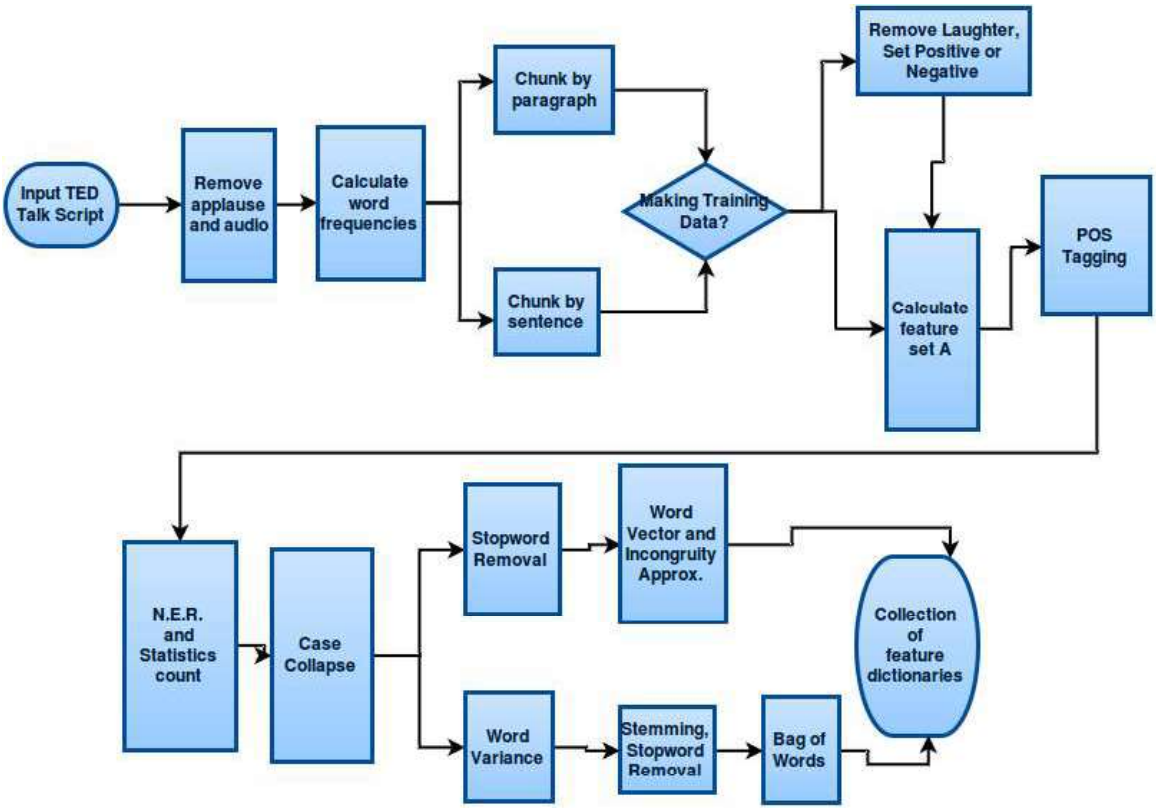

Figure 4.2: Workflow diagram for Laff-O-Tron.

4.3 Workflow

The overall structure for how Laff-O-Tron parses a talk can be seen in Figure 4.2. The output is a collection of feature dictionaries each corresponding to a chunk that was parsed.

4.3.1 Named Entity Recognition

For the NER step, we use a system that we constructed by hand to attempt to identify a subset of named entities: people, statistics, quantities, years, and monetary values. However, this system is far from complete. It is unable to identify numbers that are spelled out and is unable to capture some of the ways speakers gave statistics. While

some speakers would give percentages, others may say something such as "x in y things" or "out of every x things, there are y others". We hope to, in the future, use a powerful NER system to identify entities in talks, as we believe it will help identify analogies; a very popular TED Talk laughter inducer [33].

4.3.2 Stemming

For stemming purposes, we chose to use the Snowball stemmer provided by NLTK. We recommend experimenting with other stemmers in future work.

4.3.3 Feature Set A

Feature Set A is made up of the following features: depth, sentiment features, ambiguity, and the previous laugh features. They are clustered at the same location in the workflow diagram to save space.

4.4 Features

4.4.1 Bag-of-words

The bag-of-words model views the text, in our case chunks, as a bag of words. This means that order and grammar are disregarded, and only which words are present matters. For Laff-O-Tron, we use bag-of-words by having each word, after being stemmed, be its own feature. This model exploits the fact that some words are more often used in a humorous context than others. It also helps find subjects that are not talked about humorously. For example, we found that the subject of diseases is much more commonly used non-humorously.

Bag-of-words was used in multiple previous implementations including knock knock joke recognition and identifying humor anchors [34] [31].

Stopword Removal - We remove stopwords when making the incongruity approximation and bag-of-words features. Doing so improves the accuracy and increases the efficiency of some of the classifiers by shrinking the number of features. This increase in efficiency is essential when running Laff-O-Tron with the full data set on my personal laptop.

4.4.2 Ngrams

While the features from the bag-of-words model explore the idea that certain words are often used more humorously than others, ngram features explores a similar idea with word and letter combinations. For Laff-O-Tron, we have each ngram be its own feature.

Ngrams were previously used for knock knock joke recognition and identifying humorous yelp comments as well [31] [22].

Word Grams - Word grams are used to find word pairs and triples that tend to have more laugh potential. This helps capture phrases that have high and low laugh potential.

Character Grams - While word grams find phrases, useful character grams could help find character combinations. Unfortunately, this proved very ineffective and made the accuracy worse. It also took up too much memory to be able to run once our data set became too large. Because of this, character grams will not show up in the results section.

4.4.3 Parts of Speech Features

Parts of speech features were some of the first features we chose to look at for Laff-O-Tron. After reading about some of the related works, we found that POS was used

into two separate ways. We then implemented both of these methods.

Personal Nouns and Proper Nouns - *"Jokes seem to constantly make reference to human-related scenarios, through the frequent use of words such as you, I, man, woman, guy, etc." [15]*

The first and major use of POS in computational humor research has to do with identifying personal pronouns and proper nouns. The reasoning behind this is that humor is often human centric [15]. This follows the theories of release, relief or relaxation, as well as the theories of superiority, disparagement, or criticism [18]. Because of this numerous other works, such as identifying humor in tweets and identifying humor anchors, have found positive results when looking at personal pronouns and proper nouns [35] [34]. With this in mind, we created features that looks at personal pronouns and proper nouns in a chunk.

The first two features we have are what percentage of the chunk was made up of proper nouns and what percentage of the chunk was made up of personal pronouns. We calculate this by using a POS tagger to identify which words in a chunk were personal pronouns and which ones were proper nouns. We then divide the number of each by the total number of words in a chunk. The last feature is the number of personal pronouns and proper nouns per noun in a chunk. This is calculated by adding together the amount of pronouns and nouns that were in a chunk, and then dividing that by the total number of nouns in that chunk. The resulting ratio is used as a feature in Laff-O-Tron.

Noun, Verb, and Adjective Percentages - When working with short text, sometimes "sentence-level syntactic parsing is not suitable" [35]. This leads us to look at word-level syntax, which is where the second use of POS comes in. This is important as Laff-O-Tron often looks at short text as it uses sentence chunking and runs into short paragraph chunks on occasion. This is why we chose to look at what

proportions of the chunk were nouns, verbs, and adjectives. To do this, we use a POS tagger to tag each word in a chunk, and count how many are nouns, verbs, and adjectives. We then divide those numbers by the number of words in the chunk to get the percentage each POS makes up in the chunk. These percentages are features used by Laff-O-Tron.

4.4.4 Sentiment Analysis

"Humor is essentially associated with sentiment and subjectivity..." [34]

When looking at the humor theories discussed before, one can see that a major part of humor is the human aspect. When detecting one-liners, they noticed how many one-liners are human centric [17]. Thus, others have concluded a major indicator of humor could be subjectivity [34]. It tends to focus on personal beliefs and opinions, such as in the joke "Only adults have trouble with child-proof bottles" [15]. Although this joke has a major incongruity factor, it is still based on the personal beliefs a person has about child-proof bottles.

When talking about the polarity of humor, many have found that humor tends to be negative [15] [18]. "Humorous texts seem to often include negative word forms" such as doesn't, isn't and don't [15]. An example can be seen in "If at first you don't succeed, skydiving is not for you" [15]. However, it doesn't just end at negative verb forms. Negative adjectives with negative connotations show up often as well. These are adjectives like "bad" and "wrong". This can be seen in the joke "When everything comes your way, you are in the wrong lane" [15].

This doesn't mean that humor is a negative thing, but that it tends to shine a negative light upon things. It is for this reason we decided to incorporate sentiment analysis into Laff-O-Tron. Since TextBlob provided a way to measure subjectivity and polarity, we chose to use that.

Subjectivity and Polarity - For each chunk we measure the subjectivity and polarity of the whole chunk and store each of them as a feature. Subjectivity and polarity are each returned from TextBlob as a number between -1 and 1. We round the values up to the nearest tenth as well as increment it by 1 to remove any negative values. However, in order to allow our Naive Bayes classifier to use the feature, we also store a binned value as a feature. To bin the values, we measure if the original value is greater than .1 or less than -.1. If so we set the binned value to 2 and 0 respectively, setting every other value at 1 to imply that the polarity or subjectivity is neutral.

Polarity Change - Aside from the polarity of the chunk, we also investigate how the polarity has changed since the previous chunk. To do this, we take the difference between the current chunk and the previous chunk. We then add 2 to prevent negative values. We make sure that we subtract both values after we increment them both by 1 to prevent poor data. Our goal is to see if changes in polarity, either positively or negatively, could help indicate whether something had laugh potential or not. As with subjectivity and polarity, we round the value to the nearest tenth, and have another feature to store the binned value as well.

This feature is unique to Laff-O-Tron as previous attempts at humor recognition worked on text with very little lines of substance: such as one-liners and knock knock jokes.

4.4.5 Depth

The depth feature indicates how deep into the talk the chunk is. This feature is measured as a percentage to normalize it across varying length talks. This paragraph, for example, would have a depth of .48257. This feature was added under the theory that jokes may occur near the beginning or end of talks more often. This feature is another that is unique to Laff-O-Tron for the same reason Polarity Change.

4.4.6 Length

The length feature is the length of the chunk in words. We chose this thinking that there might be a connection with chunk length and laughter potential. Shorter chunks may have higher laughter potential. But longer chunks may, as opposed to shorter chunks, may be filled with facts and statistics, or dredge on too long to be considered humorous. These were ideas we wanted to test when we created this feature.

4.4.7 Punctuation

We believed that the punctuation used may be a strong indicator that the audience will laugh. We focused on three kinds of punctuation: exclamation marks, question marks, and quotation marks.

Exclamation Points! - One of our features is a Boolean set to true if there is an exclamation point used in the chunk, and false otherwise. Exclamation marks indicate a high energy statement, which led us to think that it would encourage laughter as it is also high energy.

Question Marks? - We include a question mark feature to indicate whether a question was asked in the chunk or not. We were unsure whether questions would have a connection to laughter potential when originally putting in this feature. However, we believed that questions were asked during critical moments in a talk, and added it in. This feature is a Boolean set to true if there is a question mark used in the chunk.

Quotes - *"An easy way to get a laugh without being a comedian or telling a joke is to quote somebody else who said something funny" [33].*

Quoting something funny someone said is an effective way to get a laugh and is something that TED speakers do all the time. These quotes can be from famous

people, anonymous people, family or friends. For example, Carmen Agra Deedy quoted her mother, who said, "I gave shame up with pantyhose-they're both too binding" [33]. As such, we have a feature that is a Boolean set to true if there are quotes used in the chunk.

4.4.8 Word Variance

Word variance is calculated by summing up the number of different words in a chunk and then dividing it by the number of words total. This is intended to give us an idea on how diverse the vocabulary in the chunk was. The idea behind the feature is that the more various the words in the chunk, the more complex it may be. It is also able to capture chunks that are too short to have much repetition. We believe that these short or overly complex chunks have a lower laugh potential.

4.4.9 Incongruity Approximation

"Humor often relies on some type of incongruity, opposition, or other forms of apparent contradiction" [17]

Incongruity, inconsistency, and contradiction are essentials in humor detection as they have their own branch of humor theory. But, as important as they are, what do you look for in text to identify them? This is a problem as "Direct identification of incongruity is hard to achieve..." [34] But there are other things that we can look for that could indicate incongruity. For detecting one-liners, since it "is relatively easy to identify the presence of antonyms in a sentence" using WordNet, they approximated incongruity using antonyms [17]. This worked decently well at the time, but the more modern humor anchor detection took it a step further and decided to "measure the semantic disconnection in a sentence" [34]. For this, they used the Word2Vec package of the Gensim library. We decided to approximate incongruity using this method.

Incongruity Features - To approximate the incongruity of each chunk, we first split the chunk into sentences. Then, for each sentence, we tokenize them into words and then remove all stop words, words that had non-letters in them, and duplicate words. We then calculate the cosine distance between each of the remaining words, so that each word is compared to every other word. We keep track of the max distance and min distance between the words in each of the sentences. We also keep track of the average distance between words in each sentence. The result is three features: the max distance across all sentences, the min distance across all sentences, and finally the average of each sentence averaged.

4.4.10 Swear Words and Adult Slang

"The sex was so good that even the neighbors had a cigarette" [17]

Humor based on adult slang is very popular [17]. Therefore, we followed the example setup by one-liner detection and considered it a feature. We created a small dictionary of adult words, and for each chunk we check to see if any of them are present. If so, we set a feature to true, and false otherwise. By changing the size of the dictionary, the effectiveness of this feature could change.

4.4.11 Ambiguity Approximation

"What do you do with a dead chemist? Barium!"

"Humor and ambiguity often come together when a listener expects one meaning, but is forced to use another" [34]. Because of this, ambiguity falls under the incongruity humor theories. With this in mind, we looked into methods of measuring ambiguity in a sentence. Ambiguity was looked into before by multiple papers, including one-liner detection and humor anchor extraction [17] [34]. One proposed method of capturing ambiguity used sense combination. This can be summed up as

approximating how many different ways each word in a sentence can be interpreted in that sentence, and approximating how many kinds of sentences can be made from it.

We chose to approximate ambiguity by using the Stanford parser to provide us with the number of ways a sentence can be parsed. We assumed that the more ways a sentence can be parsed, the more ambiguous it is. Unfortunately, this did not work, and will not show up in the results section. This is because the Stanford Parser port we used produced the same number of ways to parse each sentence: 50.

4.4.12 Counting Statistics

After using the NER system to identify statistics, we decided to count the number of statistics in a chunk and use it as a feature. We believed that, except in exceptional circumstances, statistics are rarely used for humorous purposes. Using statistics to achieve humor through incongruity is possible, but we believed statistics were a much higher indicator of a non-humorous tone than a humorous one. This is a feature that is linked to the NER system and needs to be able to detect the many different ways to spot a statistic.

4.4.13 Word Frequencies

Word and character frequencies are a metric used in NLP for various purposes. For example, character frequencies can be used to help identify a language [5]. With this in mind, we wondered if word frequencies would play a role in laughter prediction. This idea had two trains of thought. First, we wondered if using words commonly used in everyday chatter would help indicate that the talk was not as serious. Not taking yourself too seriously is a method used to encourage laughter and improve a public speech [33]. Second, we thought that using words rarely used in everyday

chatter may indicate the speaker was setting up for something. This setup could be for a joke or an important topic, and thus may play an important role.

Getting Word Frequencies - To approximate how commonly a word would be used in everyday chatter, we decided to gather the word frequencies from the massive Brown corpus. The Brown corpus contains 1.15 million words, all tagged and categorized into 15 genres including, but not limited to: news, romance, adventure, and science fiction [5]. This is a massive corpus, and we chose to use it because of its sheer size, hoping that it would give us an approximation of realistic word frequencies.

Before getting the word frequencies across the Brown corpus, we case collapse and stem all of the words in the corpus. Doing this ensures that words like "run" and "running" would fall under the same word.

Rare Words - When approximating rare words, we case collapse and stem all the words in a chunk. We also remove any stopwords in the chunk so that they do not skew the word rarity. To approximate how rare the words were in a chunk, we find the highest frequency word and the lowest frequency word, and the average frequency across all words in the chunk. The maximum, minimum, and average frequencies are each a feature for that chunk.

Hapax Legomenon - A hapax legomenon, or hapax for short, is a word that only occurs once within a context. For Laff-O-Tron, a context is the entirety of a TED talk. Although there is case collapsing, there is no stemming done when calculating the hapaxes. During each chunk, we count the number of hapaxes and use that number as another feature. This feature was added because we thought that, just like with looking for rare words, a higher hapax count could be a sign of a setup.

4.4.14 Previous Laughter Features

One good indication of future laughter could be the presence, or lack, of previous laughter. Unfortunately, knowing when the audience previously laughed may be considered by some as cheating, considering Laff-O-Tron should be putting in the laughter itself. None the less, there may be uses of investigating features based on previous laughter being known. For example, if Laff-O-Tron was made to work in real-time, it could update its predictions after each chunk. Another use would be using parts of a previous talk that had proven to be very laugh positive. Laff-O-Tron could then take this into consideration when predicting. Either way, investigating features based on previous laughter proved to have interesting results.

Number of Laughs So Far - Our first feature based on previous laughter is keeping track of the number of laugh positive chunks that had appeared before the current chunk. This number is stored as a feature for the chunk, and proved to be effective by itself.

Chunks Since Last Laugh - Our second feature keeps track of how many chunks there have been since the last laugh positive chunk. We make sure that this does not count the current chunk, as then it would be set to zero on every laugh positive chunk.

4.5 Classifiers Used

For our classifiers, chose to use Naive Bayes; linear SVM; Random Forest; AdaBoost with a decision stump as the weak classifier (which we will refer to as our AdaBoost classifier); and a combination of the Naive Bayes and boosted decision stump classifier.

4.5.1 Maximum Entropy

"I used to be a watchmaker; it is a great job and I made my own hours"

Due to the extremely long run time of Maximum Entropy, it turned out to be too infeasible to run multiple times. Initial testing also revealed that its results were vastly inferior to Naive Bayes or our AdaBoost. Because of this, we abandoned Maximum Entropy early on and kept few results.

4.5.2 AdaBoost and Decision Stumps

When tuning AdaBoost, there are two things to adjust: the weak classifier and the number of weak predictors. For the weak classifier we found that the default decision stump classifier, a decision tree that has a depth of only one, was the best option between decision stumps and Naive Bayes. Using Naive Bayes as the weak classifier, the new classifier still could not take advantage of dependent features, and the accuracy did not improve much. However, with decision stumps, it was able to take advantage of many more features that were dependent. The results were an average of 2% higher accuracy across multiple feature sets.

After picking decision stumps, we experimented with varying amounts weak classifiers. We found that 35 is the minimum amount of weak classifiers, and that there is very little variation up to 100 weak classifiers. However, based on which features were present, the optimal amount of weak classifiers varied. Because of this, we decided to use the default of 50 classifiers.

4.5.3 Combination Classifier

We implemented a combination classifier that used our AdaBoost classifier and Naive Bayes classifier, and marks something as true only if both classifiers say it is true.

Chapter 5

SYSTEM EVALUATION

"It turned out to be perfect and bug free on the first go. (Nobody)"

The goal of Laff-O-Tron is to be able to accurately predict when laughter would show up in a TED talk. This goal entails both correctly predicting a laugh when one would occur, and not predicting a laugh when one wouldn't occur. To test this, we created a collection of positive data and a collection of negative data pulled from the TED talk data we collected. After balancing this data, we then evaluated the system by looking at seven measures across a variety of classifiers. The seven measures we looked at were: accuracy and the precision, recall, and F-measure of both the negative and positive data. We were able to use these measurements to identify both useful and unimportant features, and eventually find the most efficient feature set and classifier combination.

5.1 Data Collection

We used the transcripts of 1787 TED Talks scraped from the TED website. We then limited the talks by length of the script in order to remove any songs, extremely short, or extremely long TED talks that would show up. This is important as songs are not the environment we are studying. Talks with incredibly small scripts are not helpful either, as they are often focused on something the audience sees, which cannot be captured by the transcript well. We ignored long talks to prevent the unusually large talks from overpowering the smaller talks, as they would supply significantly more chunks of data.

The amount of positive and negative data varied based on if we used sentence or

paragraph chunks.

Data by sentence chunks:

- 3236 positive chunks
- 98774 negative chunks

Data by paragraph chunks:

- 1509 positive chunks
- 18087 negative chunks

This massive imbalance of positive to negative chunks is why we decided to balance the data set. We discuss the results of an unbalanced data set in the experiments section.

5.2 Training and Testing

To evaluate Laff-O-Tron, we first went through each TED talk and broke it up into both paragraph chunks and sentence chunks. We skipped chunks that were just applause or audio. Then, if a laugh was recorded in the chunk or immediately after the chunk (which was more often the case), the chunk would be marked Laughter Positive. Otherwise it was marked Laughter Negative. The laughs themselves were also removed from the chunks.

We then, like many previous humor recognition attempts, balanced our data so that the number of positives and negative chunks were even [34] [22] [17]. We balanced the data by removing negative data from our training and testing groups so that it was equal to our positive data. Tests with unbalanced data are discussed in the experiments section. We then trained Laff-O-Tron using approximately 80% of the

balanced data, and then tested it against the other 20% of the balanced data which it had never seen before. When testing performance, we would run the test at least 25 times, with each run having a new balanced training set randomly picked from the training data. This method is known as "bagging".

5.3 Evaluation Methods

We evaluated each classifier and feature set combination using the seven measures mentioned above: accuracy, and the precision, recall, and F-measure of the positive and negative data. Accuracy is simply what percentage of the test data it accurately predicted. It is important to note that a good system should have a higher accuracy than one that tags all of the test data with the same tag. With our balanced data set, this means greater than 50%. Precision "indicates how many of the items that we identified were relevant..." [5]. This can be thought of as "What percent of positives were true positives?" Recall "indicates how many of the relevant items that we identified" [5]. Another way to look at this is "What percent of all of the positives did we correctly identify?" Finally, F-measure "combines the precision and recall to give a single score" [5]. It is defined as:

(2 * Precision * Recall) / (Precision + Recall)

5.4 Finding the Best Feature Set

We followed a procedure to try and find the best feature set without going through every single possible permutation. We first looked at all of the feature types individually and made an initial set of all features whose accuracy or positive precision was significantly above random. After we created this initial feature set, we went through each feature type and either added or removed it to create a new feature set. If this

new feature set showed better results, we would continue to add or remove features from this feature set instead. After having gone through each feature once, we did through each feature again so that the earlier features have a chance to interact with the later features that might not have been present before. The result was a feature set we believe to have approximately the best results.

5.5 Results

To help prevent reporting outlier values, all of our results shown are the average of running each test at least 25 times with randomly chosen test and training data. The variables that were changed across each test were the:

- Classifier used
- Features looked at in testing and training
- The chunk type used

For a table of the complete results of each feature across each classifier, please see Table 8.1 and Table 8.2 in the Full Experimental Results chapter.

5.5.1 Sentences Versus Paragraphs

As a whole, paragraph chunks performed better than sentence chunks for each feature. Paragraph chunks also had a larger impact from adding more of our features. This resulted in the optimal feature set for paragraphs to be larger than that for sentences.

A major difference between paragraphs and sentences are which classifiers proved better for the chunks. The Naive Bayes classifier works much better with sentence chunks than paragraphs chunks. This is because the Naive Bayes classifier would become bogged down by too many features when looking at the bag-of-words and

ngrams features for paragraphs. However, it was not bogged down when looking at sentences, and as such was the most useful for those features. The AdaBoost, SVM, and combination classifiers were the most effective for the paragraph sized chunks, being able to utilize much more features than Naive Bayes.

5.5.2 Best Results

When testing Laff-O-Tron, we collected results from four categories.

- Paragraph sized chunks with features including the previous laugh features was our first category. This category shows Laff-O-Tron with the most support from the text.
- Paragraph sized chunks with features not including the previous laugh features was our second category. This category shows Laff-O-Tron when being given a script to predict laughter in.
- Sentence sized chunks with features including the previous laugh features was our third category. This category allows Laff-O-Tron to be more dynamic as the text input doesn't need to be cut into paragraphs beforehand. This could be useful as a live system with an audience when combined with speech to text functionality.
- Sentence sized chunks with features not including the previous laugh features was our final category. This category setup is the most dynamic but supplies Laff-O-Tron with the least amount of text support.

As can be seen in Table 5.2, category 1 did the best across every measure, while category 4 did the worst. This makes sense, as the lack of text support hindered Laff-O-Tron. The difference between categories 2 and 3 was small, with category 2

Table 5.1: Table containing which features and classifier were used in the optimal setup for each category of our results. An "X" represents that the features were used.

Features	Category 1	Category 2	Category 3	Category 4
Words	X	X	X	X
Ngrams	X	X	X	X
Pronouns	X	X	X	X
POS %			X	X
Sentiment	X	X	X	X
Laugh Features	X		X	
Chunk Length	X	X		
Has !	X	X	X	X
Has ""	X	X	X	X
Incongruity	X	X	X	X
Swearing	X	X	X	X
Frequency			X	
Hapaxes	X	X	X	X
Classifier	SVM	SVM	Combo	Naive Bayes

having a slightly lower positive precision but significantly higher positive recall. We noticed that the impact of using paragraphs was slightly greater than the impact of using previous laugh knowledge. This may have been because features were much more powerful with paragraph chunks, which we will cover in section 5.5.3. Paragraph chunks were also able to use more features effectively, which can be seen in Table 5.1.

5.5.3 Results Across Features

This section analyzes the accuracy and effects of the several features we tested for Laff-O-Tron. We show the results for each feature used by itself in the figures 5.1

Table 5.2: Table containing the results of the optimal setup for each of our categories.

Category	Accuracy	Pos Prec.	Pos Rec.	Pos F1.	Neg Prec.	Neg Rec.	Neg F1
1	0.75	0.75	0.73	0.74	0.74	0.76	0.75
2	0.70	0.72	0.67	0.69	0.69	0.73	0.71
3	0.69	0.76	0.57	0.65	0.65	0.82	0.73
4	0.66	0.65	0.71	0.68	0.68	0.61	0.64

and 5.2. It should be noted that some of the features proved to be more useful when added in than their individual results would imply.

Bag-of-words and Ngrams - The bag-of-words and ngrams features we use proved to be very effective across classifiers and chunk sizes except when using the Naive Bayes classifier with paragraph chunks. We believe this is because paragraph chunks provided too many features to Naive Bayes. It has been found that when a learning algorithm is provided with too many features it may suffer from overfitting [5]. These two of feature sets provide a massive amount of features, as every non-stopword is used at least once in each feature set. However, when the Naive Bayes classifier is used with smaller chunk sizes, it looks at less features and suffers less from overfitting. Looking in Table 8.2 one can see that the positive recall was very high and the negative recall was very low, leaving us to assume that it tagged nearly everything as laughter positive.

Ngrams specifically showed results with high precision and low recall when looking at sentences or when using the Random Forest classifier.

The bag-of-words model working well follows an idea proposed by Oliveria and Rodrigo, when investigating humor in Yelp reviews, that "shallow" learning methods, like the ones we used, work well with bag-of-words models [22].

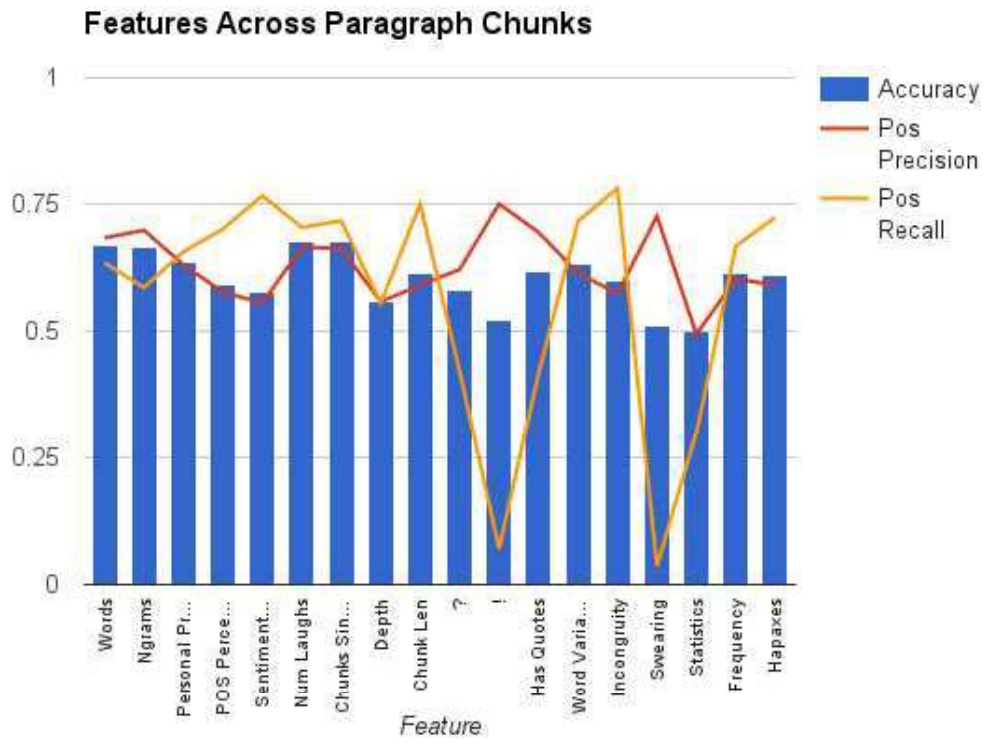

Figure 5.1: The accuracy, positive precision, and positive recall across each feature when using paragraph chunks. The values shown are the best values from all of the possible classifiers for that feature.

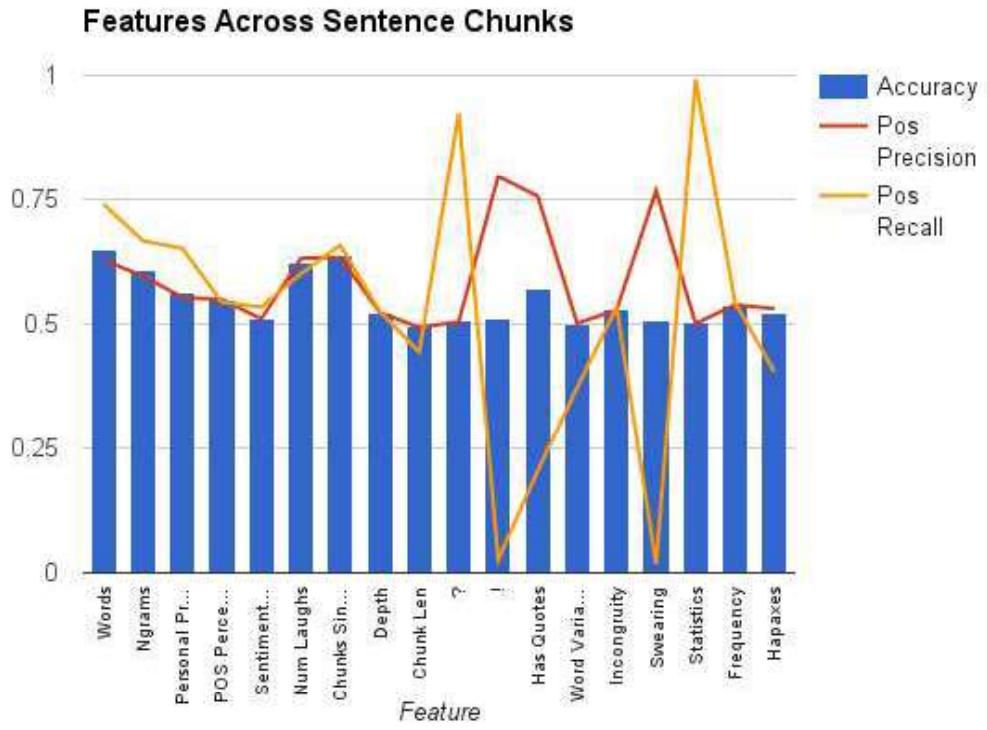

Figure 5.2: The accuracy, positive precision, and positive recall across each feature when using sentence chunks. The values shown are the best values from all of the possible classifiers for that feature.

Parts of Speech Features - The POS features proved to be useful when looking at paragraph sized chunks. However, they were found to be not too helpful when used with sentence chunks. The AdaBoost classifier in particular was found to be able to utilize them the most effectively, and as such the combination classifier used them well too. The personal pronouns feature in particular was found to be very useful to the AdaBoost classifier.

Sentiment Analysis - The results of the features using sentiment analysis proved to be disappointing. Although they helped slightly when incorporated into the final model, it did not provide the level of improvement seen in previous research. We have two ideas for why this may be the case. First, as discussed before, humor tends to have a negative polarity. But, in the TED Talk environment, this was less often the case. It turns out that the humor in our data set was more often marked positive or neutral than negative. We believe this is because of the environment as people are less often to speak negatively about a subject during a public speech like those at TED. Our other idea is that our sentiment analysis tool may need improvement before we could use it for the complex task of humor recognition.

Depth - The depth of the chunk in a talk is not useful in predicting laughter.

Length - The length of a chunk is useful for paragraph sized chunks. When looking at paragraph sized chunks, we more often found that unusually short paragraphs are more often negative than positive.

Length is not useful when looking at sentences.

Punctuation - Our punctuation features have a noticeably high positive precision, with the exception of question marks. Exclamation points have the highest positive precision of the three, but also a really low positive recall. This is because, although it has a strong connection to laughter, it is not often used. Quotations work in a similar way, but quotations are used more often than exclamation points. As such,

the accuracy and positive recall are higher, but positive precision is lower. Because of their strong connection to laughter, we included both quotations and exclamation point features in our final set.

Question marks are not useful with sentence chunks, but had an impact with paragraph chunks. With paragraph chunks, question marks are more often associated with positive data. Despite this, we did not see an increase in our results when using question marks, leading us to believe they were a redundant feature.

Word Variance - Word variance is only useful with paragraph sized chunks. The SVM, AdaBoost, and combination classifiers use this feature the most effectively. Despite this, we chose not to put it in our final feature set as it was shown not to improve accuracy when combined with counting hapaxes. We believe this because they are redundant together.

Incongruity Approximation - Our incongruity approximation features result in approximately random accuracy with sentence chunks, but are effective when using paragraph chunks.

Swear Words and Adult Slang - "Really ****ing high precision. (Andrew Acosta)"

Features involving swear words and adult slang have a really high positive precision but really low recall. This is because adult slang and swear words are rarely used, but when they are, they are usually used for humorous purposes. Other times they are used are when the talk is about an adult subject such as the science of orgasms.

Our dictionary for identifying adult slangs and swear words was made by hand and therefore only covers a subsection of possible words. Adjusting the dictionary can possibly improve the results of this feature so that it has a higher recall.

Counting Statistics - Counting statistics in a chunk was found to not be useful in predicting laughter. This could be a result of how we are identifying statistics, as we

are unable to accurately identify every way a person could state a statistic.

Word Frequencies - The word frequency features are not effective for sentence chunks, but are effective in paragraph chunks. Unfortunately, word frequency features, but not the hapax count feature, in paragraph chunks makes SVM ineffective. Beyond individual values, incorporating word frequency features with any other features drops the accuracy and positive f-measure significantly.

We found that the hapax count feature has a connection between positive data and a high number of hapaxes for paragraph chunks.

Previous Laugh Features - "Laugh features without the laughable results. (An advertisement for these features)"

Using the previous laugh related features noticeably improves the performance of Laff-O-Tron. The laugh features proved to be the most powerful features by themselves, with the exception of the bag-of-words features for sentences. This leads us to believe that an effective tool for predicting laughter is knowledge about previous laughter. This could be because it gives us a glimpse of the atmosphere of the talk.

5.5.4 Results Across Classifiers

Each classifier provided slightly different results across features and chunk sizes. Here we will take a look at how each classifier performed.

Naive Bayes - In our tests, Naive Bayes shone when working with sentence chunks. It made the best use of the bag-of-words features compared to the other classifiers. However, it struggled when using the POS features with sentence chunks. It was also the quickest to become overloaded with too many features, which caused it to fall behind with paragraph chunks as we looked at more and more features. Also, as it uses the naive assumption that features have no connection to each other, it

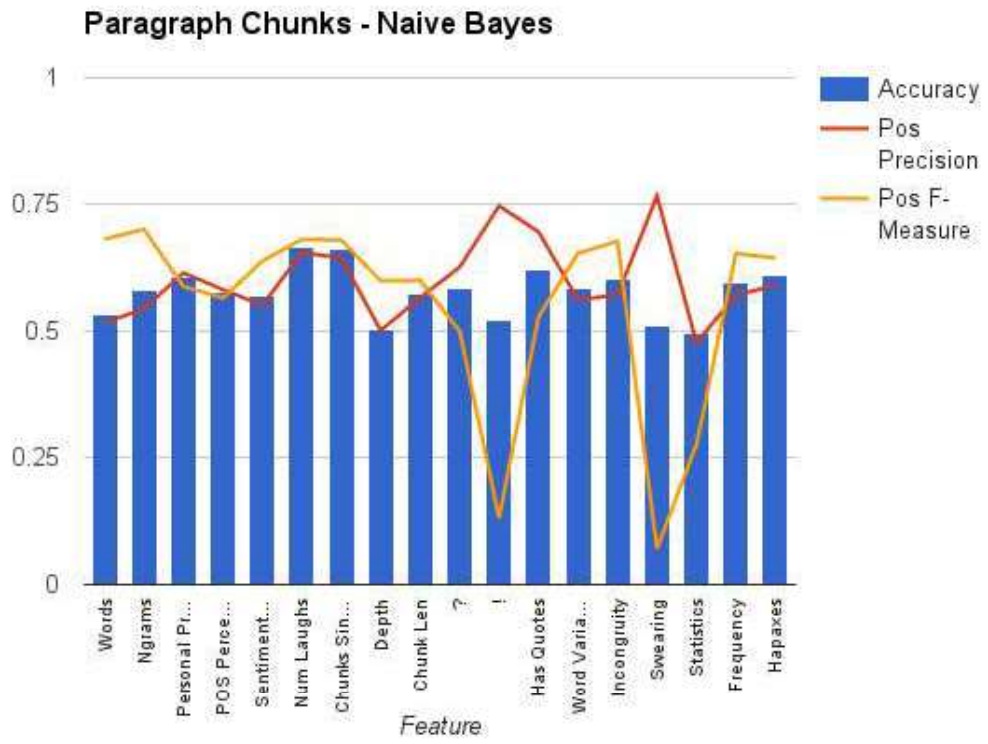

Figure 5.3: The accuracy, positive precision, and positive F-Measure across each individual feature group using the Naive Bayes classifier with paragraph chunks.

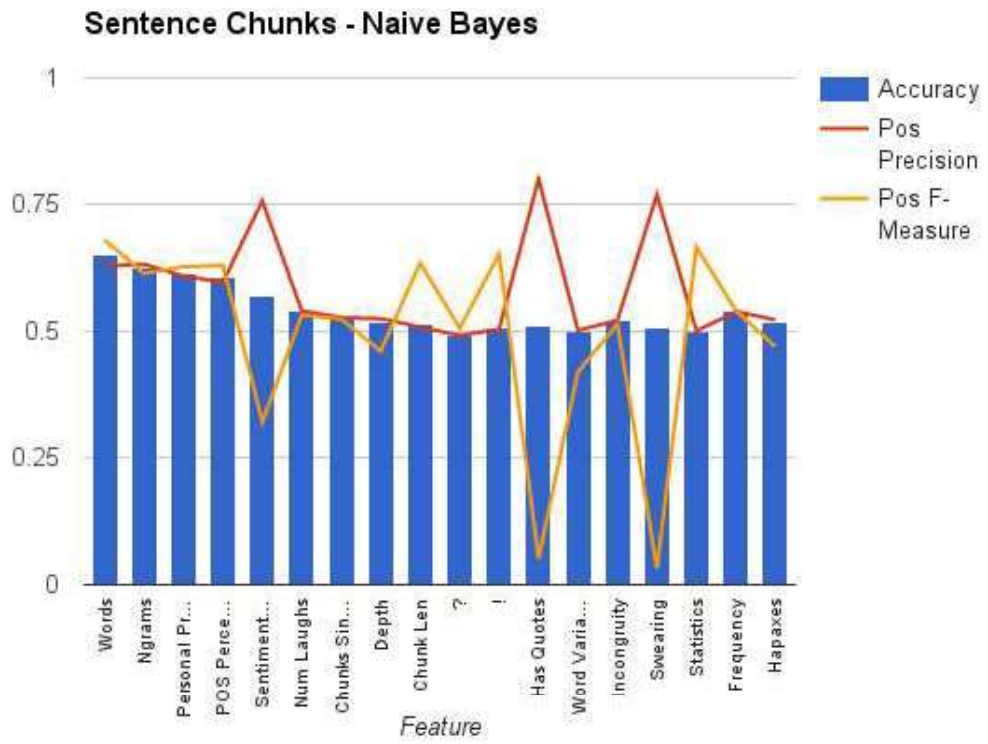

Figure 5.4: The accuracy, positive precision, and positive F-Measure across each individual feature group using the Naive Bayes classifier with sentence chunks.

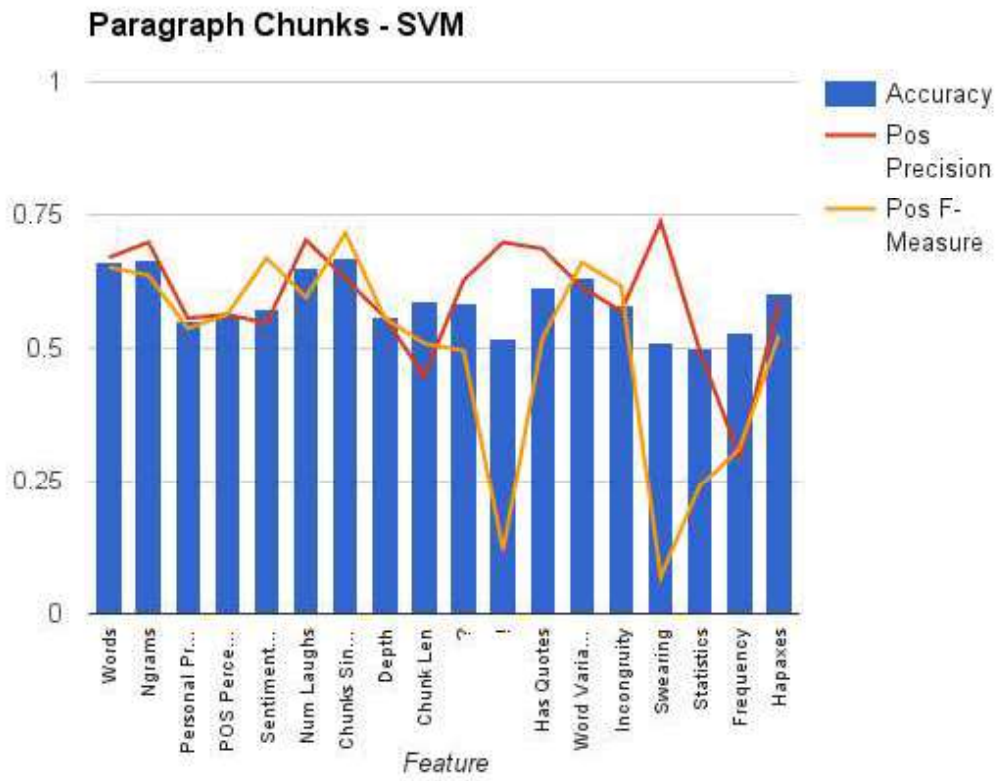

Figure 5.5: The accuracy, positive precision, and positive F-Measure across each individual feature group using the SVM classifier with paragraph chunks.

fell behind even more as we looked at more and more types of features. As a whole, Naive Bayes was useful when looking at sentence sized chunks without many features, which is why it was the best classifier for category 4.

SVM - During testing, the most notable trend amongst the SVM classifier was that looking at the word frequency feature ruined its accuracy. Looking at Figure 5.5 and Figure 5.6, one can see the accuracy is around .5 and the positive F-measure is approximately .3 for paragraphs chunks. Beyond that, it worked very well with paragraph data, often having the highest positive precision. When working with sentences however, it was overshadowed by the AdaBoost classifier.

AdaBoost - AdaBoost consistently provided good results independent of chunk

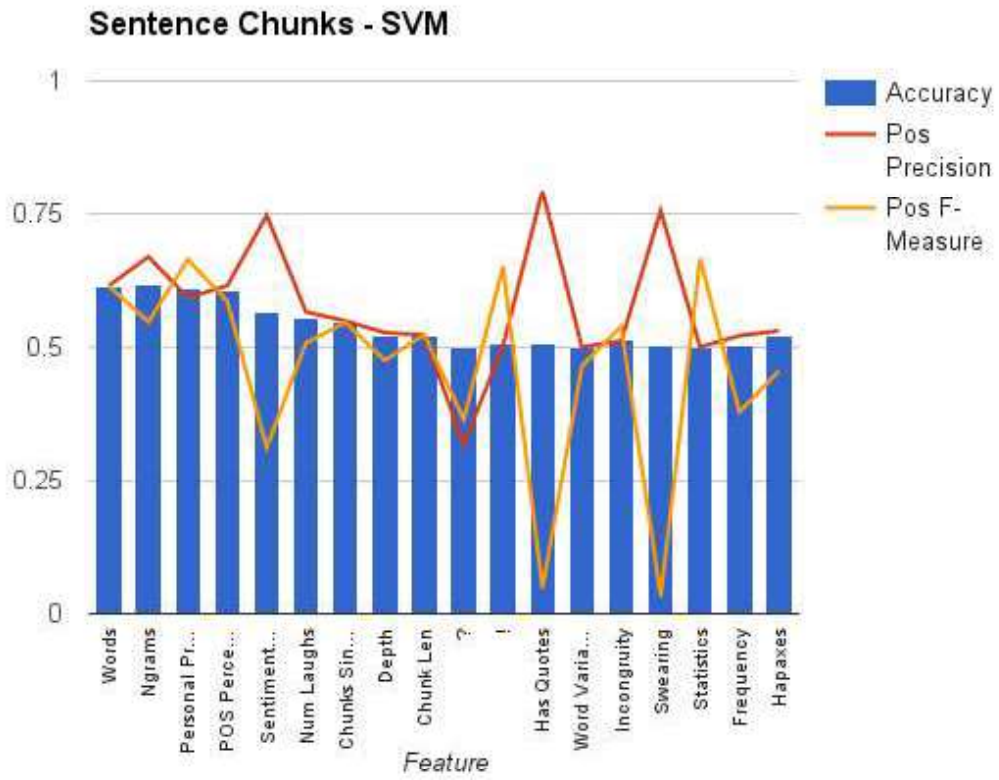

Figure 5.6: The accuracy, positive precision, and positive F-Measure across each individual feature group using the SVM classifier with sentence chunks.

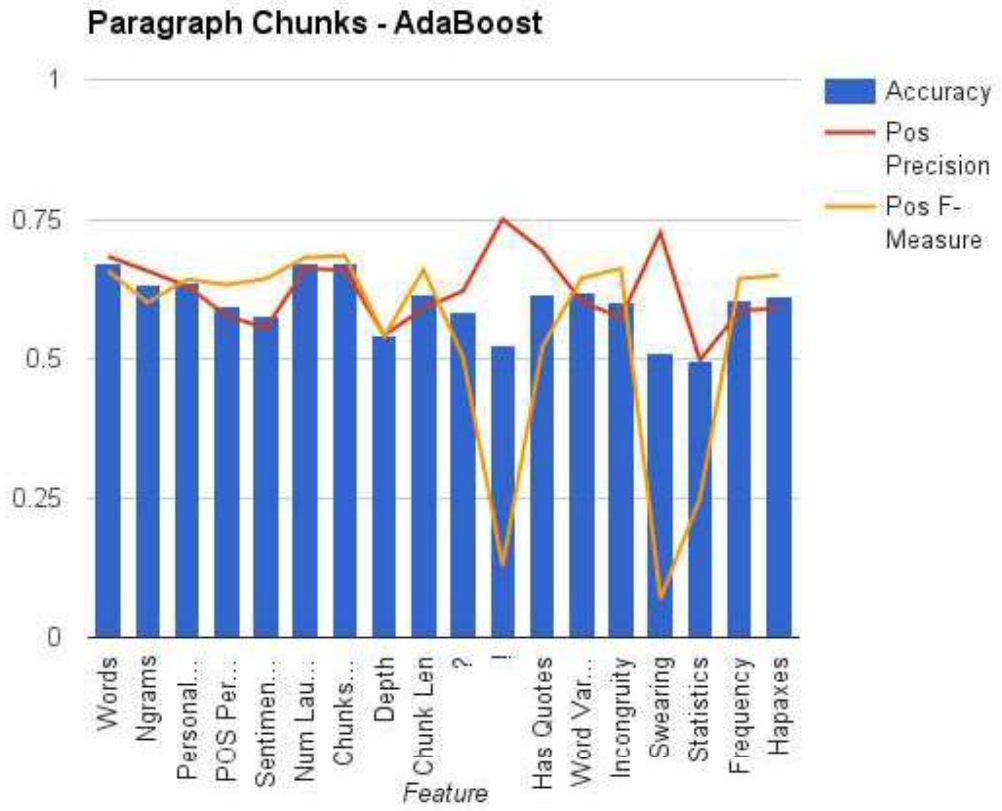

Figure 5.7: The accuracy, positive precision, and positive F-Measure across each individual feature group using the AdaBoost classifier with paragraph chunks.

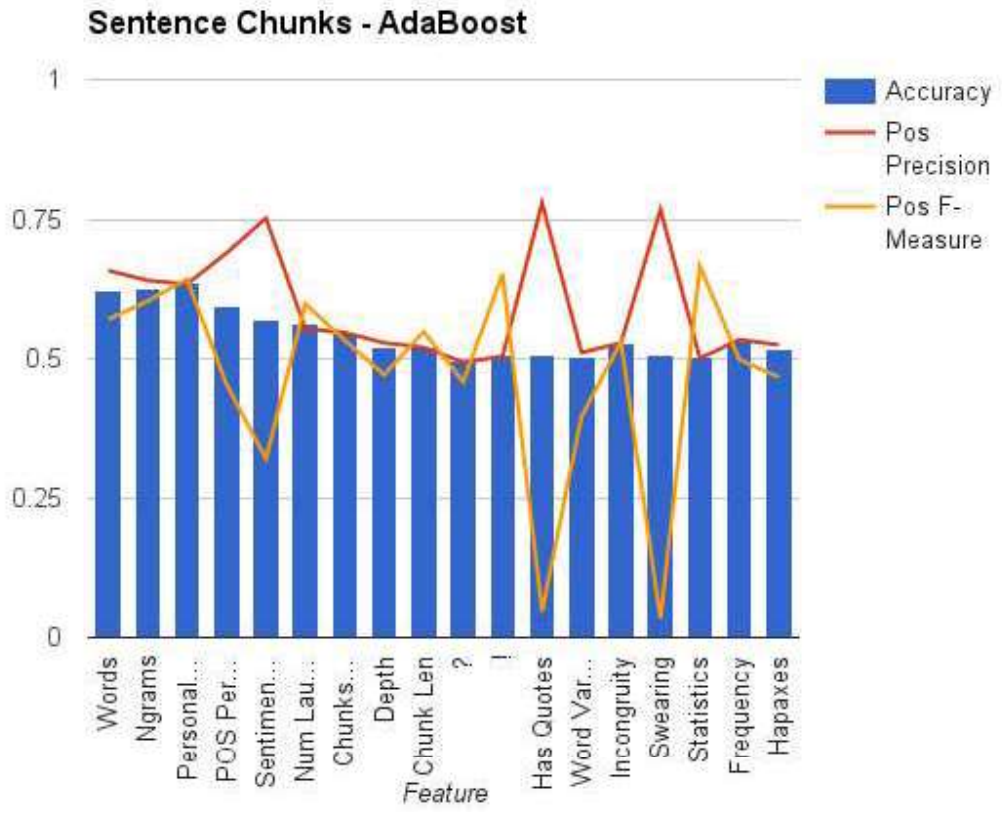

Figure 5.8: The accuracy, positive precision, and positive F-Measure across each individual feature group using the AdaBoost classifier with sentence chunks.

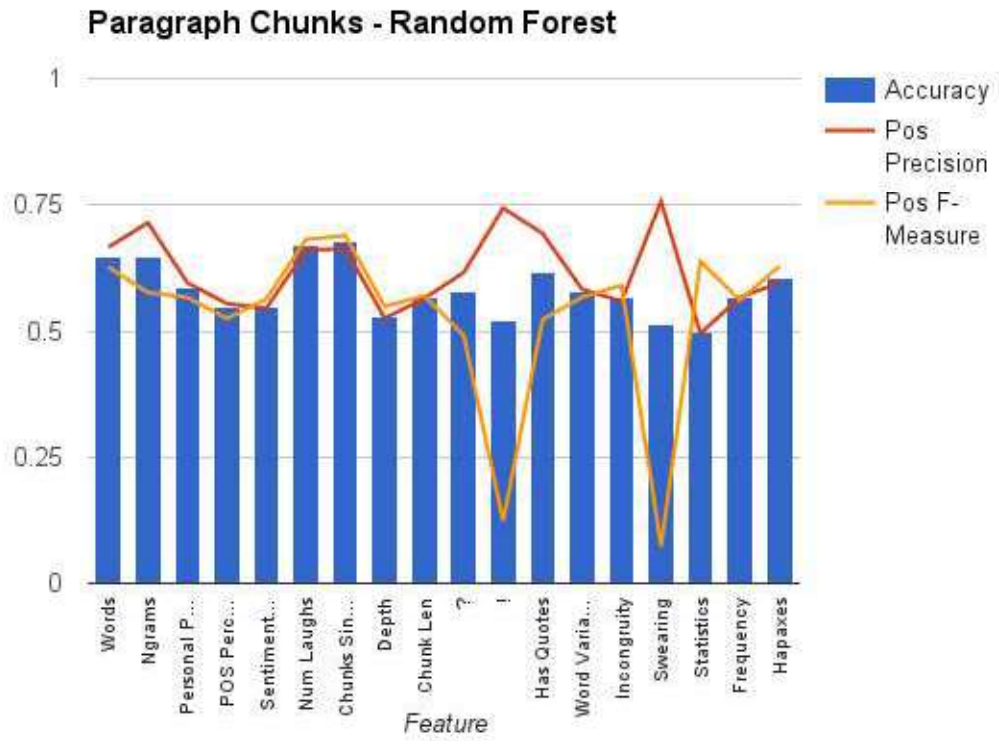

Figure 5.9: The accuracy, positive precision, and positive F-Measure across each individual feature group using the Random Forest classifier with paragraph chunks.

size or number of features while testing. When looking at paragraphs, it almost always had the best or second best results, combination classifier excluded. This was the same when looking at sentence chunks. As such, if any future works could only use one classifier, it should be AdaBoost.

Random Forest - The Random Forest classifier did not perform noticeably well when compared to the other classifiers during testing. When looking at paragraph chunks and combining multiple features, it would do better than the Naive Bayes classifier, but wouldn't come close to the results of AdaBoost or SVM. When using sentence chunks, Random Forest produced results close to Naive Bayes or AdaBoost, but never exceeded them.

Combination Classifier - Combining the AdaBoost classifier and Naive Bayes clas-

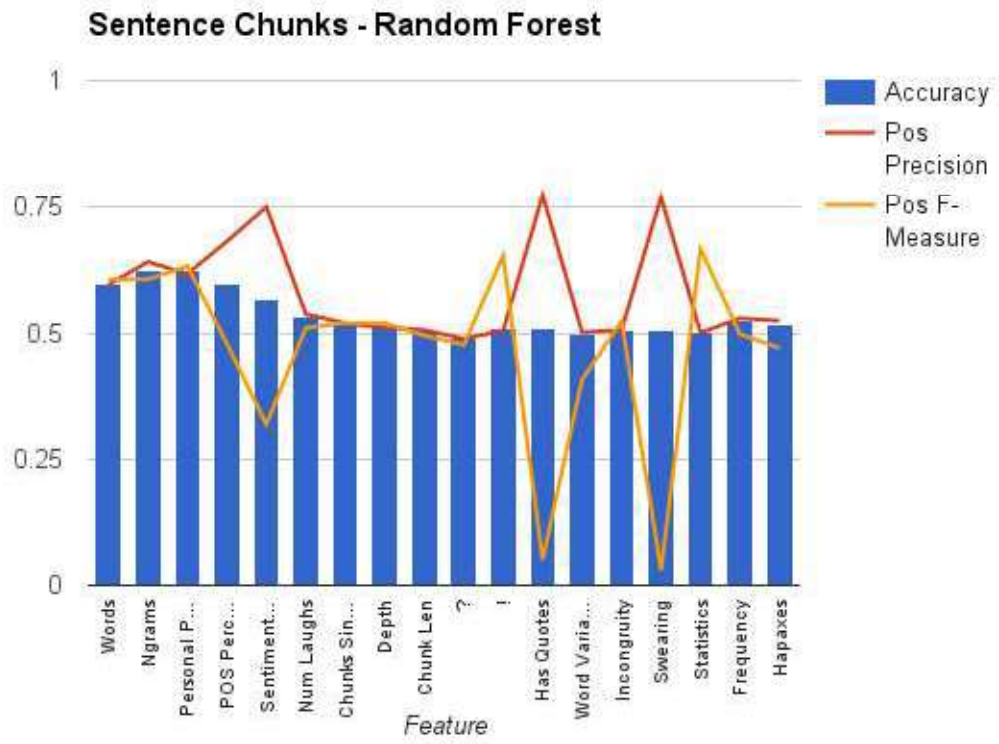

Figure 5.10: The accuracy, positive precision, and positive F-Measure across each individual feature group using the Random Forest classifier with sentence chunks.

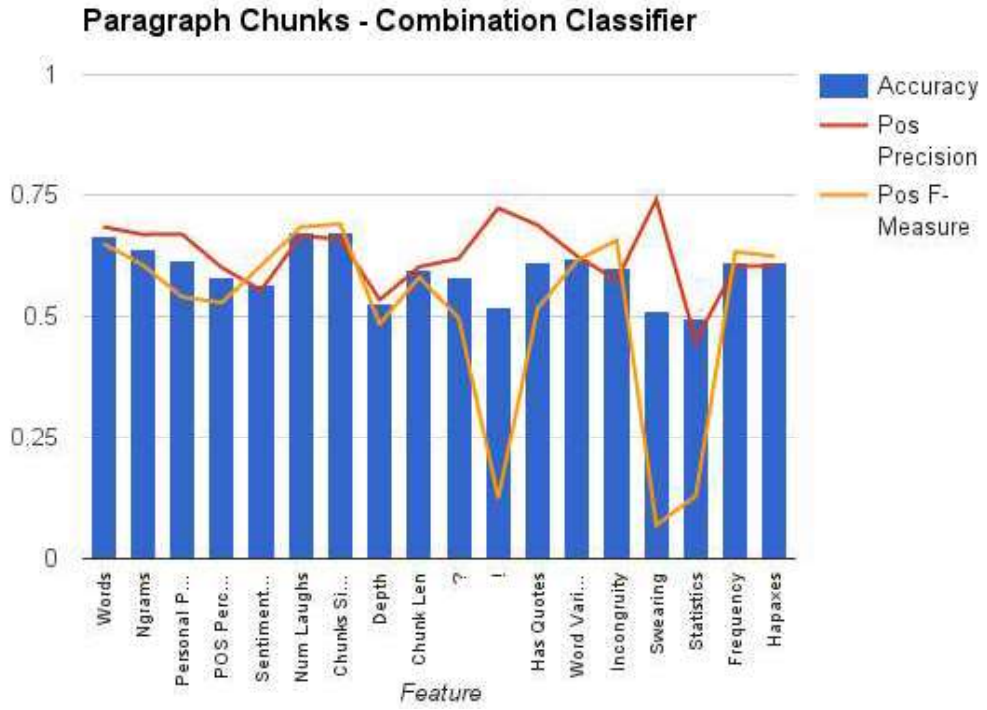

Figure 5.11: The accuracy, positive precision, and positive F-Measure across each individual feature group using the combination classifier with paragraph chunks.

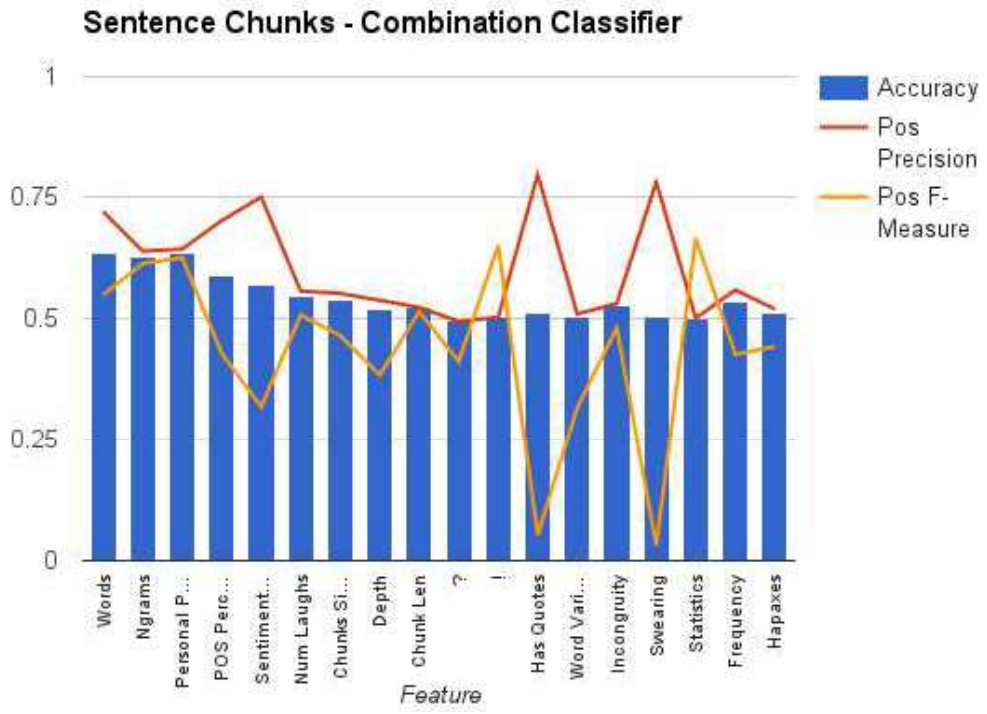

Figure 5.12: The accuracy, positive precision, and positive F-Measure across each individual feature group using the combination classifier with sentence chunks.

sifier created a new classifier with higher positive precision but lower recall. We chose to pick these two classifiers as they were the most effective when looking at sentence chunks. This resulting classifier increases positive precision at the cost of positive recall, but the increase is significantly greater than the loss.

5.6 Experiments

5.6.1 Uneven Data

Using the full data set did not make sense due to the massive imbalance of positive to negative data. So we experimented with training our classifier on three times as much negative data than positive data. Testing was still done on a balanced data set. We found that the classifiers started marking nearly everything as positive, leading to a high positive recall, but low positive precision. This also meant that its negative precision was high but it had a really low negative recall. This trend was made worse when we increased it to four times the negative data.

5.6.2 Up to the First Laugh

At the start of our project our positive data was everything in a talk up to the first laugh. For negative data, we took talks without laughter and cut them around the same lengths. We also included talks with laughter that would come after the cut, so that some of the negative data came from talks that did have laughs later on. After some time, we abandoned this method as our initial results did not show promise.

Chapter 6

FUTURE WORK

"If it ain't broke, then you need to add more features! (Jarrel)"

Laff-O-Tron, having looked at an environment untouched by previous works, investigated many features during testing. However, there are many more things that could be investigated, and many features whose performance may be improved with new methods. Below are a few of things that can be done to possibly improve the power of Laff-O-Tron

6.1 Improving NER and Adult Slang Recognition

Laff-O-Tron's current NER system and dictionary of adult slang are in need of improvement. Although the adult slang dictionary proved to be effective, using our NER system actually hinders performance as it can not detect the majority of entities. An improved NER system would help normalize the data and possibly lead to finding useful patterns. This would also improve detecting statistics and facts, which may make the statistics count feature useful. Expanding the adult slang dictionary may make it more useful as well and increase its recall. The work on one-liner detection provided an example for how to expand the adult slang dictionary by using WordNet Domains to extract all synsets labeled with "Sexuality" [17].

6.2 Using the Probability Distribution

Because the classifiers are able to return the probability of each of tag when classifying, it may be possible to improve their results by adjusting the cutoff for marking something as positive. This will give us more control of the precision and recall values.

6.3 Changes to Word Frequencies

To approximate the frequencies of everyday words, we chose to use the Brown corpus. This may have been a mistake as it was created in 1961, and the way people speak has changed. Using a more modern corpus may improve the usefulness of word frequency analysis. Another idea would seeing if changing the genre of the corpus could lead to identifying humorous, or non-humorous, patterns. For example, would using the government genre of the Brown corpus lead us to finding non-humorous patterns as government is talk is generally non-humorous? Would using the adventure genre find more humorous patterns? Finally, stemming the text before gathering the hapaxes, and counting the stemmed hapaxes in a chunk, may improve the effectiveness of that feature as well.

6.4 Ambiguity Improvement

"Did you hear about the guy whose whole left side was cut off? He's all right now. (Jarrel)"

The method we attempted to use in Laff-O-Tron to approximate ambiguity did not work. We hoped to use the Stanford parser to approximate ambiguity by discovering the number of ways a chunk could be parsed, but it proved ineffective and took too long to implement. Other methods, like those described in the study on humor anchor extraction, have the possibility of improving Laff-O-Tron's performance and could be implemented in future works.

6.5 Deep Neural Networks

As discovered by Oliveria and Rodrigo when investigating humor detection in Yelp reviews, deep learning can get significantly better results when used properly [22]. By using deep neural networks instead of the "shallow" methods used by Laff-O-Tron, the performance can be improved by using features that look more at the expressive relationships between semantics and humor.

6.6 Different Classifiers

Due to time constraints, we limited the amount of classifiers to test. Other works, such as humor detection in tweets, used other classifiers such as Gradient Boosted Regression Trees [35]. As seen in our evaluation section, classifiers varied in performance based on the features detected. So it is possible we used a less than optimal classifiers for this project.

6.7 Analogy Detection

Analogy detection is something that we believe could be incorporated to help identify humor in a TED talk. Although that is an entirely different issue altogether, it would directly impact what we could analyze in a public talk. This is because "Many popular TED presenters provoke laughter by using analogies" [33]. For example: "If you hear an expert talking about the Internet and saying it does this or it will do that, you should treat it with the same skepticism that you might treat the comments of an economist about the economy or a weatherman about the weather." Danny Hillis, inventor, TED 2013

6.8 Phonetics

"BAM! Dead daughter! Sure it sounds nice because of the alliteration. But I assure you it's not." (Robin Williams in The Crazy Ones)

The phonetic properties of a humorous sentence are so important; some studies have found that they are often at least as important as their context. This is because "many humorous texts play with sounds, creating incongruous sounds or words" [34]. Word repetition, rhyme, and alliteration are all phonetic components often used in humor. Adding in a system to analyze the phonetic properties of chunks in a TED Talk could allow Laff-O-Tron to identify phonetic jokes that it could not catch before.

Previous works such as one-liner detection, Twitter humor recognition, and humor anchor extraction attempted to approximate this in different ways. For one-liner detection, they focused on repetitions in text, ignoring repetitions in things like stopwords [17]. While in the humor anchor extraction paper, they used a pronunciation dictionary to look at the phonetic sounds of words [34].

It is highly recommended that the user look up puns immediately after reading this for examples of the importance of phonetics.

Chapter 7

CONCLUSION

"The conclusion is the part where you get tired of thinking. (Arthur Bloch)"

Laff-O-Tron serves to widen humor recognition into the realm of public speaking. Laughter is an essential part of being human, and having computers be able to laugh along with us, without understanding why, is another step towards making a more human AI. More importantly, it can improve the ways humans interact with computers by making them appear more human. Laff-O-Tron can also be used by advertisers and public speakers to measure the humorous effects of their speech. We have demonstrated that it is possible to predict laughter in a public speaking environment by looking at a variety of features across various machine learning classifiers. We used 7 metrics: accuracy and positive and negative precision, recall, and f-measure. Our final results showed that using previous laughter and paragraph sized chunks, Laff-O-Tron can accurately predict laughter with 75% accuracy. Looking at sentences or not using previous laughter gave us 69-70% accuracy. And using sentence chunks and no previous laugh features resulted in 66% accuracy. Furthermore, we have learned more about the various kinds of humor centric features and how to properly use them in a public speaking environment.

Our Laff-O-Tron system is far from perfect and there is much more experimentation to be done to improve its capabilities. Despite this, as the first system to try and predict laughter in a TED Talk, it performed very well. And it's always good to see that your data is more laughable than your results.

Chapter 8

FULL EXPERIMENTAL RESULTS

"Artificial intelligence is no match for natural stupidity. (Anonymous)"

Table 8.1: Using Sentence Chunks, by classifier, each feature's accuracy, positive precision, positive recall, positive F1 score, negative precision, negative recall, negative F1 score.

Begin of Table							
Classifier	Accuracy	Pos Prec.	Pos Rec.	Pos F1.	Neg Prec.	Neg Rec.	Neg F1
Words							
Naive Bayes	0.65	0.63	0.74	0.68	0.68	0.56	0.62
SVM	0.62	0.62	0.61	0.61	0.61	0.62	0.62
AdaBoost(50)	0.62	0.66	0.50	0.57	0.60	0.74	0.66
Random Forest	0.60	0.55	0.62	0.61	0.60	0.58	0.59
COMBO	0.64	0.72	0.44	0.55	0.60	0.83	0.69
N-grams							
Naive Bayes	0.61	0.60	0.67	0.63	0.62	0.55	0.58
SVM	0.61	0.62	0.56	0.59	0.60	0.65	0.62
AdaBoost(50)	0.59	0.69	0.34	0.45	0.56	0.85	0.68
Random Forest	0.60	0.68	0.37	0.48	0.57	0.83	0.67
COMBO	0.59	0.70	0.31	0.43	0.56	0.87	0.68
Personal Pronouns and PN/N							
Naive Bayes	0.54	0.54	0.53	0.53	0.54	0.55	0.55
SVM	0.55	0.57	0.46	0.51	0.55	0.65	0.59

Continuation of sentence features table							
Classifier	Accuracy	Pos Prec.	Pos Rec.	Pos F1.	Neg Prec.	Neg Rec.	Neg F1
AdaBoost(50)	0.56	0.55	0.65	0.60	0.58	0.48	0.52
Random Forest	0.53	0.54	0.49	0.51	0.53	0.58	0.55
COMBO	0.55	0.56	0.47	0.51	0.54	0.63	0.58
POS percentages							
Naive Bayes	0.54	0.54	0.53	0.53	0.54	0.55	0.55
SVM	0.55	0.57	0.46	0.51	0.55	0.65	0.59
AdaBoost(50)	0.56	0.55	0.65	0.60	0.58	0.48	0.52
Random Forest	0.53	0.54	0.49	0.51	0.53	0.58	0.55
COMBO	0.55	0.56	0.47	0.51	0.54	0.63	0.58
Sentiment analysis							
Naive Bayes	0.52	0.53	0.41	0.46	0.52	0.63	0.57
SVM	0.52	0.53	0.43	0.48	0.52	0.61	0.56
AdaBoost(50)	0.52	0.53	0.43	0.47	0.52	0.61	0.56
Random Forest	0.51	0.51	0.53	0.52	0.51	0.49	0.50
COMBO	0.52	0.54	0.30	0.38	0.51	0.74	0.61
Laughs previous							
Naive Bayes	0.62	0.63	0.60	0.61	0.62	0.65	0.63
SVM	0.62	0.67	0.46	0.55	0.59	0.77	0.67
AdaBoost(50)	0.62	0.64	0.57	0.60	0.61	0.67	0.64
Random Forest	0.63	0.64	0.58	0.61	0.61	0.67	0.64
COMBO	0.63	0.64	0.59	0.61	0.62	0.66	0.64
Sentences since last laugh							
Naive Bayes	0.62	0.61	0.65	0.63	0.62	0.58	0.60
SVM	0.61	0.59	0.78	0.66	0.59	0.44	0.50

Continuation of sentence features table							
Classifier	Accuracy	Pos Prec.	Pos Rec.	Pos F1.	Neg Prec.	Neg Rec.	Neg F1
AdaBoost(50)	0.64	0.63	0.66	0.64	0.65	0.62	0.63
Random Forest	0.62	0.62	0.65	0.63	0.63	0.60	0.61
COMBO	0.63	0.64	0.61	0.62	0.63	0.66	0.64
Depth							
Naive Bayes	0.51	0.51	0.84	0.63	0.54	0.19	0.28
SVM	0.52	0.52	0.53	0.52	0.52	0.52	0.52
AdaBoost(50)	0.52	0.52	0.59	0.55	0.53	0.46	0.48
Random Forest	0.51	0.51	0.49	0.50	0.51	0.53	0.52
COMBO	0.52	0.52	0.51	0.51	0.52	0.53	0.53
Sentence Length							
Naive Bayes	0.49	0.49	0.52	0.51	0.49	0.46	0.47
SVM	0.50	0.32	0.49	0.37	0.21	0.15	0.14
AdaBoost(50)	0.49	0.49	0.44	0.46	0.49	0.54	0.51
Random Forest	0.49	0.49	0.47	0.48	0.49	0.51	0.50
COMBO	0.46	0.50	0.36	0.41	0.50	0.63	0.55
Is Question							
Naive Bayes	0.51	0.50	0.93	0.65	0.55	0.01	0.15
SVM	0.51	0.50	0.92	0.65	0.53	0.09	0.15
AdaBoost(50)	0.51	0.50	0.92	0.65	0.55	0.09	0.16
Random Forest	0.51	0.50	0.92	0.65	0.55	0.10	0.16
COMBO	0.50	0.50	0.92	0.65	0.53	0.09	0.15
Is Exclamation							
Naive Bayes	0.51	0.80	0.03	0.05	0.51	0.99	0.67
SVM	0.508	0.79	0.02	0.05	0.50	0.99	0.67

Continuation of sentence features table							
Classifier	Accuracy	Pos Prec.	Pos Rec.	Pos F1.	Neg Prec.	Neg Rec.	Neg F1
AdaBoost(50)	0.51	0.78	0.02	0.05	0.50	0.99	0.67
Random Forest	0.51	0.77	0.03	0.05	0.50	0.99	0.67
COMBO	0.510	0.80	0.03	0.05	0.51	0.99	0.67
Has Quote							
Naive Bayes	0.57	0.76	0.20	0.32	0.54	0.93	0.68
SVM	0.57	0.75	0.20	0.31	0.54	0.93	0.68
AdaBoost(50)	0.57	0.75	0.20	0.32	0.54	0.93	0.68
Random Forest	0.57	0.75	0.20	0.32	0.54	0.93	0.68
COMBO	0.57	0.75	0.20	0.32	0.54	0.93	0.68
Word Variance							
Naive Bayes	0.50	0.50	0.37	0.42	0.50	0.63	0.55
SVM	0.50	0.50	0.44	0.47	0.50	0.57	0.53
AdaBoost(50)	0.50	0.51	0.35	0.40	0.50	0.66	0.56
Random Forest	0.50	0.50	0.36	0.41	0.50	0.64	0.55
COMBO	0.50	0.51	0.24	0.32	0.50	0.77	0.61
Incongruity							
Naive Bayes	0.52	0.52	0.51	0.51	0.52	0.54	0.53
SVM	0.51	0.51	0.57	0.54	0.52	0.45	0.48
AdaBoost(50)	0.53	0.53	0.53	0.53	0.53	0.52	0.53
Random Forest	0.51	0.51	0.54	0.52	0.51	0.47	0.49
COMBO	0.53	0.53	0.44	0.48	0.52	0.61	0.56
Swearing							
Naive Bayes	0.51	0.77	0.02	0.03	0.50	0.99	0.67
SVM	0.51	0.76	0.02	0.03	0.50	0.99	0.67

Continuation of sentence features table							
Classifier	Accuracy	Pos Prec.	Pos Rec.	Pos F1.	Neg Prec.	Neg Rec.	Neg F1
AdaBoost(50)	0.51	0.78	0.02	0.03	0.50	0.99	0.67
Random Forest	0.51	0.78	0.02	0.03	0.50	0.99	0.67
COMBO	0.51	0.78	0.02	0.03	0.50	0.99	0.67
Statistics							
Naive Bayes	0.50	0.50	0.99	0.67	0.60	0.01	0.02
SVM	0.50	0.50	0.99	0.67	0.58	0.01	0.02
AdaBoost(50)	0.50	0.50	0.99	0.67	0.61	0.01	0.02
Random Forest	0.50	0.50	0.99	0.67	0.57	0.01	0.02
COMBO	0.50	0.50	0.99	0.67	0.58	0.01	0.02
Frequency							
Naive Bayes	0.54	0.54	0.54	0.54	0.54	0.54	0.54
SVM	0.50	0.52	0.47	0.38	0.41	0.53	0.41
AdaBoost(50)	0.53	0.53	0.47	0.50	0.53	0.59	0.56
Random Forest	0.53	0.53	0.47	0.50	0.52	0.58	0.55
COMBO	0.54	0.56	0.35	0.43	0.53	0.73	0.61
Hapax count							
Naive Bayes	0.52	0.52	0.44	0.47	0.52	0.60	0.55
SVM	0.52	0.53	0.40	0.46	0.52	0.64	0.57
AdaBoost(50)	0.52	0.52	0.44	0.47	0.52	0.59	0.55
Random Forest	0.52	0.52	0.44	0.47	0.52	0.60	0.55
COMBO	0.51	0.52	0.40	0.44	0.51	0.62	0.56
Frequency, Hapax count							
Naive Bayes	0.55	0.54	0.57	0.55	0.55	0.52	0.54
SVM	0.50	0.51	0.47	0.37	0.45	0.53	0.40

Continuation of sentence features table							
Classifier	Accuracy	Pos Prec.	Pos Rec.	Pos F1.	Neg Prec.	Neg Rec.	Neg F1
AdaBoost(50)	0.54	0.54	0.51	0.52	0.53	0.56	0.55
Random Forest	0.51	0.52	0.45	0.48	0.51	0.57	0.54
COMBO	0.55	0.57	0.39	0.46	0.54	0.70	0.61
End of Table							

Table 8.2: Using Paragraph Chunks, by classifier, each feature's accuracy, positive precision, positive recall, positive F1 score, negative precision, negative recall, negative F1 score.

Begin of Table							
Classifier	Accuracy	Pos Prec.	Pos Rec.	Pos F1.	Neg Prec.	Neg Rec.	Neg F1
Words							
Naive Bayes	0.53	0.52	1.00	0.68	0.96	0.07	0.13
SVM	0.66	0.67	0.63	0.65	0.65	0.69	0.67
AdaBoost(50)	0.67	0.68	0.63	0.66	0.66	0.71	0.68
Random Forest	0.65	0.67	0.59	0.63	0.63	0.70	0.66
COMBO	0.67	0.68	0.62	0.65	0.65	0.71	0.68
N-grams							
Naive Bayes	0.58	0.54	0.98	0.70	0.91	0.18	0.30
SVM	0.67	0.70	0.59	0.64	0.64	0.75	0.69
AdaBoost(50)	0.63	0.66	0.55	0.60	0.61	0.71	0.66
Random Forest	0.64	0.71	0.48	0.56	0.61	0.80	0.69
COMBO	0.64	0.67	0.56	0.61	0.62	0.72	0.67

Continuation of paragraph features table							
Classifier	Accuracy	Pos Prec.	Pos Rec.	Pos F1.	Neg Prec.	Neg Rec.	Neg F1
Personal Pronouns and PN/N							
Naive Bayes	0.61	0.61	0.56	0.59	0.60	0.65	0.62
SVM	0.55	0.56	0.52	0.54	0.55	0.59	0.57
AdaBoost(50)	0.64	0.63	0.66	0.64	0.64	0.61	0.63
Random Forest	0.59	0.60	0.54	0.57	0.58	0.63	0.60
COMBO	0.61	0.67	0.45	0.54	0.59	0.77	0.67
POS percentages							
Naive Bayes	0.58	0.58	0.55	0.57	0.57	0.60	0.59
SVM	0.56	0.56	0.57	0.56	0.56	0.56	0.56
AdaBoost(50)	0.59	0.58	0.70	0.63	0.62	0.49	0.54
Random Forest	0.55	0.55	0.50	0.52	0.54	0.60	0.57
COMBO	0.58	0.60	0.47	0.53	0.57	0.68	0.62
Sentiment analysis							
Naive Bayes	0.57	0.55	0.76	0.64	0.61	0.38	0.47
SVM	0.57	0.55	0.86	0.67	0.68	0.29	0.40
AdaBoost(50)	0.58	0.56	0.77	0.64	0.62	0.39	0.47
Random Forest	0.55	0.54	0.58	0.56	0.55	0.51	0.53
COMBO	0.57	0.55	0.67	0.60	0.58	0.46	0.52
Laughs previous							
Naive Bayes	0.67	0.65	0.71	0.68	0.68	0.63	0.65
SVM	0.65	0.70	0.52	0.60	0.62	0.78	0.69
AdaBoost(50)	0.67	0.66	0.70	0.68	0.68	0.64	0.66
Random Forest	0.67	0.66	0.70	0.68	0.68	0.64	0.66

Continuation of paragraph features table							
Classifier	Accuracy	Pos Prec.	Pos Rec.	Pos F1.	Neg Prec.	Neg Rec.	Neg F1
COMBO	0.67	0.67	0.70	0.68	0.69	0.65	0.67
Sentences since last laugh							
Naive Bayes	0.66	0.64	0.72	0.68	0.68	0.60	0.64
SVM	0.67	0.63	0.83	0.72	0.75	0.51	0.61
AdaBoost(50)	0.67	0.66	0.72	0.68	0.69	0.63	0.66
Random Forest	0.68	0.66	0.72	0.69	0.69	0.63	0.66
COMBO	0.67	0.66	0.73	0.69	0.70	0.62	0.66
Depth							
Naive Bayes	0.50	0.50	0.74	0.60	0.50	0.26	0.34
SVM	0.56	0.56	0.56	0.56	0.56	0.56	0.56
AdaBoost(50)	0.54	0.54	0.54	0.54	0.54	0.54	0.54
Random Forest	0.53	0.53	0.57	0.55	0.53	0.48	0.50
COMBO	0.53	0.53	0.45	0.48	0.52	0.61	0.56
Sentence Length							
Naive Bayes	0.57	0.57	0.64	0.60	0.59	0.51	0.55
SVM	0.59	0.45	0.61	0.51	0.54	0.33	0.40
AdaBoost(50)	0.61	0.59	0.75	0.66	0.66	0.48	0.55
Random Forest	0.57	0.56	0.58	0.57	0.57	0.55	0.56
COMBO	0.60	0.60	0.56	0.58	0.59	0.63	0.61
Is Question							
Naive Bayes	0.58	0.63	0.42	0.50	0.56	0.75	0.64
SVM	0.58	0.63	0.41	0.50	0.56	0.76	0.64
AdaBoost(50)	0.58	0.62	0.42	0.50	0.56	0.74	0.64
Random Forest	0.58	0.62	0.41	0.49	0.56	0.74	0.64

Continuation of paragraph features table							
Classifier	Accuracy	Pos Prec.	Pos Rec.	Pos F1.	Neg Prec.	Neg Rec.	Neg F1
COMBO	0.58	0.62	0.42	0.50	0.56	0.74	0.64
Is Exclamation							
Naive Bayes	0.52	0.75	0.07	0.13	0.51	0.98	0.67
SVM	0.52	0.70	0.07	0.12	0.51	0.97	0.67
AdaBoost(50)	0.52	0.75	0.07	0.13	0.51	0.98	0.67
Random Forest	0.52	0.74	0.07	0.12	0.51	0.98	0.67
COMBO	0.52	0.72	0.07	0.12	0.50	0.97	0.67
Has Quote							
Naive Bayes	0.62	0.70	0.43	0.53	0.59	0.81	0.68
SVM	0.61	0.69	0.42	0.52	0.58	0.81	0.68
AdaBoost(50)	0.62	0.69	0.42	0.52	0.58	0.82	0.68
Random Forest	0.62	0.69	0.42	0.52	0.58	0.81	0.68
COMBO	0.61	0.69	0.41	0.52	0.58	0.81	0.68
Word Variance							
Naive Bayes	0.59	0.56	0.78	0.65	0.64	0.39	0.49
SVM	0.63	0.61	0.72	0.66	0.66	0.55	0.60
AdaBoost(50)	0.62	0.60	0.70	0.64	0.64	0.54	0.58
Random Forest	0.58	0.58	0.55	0.57	0.57	0.60	0.59
COMBO	0.62	0.62	0.61	0.62	0.62	0.63	0.62
Incongruity							
Naive Bayes	0.60	0.57	0.83	0.68	0.69	0.37	0.49
SVM	0.58	0.57	0.67	0.62	0.60	0.49	0.54
AdaBoost(50)	0.60	0.57	0.78	0.66	0.66	0.42	0.51
Random Forest	0.57	0.56	0.63	0.59	0.57	0.51	0.54

Continuation of paragraph features table							
Classifier	Accuracy	Pos Prec.	Pos Rec.	Pos F1.	Neg Prec.	Neg Rec.	Neg F1
COMBO	0.60	0.57	0.77	0.66	0.65	0.43	0.52
Swearing							
Naive Bayes	0.51	0.77	0.04	0.07	0.51	0.99	0.67
SVM	0.51	0.74	0.04	0.07	0.51	0.99	0.67
AdaBoost(50)	0.51	0.73	0.04	0.07	0.51	0.99	0.67
Random Forest	0.51	0.76	0.04	0.07	0.51	0.99	0.67
COMBO	0.51	0.74	0.04	0.07	0.51	0.99	0.67
Statistics							
Naive Bayes	0.50	0.48	0.37	0.28	0.45	0.62	0.43
SVM	0.50	0.49	0.30	0.24	0.47	0.70	0.49
AdaBoost(50)	0.50	0.50	0.33	0.25	0.45	0.67	0.46
Random Forest	0.50	0.50	0.95	0.64	0.42	0.05	0.04
COMBO	0.50	0.45	0.14	0.13	0.48	0.85	0.58
Frequency							
Naive Bayes	0.59	0.57	0.76	0.65	0.64	0.42	0.51
SVM	0.53	0.30	0.41	0.31	0.46	0.33	0.30
AdaBoost(50)	0.60	0.59	0.72	0.64	0.64	0.49	0.55
Random Forest	0.57	0.57	0.56	0.56	0.56	0.57	0.57
COMBO	0.61	0.60	0.67	0.63	0.63	0.56	0.59
Hapax count							
Naive Bayes	0.61	0.59	0.71	0.64	0.64	0.51	0.56
SVM	0.60	0.59	0.52	0.52	0.56	0.60	0.57
AdaBoost(50)	0.61	0.59	0.72	0.65	0.65	0.50	0.56
Random Forest	0.61	0.59	0.67	0.62	0.62	0.55	0.58

Continuation of paragraph features table							
Classifier	Accuracy	Pos Prec.	Pos Rec.	Pos F1.	Neg Prec.	Neg Rec.	Neg F1
COMBO	0.61	0.61	0.65	0.62	0.62	0.58	0.60
Frequency, Hapax count							
Naive Bayes	0.60	0.58	0.75	0.65	0.65	0.45	0.53
SVM	0.54	0.51	0.57	0.42	0.60	0.35	0.34
AdaBoost(50)	0.61	0.60	0.67	0.63	0.62	0.55	0.58
Random Forest	0.57	0.58	0.51	0.54	0.56	0.63	0.59
COMBO	0.61	0.60	0.62	0.61	0.61	0.59	0.60
End of Table							

BIBLIOGRAPHY

[1] Cal Poly Github. `http://www.github.com/CalPoly`.

[2] Introduction to support vector machines. `http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html`.

[3] Scitkit-learn: machine learning in python. `http://scikit-learn.org/stable/index.html`.

[4] Word2vec: Neural word embeddings in java. `http://deeplearning4j.org/word2vec`.

[5] S. Bird, E. Klein, and E. Loper. *Natural Language Processing with Python*. Beijing: O'Reilly, 2009.

[6] P. Dybala, M. Ptaszynski, R. Rzepka, and K. Araki. Multi-humoroid: joking system that reacts with humor to humans' bad moods. *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1 (AAMAS '10), Vol. 1. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC*, 1:1433–1434, 2015.

[7] L. Friedland and J. Allan. Joke retrieval: Recognizing the same joke told differently. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 883–892, New York, NY, USA, 2008. ACM.

[8] J. Hulstijn and Nijholt. *Proceedings of the International Workshop on Computational Humor*. Number 12 in Twente Workshops on Language Technology, Enschede, Netherlands, 1996.

[9] C. Kiddon and Y. Brun. That's what she said: double entendre identification. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics:shortpapers*, 2:89–94, 2011.

[10] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31:249–268, 2016.

[11] G. Lessard and M. Levison. Computational modeling of linguistic humour: Tom Swifties. In *ALLC/ACH Joint Annual Conference, Oxford*, pages 175–178. 1992.

[12] T. C. LLC. "our organization" ted: Ideas worth spreading. `https://www.ted.com/about/our-organization`.

[13] S. Loria. Textblob: Simplified text processing. `http://textblob.readthedocs.io/en/dev/index.html`.

[14] B. M. Marie-Catherine de Marneffe and C. D. Manning. Generating typed dependency parses from phrase structure parses. *In LREC 2006.*

[15] R. Mihalcea and S. Pulman. Characterizing humour: An exploration of features in humorous texts. *Lecture Notes in Computer Science*, 4394:337–347, 2007.

[16] R. Mihalcea and C. Strapparava. Making computers laugh: Investigations in automatic humor recognition. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 531–538, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[17] R. Mihalcea and C. Strapparava. Learning to laugh (automatically): Computational models for humor recognition. *Computational Intelligence*, 22(2):126–142, 2006.

[18] R. Mihalcea and C. Strapparava. Contemporary LINGUISTIC THEORIES OF HUMOUR. *Folklore (n. d.):*, 27-58.(30), Nov. 2015.

[19] T. M. Mitchell. *The Discipline of Machine Learning.* Carnegie Mellon University, July 2006.

[20] J. Morkes, H. K. Kernal, and C. Nass. Humor in task-oriented computer-mediated communication and human-computer interaction. *In CHI 98 Conference Summary on Human Factors in Computing Systems*, 98:215–216, 1998.

[21] J. L. Nigam, Kamal and A. McCallum. Using maximum entropy for text classification. *Proceedings of the 2nd International Workshop on Natural Language Understanding and Cognitive Science (2005)*, 2016.

[22] L. D. Oliveira and A. L. Rodrigo. Humor detection in yelp reviews. *Encyclopedia of Humor Studies (2016)*, 2016.

[23] . M. D. Petrovic, S. Unsupervised joke generation from big data. *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 228–232, 2013.

[24] A. Purandare and D. Litman. Humor: Prosody analysis and automatic recognition for friends*. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pages 208–215, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

[25] S. Raschka. Naive bayes and text classification introductory and theory. 4, Oct. 2014. `http://sebastianraschka.com/Articles/2014_naive_bayes_1.html`.

[26] V. Raskin. A little metatheory: Thoughts on what a theory of computational humor should look like. In *AAAI Technical Report FS-12-02*, pages 62–67. 2012.

[27] G. Ritchie, R. Manurung, H. Pain, D. O. Annalu Waller, and R. Black. A practical application of computational humour. Nov. 2015.

[28] R. E. Schapire and Y. Freund. *Boosting: Foundations and Algorithms*. Cambridge, MA: MIT, 2012.

[29] L. Schubert. Computational linguistics. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. The Metaphysics Research Lab, spring 2015 edition, 2015.

[30] D. Shahaf, E. Horvitz, and R. Mankoff. Inside jokes: Identifying humorous cartoon captions. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1065–1074, New York, NY, USA, 2015. ACM.

[31] J. M. Taylor and L. J. Mazlack. Computationally recognizing wordplay in jokes. In *In Proceedings of CogSci 2004*, 2004.

[32] S. S. Tennyson, Elizabeth M. and E. Korpela. Random forests. *Proceedings of the Wisconsin Space Conference Proc. Wisc. Space Conf. 0.0 (2011)*, 0, 2016.

[33] N. Tsolak. The 6th public speaking secret of the worlds top minds: Laughter. `http://www.laughteronlineuniversity.com/the-6th-public-speaking-secret-of-the-worlds-top-minds`.

[34] A. L. C. D. Yang, Diyi and E. Hovy. Humor recognition and humor anchor extraction. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (2015)*, pages 2367–2376, 2015.

[35] R. Zhang and N. Liu. Recognizing humor on twitter. *CIKM '14*, pages 889–898, 2014.

[36] R. Zhang and N. Liu. Recognizing humor on twitter. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 889–898, New York, NY, USA, 2014. ACM.