Introduction

Humor is a trait that is seldom found in the people to begin with. Our task is to teach our program humor. Our goal is to learn to characterize the sense of humor represented in the show by classifying from the given set of tweets which is funnier based on the there F.Q. (Funny Quotient :)
(Yes we're calculating the FQ for our project, and yes I just made that term)

Importance

There have been numerous computational approaches to humor within the last decade. However, the majority of this work decomposes the notion of humor into two groups: humor and non-humor. This representation ignores the continuous nature of humor, while also not accounting for the subjectivity in perceiving humor. Humor is an essential trait of human intelligence that has not been addressed extensively in the current AI research, and we feel that shifting from the binary, "objective" approach of humor detection is a good pathway towards advancing this work. - **From Semeval Task page**

Examples

An example of the data would be as follows:

Hashtag	Tweet	Label
#Before_Facebook	I actually LIKED things	0 (rest of the tweets)
#420_celebs	Edible Murphy	1 (top 10)
#Bad_Kickstarter_In_5_Words	Feed the Children... to Lions	2 (selected tweet)

Method

Our method to tackle this task is to build upon the code provided by the semeval team. Given their model our approach was to understand it and extend it further to include more features and increase the accuracy.
To explain more about the methodology we will go step by step on how we are pre-processing the data and how we are the building our model

1. Preprocessing of data - reading in the data and saving it into our data structure to use it with our model (def create_data)
2. Feature Extraction - we extract features from our corpus to add to our model to increase it's accuracy. These features are PoS, rhyming freq, profanity, punctuation etc. (def load_hashtag) - Note - we're only adding rhyming freq to our model now, but we're able to extract the other features as well
3. Model Design - we fit our data to our model. We are using SVM to train our model (def run_loo_exp)
4. Testing - we test our model on the test data to get the accuracy of our model. For testing we leave one file out off our training data and test our model on it. So if there are 10 files we train on 9 and test on 1. We repeat this process 10 times to test each file once and train on other files (def run_loo_exp)

Code Setup

To run the code please run the adjoining runit.sh file. To make it easier for the user to run the code and install all the dependencies we have included an install.sh which will install all the dependencies for you.

Feature extraction

Rhyming Frequency(called in the def load_hashtag)

1. Tokenize the word
2. Generate the sound syllables for every token using CMU dictionary corpus.
3. Compare individual token with every other token in a tweet to see if the last sounds are rhyming with each other.
4. Calculate frequency of every rhyming pair in a tweet and generate a list.
5. We can also save these pairs to further analyze the rhyming words or to add it to our model

Hashtag Context Feature

1. Split the hashtag into individual words and lemmatize each word.
2. Get context vector for each word using word2vec tool and add it to a file.
3. Lemmatize each word from a given tweet and compare to the context list from 2. Number of words in a tweet that are similar to any of the hash tag words constitute the frequency feature for context.

Profanity, Punctuation, Tweet length

1. All tweets are preprocessed into a global array
2. Profanity, punctuation, and tweet length functions are called based on their tweet id
   a. Profanity - profanity in the tweet is matched and counted using a profanity dictionary located in the file “en”
   b. Punctuation - regular expressions used to match and count the number of punctuation marks
   c. Tweet length - simple python string length method is used to count the length of each tweet

Parts of Speech

1. The input tweets were separated into tokens by tokenize function.
2. The punctuation’s were eliminated.
3. Tokens are tagged with parts of speech using averaged perceptron tagger from NLTK
4. Hashtag’s and user information from the tweets were retained as they were misinterpreted by the tagger.
5. GATE Twitter POS tagger was used in hope to increase the accuracy as the tagger was robust to deal with Twitter data.

Adjective and Adverb Intensity

1. This feature was inspired from Barbieri, Francesco, and Horacio Saggion. "Automatic detection of irony and humor in twitter." The research group used Potts, C. 2011. Developing adjective scales from user- supplied textual metadata. NSF Workshop on Restructuring Adjectives in WordNet to evaluate the intensity of adjective and adverbs.
2. Potts adjective intensity was used in extracting the intensity of adjective and adverbs in the tweets.

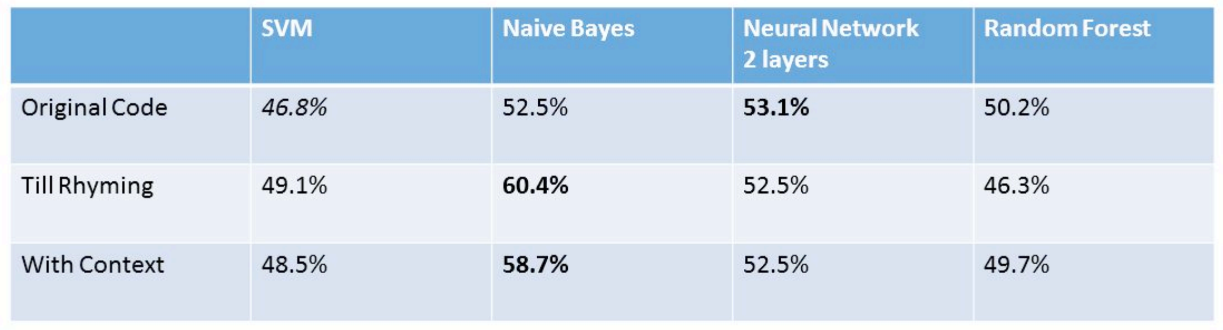
Structure of data:
The training/trial data consists of a single directory with several files. Each file corresponds to a single hashtag, and is named appropriately. For example, for the hashtag #FastFoodBooks, the file is called Fast_Food_Books.tsv. We add the underscore between hashtag tokens for easier parsing of the hashtags. We believe a better semantic understanding of the hashtag will contribute to a better performance in the task.

The tweets are labeled 0, 1, or 2. 0 corresponds to a tweet not in the top 10 (most of the tweets in a file). 1 corresponds to a tweet in the top 10, but not the winning tweet (usually, 9 tweets per hashtag). 2 corresponds to the winning tweet (one tweet per hashtag). - **from semeval**

Step-by-Step Run Procedure:-
The script uses the training data. The script performs leave-one-out experiments on the hashtag files in the directory given as an argument to the script.

Files and code:-

The submission contains different codes for the different models we used for our analysis. These models include the same features and they are evaluated on the same dataset to calculate these accuracy. For completion, I also modified the original code with different models to get an accurate baseline for all the models separately. The file tester.py is the main file we worked on and contains all the features, the other files are not using the latest PoS and the adjective and adverb intensity due to some compilation error with the NLTK package. The accuracy we achieve is as follows:



	SVM	Naive Bayes	Neural Network 2 layers	Random Forest
Original Code	*46.8%*	52.5%	**53.1%**	50.2%
Till Rhyming	49.1%	**60.4%**	52.5%	46.3%
With Context	48.5%	**58.7%**	52.5%	49.7%

Usage:

```
$ python htw_sample_script_naive_bayes.py small_data
```

Specifically, the script holds-out one hashtag file at a time out. It then forms appropriate tweet pairs within the remaining (training) hashtag files. Individual tweet representation is BOW frequency. The label applied to a tweet pair corresponds to whether or not the first tweet in the pair is the funnier tweet. The ordering of the pairs is random, and is chosen by a coin-flip. These pairs are then combined across all the training files to create the training matrix. An SVM model is trained on the resulting training matrix. Tweet pairs are also formed from the held-out hashtag file, and accuracy is computed on the resulting test matrix. The script reports micro-average accuracy across all held-out files, since different files have different amounts of tweets. - **From semeval**

We are using the same structure as the semeval code so that it is easier to understand our code and and also so that it runs with the same commands.

Team Member Contributions
Aman Kochhar - Machine learning algorithm design, prototype adaptation, different models for training and predicting the tweets
Brett Holden - Feature extraction (profanity,punctuation,length), dependency installation scripting and documentation, feature reformatting for new ML algorithms
Aashish Dhungana - Feature extraction (rhyming), Data cleaning for rhyming, HashTag context representation.
Prudhvi Raj Meda – Implemented NLTK Average Perceptron tagger, GATE Twitter POS tagger, Adjective and Adverb Intensity features.

Code contributions of each team member are detailed in comment blocks.

To make it easier to understand the changes we've made to the original code we are including the original code as well that was provided by the semeval team. This code is under the folder labeled "orig"

Sources
https://github.com/LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words http://alt.qcri.org/semeval2017/task6/
https://code.google.com/p/word2vec/.
https://gate.ac.uk/wiki/twitter-postagger.html
http://web.stanford.edu/~cgpotts/data/wordnetscales/
https://www.cs.cmu.edu/~biglou/resources/bad-words.txt