Is This A Joke?

Jim Cai*[1] and Nicolas Ehrhardt†[1]

[1]jimcai@stanford.edu
[2]ehrhardn@stanford.edu

Abstract

The deconstruction of humor in text is well studied in linguistics and literature; computationally, techniques center around modeling text structure and hand crafting features surgically designed to emulate linguistic tendences in specific types of jokes. We take a more hands-off approach, leveraging light linguistic features in conjunction with learned semantic word vectors in a neural network to learn humor. The system performs at a reasonable level and the extensions are promising.

I. Introduction

The role of humor in text is arguably different than in speech. What is lacking in intonation and timing is made up for with an emphasis in meaning. There are many reasons why a certain sentence can be humorous but rather than conduct a linguistic analysis, perhaps we may be able to computationally determine a humor metric.

I. Related Work

There has been interesting and quite succesful in the work of computational humor. Taylor and Mazlack [4] utilize Raskin's Theory of Humor in a computational setting, translating overlap in common phrases into the functional equivalent of overlap in n-grams in order to produce humorous phrases. Kiddon and Brin [3] craft elegant features in order to arrive at the problem of double entendre classification, specifically modeling the structure of erotic text and synonyms for sexually suggestive words and utilizing a SVM on those features.

Most closely related, Bengio et al. [9] utilize a hidden neural network with 1 layer in order to characterize word sequences; the performance on a variety of NLP tasks is on par with the benchmarks.

*CS 224N, CS 229
†CS 229

II. Features

I. Embeddings

Collobert et al. [1] trained a set of 50-dimensional word embeddings using a neural network with hidden layers. The scoring function compared pairs of phrases of a fixed size, where the negative example was the same as the positive example with the target word replaced with a random word. Before we utilized this embedding for our task, we were investigated the degree to which these embeddings captured semantic meaning.

It is unintuitive to perform rudimentary clustering on all 50 dimensions, especially denoting the curse of dimensionality. Therefore, we applied PCA on the set of word embeddings. Initially, we attempted to visualize the entire dataset by plotting against the first two principal components; this proved to be unhelpful because the dataset of 100k words is too dense. Nearest neighbors provided a more straightforward method to evaluate the clustering. K-nn was run on the first $d = 10$ principal components, with d chosen empirically. For

comparison, the results for using the first 5,10 principal components are as follows:
d=5:

- ten: `first`, `two`, `DGDG.DG`, `home`, `living`
- husband:`wife`,`friend`,`actress`,`wrestling`
- virginia:`washington`, `texas`, `chicago`,$\cdots$
- cold: `single`,`DG.DGDG`,`big`, `heavy`, `upper`

d=10:

- ten: `five`, `six`, `seven`,$\cdots$
- husband:`father`,`wife`,`mother`, `daughter`, `lord`
- virginia:`boston`,`philadelphia`, `minnesota`
- cold: `natural`, `sea`, `big`, `earth`, `heart`, `heavy`

While still not entirely perfect upon inspection it is conceivable that these words are related. especially on the kinds of co-occurences with other words that they share; this is what is endowed by the neural net that they were derived from. Turian et al. [5] also study these embeddings extensively and find that they perform well in various word representation tasks.

II. Context

Inspired by Erk and Pado [6] we intend to derive a sense of words into a sentence with respect to the parts of speech of adjacent words. To do so, we model a word's contextual information alongside its word embedding as:

$$(R_x^{-1}, R_x^{+1}, v_x,) \in \mathbb{R}^{150}$$

Where $v_x \in \mathcal{V}$, our word embedding space and R_x^{-1}, R_x^{+1} are applications of $\mathcal{R} \to \mathcal{V}$ (role space $\to$ vector space). The role space consists of an n-dimensional vector that represents the contribution of a particular tag before/after a certain word. Depending on a particular word's neighbors, we extract the mean contribution of occurences with such tags; this will vary depending on the training corpus.

Mitchell and Lapata [7] discusses various tradeoffs in combining semantic word vectors; they argue that composition by dot product is effective because it captures the commonality between sentences and accentuates it. However since we would like to keep different sources of variability in the vector, we have chosen to take the mean of the word vectors that have a particular tag. For this work we utilize the Brown corpus and POS tagger provided by python's nltk library.

For example, we have the sentence "Cats like Hats" with the accompanying parts of speech, ((cats,'N'),('like','P'),('hats','N')). We then store the word vector for "cats" in $R_{like}^{-1}[N]$ and the word vector for "hats" in $R_{like}^{+1}[N]$.

One issue with this framework is data sparsity– more specifically for a word to have useful values of this feature it would require having been observed in its different contexts (if applicable). Another issue is in the evaluation stage, if the test sentence exhibits a new POS tag –in this case we utilize a naive back-off model that looks for the minimum euclidean distance to the vector of any tag to use as the feature.

III. Model

I. Goal

In this section, we follow a similar problem formulation in [8]. Given parameters θ for our model, we arrive at an estimated probability of a sentence being funny. We seek to maximize the following probability:

$$l(\theta) = \log \prod_i^m p(y^{(i)}|x^{(i)};\theta) \tag{1}$$

which simplifies to

$$\sum_i^m y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) \tag{2}$$

where $h_\theta(x)$ is the output of our classifier.
Since we will utilize Stochastic Gradient Descent (SGD) to estimate our parameters, we

thus seek to minimize

$$J(\theta) = \frac{1}{m}\sum_{i}^{m} -l(\theta) + R \tag{3}$$

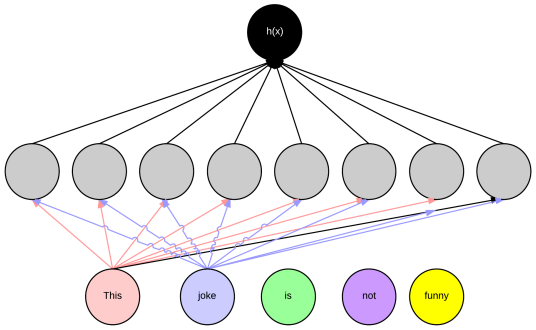
where

$$R = \frac{C}{2m}\sum_{j}^{nC}\sum_{k}^{H} W_{k,j}^2 \tag{4}$$

Adding R is equivalent to placing a Gaussian prior on the parameters to help with overfitting.

II. Hidden Neural Network

Taking the word-level features as raw input, we utilize a 1 layer hidden neural network to perform the classification. The topmost neuron is a softmax layer, outputting a classification $\in [0, 1)$. The middle layer is the hidden layer, and we illustrate the paths of the input layer with only the first two words.



As such, the parameters to our hidden neural network model are as follows:

- $U \in \mathbb{R}^{Hx1}$
- $W \in \mathbb{R}^{Hxnd}$
- $B_1 \in \mathbb{R}^{Hx1}$
- $B_2 \in \mathbb{R}^{1}$

H is the number of hidden layer neurons, n is the number of input words that have d-dimensional feature vectors. For a particular neuron in the hidden layer, it receives information from the input layer and outputs:

$$a_1 = f(W_1.X + B_1) \tag{5}$$

where $f(x)$ is a nonlinear sigmoid-like function. The softmax layer then takes each hidden neuron as an input and produces a classification according to

$$z = U^T a + B_2 \tag{6}$$

where a is a vector of outputs from the hidden layer. Finally, the topmost neuron returns

$$h(z) = \frac{1}{1 + \exp -z} \tag{7}$$

which has the same form as logistic regression and a 2-class maximum entropy model. We classify as positive if it passes the score threshold of 0.5.

III. Update Equations

The SGD update equations follow.We include the partial update to a positive example (the negative example equations are analagous). One particular design decision is to keep the vocabulary matrix L fixed due to the contextual awareness of each word feature. We define the following equations:

$$h_\theta(x) = \frac{1}{1 + \exp -j(x)} \tag{8}$$

$$g(x) = -(1 + \exp -j(x))^2(\exp j(x)) \tag{9}$$

$$j(x) = U^T f(x) + B_2 \tag{10}$$

$$f(x) = tanh(Wx + B_1) \tag{11}$$

$$f'(x) = 1 - tanh^2(Wx + B_1) \tag{12}$$

Our update equations for a single data point are as follows

$$\frac{\partial J(\theta)}{\partial U} = g(x)f(x) + \frac{\partial R(\theta)}{\partial U} \tag{13}$$

$$\frac{\partial J(\theta)}{\partial W} = g(x)U^T f'(x) \times X + \frac{\partial R(\theta)}{\partial W} \tag{14}$$

$$\frac{\partial J(\theta)}{\partial B_1} = diag((g(x)U^T \times f'(x)) \tag{15}$$

$$\frac{\partial J(\theta)}{\partial B_2} = g(x) \tag{16}$$

where in equations (14,) we use the cross product operator. The regularization gradients are

$$\frac{\partial J(\theta)}{\partial U} = C\dot{U} \tag{17}$$

$$\frac{\partial J(\theta)}{\partial W} = C\dot{W} \tag{18}$$

$$\tag{19}$$

IV. Experiments

I. Data

Initially we set out to predict the punniness of certain sentences. The original hypothesis was that sentences are punny because specific words in the sentence are used in more than one context and as such, are jarring enough to be funny to the reader. We began labeling puns but it soon became too time intensive. Our final dataset consists of 182 high quality puns; negative examples are sentences taken from the constitution by nature of it being a serious document. This is our smaller evaluation dataset–given the dimensionality of our features, this amount of data is too prone to overfitting especially considering that some of it is omitted for a test set. In our initial testing, we obtained results that were too good to be true; nonetheless it is interesting to consider training on another dataset and evaluating solely on puns.

With the advent of twitter, it is relatively simple to access curated feeds. Most notably for this experiment, we leverage this aspect for training examples. We obtain "funny" examples from joke feeds (@TheComedyJokes, @FunnyJokeBook, @funnyoneliners) and "unfunny" examples from authoritative news feeds (@BreakingNews, @TheEconomist, @WSJ). Some examples of each type of sentence found in our dataset are:

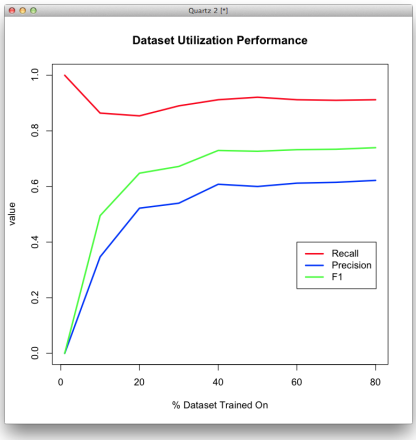
· **Pun**: i tried looking for gold, but it didn't pan out
· **Constitution**: To prove this, let Facts be submitted to a candid world
· **Joke**: Pizza is the only love triangle I want.
· **News**: Artwork purchased by city of Detroit valued at up to $866 million.

Our balanced dataset contains 6700 examples of each with the words in the sentences stemmed and retweets omitted. We then truncate the sentence length to a maximum of 10 words, with the "blank" word appended if there are fewer than 10 words in the sentence.

In SGD we essentially alter the parameters by their respective gradient of the cost function whenever we make an incorrect classification; the importance of each example is controlled by the parameter α, which downweights each gradient. We also have the regularization parameter C from equations (17, 18)

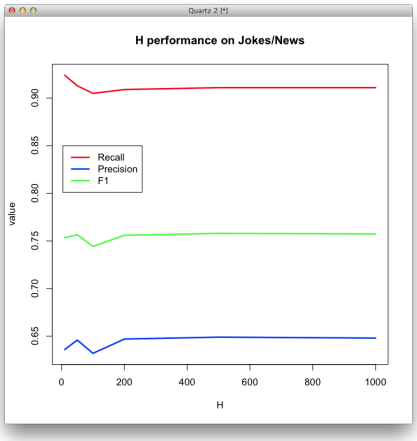
II. Results

We include a plot detailing the accuracy of the classifier with varying amounts of training and evaluation data on the full word vectors with contextual features. In these experiments we have 10 hidden states. We observe that additional data does not help the performance after 40%, or around 2000 sentence pairs.

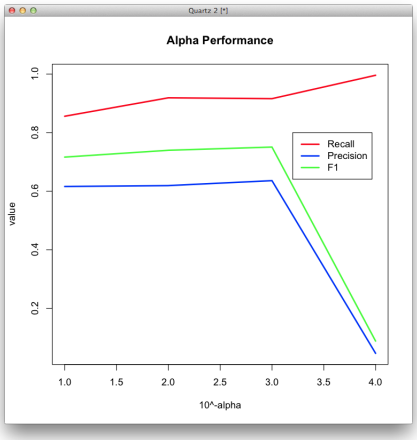


We also vary the number of hidden states. If we include too many hidden states, it is theoretically possible to predict the training data exactly and thus overfit (since the size of the weight matrix depends on the number of neurons in the hidden layer). After an initial

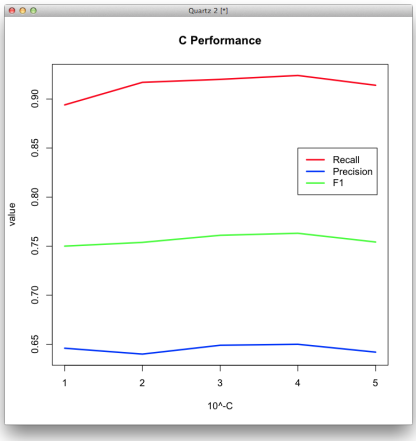
dip that is probably due to the variance of SGD, performance remains roughly constant.



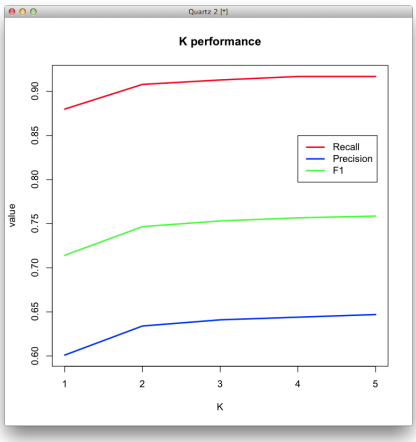
We also vary the learning parameter α. We observe that making α too small for this task kills precision.



We vary the regularization parameter C and for this particular problem it does not seem to matter much. Intuitively, since we are dealing with such high dimensional data it should be rather sensitive to regularization; however if the features are relatively evenly distributed across the dimensions regularization will not have that large a contribution.

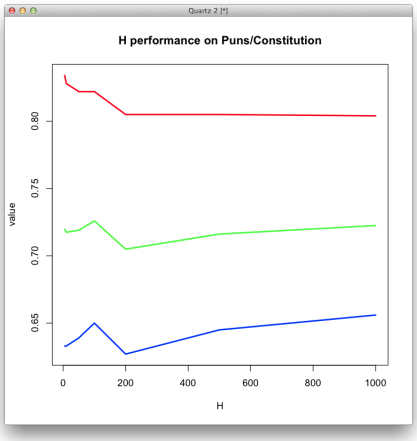


We vary the number of iterations of SGD, K and observe that there are but marginal performance gains after K=3 iterations.



III. Analysis

We evaluate our derived parameters on our dataset of puns and sentences from the constitution. Using the same parameters above when we varied the size of the hidden states, we see that 100 hidden states produces the best F1 score.



Upon inspection we do not see a distinguishable habitual error category that our neural network is predicting incorrectly.

Precision	Recall	F1
.650	.822	.726

It is interesting that even though we trained on jokes we are still able to generalize our predictions to the domain of puns. However it is difficult to make incremental progress on this model due to the nature of hidden neural network; instead, one must start afresh in thinking of the structure of the feature vector and the scoring function and the data set upon which to utilize for training and evaluation.

V. Future Work

In our initial literature review, we came across Recursive Neural Networks [10] as a good way of modeling sentence meaning because it allows for arbitrarily long sequences of words (rather than truncating sentence length or appending blanks). However we could not formulate a strong scoring function and were stranded in the swamps of implementation details.

References

[1] Collobert, Ronan. et al., (2009). Natural Language Processing (Almost) from Scratch Journal of Machine Learning Research

[2] Mihalcea, Rada and Carlo Strappavaral, (2006). Learning to Laugh (Automatically): Computational Model for Humor Recognition Computational Intelligence, Volume 22, Number 2, 2006

[3] Kiddon, Chloe and Yuriy Brin, (2011). That's What She Said Double Entendre Identification Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics:shortpapers, pages 89–94

[4] Taylor, Julia and Lawrence Mazlack, (2004). Computationally Recognizing Wordplay in Jokes

[5] Turian, Joseph et al. , (2010). Word representations: A simple and general method for semi-supervised learning

[6] Erk, Katrin and Sebastian Pado, (2008). A Structured Vector Space Model for Word Meaning in Context

[7] Mitchell, Jeff and Mirella Lapata, (2008) . Vector-based models of semantic composition. In Proceedings of ACL, 236–244.

[8] Manning, Chris and Richard Socher CS224n: PA4 assignment sheet. http://nlp.stanford.edu/ socherr/pa4_ner.pdf

[9] Bengio, Yoshua et al., (2003). A Neural Probabilistic Language Model Journal of Machine Learning Research 3 (2003) 1137–1155

[10] Socher, Richard et al. , (2013). Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank