Humor Detection in Yelp reviews

Luke de Oliveira
Stanford ICME
lukedeo@stanford.edu

Alfredo Láinez Rodrigo
Stanford ICME
alainez@stanford.edu

Abstract

We utilize the state-of-the-art in deep learning to show that we can learn by example what constitutes humor in the context of a Yelp review. To the best of the authors knowledge, no systematic study of deep learning for humor exists – thus, we construct a scaffolded study. First, we use "shallow" methods such as Random Forests and Linear Discriminants built on top of bag-of-words and word vector features. Then, we build deep feedforward networks on top of these features – in some sense, measuring how much of an effect basic feedforward nets help. Then, we use recurrent neural networks and convolutional neural networks to more accurately model the sequential nature of a review.

1 Introduction

Yelp's utility relies on being able to provide an accurate, crowd-sourced opinion of a place of business. However, how can we determine which reviews provide valuable insight into a business? Does humor play an important role on businesses reviews? We intend to predict community-determined humor in Yelp reviews. Every Yelp review has three qualities that can be voted on by other users: "funny", "useful", and "cool". We will zero in the study of "funny" reviews.

This is an interesting NLP problem in that we must learn higher order structures that constitute humor. In doing this, we will first establish a baseline using traditional methods to set the barrier for deep learning to surpass. After that, we will use improvements both in terms of deep learning state-of-the-art models and in features representation, utilizing vector representations of words. We show that, while it is difficult to capture high order structures of humor, the improvement over traditional methods is clear.

2 Problem Description

2.1 Dataset

We will use the Yelp Dataset Challenge dataset, which consists of 1.6 million reviews by 366.000 users for 61.000 businesses. Each review consists of one or more sentences commenting on the business at hand, along with votes given by other users to the review – particularly, "funny", "useful", and "cool". We will be considering the "funny" category in this project. Since the data is so vast, we will always utilize totally balanced training and test sets.

2.2 Defining Humor

Even though we have a vote total for the number of users who found a particular review funny, we note that there is a lot of noise in the votes with only one "funny" vote. By no means a systematic study, we looked at the types of reviews that were receiving one and two "funny" votes by the collective Yelp user base. In most cases, by visual inspection, we noted that these reviews weren't in fact funny. In this exposition, we tackle the binary classification problem of funny-vs-not-funny.

As we proceed, we make an arbitrary assumption – a review is "funny" if it obtains *three or more* funny votes.

2.3 Background

There is not a large body of work on so-called "computational humor". Work that exists is largely in the pure NLP domain, and uses very contrived features and simplistic tree methods or SVMs [MS06]. Part of the issue we face is that we will have no idea whether or not our models extend *out-of-domain*, to a transfer learning setting. This is because there is no labeled corpus of funny texts – partially due to the subjectivity associated with imposing a binary outcome on something such as humor. Some work has been done applying similar methods to Twitter, but no large, labelled datasets exist for thorough validation.

2.4 Preprocessing

As with any text analytics problem, we utilize tokenization. We build word vectors using the excellent `gensim` package. Note that we *do not* remove stopwords or punctuation, as some sequences of characters are quite expressive in the context of Yelp. For example, consider a question mark.

```
In [1]: w2vmodel.most_similar('?')
Out[1]:
[('kidding', 0.8124330043792725),
 ('heck', 0.7966572642326355),
 ('wtf', 0.7851443290710449),
 ('hell', 0.7803547382354736),
 ('what', 0.7729526162147522),
 ('anyways', 0.7656563520431519),
 ('why', 0.7625856995582581),
 ('mean', 0.7603179812431335),
 ('ya', 0.7591337561607361),
 ('yeah', 0.7561845779418945)]
```

Note how intuitively this is different that what we would traditionally associate with the semantic meaning of "?". As a second example, consider the character "!".

```
In [2]: w2vmodel.most_similar('!')
Out[2]:
[('awesome', 0.8051108121871948),
 ('amazing', 0.8016425967216492),
 ('omg', 0.7824660539627075),
 ('def', 0.7814326286315918),
 ('soooo', 0.7789645195007324),
 (':)', 0.770503044128418),
 ('sooooo', 0.7704253318336487),
 ('sooo', 0.7700715065002441),
 (':))', 0.7666398286819458),
 ('soo', 0.7654560804367065)]
```

We hypothesize that domain-specific embeddings such as this are crucial for a difficult task such as humor detection.

We consider the classification problem, and decided to convert this into a *class balanced* problem – selecting equal numbers of funny and not-funny reviews. This allows for more easily interpretable accuracy results.

2.5 Evaluation

In Yelp's Kaggle competition[1], they suggest a metric for evaluation of a count based target called Root Mean Squared Logarithmic Error (RMSLE). It is defined as

$$\mathsf{RMSLE}(\hat{y}, y) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(1+\hat{y}_i) - \log(1+y_i))^2}, \tag{1}$$

which allows us to focus on detecting the presence of humor rather that worrying about distinguishing between marginal humor. Note the presence of the +1 offset, allowing the logarithm to be defined at zero.

However, through experimentation, we discovered that this metric is not easy to interpret, and is not a particularly intuitive metric. Though numerically sound and probably the most logical way of predicting vote totals, we cannot grasp how well a model works in this space. In order to understand how well our models work, we convert the problem into the binary classification regime. After balancing the classes according to subsection 2.2, we optimize the classical log-likelihood and report the classification accuracy.

Though not a formal exposition, we also note that confusion matrices produced by the tested methods yielded *very* symmetric results – for the sake of brevity we do not include them as they provide little additional insight into our problem.

3 Shallow Algorithms

We will build up our examination of humor prediction in a scaffolded way. We begin with baseline models using classical machine learning on top of bag-of-words, with a maximum number of features set at 500. This helps us determine a baseline for humor prediction. Then, we use the excellent `gensim` package to learn word vectors over the dataset, and for a given review, we simply take the mean of all word utterances.

First, we consider classical methods built on top of Bag of Words representations.

Method	**Train Accuracy**	**Dev Accuracy**	**Test Accuracy**
SGD, Logistic Classifier	0.7363	0.7359	0.7229
SVM, RBF Kernel	0.7451	0.7439	0.7324
Random Forest	0.7568	0.7559	**0.7542**

Table 1: Accuracy of "shallow" methods build on BOW

We obtain a good baseline for us to build our more complicated, deep methods.

Now, we consider using shallow algorithms on top of word vector representations. First, we utilize the `gensim` package to build word vectors of dimension 100 over our entire corpus. Denote the word vector for word i as v_i. A phrase vector is simply a mean of the v_i of a phrase. Specifically, a phrase P is simply a subset of the natural numbers, i.e., $D \subset \mathbb{N}$. Thus, the phrase vector v_D is

$$v_D = \frac{1}{|D|}\sum_{i \in D} v_i.$$

In the rest of the paper, we refer to phrase vectors as PVs.

We note something very peculiar here – shallow methods are not very good at picking up on humor when we work in the word vector space. We obtain better performance using simple SGD with a log-loss on the tf-idf reweighted bag of words. Now, we can parse out exactly how much of a performance increase can be seen using deep learning.

[1] https://www.kaggle.com/c/yelp-recruiting/details/evaluation

Method	Train Accuracy	Dev Accuracy	Test Accuracy
SGD, Logistic Classifier	0.7012	0.7001	0.6992
SVM, RBF Kernel	0.7172	0.7124	0.7195
Random Forest	0.7522	0.7118	**0.7202**

Table 2: Accuracy of "shallow" methods build on PVs

4 Deep Feedforward Networks

Before we examine deep models, we first describe layer notation.

- $\mathsf{FC}(n, m)$ – Fully Connected, dense layer n inputs, m outputs.
- $\mathsf{Dropout}(p)$ – standard dropout layer à la CITE.
- $\mathsf{MaxOut}(n, m, k)$ – MaxOut layer with n inputs, m outputs, and k piecewise elements.
- $\mathsf{Conv2D}((a, b), k, n)$ – Convolution layer with filters of shape `(a, b)`, with k channels, and n features.
- $\mathsf{Pool2D}((a, b))$ – Standard max-pooling layer with pool areas of shape `(a, b)`

With both the bag-of-words and PV representations, we use two types of architectures – a classic, deep rectifier network with heavy dropout, and a slightly more modern, shallower MaxOut [Goo+13] network, once again with dropout. Here, we outline the two basic archetypes.

- Type A – $[\mathsf{FC} \to \mathsf{ReLU} \to \mathsf{Dropout}] \times p \to \mathsf{FC} \to \mathsf{sigmoid}$
- Type B – $[\mathsf{MaxOut} \to \mathsf{Dropout}] \times 2 \to \mathsf{FC} \to \mathsf{sigmoid}$

Continuing our scaffolding, we build deep feedforward nets on top of bag-of-words representations. All parameters were optimized, though the dropout rates and layer sizes did not have a large effect on performance. We utilized Adagrad [Zei12] for our optimization, and trained for 25 epochs using a batch size of 50 over a 50-50 split on training, dev, and testing sets.

Method	Train Accuracy	Dev Accuracy	Test Accuracy
Type A	0.7602	0.7690	**0.7593**
Type B	0.7790	0.7624	0.7577

Table 3: Accuracy of deep nets build on BOW

Now, we consider building a basic deep network on top of mean word vector representations. For this particular problem, we utilized Adadelta [Zei12] for our optimization, and trained for 30 epochs using a batch size of 30 over a 50-50 split on training, dev, and testing sets.

Method	Train Accuracy	Dev Accuracy	Test Accuracy
Type A	0.7802	0.7790	0.7763
Type B	0.7991	0.7829	**0.7821**

Table 4: Accuracy of deep nets build on PVs

As we can see, deep nets on top of word vectors provide a much more reasonable baseline to continue our investigation into more complicated models. In addition, the ReLU units in Model B seem to provide a great basis for learning.

There is a very important piece of wisdom to be taken away from these results – note that without deep learning, BOW representations perform better than PVs. However, when we build Deep Nets on top of BOW and PVs, the performance with PVs surpasses the performance of both Deep Nets and traditional methods on BOW, indicating that there is some meaning contained in word vectors that deep learning is able to pick up that traditional methods cannot.

5 Recurrent Neural Networks

As a next step, we decided to leverage on models that try to model sequences of word vectors. We can view the reviews as a sequence where we only have one label at the end. This many-to-one mapping lends itself nicely to Recurrent Neural Networks. We tried to explot the idea of keeping memory units to try to capture long distance dependencies in the text, and so we implemented LSTMs and GRU models. Although both of them have proved to be very similar in performance in our experiments, we have noticed a slight general improvement with GRUs, and so we will report those results. We have followed the model explained in [Chu+15].

Training with LSTMs and GRUs have been delicate since convergence was not assured, and in addition, parameters such as the optimizer were critical. However, we have found an architecture which yields good results, combining a recurrent GRU with a layer of 128 hidden units with sigmoid activation. We have used the *adam* optimizer, which outperformed other optimizers such as *adagrad*, *RMSProp* or *sgd*. These had slower or no convergence at all. The recurrent layer has a dropout of 0.5 and the sequence was padded to a maximum of 100 word vectors for memory reasons.

Method	**Train Accuracy**	**Dev Accuracy**	**Test Accuracy**
GRU, word vectors fixed	0.7903	0.7796	**0.7820**

Table 5: Accuracy of GRU network

While the results are good compared to the basic baseline, the recurrent network does not clearly outperform the deep models trained on PVs. Although the advanced complexity of Gated Recurrent Units would suggest a better capture of humorous text structures, we may conclude that understanding humor in a sequence of text is too complex for the network to process.

6 Convolutional Networks

We examine convolutional networks à la [Kim14] for the task of humor detection. We construct all models using the Theano library[Bas+12]. As a modeling choice, we truncate all reviews at 150 words after consulting with the distribution of review length.

Let $\bigoplus$ be the horizontal-wise concatenation operator. Our model takes as input a word vector matrix over a vocabulary of size V and provides word vectors of size w. We let this matrix be defined as $U \in \mathbb{R}^{V \times w}$, where $U[i]$ is the word vector (a row vector) for word i. Note that we set $U[0]$ to be the zero vector for the padding. So, for a given sentence $s = (s_1, s_2, \ldots, s_L)$, we define the sentence image as

$$X_s = \bigoplus_{i=1}^{L} U[s_i]^T$$

Where now the ith column of X_s is the word vector associated with the ith word in s. For a given n-gram size n, we can construct convolution filters of shape $(w \times n)$ followed by pooling of shape $(L - n + 1 \times 1)$.

In constructing our model, we use n-grams of size 3, 4, 5, and 6, where each filter has 10 features. Let this set be N. We then apply dropout, and have a fully connected layer that feeds into our sigmoid. Using our notation, we have

$$\begin{aligned} \mathsf{MultiConv} &= \mathsf{Concat}([\mathsf{Conv2D}((w,n),1,10) \to \mathsf{ReLU} \to \mathsf{Pool2D}((L-n+1,1))]_{n \in N}) \\ \mathsf{MultiConv} &\to \mathsf{Dropout} \to \mathsf{FC} \to \mathsf{ReLU} \to \mathsf{Dropout} \to \mathsf{FC} \to \mathsf{sigmoid} \end{aligned}$$

We used [Kim14] as our guide for parameters – we use a hidden layer size of 100. In addition, we use Adadelta [Zei12] for 20 epochs over a batch size of 100.

Method	Train Accuracy	Dev Accuracy	Test Accuracy
Conv, word vectors fixed	0.8322	0.8001	0.8010
Conv, word vectors optimized	0.8331	0.8093	**0.8101**
Conv, random word vectors	0.8193	0.7941	0.7986

Table 6: Accuracy of Convolutional networks

In addition to the architecture proposed in [Kim14], we implemented a MaxOut extension with three knots in the convolutional layers. That is, for each n-gram filter, we in fact use three replicas of the convolutional filters – then, we apply an element-wise maximum across the three replicas. We train for 25 epochs with a batch size of 100, and we use Adagrad for the optimization algorithm.

Method	Train Accuracy	Dev Accuracy	Test Accuracy
MaxoutConv, word vectors fixed	0.8329	0.8102	0.8114
MaxoutConv, word vectors optimized	0.8539	0.8115	**0.8157**
MaxoutConv, random word vectors	0.8010	0.7977	0.7912

Table 7: Accuracy of Maxout Convolutional networks

7 Conclusion

To the best of our knowledge, we have performed the first systematic study of utilizing deep learning for humor detection. Though we had to make some arbitrary decisions with respect to what to call a "funny" review, we see consistent and promising results on the class balanced problem.

We began by constructing both deep and shallow models over bag of words and mean word vector representations – we note that shallow methods perform better over bag of words based features, while deep models are more performant on the mean word vector representation of phrases. We hypothesize that this is due to the fact that the expressive deep networks can use the semantic information contained in the word vectors.

We then progressed to using recurrent neural networks, in particular GRUs. We expected that memory units in a sequence of word vectors would be able to better capture humor traits in text. While the results clearly outperform shallow methods, they yield similar accuracies to those of deep models over averages of word vectors, leading to the most probable conclusion that humor structures are too complex for this model to leverage on.

Finally, we examined convolutional networks à la [Kim14], and provided a simple MaxOut extension. We see that convolutional networks afforded the best results across all the methods we considered, reaching a maximum of 81.57% accuracy on the class balanced problem.

Overall, we learned a few key facts regarding humor detection and deep learning. As evidenced by our application of deep nets on word vector and BOW features, deep learning manages to learn more expressive relationships between semantics and our binary outcome of humor. In addition, it seems that methods which allow for the contextual nature of a sentence yield better performance. In the future, it would be wonderful to obtain (via Amazon Turk) a large, labelled dataset of funny / not-funny texts with different humor attributes (i.e., play-on-words, offensive, pun, etc.) and varied sources. Overall, we are pleased that we obtain intuitive results on a publically available dataset with an ill-posed and elusive target – user identified humor.

References

[MS06] Rada Mihalcea and Carlo Strapparava. "Learning to Laugh (Automatically): Computational Models for Humor Recognition". In: *Computational Intelligence* 22.2 (2006), pp. 126–142. ISSN: 1467-8640. DOI: 10.1111/j.1467-8640.2006.00278.x. URL: http://dx.doi.org/10.1111/j.1467-8640.2006.00278.x.

[Bas+12] Frédéric Bastien et al. *Theano: new features and speed improvements*. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop. 2012.

[Zei12] Matthew D. Zeiler. "ADADELTA: An Adaptive Learning Rate Method". In: (Dec. 2012). eprint: 1212.5701. URL: http://arxiv.org/abs/1212.5701.

[Goo+13] Ian J. Goodfellow et al. "Maxout Networks". In: (Feb. 2013). eprint: 1302.4389. URL: http://arxiv.org/abs/1302.4389.

[Chu+14] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: (Dec. 2014). eprint: 1412.3555. URL: http://arxiv.org/abs/1412.3555.

[Kim14] Yoon Kim. "Convolutional Neural Networks for Sentence Classification". In: (Aug. 2014). eprint: 1408.5882. URL: http://arxiv.org/abs/1408.5882.

[Chu+15] Junyoung Chung et al. "Gated Feedback Recurrent Neural Networks". In: (Feb. 2015). eprint: 1502.02367. URL: http://arxiv.org/abs/1502.02367.