Automatic Detection of Humor In Yelp Reviews Using Deep & Shallow Learning Methods

Abu Saleh Md Noman[*]

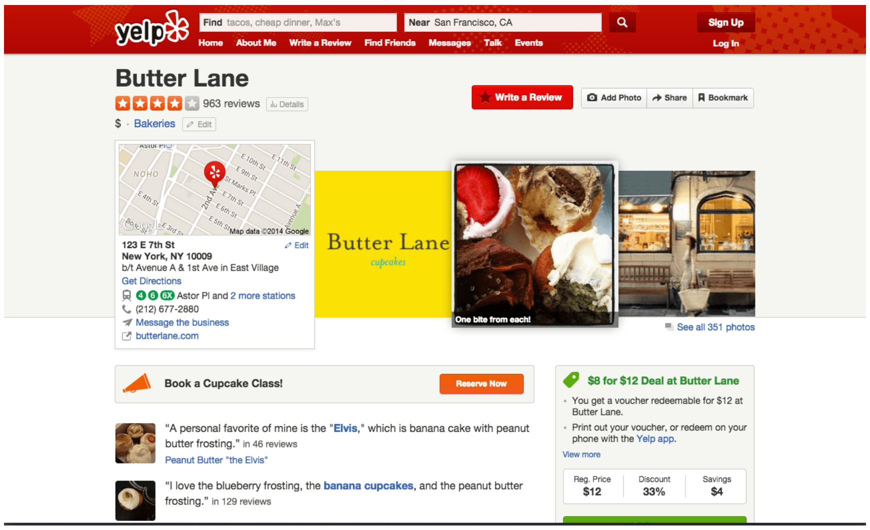
May 5, 2018

Abstract

Humor is an indispensable part of textual communication. In this project I aim to detect humor in unstructured text such as Yelp reviews. First, I develop the definition of humor and collect & preprocess the data. Then I use state of the art shallow (e.g. SVM) and deep learning (e.g. Feed-Forward Neural Network, Convolutional Neural Network) techniques to model the reviews and compare the accuracy. My finding suggests, shallow method perform better using bag of words method while deep learning captures humor better using word vectors. My findings has implication for business model developers and AI researchers alike.

1 Introduction

The main business model of Yelp is providing user-generated content to users or reviews. The people here post reviews on different service such as restaurant, local business, nightlife, home service etc. They discuss the sentiment associated with those services and other users can see the ratings and make further decisions.



Butter Lane

963 reviews

$ · Bakeries

123 E 7th St
New York, NY 10009
b/t Avenue A & 1st Ave in East Village
Get Directions
4 6 6X Astor Pl and 2 more stations
(212) 677-2880
Message the business
butterlane.com

Book a Cupcake Class!

"A personal favorite of mine is the "Elvis," which is banana cake with peanut butter frosting." in 46 reviews
Peanut Butter "the Elvis"

"I love the blueberry frosting, the banana cupcakes, and the peanut butter frosting." in 129 reviews

$8 for $12 Deal at Butter Lane
- You get a voucher redeemable for $12 at Butter Lane.
- Print out your voucher, or redeem on your phone with the Yelp app.

Reg. Price	Discount	Savings
$12	33%	$4

Figure 1: Example Yelp review.

Figure 1 shows a typical Yelp review. Users can give ratings, estimate the probable cost, provide feedback & suggestions etc. A lot of previous work addressed sentiment analysis, opinion mining on this kind of crowdsourced opinions. However, it is still not clear which reviews are particularly important from

[*]Indina University, Bloomington

Yelp's point of view. Can we get any corelation between importance of a review and associated humor? Also, does humorous means they are not useful? Every Yelp review has three qualities that can be voted on by other users: "funny", "useful", and "cool". I wanted to reduce this to a binary class problem where everything apart from 'funny' is considered non-funny. Automatic detection of humor in unstructured text has not been well-studied and thus require further attention. So the goal of this project is:

Can deep learning based NLP techniques capture innate humor in Yelp reviews? How does this compare to the understanding of shallow learning findings?

The goal of all NLP technologies is to understand the inherent meaning from text by alleviating human effort. Advancement in machine learning algorithms has guranteed success in various text classification based tasks such as author profiling, sentiment analysis etc. The main motivation of this project engenders from the demand of understanding a crucial aspect of human discourse and its implication in fledgling businesses like Amazon, Yelp etc. Full computational understanding of humor will also enable intelligent agent to moderate such forums more efficiently without human efforts.

There were several challenges that I faced unlike any other text classification task. First, the insight drawn from the dataset is domain specific but I did not find enough data available online that can capture a generic picture of humor. Second, the definition of 'funny' is highly subjective and fuzzy; something funny to someone may not be fun to someone else. So the application of our model out of domain can be questionable because of the lack of labeled data for other domains as mentioned earlier. Also, I did not find a lot of related work in this regard. Humor is relatively well studied in scientific fields such as linguistics [2] and psychology [5, 8] but not computationally. Some work has been done applying similar methods to Twitter, but no large, labelled datasets exist for thorough validation. Some work uses very contrived features and simplistic tree methods or SVMs [7]. This work is motivated by [4, 1]. To this end, the contribution of this project is devising a novel extension of mixed method approach to detect humor in Yelp reviews, compare the findings and report the implications of my findings. To do this, first, I will explain the how the data was acquired and preprocessed, followed by setting up a baseline method. Later, I will show feature representation, vectorization and word embeddings from input data and how to apply them to build and compare outputs from Support Vector Machine (SVM), Feed Forward Neural Networks (FFN) and Convolutional Neural Networks (CNN).

2 Methods

2.1 Dataset

I am using the Yelp Dataset Challenge dataset, which consists of about 1.6 million reviews by 366,000 users for 61,000 businesses. The Yelp dataset is a subset of their businesses, reviews, and user data for use in personal, educational, and academic purposes. Available in both JSON and SQL files, it is used to teach students about databases, to learn NLP etc. Each yelp review has user given votes for three categories: "funny", "cool" and "useful". I am using this community provided data to get our dataset. I extracted a balanced set of 1,00,000 reviews, containing equal number of samples for both classes. This data is to be used for training and testing purposes. Upon close inspection I found that often times if a review receives only 1,2,3 funny votes, those are not actually funny. Also due to the fuzzy nature the definition of funny review is:

"A review is considered funny if it has received at least 5 funny votes."

2.2 Preprocessing

As with any text analytics problem, I utilize tokenization. I build word vectors using the gensim package. Note that I do not remove stopwords or punctuation, as some sequences of characters are quite expressive in the context of Yelp. I hypothesize that domain-specific embeddings such as this are crucial for a difficult task such as humor detection. I 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.

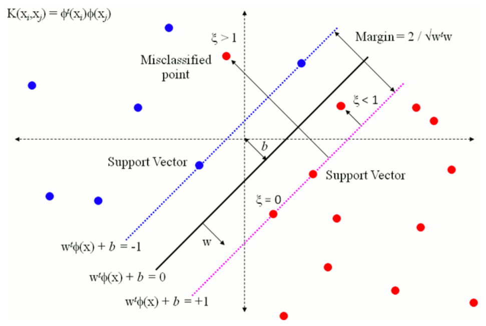


Figure 2: The figure shows a linear SVM classifier for two linearly separable classes. The hyperplane $\mathbf{w}^T x + b$ is the solid line between H_1 and H_2, and the the margin is M.

2.3 Used Approach and Modules

Different word representations (vectoriaztion) and learning algorithms used are explained here. I used SVM as shallow learning and FFN & CNN as deep learning algorithm. I reused and replicated the codes from the famous paperwork [10, 1]. For evaluation, I optimized the classical log-likelihood and report the classification accuracy.

2.3.1 SVM

Support Vector Machine (SVM) is one of the most widely used classification methods. SVM cares only about the data points near the class boundary and finds a hyperplane that maximizes the margin between the classes. Let the input be a set of N training vectors $\{\mathbf{x_n}\}_{n=1}^N$ and corresponding class labels $\{y_n\}_{n=1}^N$, where $\mathbf{x_n} \in \mathbb{R}^D$ and $y_n \in \{-1, 1\}$. Initially we assume that the two classes are linearly separable. The hyperplane separating the two classes can be represented as:

$$\mathbf{w}^T \mathbf{x}_n + b = 0,$$

such that:

$$\begin{aligned} \mathbf{w}^T \mathbf{x}_n + b &\geq 1 \quad \text{for} \quad y_n = +1, \\ \mathbf{w}^T \mathbf{x}_n + b &\leq -1 \quad \text{for} \quad y_n = -1. \end{aligned}$$

Let the two hyperplanes (Figure 2) separating the classes such that there is no other data point between them. Our goal is to maximize the margin M between the two classes. The objective function:

$$\begin{aligned} \max_{\mathbf{w}} \ & M \\ \text{s.t.} \ & y^n(\mathbf{w}^T \mathbf{x_n} + b) \geq M, \\ & \mathbf{w}^T \mathbf{w} = 1. \end{aligned}$$

The margin M is equal to $\frac{2}{\|\mathbf{w}\|}$. We can rewrite the objective function as:

$$\begin{aligned} \min_{\mathbf{w}} \ & \frac{1}{2} \mathbf{w}^T \mathbf{w} \\ \text{s.t.} \ & y^n(\mathbf{w}^T \mathbf{x_n} + b) \geq 1 \end{aligned}$$

Now, let's consider the case when the two classes are not linearly separable. We introduce slack variables $\{\xi_n\}_{n=1}^N$ and allow few points to be on the wrong side of the hyperplane at some cost. The modified

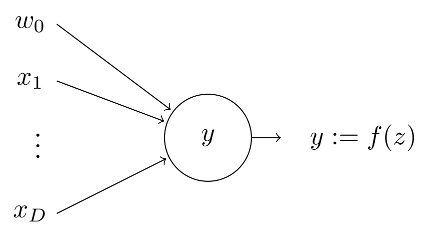


Figure 3: Single processing unit and its components. The activation function is denoted by f and applied on the actual input z of the unit to form its output $y = f(z)$. $x_1, \ldots, x_D$ represent input from other units within the network; w_0 is called bias and represents an external input to the unit. All inputs are mapped onto the actual input z using the propagation rule.

objective function:

$$\begin{aligned} \min_{\mathbf{w}} \ & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\xi_n \\ \text{s.t.} \ & y^n(\mathbf{w}^T\mathbf{x_n} + b) + \xi_n \geq 1, \\ & \xi_n \geq 0, \ \ \forall n. \end{aligned}$$

The parameter C can be tuned using development set. This is the primal optimization problem for SVM.

The advantages of SVM are:

- Produce very accurate classifiers.
- Less overfitting, robust to noise.

While the disadvantages are:

- SVM is a binary classifier. To do a multi-class classification, pair-wise classifications can be used (one class against all others, for all classes).
- Computationally expensive, thus runs slow.

A kernel is a similarity function. It is a function that is provided to a machine learning algorithm. It takes two inputs and spits out how similar they are.

My Approach: For this project, I have used the scikit-learn python implementation, and performed experiments using both linear and RBF kernel. The input was vectorized using bag of words based on tf-idf scores. The parameters were chosen: **C = 1** (The C parameter trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly by giving the model freedom to select more samples as support vectors.) **Gamma = auto** (the gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. The gamma parameters can be seen as the inverse of the radius of influence of samples selected by the model as support vectors.)

2.3.2 NN & FFN

An artificial neural network, also referred to as "neural network", is a set of interconnected processing units [9]. A processing unit receives input from external sources and connected units and computes an output which may be propagated to other units. These units represent the neurons of the human brain which are interconnected by synapses. A processing unit consists of a propagation rule and an activation function. The propagation rule determines the actual input of the unit by mapping the output of all direct predecessors and additional external inputs onto a single input value. The activation function is then applied on the actual input and determines the output of the unit. The output of the processing unit is also called activation. This is illustrated by Figure 3 showing a single processing unit where f denotes the activation function, z the actual input and y the output of the unit. The perceptron consists of D input units and C output units. Every input unit is connected to every output unit as shown in Figure 4.

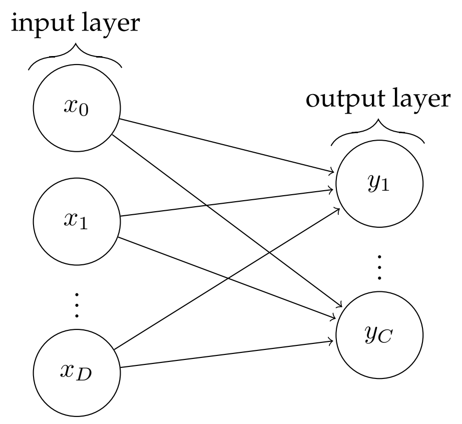


Figure 4: The perceptron consists of D input units and C output units. All units are labeled according to their output: $y_i = f(z_i)$ in the case of output units; x_i in the case of input units. The input values x_i are propagated to each output unit using the weighted sum propagation rule. The additional input value $x_0 := 1$ is used to include the biases as weights.

Since we can model every neural network in the means of a network graph we get new neural networks by considering more complex network topologies. One type of topology is FFN. In a feed-forward topology we prohibit closed cycles within the network graph. This means that a unit in layer p may only propagate its output to a unit in layer l if $l > p$. Thus, the modeled function is deterministic. In this project I consider feed-forward networks only. As demonstrated in Figure 4 the single-layer perceptron implements a feed-forward topology.

My Approach: I have built a feed forward network on the top of word vectors generated using gensim library of python. Pybrain library was used to build the feed forward network. The network was constructed to have 100 input nodes, 20 hidden layers and one output node. Here I have generated a dataset that supports 100 dimensional inputs and one dimensional targets since each input vector is 100 dimensional target is funny or non funny represented in 1 dimension (1 or 0) . Finally the network was trained till convergence.

2.4 CNN

The idea of CNNs were first introduced in image classification. CNNs are basically just several layers of convolutions with nonlinear activation functions like ReLU or tanh applied to the results. Unlike FFN, we use convolutions over the input layer to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. Each layer applies different filters and combines their results. Instead of image pixels, the input to most NLP tasks are sentences or documents represented as a matrix [3]. Each row of the matrix corresponds to one token, typically a word, but it could be a character. That is, each row is vector that represents a word. Typically, these vectors are word embeddings (low-dimensional representations) like word2vec or GloVe, but they could also be one-hot vectors that index the word into a vocabulary. For a 10 word sentence using a 100-dimensional embedding we would have a 10 by 100 matrix as our input. In NLP we typically use filters that slide over full rows of the matrix (words). Thus, the "width" of our filters is usually the same as the width of the input matrix. The height, or region size, may vary, but sliding windows over 2-5 words at a time is typical. Figure 5 summarizes this idea:

My Approach: The word representation was Word2Vec and random initial values. I used both static and non-static channels for testing. I constructed all models using the Theano library. As a modeling choice, I truncate all reviews at 150 words after consulting with the distribution of review length. In constructing the model, I use n-grams of size 3, 4, 5, and 6, where each filter has 10 features. Let this set be N. I then apply dropout, and have a fully connected layer that feeds into the sigmoid. The code I used was provided by [10]. The pre-trained Google's news corpus word vector representations are used. As suggested by the paper, I used a hidden layer size of 100 for 20 epochs over a batch size of 100.

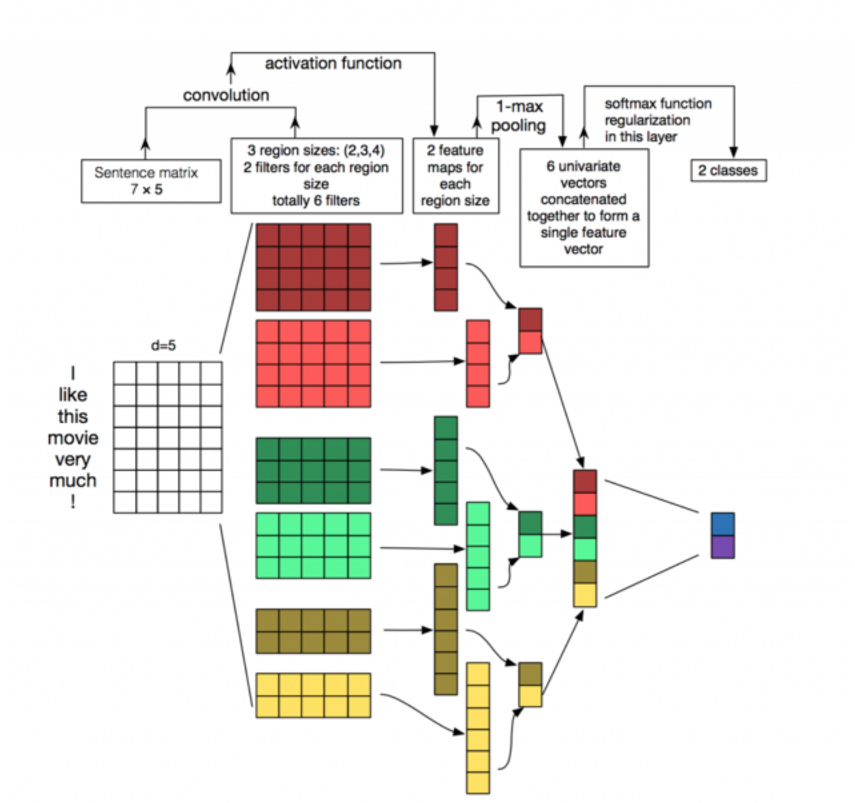

activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Figure 5: Illustration of a Convolutional Neural Network (CNN) architecture for sentence classification. Here we depict three filter region sizes: 2, 3 and 4, each of which has 2 filters. Every filter performs convolution on the sentence matrix and generates (variable-length) feature maps. Then 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer then receives this feature vector as input and uses it to classify the sentence.

3 Results & Discussion

3.1 SVM

Using the single fold approach, 1/5th of the dataset taken as testing data, rest as training data the accuracy was 83.4% for linear kernel and 50.01% for RBF kernel. Using the 5-fold cross validation approach the accuracies were given in Table 1.

Table 1: Accuracies on different iteration for SVM

Iteration	Linear	RBF
1	.83	.499
2	.834	.491
3	.829	.499
4	.834	.498
5	.831	.494
Average	.831	.496

Linear kernel gives a reasonably good accuracy. RBF kernel on the other hand gives an accuracy of nearly 50%, which is in fact a poorer performance than a random classifier considering that we are only trying to do binary classification. This may be because the parameters haven't been tuned, and multiple reruns of the RBF classifier with various parameters adjustments need to be done to come up with optimum values of c and gamma.

To test the efficacy of the trained model I used a secondary dataset consisting of about 16,000 one-liner jokes (funny), and 2000 quotes (not funny) was taken (collected from LTRC). The accuracy for Linear kernel was 57% while the RBF kernel was 67%. This is opposite to what I saw in Yelp dataset. So it can not be decided which kernel performs better and in which case. This could probably be because of the nature of data, fuzziness of humor in different context etc. Upon experiment with other humor dataset, we can draw a conclusion.

3.2 FFN

25% of the dataset is taken as test data, rest as training data. The training error score was 0.25 (details attached with submission). The results were:

- Training accuracy: 0.77
- Validation accuracy:
  - 1st Epoch: 0.763
  - 2nd Epoch: 0.755
  - 3rd Epoch: 0.741

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. However, the performance of FFN on word vectors is comparatively poorer to SVM on tf-idf based BOW representations.

3.3 CNN

Using a hidden layer size of 100 for 20 epochs over a batch size of 100, the achieved accuracies for both Word2Vec and random word vectors were given in Table 2.

Table 2: Accuracies on different channels for CNN

Channel	Word2Vec	Random
Static	.801	.744
Non-Static	.806	.781

Note that, CNN-rand is Randomly initialized word vectors. CNN-static uses pre-trained vectors from word2vec and does not update the word vectors. CNN-non-static same as CNN-static but updates word vectors during training. We can see, in general, non-static channel performs better while when a CNN is fed with word vectors which is relations among the words of a sentence it gave better results than the CNN with randomly initialized values.

4 Conclusion

The goal of this project was to automatically detect humor and compare methods used. I introduced both deep and shallow models over bag of words and word vector representations. The findings suggest that, shallow methods perform better over bag of words based features, while deep models perform well on the mean word vector representations. This is probably because word vectors are mapped into vector spaces in a way so that similar words are mapped closer therefor deep NNs can leverage this semantic information. Then I examined CNN as discussed by Kim [6], which yielded nearly 80% accuracy for the balanced dataset which is comparatively less than SVM. Therefore, although the classification accuracy is exciting I am not convinced whether Deep learning technique in my dataset, outperforms shallow algorithms. So, as a future work it will be intriguing to see how these algorithms work in presence of more crowsourced and hand annotated humor data. Also, I plan to use more modules such as logistic classifier (SGD), Random forest based ensemble methods and Recurrent Neural Networks (GRU, LSTM etc.) The implications of the findings are manifold: for AI researchers (automatic detection of humors in text), business analyst (nuanced understanding of humorous reviews) and further research can shed more light into this.

Acknowledgments

Thanks to [6, 4] for inspiring me to this project. I am also grateful to [1] for the implementation and codes.

References

[1] Humour-detection. http://srishti-1795.github.io/Humour-Detection/.

[2] S. Attardo, D. H. Attardo, P. Baltes, and M. J. Petray. The linear organization of jokes: analysis of two thousand texts. *Humor-International Journal of Humor Research*, 7(1):27–54, 1994.

[3] D. Britz. Understanding convolutional neural networks for nlp. http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/, Nov 2015.

[4] L. de Oliveira and A. L. Rodrigo. Humor detection in yelp reviews.

[5] S. Freud. *Jokes and their relation to the unconscious*. WW Norton & Company, 1960.

[6] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

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

[8] W. Ruch. Computers with a personality? lessons to be learned from studies of the psychology of humor. In *Proceeding of The April Fools Day Workshop on Computational Humor*, pages 57–70, 2002.

[9] D. Stutz. Introduction to neural networks. *Selected topics in human language technology and pattern recognition WS 13/14*, 2014.

[10] Yoonkim. yoonkim/cnn_sentence. https://github.com/yoonkim/CNN_sentence, Feb 2016.