Multiclass Sentiment Prediction using Yelp Business Reviews

April Yu
Department of Computer Science
Stanford University
Stanford, CA 94305
aprilyu@stanford.edu

Daryl Chang
Department of Computer Science
Stanford University
Stanford, CA 94305
dlchang@stanford.edu

Abstract

Many e-commerce and related sites allow text reviews and these provide much more detail than the quantitative metric of star ratings. However, star ratings provide insight on a establishment or product very quickly. Is it possible to translate the detail of text reviews into the quick usefulness of a numerical rating? We hope to explore this through multiple deep learning techniques to achieve the highest accuracy.

1 Introduction

Sites like Yelp and Amazon allow users to formalize their thoughts and opinions of businesses and products through the form of text reviews. These text reviews provide the highest granularity of detail that restaurants are able to receive directly from their customers. However, it is impossible to quantify performance from text reviews alone, which is why sites implemented a numerical rating system to complement the text reviews. If these numerical values could be learned from the text reviews themselves, users would not have to summarize their entire experience into a number between 1 and 5. We hope to use pre-trained word2vec and GLoVe vectors to run various types of neural networks on text reviews to perform multi-class sentiment analysis.

2 Problem Statement

Using the Yelp Challenge Academic Dataset and pre-trained word2vec and GloVe word vectors, we hope to run various neural networks on text reviews to predict the star rating associated with that particular review. We plan on setting a preliminary baseline for our work by averaging the word vectors and then feeding that into the Naive Bayes algorithm. With this baseline metric in mind, we then plan on implementing a recursive neural network, a matrix-vector recursive neural network and a recursive neural tensor network to achieve a multi-class sentiment prediction with the highest testing accuracy. In all approaches, we plan on using pre-trained word vectors instead of randomly initializing and then training word vectors.

We plan on segmenting our dataset into 2 sections: development and testing. We will run cross-validation of our various algorithms on the development set throughout development and will only touch the test set when it comes time to test the accuracy of the various algorithms.

3 Technical Approach and Models

As one of our baselines, we plan to use the bag-of-words model, which vectorizes a document as the occurrence counts for each word, sometimes using a reweighting scheme such as tf-idf to place more importance on unique words. However, the bag-of-words model does not account for more complex interactions, such as word order and long-range dependencies. Socher gives the example of white blood cells destroying an infection and an infection destroying white blood cells having the same bag-of-words representation, despite the former being positive and the latter being negative.

Therefore, we also plan to implement models based on vector space models (VSMs) learned from co-occurrence counts, as in GloVe and word2vec. One possible approach is semi-supervised recursive autoencoding as proposed by Socher et al., in which a document can be summarized by applying a series of autoencoders that encode vectors into a compressed hidden representation. As shown in Figure 1, the encoding process can be visualized as a binary tree in which children nodes (vectors) are progressively encoded until the entire document is summarized as a single vector. This vector can then be used as an input for different tasks; in sentiment analysis, for example, we can add a softmax layer to predict the polarity of the document using the document vector.

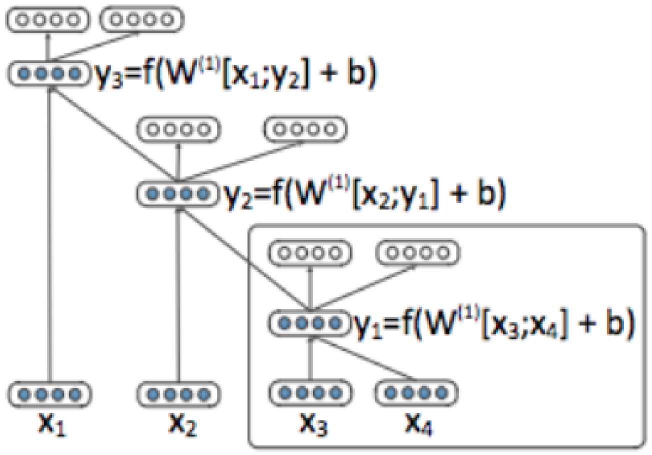
$y_3=f(W^{(1)}[x_1;y_2] + b)$
$y_2=f(W^{(1)}[x_2;y_1] + b)$
$y_1=f(W^{(1)}[x_3;x_4] + b)$
x_1
x_2
x_3
x_4

Figure 1: An illustration of the recursive autoencoding process.

The objective loss function we desire to minimize when learning the parameters is the reconstruction error. At each non-terminal node, the input vectors $i^{(1)}$ and $i^{(2)}$ can be reconstructed from the encoded representation p using the following equation:

$$[i'^{(1)}; i'^{(2)}] = W^{(2)}p + b^{(2)}$$

The reconstruction error is then the squared loss between the original inputs and the reconstructed inputs:

$$E = \frac{1}{2}\|[i^{(1)}; i^{(2)}] - [i'^{(1)}; i'^{(2)}]\|^2$$

Finally, the objective loss function is calculated as the sum of the reconstruction errors at all of the non-terminal nodes in the tree.

In order to adapt the recursive autoencoder to the task of sentiment prediction, we add a sentiment distribution calculation at each non-terminal node p by adding the following calculation:

$$D(p) = softmax(W^{(label)}p)$$

We then calculate the cross-entropy between the ground truth sentiment distribution for the document and the distribution calculated at each non-terminal node. The cross-entropies are summed up and added to the overall objective loss function. Finally, stochastic gradient descent is used to learn the weight and bias parameters.

We also intend to apply a matrix-vector recursive neural network (MV-RNN) proposed by Socher et al. (2012), which allows the compositional function to be parameterized by the words that participate in it. MV-RNN represents every word and longer phrase in a parse tree as both a vector and a matrix. When two components are combined, the matrix of one is multiplied with the vector of the other and vice versa. The original proposal for MV-RNN initializes random word vectors. Instead, we plan on utilizing pre-trained GloVe and word2vec.

With MV-RNN, the number of parameters depends on the size of the vocabulary and can become very large. To combat this, we also plan on implementing a recursive neural tensor network (RNTN) to compare its performance against the MV-RNN.

All notation above is borrowed from the relevant paper by Socher et al.

4 Intermediate/Preliminary Experiments & Results

In establishing our baseline metric for this project, we averaged the word vectors and ran them through the Naive Bayes algorithm to determine the associated numerical value. With Naive Bayes, we achieved a 24% accuracy with 5 possible classes.