Deep Learning for Amazon Food Review Sentiment Analysis

Jiayu Wu, Tianshu Ji

Abstract

In this project, we study the applications of Recursive Neural Network on sentiment analysis tasks. To process the raw text data from Amazon Fine Food Reviews, we propose and implement a technique to parse binary trees using Stanford NLP Parser. In addition, we also propose a novel technique to label tree nodes in order to achieve the level of supervision that RNN requires, in the context of the lack of labeling in the original dataset. Finally, we propose a new model RNNMS (Recursive Neural Network for Multiple Sentences), and have better results than our baseline in terms of every metrics we consider.

1 Introduction

Sentiment Analysis is an important task in NLP. Its purpose is to extract a single score from text, which makes it more convenient to analyze a large corpus of text. Various methods has been used to solve sentiment analysis problems, including bag-of-words and n-grams, and the arrival of deep learning, especially recursive neural network, provides a novel and powerful way to extract sentiment from text data.

Recursive neural network has been shown to have a stellar performance using Stanford Sentiment Treebank data [1]. However, many of text datasets are not as well labeled as Stanford Sentiment Treebank. For example, the data we have only has one label for each review which is composed of multiple sentences. Therefore, the goal of our project is to test whether recursive neural network is still effective with insufficient tree labeling. Moreover, most RNNs are designed to only consider one single sentence as input, and thus we propose RNNMS, Recursive Neural Network for Multiple Sentences, to handle multiple sentences at once.

2 Background and Related Work

Recent work has been focused on other complicated RNN models such as recursive neural tensor network [1], which is robust in detecting negating negatives, and Tree LSTM[2], which has the idea of forget gate inherited from LSTM. is a hot model and certainly worth our studies in a project.

Looking at last years project [3], the accuracy of that was 59.32% to 63.71%, depending on different Recursive Neural Network models. They developed vanilla one hidden layer, two hidden layer recursive neural networks and RNTN. In our project, we achieved 10% more than their result, which is a significant improvement. Better tree parser and amplify labeling internal nodes techniques are attributed to our better result.

Meanwhile, Stanford TreeBank, due to the strong supervision, that is to say, thoroughly labeled internal nodes, achieved very good test accuracy (more than 80%). It is so far the best data set which to be used for Recursive Neural Network. On the other hand, we can assume that lack of labeling is one of the big challenges for Recursive Neural Network.

Back to the Kaggle challenge, although there is no current winner accuracy right now, the data set and the question were actually drawn from a paper coming from Stanford.[4] Though, in their paper, the main challenge was not sentiment analysis, their highest test accuracy was about 40% in their studies of users tastes and preferences changing and evolving over time. This low accuracy also showed that this was a challenging data set to analyze on.

3 Approach

3.1 Dataset

3.1.1 Data preprocessing

The Amazon Food Review dataset has 568, 454 samples. 52268 reviews have a score of 1, 29769 reviews have a score of 2, 42640 reviews have a score of 3, 80655 reviews have a score of 4, and 363122 reviews have a score of 5.

However, we found that it is difficult to distinguish reviews with score 4 and reviews with score 5, and same difficult for reviews with score 1 and reviews with score 2.

For example, consider the following review:

”good flavor! these came securely packed... they were fresh and delicious! i love these Twizzlers!”

This review turns out to have a score of 4 while it would also make sense if the review had a score of 5.

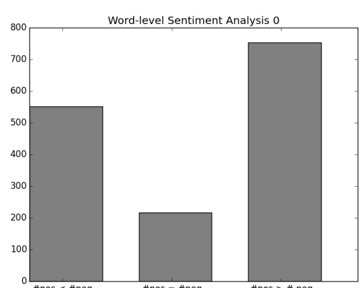
Therefore, our solution is to binarize the labels to ”positive” and ”negative” by aggregating score 4 and 5 as ”positive”, score 1 and 2 as ”negative”, and ignore samples with score 3 since we are more interested in reviews with a clear attitude.

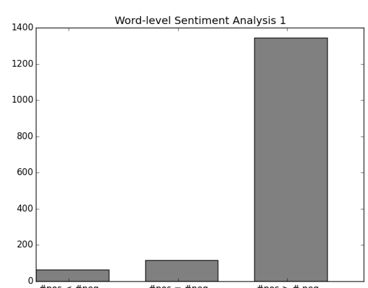
After we binarizing review scores, we notice the dataset is quite imbalanced, i.e. about 80% of the reviews are positive. If we have a classifier which would always predict a review as positive, then it is able to easily achieve 80% accuracy. To solve this problem, we use undersampling technique. We randomly drop positive reviews to make the number of positive reviews are roughly the same as that of negative reviews.

3.1.2 Dataset stats

Before we start building deep learning models, we first examine the features of our dataset in order to construct an appropriate model.

Using twitter sentiment words [5], we can obtain sentiment label for each word in our reviews. We want to take a look at the difference of positive reviews and negative reviews from the word-level perspective.





The left figure is the summary for negative reviews, and the right for positive reviews. Both of the figures have three categories. The first category is the number of reviews that contain more negative words than positive ones, the second is the number of reviews that contain equally many positive and negative words, and the last one is the number of reviews that contain more positive words than negative ones.

We notice that while most of the positive reviews have more positive words than negative words, it is surprising to find that even around half of the negative reviews have more positive words than negative words.

The stats we just show are important because they show that doing sentiment analysis only on the word level may not work well for the dataset we have. Therefore, the baseline we have that just essentially uses bag-of-word model is not likely to perform very well. We need some model that is able to look at the bigger picture, that is to take sentence structure into consideration.

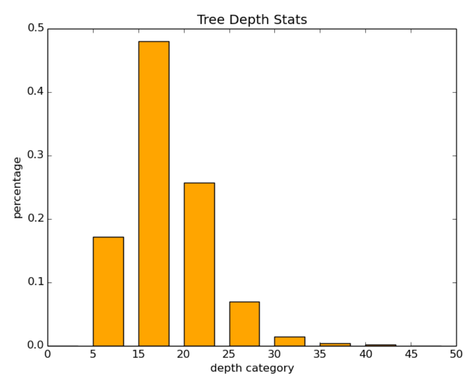
3.1.3 Tree parser

One of the most important parts of our project is to construct binary trees, in order to feed our Recursive Neural Networks.

Stanford NLP Parser is a powerful tool to parse sentences to trees based on a specified NLP model. We choose exhaustive PCFG(Probabilistic Context-Free Grammars) parser as our parser to process the reviews.

The problem of the PCFG parser is that the tree returned is not always a binary tree. An internal node may have more than two children. Since our model can only handle two children's hidden layer output, then we need to convert trees into their binary form. Therefore, we add the TreeBinarizer class in processResults() of ParseFiles class, and then we can have binary trees. However, there is still one more problem of these trees. Some internal nodes may only have one child, since, again, we need each internal node to have two children for our model. For example, $NN \rightarrow man$. To solve this problem, we employ the following technique: for each internal node that has only one child, we delete this node and elevate its child one level up. For example, suppose we have $NP \rightarrow theNN$ and $NN \rightarrow man$, notice that NP node has two children while NN node has only one child. In this case, we treat the above the structure as $NP \rightarrow the\ man$.

Now we have binary trees ready. The following figure is a histogram of tree depth distribution. We



find that most of the trees we generate from reviews have a depth between 15 and 20. Notice that if a tree is too deep, not only the model may not perform well but also it takes too long to train the model using this tree. Therefore, we prune the trees that have a depth more than 20.

Another potential problem of the tree is the lack of labeling. Training Recursive Neural Network usually needs comprehensive supervision, namely every node is labeled. However, given the nature of our dataset, one review, composed of multiple trees, only has one label. It may be very difficult for RNN to learn well with such low-level supervision.

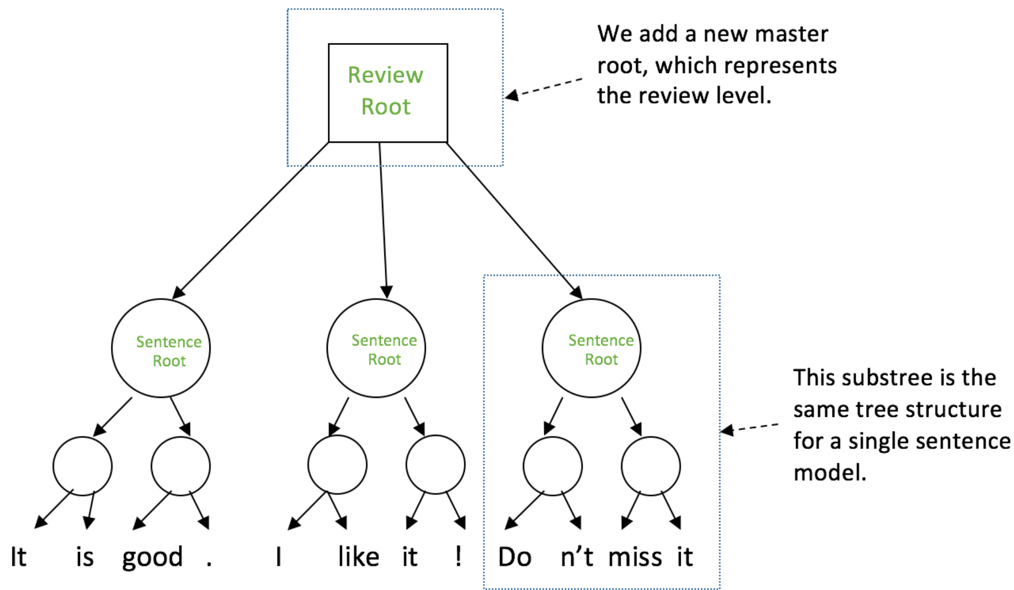
Therefore, in order to increase the level of supervision in our model, we first label words located at the leaf level. With the help with twitter sentiment words [5], we are able to label the words as positive, negative or neutral.

However, there are still about half of the nodes that are not labeled. It is difficult to label each node, or phrase, due to the lack of phrase sentiment banks. Thus we propose a novel technique to label the internal nodes without laborious human labeling over thousands even millions of nodes. For each internal node, we check its two children's label: if their label is identical, then we set the internal node's label to the same label; if their label is not identical, then we are not sure about the node's sentiment and we just set the label to be neutral.

Notice that although we set some internal nodes' label to be neutral, the model that we use actually will ignore all neutral labels and only considers positive/negative labels when calculating loss function.

3.2 Models

We use a one hidden-layer Recursive Neural Network as our model. In most previous work, RNN is fed with single sentences. However, we want to feed reviews(paragraphs) to RNN. Therefore, we propose the model RNNMS (Recursive Neural Network for Multiple Sentences).



We first define the hidden layer:

For the review root, we have

$$h^{(1)} = tanh((\frac{1}{N}\sum_{i=1}^{N} h^{(1)}_{child\_i})W^{(r)} + b^{(r)}) \tag{1}$$

For other nodes, we have

$$h^{(1)} = \max([h^{(1)}_{left}, h^{(1)}_{right}]W^{(1)} + b^{(1)}, 0) \tag{2}$$

Then we define the output layer:

For the review root, we have

$$\hat{y} = softmax(h^{(1)}U^{(r)} + b^{(r)}_s) \tag{3}$$

For other nodes, we have

$$\hat{y} = softmax(h^{(1)}U^{(1)} + b^{(1)}_s) \tag{4}$$

$h \in R^{1\times d}$, $\hat{y} \in R^{1\times C}$, $W^{(1)} \in R^{2d\times d}$, $b^{(1)} \in R^{1\times d}$, $U^{(r)} \in R^{d\times C}$, $b^{(r)}_s \in R^{1\times C}$, $U^{(1)} \in R^{d\times C}$, $b^{(1)}_s \in R^{1\times C}$. And we choose the embedding size $d = 50$, and the label size $C = 2$.

Finally, we define the loss function:

$$J = \beta CE(y_r, \hat{y_r}) + (\sum_{all\ nodes\ with\ labels} CE(y, \hat{y})) + l2(\sum W_{ij}^{(r)} + \sum W_{ij}^{(1)} + \sum U_{ij}^{(r)} + \sum U_{ij}^{(1)}) \quad (5)$$

β is the weight of the review root's cross entropy loss. We amplify the effect of the review root by setting β = the number of all nodes with a positive/negative label - 1. We increase the weight of the review root because due to the lack of labeling for internal nodes, we need to make good use of the review label and after all what we really care about is the prediction accuracy at the review root level.

Recursive neural network is the way of using the same set of weight and applying recursively on the same structure. Recursive neural network has been used as a useful tool in natural language processing, especially in sentiment analysis, because it processes the sentence as how a human understands a sentence.

There are two major differences between our RNNMS and the naive RNN.

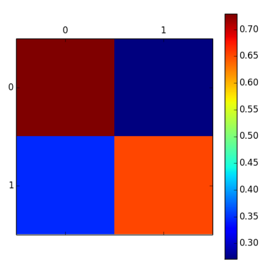
First, at review root level, we average the hidden layer output of all its children. This technique is similar to averaging every word's vector in a sentence when we want to do sentiment analysis on a single sentence. We incorporate the idea of bag-of-sentence to the master root level of our model because we believe the average value can be a good generalization of the sentences it contain. For example, if a positive review contains four sentences, and all sentences are positive, then the average is likely to be also treated as positive. Even there is one negative sentence, the averaging operation can still put the review level hidden layer output to the positive side.

Second, we have a different output layer for review root than that for all other nodes. The reason behind this is that while the RNN model assumes there exists a recursive grammatical structure for each sentence, the relationship between the review and sentences is not captured by any recursive grammatical rules. Therefore, we need a different pair of U and bs to fit the review root level's output.

4 Experiments & Results

4.1 Baseline

Our baseline is a Naive Bayes classifier and we use the average of all word vectors of a review as the feature vector for a review.



	precision	recall	f1
negative	0.66	0.73	0.69
positive	0.72	0.65	0.69
average	0.69	0.69	0.69

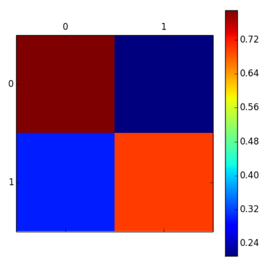
The results show that how well the baseline with a basic bag-of-word model can perform. Notice that the baseline actually performs very well. Due to the nature of the problem, the sentiment analysis of single sentence like movie reviews, accuracy never reached above 80% for 7 years [6]. We think the reason behind this is that while movie reviews have a lot of sarcasm, which is very difficult for any model to grasp, amazon food reviews are much more straight forward, and thus most of the sentiments are expressed directly at the word level. For example, a user may write a lot of positive words to say a food is good. It is possible to judge a food review's sentiment only by identifying positive words in a food review,. However, it is usually not enough to predict a movie review's sentiment only by looking for positive or negative words. Therefore, given the nature of our dataset, the baseline actually sets a high bar for our RNNMS model.

Unlike many other models using bag-of-word or n-gram, Recursive Neural Network is able to learn to grasp the semantic structure of a sentence because it considers the semantic composition of a sentence during training. Therefore, Recursive Neural Network is expected to perform better than character-level n-gram models and bag-of-word models for sentiment analysis task.

4.2 RNNMS results

For training our model, we initialize the embedding matrix using 50 dimensional GloVe word vectors trained by twitter data because we think tweets are both semantically and grammatically similar to online food reviews.

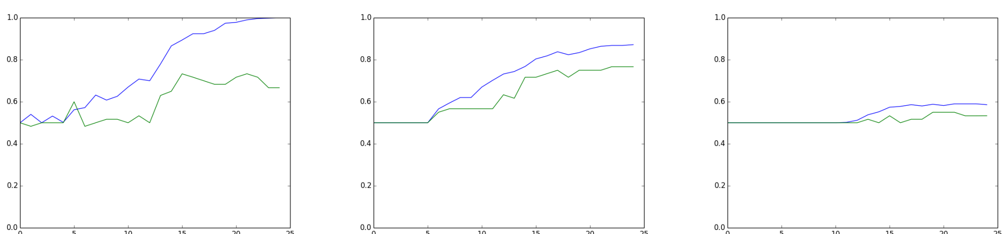
Here is the best result from our RNNMS with learning rate = 0.1 and l2 regularization = 0.01.



	precision	recall	f1
negative	0.72	0.79	0.76
positive	0.77	0.70	0.73
average	0.75	0.75	0.74

The results show that all metrics of our RNNMS outperform the baseline we have. The 6% boost of average accuracy may be the result of more understanding of the grammatical structure of a review.

4.3 Hyperparameter tuning



(a) lr=0.1, l2=0.005 (b) lr=0.1, l2=0.01 (c) lr=0.1, l2=0.02

The above three plots are train accuracy and validation accuracy vs. epoch for three different pairs of learning rate and l2 regularization.

Notice that in (a), train accuracy rises to almost 1 but the validation accuracy first hits above 0.7 but later drops and remains around 0.65, which indicates the overfitting problem due to a small l2 regularization parameter. In (c), both train accuracy and validation accuracy grows very slow and plateau at around only 0.6, which indicates the underfitting problem due to a large l2 regularization parameter.

For the pair with best test accuracy, which is lr=0.1 and l2=0.01, we see that both the train accuracy and validation accuracy goes up significantly since epoch 5 and plateau since epoch 20. Therefore it shows that our model converges quickly and does not require a large number of training epochs. It is possible that the reason behind this is that a lot of food reviews are quite similar, and thus when given similar training reviews, the model is able to learn very fast.

4.4 Activation function comparison

For the review root, we change the non-linearity

$$h^{(1)} = tanh((\frac{1}{N}\sum_{i=1}^{N} h^{(1)}_{child\_i})W^{(r)} + b^{(r)}) \tag{6}$$

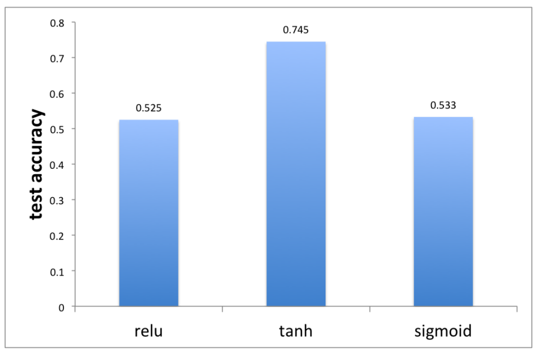
to

$$h^{(1)} = relu((\frac{1}{N}\sum_{i=1}^{N} h^{(1)}_{child\_i})W^{(r)} + b^{(r)}) \tag{7}$$

and

$$h^{(1)} = sigmoid((\frac{1}{N}\sum_{i=1}^{N} h^{(1)}_{child\_i})W^{(r)} + b^{(r)}) \tag{8}$$

The test accuracy for each case are as follows:



tanh function has the highest test accuracy. Therefore, we chose tanh function in our best model. An insight that we got from piazza is that ReLU and sigmoid both saturate at 0 on the negative side, whereas tanh saturates at -1. Thus, if a weight gets multiplied by the output of a tanh, the size of the weight matters for negative values of the input to the tanh. It does not matter for any negative ReLU inputs and for sufficiently large sigmoid inputs in absolute value. [7]

5 Conclusion

- RNNMC performs better than the baseline. Even with insufficient labeling of trees, RNNMC is still able to outperform in every metrics we have than the baseline naive bayes classifier using averaged word vectors as input features, which means that understanding phrase-level structure does help sentiment analysis task.
- For recursive neural network, labeling every node is very important. While this model can achieve as high as above 80% accuracy using Stanford Sentiment Tree Bank dataset, Our results show that without sufficient labeling, this model is not able to achieve an accuracy above 80%, which means RNN family needs strong supervision. However, most of the online reviews and other documents only have very limited labels, therefore our results are meaningful because it shows that even without sufficient labeling of tree nodes, it stills performs well. Moreover, we have proposed and tested a novel technique in order to increase the level of supervision by adding label to some nodes of a tree.
- The recursive hidden layer should only be shared among tree nodes that are intrinsically similar, that is to see we probably should not use the same recursive hidden layer for aggregating sentences for the review root node. The reason is that the relationship between sentences, as we think, should be intrinsically different than that between phrases and words. Therefore, when we design recursive neural network, we should think about whether the structure we model should have a recursive property. Otherwise, we need to design a different hidden layer and output layer for the review root level, like what we did in our RNNMS.

6 Reference

1. Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.

2. Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *CoRR*, abs/1503.00075, 2015.

3. Ye Yuan and You Zhou. Twitter Sentiment Analysis with Recursive Neural Networks. http://cs224d.stanford.edu/reports/YuanYe.pdf

4. J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. *WWW*, 2013.

5. Jeffrey Breen. twitter-sentiment-analysis-tutorial-201107. https://github.com/jeffreybreen/twitter-sentiment-analysis-tutorial-201107/blob/master/data/opinion-lexicon-English

6. Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 115124. Association for Computational Linguistics, 2005.

7. *Piazza Discussion*