Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank

1 Citation

Socher, Richard, et al. "Recursive deep models for semantic compositionality over a sentiment treebank." Proceedings of the 2013 conference on empirical methods in natural language processing. 2013.

`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.383.1327&rep=rep1&type=pdf`

2 Abstract

We create the Stanford Sentiment Treebank dataset, which has fine-grained (i.e. very negative, negative, neutral, positive, very positive) sentiment labels for the parse trees of 215K phrases. We make a Recursive Neural Tensor Network that gets state of the art binary (85.4%) and fine-grained (80.7%) sentiment classification.

3 Introduction

Existing semantic vector spaces struggle with long phrases and can't deal with hard negation (e.g. the words "doesn't", "isn't", etc.) well. We create a dataset to help with this. Basically, we took movie reviews, ran the Stanford parse to get parse trees, and used Mechanical Turk to label each node in the tree.

We make a Recursive Neural Tensor Network (RNTN) that starts with the word vectors of the phrase and recursively compute vectors for parent nodes.

4 Related Work

Other semantic vector spaces struggle to model hard negation (e.g. "not", "doesn't"), and antonyms (i.e. oppposites). People have experimented with different ways to combine word vectors. Another approach to compositionality is by mapping sentences to logical forms, but these only work in restricted domains. And of course, people have done research into deep learning and sentiment analysis.

5 Stanford Sentiment Treebank

Bag-of-word models don't consider word order, which severely limits their power. To build our dataset, we start with a movie review dataset, taken from the Rotten Tomatoes website. It has about 10K sentences, where the label is taken from the sentiment of the overall review.

We ran the Stanford Parser on the sentences to get parse trees. We then used Amazon Mechanical Turk to crowdsource the labeling of the trees. We show the sentence to an annotator and provide them a slider to indicate sentiment. We find that short sentences tend to be neutral and that people tend to use 5 levels of sentiment (very negative, negative, neutral, positive, very positive).

6 Recursive Neural Models

We turn an n-gram into a parse tree, label leaf nodes with word vectors, and hand it to the neural network to label the other nodes. Each word is represented by a d-dimensional vector, and we learn the embedding matrix L that maps words to vectors. To label the leaf nodes with a sentiment, we just do:

$$y^a = \text{softmax}(W_s a)$$

How do we compute the parent vectors? We try several approaches. Consider a parent p with children c_1 and c_2. We can do the following (you can also use a bias term in this equation).

$$p = \tanh(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix})$$

Another approach is to associate a matrix and vector with each node. This way, when computing a parent, we can combine the children's matrices and vectors together. This is based on a model called matrix-vector RNN (MV-RNN). The equations are the following (capital letters indicate matrices):

$$p = \tanh(W \begin{bmatrix} C_2 c_1 \\ C_1 c_2 \end{bmatrix}), P = \tanh(W_M \begin{bmatrix} C_1 \\ C_2 \end{bmatrix})$$

The problem with the above is that it has too many parameters. So, we'll use a Recurrent Neural Network (RNN) instead. We'd like words to interact somehow, preferably with something more than just through an activiation function. First, we define $V^{[1:d]} \in \mathbb{R}^{2d \times 2d \times d}$. We then compute:

$$p = \tanh(\begin{bmatrix} c_1 \\ c_2 \end{bmatrix}^T V^{[1:d]} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix})$$

Once we do this, we can apply our softmax classifier $y^i = \text{softmax}(W_s x^i)$ for each node i in the parse tree. We then use log-likelihood and regularization to define the following cost function (the one-hot-encoded sentiment ground truth for node i is t^i):

$$E(\theta) = \sum_i \sum_j t^i_j \log y^i_j + \lambda ||\theta||^2$$

The paper describes how to compute derivatives for this model.

7 Experiments

The best (from cross validation) vector size was between $d = 25$ and 35 and batch size was between 20 and 30. We train with Adagrad for 3-5 hours. We compare against Naive Bayes (NB), SVM, Naive Bayes with bag of bigrams features (biNB), RNN, and MV-RNN. Our RNTN beats all of them on fine-grained (i.e. 5-class) and binary sentiment classification. When we focus on sentences of the form "X, but Y", RNTN is best with 41% accuracy. We looked at negating positive sentences (e.g. "I liked every single minute of the film" becomes "I didn't like a single minute of the film") and negating negative sentences (e.g. "it is incredibly dull" becomes "it is definitely not incredibly dull"). RNTN beats other models on both these cases. We also looked at the most positive and negative n-grams - they make sense.

8 Conclusion

Our RNTN, which is trained on our Stanford Sentiment Treebank dataset, sets state of the art on binary and fine-grained sentiment classification.