Aspect Specific Sentiment Analysis of Unstructured Online Reviews

Elliot Marx
Department of Computer Science
Stanford University
emarx@stanford.edu

Zachary Yellin-Flaherty
Department of Computer Science
Stanford University
zachyf@stanford.edu

Abstract

In this paper, we address the problem of aspect-specific sentiment analysis. Given product reviews, our goal is to extract not only the general sentiment of the review, but the aspects mentioned in the review and the sentiments specific to these aspects. We approach this problem by both jointly and sequentially predicting the aspects and sentiments of a review. Within these frameworks, we explore forms of both recursive and recurrent neural nets. To handle sentences with multiple aspect-sentiment pairs, we develop approaches to predict multiple classes. On our dataset with 17 classes (and multiple classes per example), we achieve 51.8% accuracy in predicting aspect-sentiment pairs, a vast improvement over our baseline using Naive Bayes and tf-idf features with 37.3% accuracy.

1 Introduction

Automatically synthesizing the meaning of customer reviews is helpful to company and consumer alike. Summarizing this information allows consumers to find items with qualities important to them and companies to develop a quick look into user satisfaction.

However, the technology for effectively synthesizing this large volume of reviews is underdeveloped. Hundreds of reviews on Amazon of a single product are reduced to a simple distribution of overall reviews and a few of the most "helpful" reviews.

Reviews for products online are seldom fully negative or positive in sentiment. Rather, they describe the positive and negative core aspects of a product. To demonstrate this issue, consider an excerpt from this 3/5 star review for a laptop:

"The faux leather cover is a wee bit cheesy for my taste, but I loved the price and the performance."

For a purchaser, this review may not be useful when viewed only as a contribution to a mean score. The aspects "performance" and "price" have highly positive sentiment, while "appearence" receives a slightly negative sentiment. For a customer indifferent to the aesthetic of a laptop, this review should contribute higher than a 3/5 score to the mean.

More useful to the consumer are summary statistics for each of a product's features. Our goal is to bring this structure to Amazon product reviews using deep learning.

1.1 Problem Statement

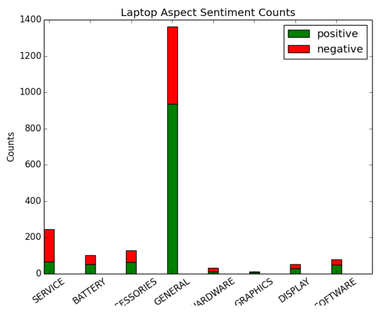
The problem is twofold: identify the product aspects in the review, and then classify the sentiment attached to each aspect. Formally, we are given a set of reviews $R = \{r_1, r_2, r_2, \dots\}$. From this, we identify aspect-sentiment pairs $\{(a_i^1, s_i^1), (a_i^2, s_i^2), \dots\}$ for each review r_i.

1.2 Dataset Description

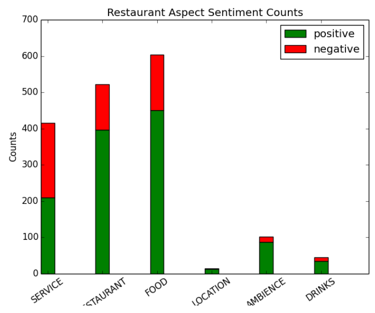
We will be training on two datasets for evaluation: laptop and restaurant reviews. The datasets are described as follows:

Dataset	**Reviews**	**Sentences**	**Aspects**
Laptops	450	2501	Service, Battery, Accessories, General, Hardware, Graphics, Display, Software
Restaurants	350	2000	Service, Overall, Food, Loc., Ambiance, Drinks

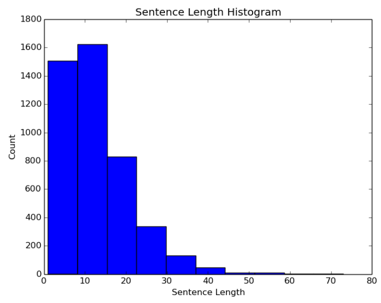
We are given a list of sentences for each review. For each sentence, we have a set of tuples, each of which indicates the aspect and the sentiment of the aspect (positive or negative). The original dataset contained 24 different aspects for laptop reviews, but we merged similar aspects (ie. Customer Service, Support, and Warranty) to create a larger number of examples of each class. The graphics below offer some details about our dataset.



(a) Laptop tag histogram



(b) Restaurant tag histogram



(c) Sentence length histogram

Figure 1: Distributions of our dataset

2 Background and Related Work

Many researchers have approached the problem of aspect-specific sentiment analysis, though only recently with tools from deep learning. There are two major ways to approach the problem: the Separate Aspect Sentiment Model (SAS) and the Joint Multi-Aspect Sentiment Model (JMAS). In SAS models, we predict the aspect of a given review independently of the sentiment for the class. Then, given the aspect, we predict the sentiment of the aspect. In JMAS models, we predict aspect-sentiment pairs, thus jointly predicting which (and possibly multiple) pairs are present in a given system.

The first systems developed for aspect-sentiment analysis used SAS models. Popescu and Etzioni start with rule-based systems for both identifying the product features and classifying their sentiment [10]. In [4], Hu and Liu use a more advanced mining-based algorithm to determine features, and wordnets to capture sentiment. The authors in [5] formulate the problem as a weighted bipartite cover to learn the parts of reviews that mention aspects of interest.

More recently, there has been exploration into JMAS models. Inspiring our work, Himabindu et. al use hierarchical deep-learning frameworks to extract aspect-sentiment pairs by jointly modeling features and sentiments in [3]. Their work requires finely-labeled training data giving the aspect-sentiment pairs at each node in the tree. Such models are as useful when data is labeled only at the phrase level.

In this work, we adapt these hierarchical methods designed for tree-labeled datasets to data labeled only at the sentence level, and compare their performance to advanced recurrent nets, such as LSTMs and GRUs.

3 Approach

In our deep-learning models, we represent each word with a word vector and represent each review by combining these vectors in different manners depending on the model. We explore both hierarchical and recurrent frameworks to learn the aspect-sentiment pairs for sentences. First, we briefly describe these two models.

3.1 Join Aspect Sentiment Model (JMAS)

We employ the Joint Multi-Aspect Sentiment Model from [3]. In this model, we create a class for each pair $(aspect, sentiment)$, so that our label $y^i \in \mathbb{R}^{2n}$, where n is the number of aspects. Further, we allow our model to predict multiple aspect-sentiment pairs, as our dataset exhibits a majority of such examples.

3.2 Separate Aspect Sentiment Model (SAS)

In order to take advantage of the known success of recursive neural tensor networks (RNTN) to model sentiment [6], we also explored predicting aspect and sentiment independently. We initially predict sentiment with an RNTN and then predict aspect with recurrent models (LSTM and GRU). For each aspect predicted by the recurrent network, we predict a pair of that aspect coupled with the with sentiment output by the RNTN.

4 Models

In the context of one or both of the JMAS and SAS frameworks, we train the following models:

4.1 Baseline

In order to assess the effectiveness of other neural networks, we implement a simple baseline from traditional NLP. We treat each sentence in each review as a separate review. We extract tf-idf vectors of words from these sentences as our features. From these features, we train a multi-label one-vs-all Support Vector Machine classifier and a Multinomial Naive Bayes classifier. For this rudimentary baseline, we used only the JMAS framework.

4.2 Recurrent Neural Nets

We employ many different recurrent neural nets for this task. In each, we apply dropout to our hidden layers, as described in [7] to prevent overfitting. For each of the following models, we train under both the JMAS and SAS frameworks. We use the framework provided with Keras.io with added infrastructure.

4.2.1 Simple and Deep Recurrent Neural Net

In the simple recurrent neural net, we simply combine the result of our previous hidden layer with the word vector at the current timestep as follows:

$$h_t = W\sigma(h_{t-1}) + W^{(hx)}x_t$$

Then, we apply a linear transformation to the final hidden layer, and take the softmax of the result to generate class probabilities.

In the deep recurrent network, we modify the simple recurrent net to incorporate feedback from multiple previous hidden layers:

$$h_t = W^{(hx)}x_t + W^{(1)}\sigma(h_{t-1}) + W^{(2)}\sigma(h_{t-2}) + \cdots$$

In practice, we find that using a depth of 3 hidden layer provides best results.

4.2.2 GRU

In contrast to simple recurrent neural nets, GRUs have update gates that allow the model to learn how much of the past state is relevant.

Our recurrent layers h_t are computed using update and reset gates as follows:

$$\begin{aligned}
\text{Update Gate}: z_t &= \sigma(W^{(z)}x_t + U^{(z)}h_{t-1}) \\
\text{Reset Gate}: r_t &= \sigma(W^{(r)}x_t + U^{(r)}h_{t-1}) \\
\text{Proposal}: \widetilde{h_t} &= \tanh(Wx_t + r_t \circ Uh_{t-1}) \\
\text{Current Layer}: h_t &= z_t \circ h_{t-1} + (1 - z_t) \circ \widetilde{h_t}
\end{aligned}$$

When the reset gate r_t is close to 0, we can ignore the previous memory, allowing the model to drop information that is not longer relevant. The update gate z_t controls how much the past state should matter now, and helps eliminate vanishing gradient problems.

4.2.3 LSTM

LSTM is a complex recurrent neural net that create additional gates to allow flexibility in what information is backpropogated through the layers. The gates and recurrent layers are computed as follows:

$$\begin{aligned}
\text{Input gate}: i_t &= \sigma(W^{(i)}x_t + U^{(i)}h_{t-1}) \\
\text{Forget gate}: f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1}) \\
\text{Output gate}: o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1}) \\
\text{New memory cell}: \tilde{c}_t &= \tanh(W^{(c)}x_t + U^{(c)}h_{t-1})
\end{aligned}$$

The final hidden state for each timestep becomes:

$$h_t = o_t \circ \tanh(f_t \circ c_{t-1} + i_t \circ \tilde{c}_t)$$

This powerful model is currently very popular and the most powerful recurrent neural network we studied.

4.3 Recursive Neural Net

We built a single-hidden layer recursive neural network by parsing our sentences into binary trees using the Stanford Parser. We labelled our data at the root of each sentence tree, and propagated error down to the nodes. Since we do not have phrase or sentence-level labels, we set local δ terms at sub-trees to be 0. We implemented our recursive neural net by changing the assignment 3 starter code.

4.4 Objective Function for Neural Networks

The intuition behind our objective function is to minimize the distance between the output of the softmax prediction, y^i, and the true label t^i, for each training example. Note that there are potentially multiple labels for the ith example. As a result, we normalize t_i to sum to one, so that if there are k labels in t_i, each entry in t_i corresponding to a present label is $\frac{1}{k}$. Our objective function, adapted from [3], is the following:

$$E(\theta) = \sum_i \sum_j t^i_j \log y^i_j + 0.5\lambda \parallel \theta \parallel^2,$$

where t^i_j is the jth class of the ith training example. y^i is the prediction on the ith training example, a softmax function over all possible classes. θ represents all the parameters that exist in the current deep learning model.

4.5 Word Vectors

For our initial experiments, we initialized the word vectors with pre-trained GloVe vectors. Since our dataset is small relative to the size of the trained word vectors, when possible we did not back propagate into the word vectors.

We also had access to a large Amazon Review Dataset from [1] that included a set of electronics reviews. We trained a set of word vectors with the skip-gram model on all reviews that included the word "laptop", of which there were over 500,000. We initialized our model with this set of word vectors when training and testing on the laptop review dataset. This set of word vectors was more specialized, in the sense that its context was more specific to ours, and the interactions among the words in our dataset are better represented though this corpus of reviews than through the Twitter dataset. For our final laptop tests, we used this word vector representation. We had no analogous corpus for the restaurant dataset, so we experimented exclusively with the pre-trained Twitter GloVe word vectors for this dataset.

We attempted using word vectors of lower and higher dimensions (as low as 25 and as high as 100), but found no dramatic improvement and settled on 50 dimensional word vectors throughout.

We do not backpropagate into our word vectors because at this stage we do not have enough data to keep all similar words in correct locations in the vector space after training. We load the word vectors pre-trained from the GloVe dataset or the pre-trained custom laptop Amazon Review word vectors.

4.6 Evaluation

The output of each algorithm is zero or more aspect-sentiment pairs for each review. Since there can be multiple labels for each review, we use Jaccard Similarity Score. The Jaccard Similarity Score is defined to be the intersection divided by the union of the predicted and ground-truth aspect-sentiment set.

We also analyze how well our algorithms determine the correct aspect independent from sentiment. Since each review addresses multiple aspects, we again use Jaccard Similarity Score.

Finally, we evaluate our algorithm's performance on sentiment analysis. We condition on the event that our algorithm outputs the correct aspect, and then measure the accuracy on these samples.

5 Experiential Results

Table 1: Experimental Results on Laptop and Restaurant Datasets

	Laptop			Restaurant		
Approach	(aspect, sent)	aspect	sent|aspect	(aspect, sent)	aspect	sent|aspect
SAS + LSTM	**51.83** %	**67.77**%	77.98 %	**43.69**%	**55.87**%	79.78%
SAS + GRU	49.40 %	64.14 %	78.48 %	41.87 %	54.30 %	78.88%
SAS + RNN	47.82 %	62.25 %	78.04 %	33.83 %	46.27 %	75.63%
SAS + Deep-RNN	46.77 %	61.53 %	77.18 %	24.11 %	32.74 %	76.94%
JMAS + LSTM	50.71 %	63.39 %	81.47 %	42.19 %	51.97 %	**82.81**%
JMAS + GRU	49.28 %	60.37 %	**83.50**%	43.40 %	55.07 %	80.70 %
JMAS + RNN	44.36 %	55.68 %	81.31 %	27.34 %	34.05 %	82.48 %
JMAS + Deep-RNN	44.92 %	57.04 %	80.08 %	23.83 %	32.65 %	78.37 %
SVM + tf-idf	47.01 %	64.02 %	86.43 %	32.16 %	50.20 %	83.21 %
NB + tf-idf	37.26 %	61.11%	96.62 %	17.24 %	29.52 %	95.32 %

5.1 Analysis

We found LSTM with the SAS framework to be most the effective model (with other updates described in the following section). We reached 51.83% accuracy on the laptop dataset and 43.69% accuracy on the restaurant dataset for predicting aspect-sentiment pairs. The responsibility for our error is shared fairly evenly between aspect prediction and sentiment prediction given an aspect. For example, in the laptop dataset, we correctly identify 68% of the aspect pairs, but once we have correctly identified a pair, we classify its sentiment correctly with probability 78%.

Our results do not entirely align the with results from [3]. There, the authors found that the highest-performing aspect-sentiment classifier predicted aspect and sentiment jointly, while our best model predicts aspect and then conditionally predicts sentiment based on aspect. We conclude that this divergence is most likely a result of limited data in our experiments. Note that from our graphs in Figure 1 on page 2, some of the aspect-sentiment pairs have low numbers of occurrences in our dataset. Observe in our confusion matrices that it is difficult for even our highest performing models to predict these classes. By separately predicting $\Pr(aspect)$ and $\Pr(sentiment|aspect)$, we increase the size of our aspect classes as well as our sentiment classes, consequently increasing the performance on the individual and joint tasks. In [3] there is likely sufficient data to eliminate the need for this simplification because the model has the ability to predict all classes. Further, the data was labeled at the node-level in the tree, providing more information from which to learn patterns.

5.2 Model Improvements

5.2.1 Dealing with Multiple Aspects

The number of aspect-sentiment pairs per example in our dataset varies greatly. Approximately 35% of examples do not mention any aspect-sentiment pair, and 15% have 2 or more aspect sentiment pairs.

Initially, our models predicted only one of the 14 (restaurant dataset) or 18 (laptop dataset) aspect-sentiment pairs as the class. This resulted in incorrectly classifying the 35% of the dataset without any aspect-sentiment pairs. To deal with this problem, we introduced a label "none", increasing our accuracies significantly.

We tried two techniques for dealing with multiple aspect-sentiment pairs. The first was to predict all aspect-sentiment pairs that had a score greater than the prediction for the none label. This strategy on average led to a 0.5-1% boost on both our SAS and JMAS experiments, and these results are included in our accuracies table.

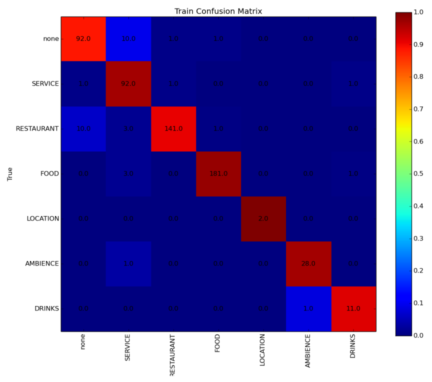
The second method trained a classifier for each aspect. This classifier would then predict one of three classes: (aspect, positive), (aspect, negative), and none. To combine the results from these predictions, we combined all predicted aspect-sentiment pairs from each classifier. While we expected this to perform well, it ultimately decreased the performance of our models. This was due to predicting too many classes per example, from which the first approach didn't suffer.

5.2.2 Training Topic-Specific Word Vectors

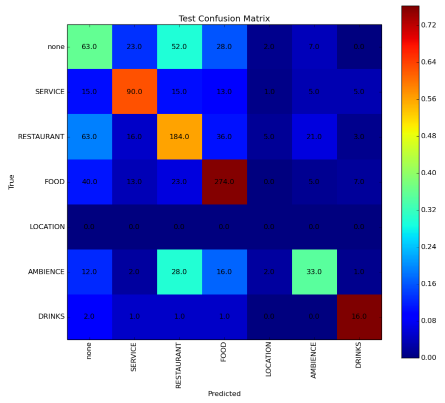
One of our major challenges was dealing with a limited set of data. This limitation led models initialized with random word vector embeddings to perform poorly. Using the word vectors trained on the Twitter dataset [9], we substantially improved our accuracies. However, the Twitter corpus is not an ideal context for these reviews. Laptops have many topic-specific terms, such as "motherboard" and "processor". Our intuition was that by training on a large dataset of laptop reviews, we could learn a better embedding our vocabulary in the context of reviews. Indeed, training word vectors from the Amazon dataset (see Section 4.5 for details) improved our results by 1-2% on our recurrent models.

5.3 Performance Across Classes

For the sake of visualizing our class-specific performance clearly, we include in the below confusion-matrices only include examples for which our model predicted a single aspect-sentiment pair and the ground truth had only one aspect. Because we are taking a subsection of the data to generate these plots, a few of the rows are sparse.

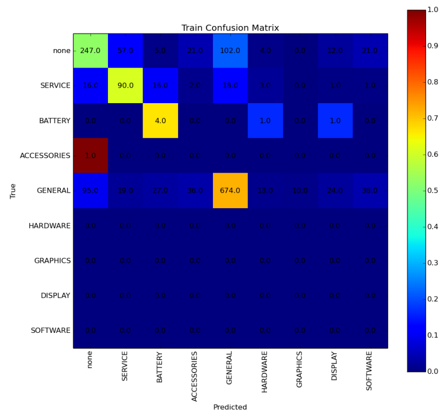


(a) Restaurant Train

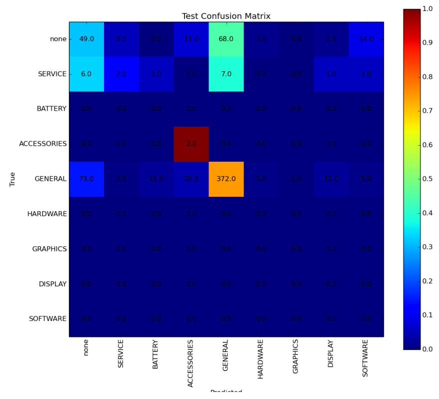


(b) Restaurant Test

Figure 2: Aspect Confusion Matrices for Restaurant Dataset



(a) Laptop Train



(b) Laptop Test

Figure 3: Aspect Confusion Matrices for Laptop Dataset

Note that in the laptop training confusion matrix (Figure 3a), the spectrum coloring is off because of an artifact in our dataset. The darkest cell (None, Accessories) has the full weight of our predictions on data labeled "accessories" because there was only a single example of the accessory label without any other labels.

We can see that one of our most-often confused pairs is (Ambiance, Restaurant). We expect to see this, as these topics are highly related. In the laptop dataset, we struggle most with Service, which we classify as General or None.

6 Conclusion

6.1 What We Learned

We learned that with limited data, it may be best to reduce the power of the model in favor of increasing the number of examples of each class (see Section 5.1 for elaboration).

With regards to the literature of recursive versus recurrent neural nets, we learned that complex recurrent neural networks are more generalizable than recursive neural networks when less data is available. Given a sentence and a label, recurrent networks can effectively fit GRU and LSTM models that perform well, while recursive neural networks are less effective than these models unless they are given the ability to exploit labels from subtrees. In the case of our dataset, phrases and words were without labels, leading the recurrent models to outperform the recursive models.

Finally, we learned that to perform well in a machine learning, you need to be close to your data. Our biggest improvements came from realizing that we had many examples without tags and that we needed to find word vectors more specific to our domain.

6.2 Future Work

In order to improve our models, we would need to add orders of magnitude of more data, including more finely labelled data. Since these examples are hand-labeled, we have a reasonable, but ultimately unideal data size. Similarly, the reviews in our dataset are labelled at the sentence. This reality led our recursive networks in particular to under-perform, especially when compared to the accuracies from our assignment 3. Word and phrase-level labelling would likely greatly improve this model's effectiveness.

Currently, our project aims to find aspect-sentiment pairs for a predetermined set of aspects for each review category. One ambitious future goal is to extend this project to determine an arbitrary aspect and its sentiment from the entire vocabulary, not just the set of predefined aspects for each category.

References

[1] McAuley, Julian. Amazon Product Data (2014). http://jmcauley.ucsd.edu/data/amazon/

[2] SemEval 2015 Aspect Sentiment Analysis Task. http://alt.qcri.org/semeval2015/task12/

[3] Lakkaraju, H., Socher, R. & Manning, C. 2014. Aspect Specific Sentiment Analysis using Hierarchical Deep Learning. NIPS Workshop on Deep Learning and Representation Learning, 2014

[4] Hu, M., and Liu, B. Mining and summarizing customer reviews. In KDD, 2004.

[5] McAuley, J.; Leskovec, J.; and Jurafsky, D. Learning attitudes and attributes from multiaspect reviews. In ICDM, 2012.

[6] Socher, R, Perelygin, A, Wu J, Chuang, J, Manning, C, Ng A, and Potts, C. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. EMNLP, 2015.

[7] Srivastava, N, Hinton, G, Krizhevsky, A, Sutskever, I, Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15, 2014.

[8] Mikolov, T et al. Distributed Representations of Words and Phrases and their Compositionality. In NIPS, 2015.

[9] Jeffery Pennington, Richard Socher, Christopher Manning. GloVe: Global Vectors for Word Representation.

[10] Popescu, A.-M., and Etzioni, O. Extracting product features and opinions from reviews. In EMNLP. 2005.