MSA: Movie Sentiment Analysis

Renan Campos, Connor Cooper, Kevin Wacome

Abstract

This paper describes the system we implemented for the machine learning course project. We participated in the sentiment analysis competition task. This system avoids the usage of hand-crafted features to generate word vector representations that encode semantic and sentiment information by following the approaches of Maas, et al (2011). We were able to generate a categorization accuracy of 87.5%, according to kaggle metrics. This accuracy is very close to the second and third place systems and the best performing system achieved an accuracy of 89.1%.

1 Introduction

When someone is shopping for an appliance, or trying to pick a movie to watch, they will often look at the reviews of that particular product. These reviews express sentiment - thoughts influenced by or proceeding from feeling or emotion. Because people consider the sentiment of others when making decisions, sentiment analysis is a well-motivated area of research.

In this paper, we present our solution to a key sentiment analysis task: being able to identify whether a movie review is positive or negative. This is a well-researched task, with many proposed solutions, as we will show in our background section. Our model allows for non-handcrafted feature sets and requires little prior knowledge. We will describe each component of our system and provide an overview of our learning model. After presenting the system in detail, we will describe the data set and experiments that were run, and see how our model holds in our Machine Learning class's sentiment analysis competition. Well conclude with our evaluation on how successful our model was and our main takeaways.

2 Relevant Literature and Related Works

Various approaches to sentiment analysis have been explored over the years. Pang, et. al, (2002) attempts the task of sentiment analysis using models known to do well in topic-classification, and explore why sentiment analysis is more difficult. The models tried include naive bayes, maximum entropy and support vector machine, and use a couple of different feature sets involving unigrams, bigrams, adjectives, part of speech, and position of words. Their data does show that these models dont perform as well on sentiment analysis as they would in topic-classification. They believe that this disparity is due to thwarted expectations, where a reviewer will describe good attributes that they expected, only to point out that these expectations were not satisfied by the movie. In order to achieve better results, the challenge of identifying thwarted expectations will need to be addressed. The authors suggest identifying features that indicate whether the current sentence is on-topic. This likely means to distinguish when the actual movie is being talked about, and when expectations are the subject. This paper also effectively articulates a way to handle the negation of phrases (good vs. not good) originally described by Das and Mike (2001). Pang, et al. (2002) suggests adding NOT_ to all the words between the negation word and the next punctuation.

Another approach introduced by Maas, et al,

(2011) utilizes different supervised and unsupervised methods to capture semantic information in a vectorized representation. They present a model to capture both semantic and sentiment similarities among words. This model learns semantic vectors that encode sentiment information. The authors semantic component learns word vectors via an unsupervised probabilistic model of documents. The supervised sentiment component of the model uses the vector representation of words to predict the sentiment annotations on contexts in which the words appear. This causes words expressing similar sentiment to have similar vector representation. These vector representations are then used as features for an SVM to classify polarity of movie reviews. The authors were able to obtain an high accuracy of 88.89% on a movie review task similar to the one this paper attempts to solve.

3 Data and Tools

The provided data set consisted of a training set and a test set. The training set consisted of 25,000 reviews- 12,500 positive and 12,500 negative. The test set consisted of 11,000 unlabeled reviews. We initially tried training on the entire dataset all at once. Unfortunately this became a time-consuming process. In order to lighten the load and receiver quicker feedback, the training was done in batches. In order to make sure that the batch-method would still produce satisfactory results, and to find the best number of batches to use, we created a development set of 1,000 files (500 pos/500 neg) to perform cross-validation on. The remaining 24,000 reviews were split into 40 batches of 600 files (300 pos/300 neg).

4 System

Our implementation employs the learning algorithms described in Maas, et al (2011). Figure 1 provides a diagram of our system's architecture. Our system was written in Python, with each component coded as a module.

4.1 Preprocessing

Before working on tokenizing and vectorizing the data, we introduced a layer of abstraction to make the data easier to work with. The training and test sets are treated as sets of Reviews, which is a class containing helper functions that efficiently retrieve information from a review, such as the raw text or its i.d., without keeping all of it in memory. This abstraction makes the main script easier to code because the names of the objects and methods are intuitive.

Tokenizing is important because it is what takes the text from the reviews and breaks it up into significant chunks for processing. Issues that were tackled for this part of the project included determining how to tokenize the text, and deciding what to keep/discard. We based our tokenizer off of a tutorial that was specific to tokenizing sentiment in twitter tweets (http://sentiment.christopherpotts.net/tokenizing.html). We only addressed a subset of the concepts that the tutorial covers as some weren't relevant to movie reviews. The tokenizer that we built is able to capture emoticons such as :) and :[. We felt this was important as these are occasionally used in informal writing to express emotion. Punctuation was also kept as a token, as certain occurrences of punctuation (such as ! or ...) can also contribute to the semantics of the sentence. Capitalization was taken out, and all letters, with the exception of those in emoticons, were lowercased, as we felt this distinction would lead to unnecessary sparseness when vectorizing later on.

To decide how much to keep from the text, we followed the original paper's logic, and created a function that will fix the vocabulary to the N most frequent words, ignoring the first M words as a sort of stop word removal mechanism. The original paper specifically used 5,000 words, ignoring the first 50. We will also use this configuration initially when testing the baseline. Ideally, more words tokenized would be better, but it may be computationally impractical to tokenize all of the words, as the dataset has over 80,000 vocabulary words.

After tokenizing the text, vectorizing is the next logical step, as the algorithms we plan on using require the data to be in this representation. We use the scikit module to help build both tf-idf weighted bag of words vectors for each Review, and one-hot vectors for each word in the Review, as these are the representations that we currently need.

The tokenizing and vectorizing step are actually done at the same time, so the vectorizer needs to

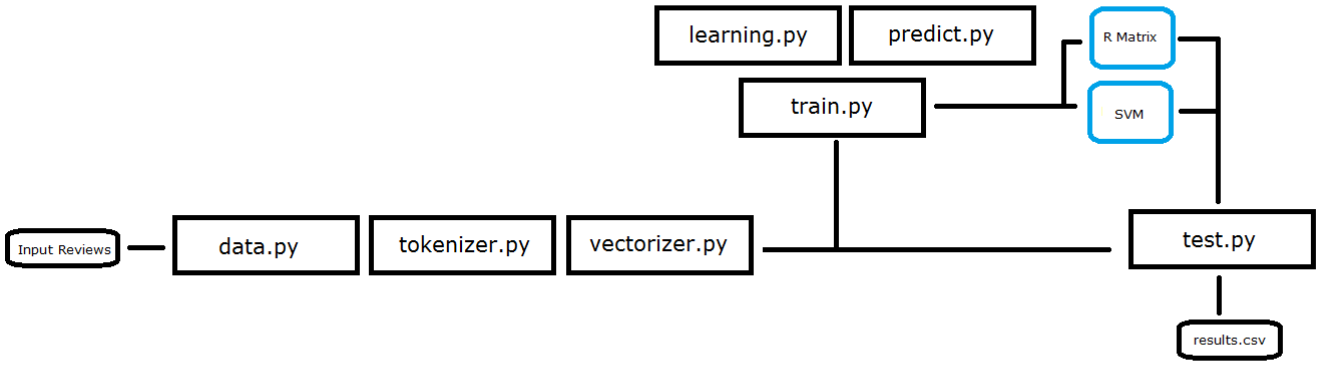

Figure 1: MSA's System Architecture.

parse through all of the training data in order to determine the N most frequent words that it will use in its vocabulary. This takes about 3 minutes to complete, so to save time, the trained vectorizers are serialized to a file so that they can be reused. Reloading these files take no longer than half a second, so this is a real time-saver.

4.2 Semantic Features

For this project we follow the approach developed by Maas, et al (2011). The goal of this approach is to encode sentiment as well as semantic information into a matrix called R. R, is a matrix where each column vector represents some word within a vocabulary V. The goal of this method is to learn this matrix R and then by computing the tf-idf scores of the words in the training set, multiply the word vectors by their corresponding tf-idf score for that word and then sum all the vectors. This resulting vector is then the representation of the document to be classified. The semantic component of this model tries to encode the relative meaning of the vocabulary words by adjusting the word vector feature dimensions given some parameters assigned to a document using gradient ascent, to maximize the probability of the training set. These word vectors will be similar for words that are semantically the same. These document parameters (θ_k) function to generate a weighted sum of the features of each word to compute the relative, as the authors state, 'energy' of a word. This 'energy' is just a real valued number used to compute softmax probability of a word occurring. The $p(w|\theta; R, b)$ is defined as $p(w|\theta; R, b) = \frac{exp(\theta^T \phi_w + b_w)}{\sum_{w' \in V} exp(\theta^T \phi_{w'} + b_{w'})}$. ϕ_w is the corresponding vector represent word w and is a column within matrix R. Each word 'energy' also has an added bias term b_w. Intuitively a high score means a high probability of a word occurring in a document. The parameters assigned to a document are also optimized using gradient ascent as well. Intuitively this is done so words belong to a certain document have a higher probability. Because certain features will be emphasized generating different word 'energy' scores. These gradient ascents are alternated until convergence. In our implementation we put a hard limit on the number of iterations to perform these alternating maximizations.

The probability of our training set is defined by the authors as the following: $p(D; R, b) = \prod_{d_k \in D} p(\hat{\theta}) \prod_{i=1}^{N_k} p(w_i|\hat{\theta}_k; R, b)$. Where D is a collection of documents (d_k). This function is generated by assuming that each document is independent. The probability of a document is represented as a joint probability, $p(d; \theta)$. This joint probability can then be represented, using Bayes Theorem, as $p(\theta)p(d|\theta; R, b)$. Also, by assuming each word within a document d_k is independent, we then represent the probability of a document as

$p(\hat{\theta}_k) \prod_{i=1}^{N_k} p(w_i|\hat{\theta}_k; R, b)$ where N_k is the number of words in a document and i is the index of the word in the document. $p(\theta)$ is a Gaussian prior. $\hat{\theta}_k$ are the estimates for the document parameters generated by performing gradient ascent. Finally, by taking $log(p(D; R, b))$ alternating maximization can be performed as, previously stated. It should be noted that regularization terms are added by the authors. Intuitively, each document probability is regularized according to the norm of its parameters and the overall corpus probability is regularized according to the Frobenious norm of the entire matrix R.

4.3 Sentiment Features

This primary short coming of the semantic vectors is their inability to adequately capture the sentiment behind a word. We solve this problem by following the approach of Maas, et al (2011), in which an addition to the original objective function above is added to incorporate sentiment features. This generates a new R matrix of word vectors, which are then used to generate document vectors in the same manner as above. The sentiment features are learned by maximizing the objective function with respect to a new parameter ψ, which is used to compute the probability of a sentiment score of a word. For simplicity's sake, we consider the case of binary sentiment, where a word is either positive or negative. Thus we want to maximize the likelihood $p(s|w_i; R, \psi, b_c)$, where s is a sentiment label, ψ is a parameter matrix, and b_c is a bias term. This probability computed using the sigmoid function, $\sigma(\psi^T \phi_w + b_c)$. This probability is weighted by a value $\frac{1}{|s_k|}$, where $|s_k|$ is the number of training examples with the same sentiment as the document which the current word has been found it. This is done to help combat the imbalance of ratings in most movie review data sets.

The addition of the sentiment features provides us with the final objective function $v||R||_F^2 + \sum_{k=1}^{|D|} \lambda||\hat{\theta}_2^2 + \sum_{i=1}^{N_K} log(p(w_i|\hat{theta}_k; R, b)) + \sum_{k=1}^{|D|} \frac{1}{|s_k|} \sum_{i=1}^{N_K} log(p(s_k|w_i; R, \psi, b_c))$. We again sum over the probabilities of every word in every document to obtain the final cost.

To train perform alternating maximization the python library Theano (Bergstra et al. 2010) is used because it allows for symbolic differentiation to compute gradients. This allows for correct gradients to be computed instead of finding the gradient function by hand.

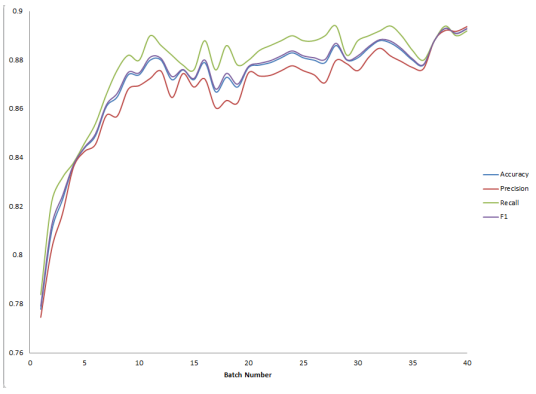

Figure 2: Cross-validation for each batch.

#	Δrank	Team Name	Score	Entries	Last Submission UTC (Best - Last Submission)
1	–	catherine	1.00000	2	Tue, 23 Feb 2016 15:17:08 (-0h)
2	–	Dizzy Hyperparameters	0.89273	3	Wed, 27 Apr 2016 03:16:10 (-26.5d)
3	↑3	JC_SP	0.87836	1	Wed, 27 Apr 2016 19:06:09
4	↑1	LRSA_CK_TV_PW	0.87691	1	Wed, 27 Apr 2016 14:08:15
5	↓1	MSA	0.87545	1	Mon, 25 Apr 2016 13:23:45
6	↓3	Movie Review Sentiment Prediction - NCPGJL	0.87509	7	Wed, 27 Apr 2016 23:44:30 (-3d)
7	↑1	breakingbad2	0.84309	2	Wed, 27 Apr 2016 20:22:26 (-0.2h)
8	↓1	Rishikesh	0.83782	2	Wed, 27 Apr 2016 20:28:05 (-2h)
9	↑1	AMS Sentiment Analysis	0.82891	1	Wed, 27 Apr 2016 19:38:36
10	↓1	WSL	0.82382	4	Wed, 27 Apr 2016 22:26:29
11	–	Malarlyer	0.82364	1	Wed, 27 Apr 2016 23:52:44
12	↑1	plew	0.74673	5	Wed, 27 Apr 2016 23:36:18 (-2.9d)
13	↑1	zaramatax	0.74545	1	Sat, 19 Mar 2016 12:18:12
14	↓2	APS	0.74509	2	Wed, 27 Apr 2016 22:12:43
15	–	QuocHuy	0.72782	2	Tue, 12 Apr 2016 21:15:19 (-9.2h)
16	–	agxcul	0.65545	4	Wed, 02 Mar 2016 05:25:25 (-0.3h)
17	–	ktagseth, vivek, janaki	0.00000	1	Wed, 27 Apr 2016 23:56:15

Figure 3: Results from the class competition.

4.4 Training and Testing

The training module processed the vectorized training set to produce a trained support vector machine and a matrix which will be described later. The training script uses learning.py and predict.py. Learning.py utilizes the learning algorithm that was previously discussed. The support vector machine is trained using the positive/negative labels and the features extracted by multiplying the R matrix with the tf-idf vectors of the training set. As mentioned earlier, a small development set was set aside to perform cross validation during each batch that was run to train the R matrix. The results of these trials were plotted in Figure 2. It is clear that performance improves the more training data the R matrix sees. With this in mind, the final model used the R matrix that was trained with all 40 batches.

The test module is responsible for producing a comma separated values file for the polarity of the test set reviews. This script simply uses the R matrix and SVM to predict the labels of the test set using features extracted in the same way as the ones for the training set. These results were submitted and placed 5th in the sentiment analysis competition, as is shown in Figure 3.

5 Conclusion

This approach provides a way to generate features that require limited prior knowledge about language structure and document content. The system outperformed many other systems based on hand engineered features, suggesting that word vectors are an effective way to determine document sentiment. The system did not attain the highest score in the class competition, however, and would likely benifit from additional tuning of hyper parameters and standard pre-processing proceedures used for sentiment analysis tasks. Such pre-processing was forgone in this work, as one of our goals was to minimize the amount of human generated data that was required to train our models, but proceedures such as negation preprocessing (Pang, et al.) would require no additional anotated data and could reasonably improve performance.

References

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. *Learning Word Vectors for Sentiment Analysis.* The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. *Thumbs up? Sentiment Classification using Machine Learning Techniques.* In EMNLP-2002, 7986.

J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio. *Theano: A CPU and GPU Math Expression Compiler*. Proceedings of the Python for Scientific Computing Conference (SciPy) 2010. June 30 - July 3, Austin, TX (BibTeX)

Peter Turney. 2002. *Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews*

Sanjiv Das and Mike Chen. 2001. *Yahoo! For Amazon: Extracting market sentiment from stock message boards.* In Proc. of the 8th Asia Pacific