SHABURY: A sentiment tagger for twitter data

Alec Pillsbury and Razi Shaban
Swarthmore College
{apillsb1, rshaban1}@cs.swarthmore.edu

Abstract

SHABURY is a sentiment tagging algorithm designed to determine the sentiment of individual tweets. The algorithm uses both SVM and tagged twitter data to decide whether or not a given tweet has a positive, negative or neutral sentiment. Our algorithm performed fairly well compared to other, non-SVM based versions of our algorithm.

1 Introduction

Automated classification of natural language is thriving field in computer science due to the variety of practical applications. One of these many applications is automated analysis of social media. Due to its widespread usage across multiple demographics, social media data provides a vast quantity of information ripe for use in both industry and government sectors. One use of automated classification is for determining whether a body of text has a positive or negative sentiment. Sentiment evaluation has a wide variety of uses, such as identifying a person's feelings about a product, sports team or their own government. This could allow for better targetting and classification of individuals themselves, allowing businesses and governments to know their consumers or citizens more accurately, allowing them to do their respective jobs better.

The SHABURY algorithm is a machine learning algorithm designed to tag raw Twitter data ('tweets') with individual sentiment tags. This algorithm was created for Task 9 of SemEval 2014 Rosenthal et al. [2014]. This algorithm is largely SVM-based, with some added functionality based on heuristics we found useful. SHABURY provided decent performance, though had some issues distinguishing between negative and neutral tweets.

2 Development

The development of our algorithm consisted of experimenting with various pre-processing techniques and statistical models. Based on these experiments, we decided to use a support vector machine (SVM) in tandem with a range of preprocessing methods. Our experimentation and final algorithm is described below.

2.1 Preprocessing

We experimented with a number of preprocessing techniques while cultivating our final list of methods. In doing this, we looked at a number of previous SemEval participant results, especially Amir et al. [2014], Jaggi et al. [2014], Mohammad et al. [2013]. In order to simplify the tweets for our statistical models, we explored case-folding, tokenization, part-of-speech tagging, and text normalization.

We ran our pre-processing tests on a Naive Bayes Classifier using cross-chunking across the SemEval training data. Performance results for the pre-processing test are in Table 1 on page 4.

2.1.1 Case-folding and text normalization

To simplify potential ambiguities across capitalization standards and usages, we converted all

letters to lowercase. While this by itself seems to decrease performance across the board, we found that it contributed positively, if marginally, to other pre-processors. In addition, we replaced all URLs with *<URL>* and all user handles (*@user*) with *<USER>*, following Amir et al. [2014].

2.1.2 Tokenization and Part-of-Speech tagging

Tokenization is the process of separating a string of text into its constituent parts in order to more easily classify strings of characters. To facilitate this, we used the ArkTweetNLP tokenizer via a Python wrapper written by Richard Wicentowski of Swarthmore College [Bhatia et al.]. The ArkTweetNLP tokenizer also facilitates part-of-speech tagging, which allowed us to further disambiguate between the various uses of individual words.

2.1.3 Negation Detector

We found that adding a negation detector significantly improved our scores, especially in the negative category. We used an algorithm similar to the one outlined by Potts, which determined whether a word should be considered negated. The algorithm did this by labelling words that appeared within a negation words "scope," which was determined by the presence of the word *not* and the placement of end-of-phrase punctuation.

2.1.4 Bigrams and trigrams

In order to collect more features, we also experimented with using word bigrams and trigrams. We used the bigram and trigram features built into the Natural Language Toolkit (NLTK) [Bird et al., 2009]. Implementing these individually significantly improved identification of negative tweets, increasing negative F1 from a baseline of 18.44 percent to 26.94 and 26.48, respectively. Combining the two of them resulted in an even greater F1 score of 38.23 percent, and layering them on top of the previous preprocessing methods led to the best base performance in Negative F1, at 40.24 percent.

3 Training

We were able to access 8111 tagged tweets provided in the SemEval-2015 training set. In order to try to improve the accuracy of our negative sentiment detection, which hung significantly below neutral and positive sentiment detection in our basic tests. We considered using emoticons, which are ways that people independently tag sentiment in their writing, as a way of collecting large amounts of untagged data. Fortunately, we found that Go et al. [2009] had already identified this possibility and built the Sentiment140 Lexicon, which was automatically compiled by identifying tweets containing positive and negative emoticons. This corpus contains 1,600,000 tweets, a small fraction of which collapses a Naive Bayes classifier. In the sections below, we discuss the different classifiers that we tested and their performance on the training data. We made great use of the implementations of these algorithms built into the Scikit-learn Python packages [Pedregosa et al., 2011].

3.1 Presence and Frequency

Our representations used the bag-of-words approach, and we distinguished in our results in Table 2 on Page 4 between presence, frequency, and term frequency-inverse document frequency (Tf-idf). Presence is a binary method of counting; a feature is either present or it is not, as opposed to the frequency approach, in which each instance of a feature is counted. Tf-idf weighting, on the other hand, is designed to lend extra weight to words which appear rarely and less weight to very common words.

3.2 Naive Bayes Classifier

Naive Bayes Classifiers are relatively simple algorithms but often perform just as well (or better) than more complex algorithms. Besides a Naive Bayes classifiers we built from scratch, we tried using two of scikit-learn's Naive Bayes implementations, BernoulliNB and MultinomialNB. MultinomialNB significantly outperformed BernoulliNB across the board, even when features were measured by presence (Bernoulli's expected input).

3.3 Support Vector Machine

Support Vector Machines (SVM) are a very effective method for classifying data. Due to the fact that traditional SVMs do not scale that well when classifying textual data, we used LinearSVC, a significantly more performant derivation. LinearSVC improved SHABURYś performance significantly, even moreso than the Naive Bayes Classifiers.

4 Analysis

Our experiments with various pre-processors, different models, and an expanded training set had mixed results. We managed to establish fairly solid neutral and positive sentiment classifiers, with our LinearSVC bigram frequency model returning F1 scores of 73.79 and 68.98, respectively. However, the aim of the majority of our experiments was to improve upon our initially abysmal negative sentiment classification, and in this we met mild improvements and overall disappointment.

4.1 Naive Bayes

The Naive Bayes classifier proved quite flexible, as we were able to increase the negative F1 score for MultinomialNB from 27.58 to 41.37 by implementing binary feature recognition (presence as opposed to frequency) and, more significantly, implementing trigrams. Comparing the Naive Bayes classifier assessed at the unigram level (overall 46.14) to the bigram (46.71) and trigram (52.12) reveals the large improvement ngrams can offer. This difference is even more notable when we consider that neutral and positive F1 scores hardly change from unigrams (69.09 and 65.78, respectively) to trigrams (63.61 and 65.78); rather, trigrams help the Naive Bayes classifier better recognize negative sentiments, increasing the negative F1 from 26.51 to 38.47 when using frequency counts. Tf-idf proved terrible at recognizing negative sentiments, with negative F1 scores of 0.17 and 0.00 at the bigram and trigram levels, respectively, despite otherwise comparable scores.

4.2 Support Vector Machines

The improvements caused by trigrams in the Naive Bayes classifier did not translate into our LinearSVC implementation. Rather, we found that implementing bigrams had a mixed effect and trigrams had a detrimental effect. When examining frequency, for example, the LinearSVC frequency unigram, bigram, and trigram negative F1 scores were 45.01, 42.45, and 36.62, respectively, dragging the overall score down from 56.73 to 52.42 despite neutral and positive F1 scores remaining stable. Binary feature recognition proved effective at the unigram level, but they only narrowly edged out frequency counts. Adding additional training data from the Sentiment140 lexicon also proved fruitless, as scores were unaffected or decreased, whether 5,000 or 50,000 extra training tweets were provided.

4.3 Logistic Regression

We used a logistic regression algorithm provided by Scikit-learn in an attempt to see if a third statistical approach could help us achieve higher negative scores, but here again we were disappointed. Rather, logistic regression seems to perform on par with our higher-performing versions of LinearSVC, but lags still in negative sentiment identification.

5 Conclusion

Our final SHABURY algorithm is a LinearSVC unigram model that uses binary feature counting and is trained on the SemEval training data and 5000 negative-tagged tweets from the Sentiment140 lexicon. This algorithm provides a viable way to identify tweets with positive and neutral sentiment, and offers modest recognition of negative tweets. Moving forward, we are convinced of a need to find new ways to detect negative sentiment, whether that be syntactic or semantic analysis, that can go beyond the methods described in this paper.

Table 1: Preprocessing performance comparison

Pre-processing	**Accuracy**	**Overall Score**	**Negative F1**	**Neutral F1**	**Positive F1**
Raw data	62.11	41.11	18.44	68.60	63.77
Case-folding	61.95	40.40	16.82	68.26	63.97
Tokenization	63.72	43.46	20.79	69.47	66.12
POS tagger	63.79	42.84	19.44	69.74	66.23
Text normalization	62.39	41.62	19.52	69.01	63.73
Negation detection	61.96	42.97	22.19	68.11	63.75
Word bigrams	61.13	45.51	26.94	66.27	64.07
Word trigrams	61.41	45.40	26.48	66.46	64.32
Bigrams + trigrams	54.16	50.51	38.23	53.80	62.78
First five together	63.64	46.14	26.51	69.09	65.78
First five + bigrams	63.50	45.92	26.27	69.00	65.56
First five + bi/tri	61.34	53.10	40.24	63.62	65.97

Table 2: Training model comparison

Features	**Count**	**Accuracy**	**Overall Score**	**Negative F1**	**Neutral F1**	**Positive F1**
MultinomialNB	*Frequency*	63.70	46.71	27.58	69.01	65.84
+ trigram		61.16	52.12	38.47	63.61	65.78
	Presence	63.98	46.88	27.41	69.26	**66.35**
+ trigram		60.64	**53.53**	**41.37**	62.18	65.70
	Tf-idf	61.06	27.46	0.17	71.20	54.76
+ trigram		**67.10**	23.28	0.00	**76.24**	46.55
LinearSVC	*Frequency*	67.27	56.73	45.01	72.17	68.45
+ bigram		**68.36**	55.71	42.45	**73.79**	**68.98**
+ bi/trigram		67.78	52.42	36.62	73.54	68.22
	Presence	67.30	**57.20**	**46.12**	72.05	68.28
+ bigram		67.67	54.48	40.15	73.01	68.81
+ bi/trigram		67.95	51.60	35.16	74.14	68.04
	Tf-idf	66.93	54.00	41.92	72.96	66.09
+ bigram		65.73	49.67	35.35	72.68	64.04
+ bi/trigram		61.01	42.92	28.38	69.24	57.46
LinearSVC+5k	*Frequency*	65.84	56.15	45.57	70.53	66.73
+ bigram		66.18	54.27	41.29	71.29	67.26
+ trigram		65.28	50.49	35.18	71.07	65.81
LinearSVC+10k	*Frequency*	65.54	53.40	40.31	70.81	66.49
+ trigram		65.20	51.38	37.13	70.83	65.64
LinearSVC+50k	*Frequency*	66.05	54.14	41.07	71.14	67.21
+ trigram		65.22	51.03	36.36	70.93	65.70
LogisticRegression	*Frequency*	**68.21**	**53.54**	**38.27**	**73.98**	**68.80**
+ trigram		67.15	49.37	31.69	73.66	67.04

References

Silvio Amir, Miguel Almeida, Bruno Martins, Joao Filgueiras, and Mario J. Silva. TUGAS: Exploiting Unlabelled Data for Twitter Sentiment Analysis . 2014.

Archna Bhatia, Dipanjan Das, Chris Dyer, Jacob Eisenstein, Jeffrey Flanigan, Kevin Gimpel, Michael Heilman, Lingpeng Kong, Daniel Mills, Brendan O'Connor, Olutobi Owoputi, Nathan Schneider, Noah Smith, Swabha Swayamdipta, and Dani Yogatama. Tweet NLP . URL `http://www.ark.cs.cmu.edu/TweetNLP/`.

Steven Bird, Ewan Klein, and Edward Loper. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, 1st Ed. . 2009. URL `http://www.nltk.org/book/`.

Alec Go, Richa Bhayani, and Lei Huang. Twitter Sentiment Classification using Distant Supervision . 2009.

Martin Jaggi, Fatih Uzdilli, and Mark Cieliebak. Swiss-Chocolate: Sentiment Detection using Sparse SVMs and Part-Of-Speech n-Grams . 2014.

Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. NRC-Canada: Building the State-of-the-Art in Sentiment Analysis of Tweets . 2013. URL `http://www.umiacs.umd.edu/~saif/WebPages/Abstracts/NRC-SentimentAnalysis.htm`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. volume 12, pages 2825–2830, 2011.

Christopher Potts. Sentiment Symposium Tutorial . URL `http://sentiment.christopherpotts.net/lingstruc.html#negation`.

S. Rosenthal, A. Ritter, P. Nakov, and V. Stoyanov. SemEval-2014 Task 9: Sentiment Analysis in Twitter. 2014.