Neil Patel

LING406 Term Project
Sentiment Analysis, Extra Credits 1&2
05/10/2018

Introduction

Sentiment Analysis, the understandings of overarching opinions and feelings in a written piece of text, is a domain that has been exploding like crazy in the last 5-10 years. With the advent of both cheap, fast computers (for easy calculations) and the internet(for data), sentiment analysis on a wider scale finally makes sense.

Its use cases may be many, but they all boil down to 1: knowing what people are thinking makes it easier to make a decision, be it for ads to display, products to sell, or even how to approach said person. This summer, I worked on analyzing Twitter about Kohl's department stores over the last 2 years. Using sentiment analysis, I was able to show that there are, in fact, less negative tweets about the holiday in-store experience then the online experience, and that certain geographical locations saw tons of positivity.
Outside of things I've worked on, it can help politicians track how the populace feels about laws (and their campaigns)

For my term project, I implemented a basic bag-of-words classifier and then tried out features like stopward extraction, negation consideration, alphabetical considerations,weighting, and size limitation. I also tried out various ML algorithms. This project ran against 2 datasets to allow for comparisons between different types of data.

Problem Definition

As we discussed above, sentiment analysis is the process of trying to "mine" text information, be it words, sentences, or documents, for opinions, views, and attitudes. These can be subjective or objective, polar, or on a 1-5 scale.

Though we can look at sentiment analysis from both the computational linguistics and the statistics/machine learning side, lets take a stab at the computational linguistics side as this is a linguistics class. What we're trying to do with sentiment analysis is to find patterns in the language and generate features and methods to define these words and patterns simply.

The system build for this project takes in review documents classed as negative or positive and trains on them , allowing for predications of a new text as positive or negative. Since the yelp dataset is on a 1-5 scale, it first needed to be converted over to a positive/negative scale.

Previous Work

Though there are literally thousands upon thousands of papers written about sentiment analysis, certain papers were either considered crucial enough that my boss suggested reading them or no real introduction to sentiment analysis could avoid mentioning them. Here are 3 papers I read that were extremely interesting and helpful for my project.

Twitter as a corpus for sentiment analysis and opinion mining by Alexander Pak and Patrick Paroubek was a game changing paper for me; it was the first paper where I actually understood what I was reading, and the first time I learned about the term "stopwords" (words like "the", "a", "an", etc.) . Pak and Paroubek first created a unigram classifier, before continuing on to build bi and tri-gram representations. They then built out both multinomial Naïve Bayes and SVM classifiers. In the case of this paper, they actually saw the most accuracy with the usage of bigrams, and a direct correlation between the size of the dataset and their F-measure. Their Naïve Bayes classifiers also saw better results then their SVM model.

Twitter sentiment analysis: The good the bad and the omg! was a paper that I found really interesting this summer; they did something interesting in that they used 3 different datasets from twitter with different identifying information (ie Hashtags, emoticons, etc.)
Their preprocessing had them using POS, but more interestingly, counted informal modifiers like ALLCAPS or happyyyyyyy. They built unigram and bigram models with these and other features, and were able to conclude that using parts of speech brought the overall accuracy down, and the best classifier used everything but. They actually posited that perhaps since tweets are so short, people aren't really utilizing grammar the way they would in a formal setting.

And no collection of papers is complete without something by Bo Pang and Lillian Lee of Yahoo Research; in this case, their 2004 *A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts.* This paper showed a statistically significant increase (~4%) in accuracy while only utilizing 60% of the provided words, something that was unheard of at the time. This paper suggested something well beyond the scope of this project; employing a subjectivity filter to remove objective sentences from the reviews before applying an SVM on them. However, even there, they tried out Naïve Bayes and SVM classifiers for their subjectivity filters!

Approach

Feature Representation

My Bag-of-Words (BoW) Model

A bag of words model builds a dictionary out of individual words as a feature representation. The feature considers whether the corresponding word exists or not, and is roughly equivalent to a unigram model.

Feature Collection:

Is every word in the feature set useful? Many of the words in the review are un-necessary; I decided to try out several different features to help alleviate this.

1) **Removing Punctuation:** iterating through the tokens and removing punctuation seems like it would help keep useless information out
2) **Removing Insignificant Words** : single letter words / mistypes are also probably not useful for my sentiment analysis
3) **Using only alphabetical tokens:** This seems important because it seems doubtable that 3leet or 9hundred would be reasonably common features
4) **Removing stopwords:** words like “The”, “a”, “at” and the like really don’t help much with the actual sentiment of the review
5) **nltk.negation**: Perhaps trying to identify words that appear between the noun and a negation word as negative will help because “this is a good movie” is different then “This is not a good movie”.
6) **Weights:** Weighting seems important; rather then a constant +1 per word, maybe words that show up very often in negative reviews should give more then a +1 for negative?
7) **A decision threshold:** This actually seemed like the single most important thing to me in my head. Words like “movie”, “film”, “cinema” etc show up really often in both the positive and negative sides; however they’re being determined as belong to positive or negative sometimes by 3-5 occurances of the word in one side versus the other. What if we only used words for negative that are at least 20% more common for the negative side? What about words that are 40% more common on the positive side as being the point givers for positive sentiment?

Feature Size:

Once the rest of these features have been selected, I actually wanted to see if the size of the positive/negative bag’s of words make a difference, and limited them to various trials like 100,200,500,1000,2000,5000,6000,8000 and 10000.

All of these features and sizes are play-aroundable with by executing experiment_BoW.py; it prompts for these parameters at the start. Everything except negation was implemented by hand, and the list of stopwords also came from nltk.

Machine Learning Classifiers

I chose to apply 4 different machine learning classifiers to the movie review dataset built on top of my BoW's code. I implemented the Naïve Bayes by scratch, but chose to import the Logistic Regression, Decision Tree, and Support Vector Machine from sci-kit learn as the performance would be much faster the something that I could implement on my own.

Though there are 5 files for machine learning classifiers in the provided code folder, these classifiers should run when you execute an experiment because I wished to be able to see how certain classifiers were affected by certain feature sets. The kNN file is extra as I decided against using kNN's for this project due to time concerns.

Naïve Bayes: A multinomial naïve bayes classifier actually makes a lot of sense because the only real assumption is that the words are independent of each other (Which is not quite true but it's an acceptable assumption). The classifier computes the prior probabilities of each feature for the training data, then estimates the likelihood probability from the given testing sequences' words' prior probabilities. As this was my 3rd time implementing some form of Naïve Bayes (once in CS448AI, once in this class), It was easy for me to implement this by hand.

Regularized Logistic Regressions: A logisitic regression classifier chooses to only computer parameters that maximize the likelihood of a solution on the training corpus, making the naïve bayes a specialized case of one. Words with higher weights are better features, here.

SVM- Support Vector Machine: An SVM can be thought of as trying to separate 2 sets of points by a line, but in many many dimensions. The line is then considered a hyperplane, and is found using support vectors (significant features in the feature set).

Decision Tree: The point of a decision tree is to build classification trees out of the input data. Each level chooses features as split nodes. The sentences are split according to the words in them.

Results

To see the raw results from my tests, open the Data_from_reviews Excel file. It shows the parameter options and outputs for each test I ran.

Methodology:

The way I chose to test my code was to add one feature at a time to see if they provided a statistical gain in accuracy for most/all of my models. For example, I first ran my code with no parameters, then with the features with punctuation removed. This turned out to be a beneficial feature so I added the alphabetical limitation, and then the stopword removal, and so forth. You can see a condensed table of these trials below; for ease of filling up I stopped filling the last columns after the first row as its definitely a feature that brings down the accuracy. In the case of several of the features, ie negation, weighting, and punction, that seem to bring down the

accuracy of several classifiers, I ran more tests to make sure they should be included before continuing on.

Tests:

	Straight Bag_of_words	BoW + punctuation removal	BoW+PR+ limit to alpha	BoW+PR +alpha+ stopwords	BoW+PR +A+SW+Negation	BoW+PR +A+SW+Neg +Weighting
BoW	64.6%	65.6%	65.6%	64.6%	62.2%	53.6%
NB	82%	81%	81.3%	82.6%	71.6%	
SVM	78%	50%	76.6%	76.8%	82.2%	
LR	75.4%	59.4%	77.4%	82.4%	82.6%	
DT	61.4%	63.2%	63.6%	68.8%	61.4%	

After testing these different features, I tested all the classifiers against what the threshold margins to determine if a word should be included in the positive or negative bag of words should be; this data can again be found in the data file, but the values in the lead were 1.4 and 1.5 (ie a 40 or 50% more frequent appearance in one category versus the other). I chose 1.5 because it kept more of the classifiers recalls higher.

I finally tested to show that the ideal size of the positive and negative bag of words is 7000; any more and they start using words that are so infrequent that they just provide skew, and any less steals some features that could be useful.

Lets consider my findings in 2 parts:

Bag of Words features:

Overall most of my features provided added value/accuracy when put into effect on the data set. Although some make obvious sense (limiting to alphabetical, removing stopwords, etc.) lets talk about a couple that made a lot less sense.

For example, the negation aspect. I'm going to bring this up more later in the report, but I personally felt as though negation should have a much larger positive impact then its negative impact. It seemed to me that identifying likes as likes_NOT would improve the ability to specify between negative and positive words.

Lets also look at the weightages. I personally felt as though by modifying the scoring per word rather then adding 1 regardless, I would get a more accurate result. I think that perhaps my methodology, which was counts_of_this_word/counts_of_total_words was less accurate then what I had hoped to try, count_of_this_word/count_of_words_on_average, but I ran out of time.

Machine Learning Classifiers:

Naïve Bayes: My personal favorite ML classifer was the Naïve Bayes classifier. It runs extremely fast (no vectorization needed!) and starts and stays relatively accurate, although starts loosing out as feature sets become extremely large. This makes sense as Naïve Bayes ignores relationships between features, where as SVM doesn't.

SVM: The SVM was extremely accurate, which makes sense as it can deal with many features without assuming independence. They took forever to train though.

Decision Trees: Personally I felt as though decision trees had a lot of lost potential as I had heard stories of random forest winning all the major Kaggle competition for years; I think this has to do with the fact that I used the basic sklearn implementation and this lead to tons of overfitting on the training data.

Logistic Regression:
The logistic regression got pretty accurate pretty quickly and stayed there, rather then improving as the feature size increased. This seems to make some sense as it really does try to fit as well as possible to the correlations between features by increasing their weights, so in our case the additional features just weren't providing the additional correlation. This makes sense as we determined the later features are basically just noise.

In my specific case, with a size of 7000 and a threshold of 50%, as well as the features for alpha-only, punctuation, insignificance, and stopwords, my SVM model was the most accurate with an 83% accuracy.

Discussions and Conclusions

Honestly I've learned a lot about sentiment analysis from a computational linguistics perspective. Where as in the past, my exposure to sentiment analysis has been to import an nltk or scikit learn package to take care of the entire process, this was my first time really building it on my own. It was also my first time implementing all the classifiers I know and love to text data. The SVM , specifically, really confused me as to how to convert the reviews into different numeric attributes.

To improve my project, there are a couple of linguistics challenges that would need to be solved. The most glaring one is a better way to deal with negation, since nltk.negation clearly didn't help. I understand that this is a complex topic; I read an interesting 2010 paper (Wiegand) cited below that really made me think about a lot of the challenges behind having a computer determine negation in a sentence or phrase.

Another improvement that I could try is the weightage; I feel as though there was a lot of potential in some kind of a weighting function, just not in the specific method that I had tried to implement it.

I would also try to improve my project using stemming, although I feel like stemming is a double edged blade because although stemming would decrease the feature size, it would remove some of the information about relations between words.

If I had to do this project again, I think that I would reformat the way I have my code put out its output; something that allows me to take a more "at a glance" view and figure out if it has improved or declined in performance, because right now I left a lot of it to do by hand.

A real breakthrough in this sort of a project would be an improvement in a linguistical morphology model; if it would somehow be possible to capture the intonation of a review (something that humans inherently add when reading something), it would be a very useful feature.

Extra Credit 2: Yelp Dataset

Approach

For the extra credit portion of this project I decided to try a slightly different approach. First, I am going to run the code with the no parameters (Just BoW), then run the code with the same parameters that got me the best result on the last dataset. (4x features, BoW sized 7000, and a threshold of 1.5)

Approach

Although I was able to repurpose bits and pieces of my code, for the most part I wrote all new functions. I had to come up with a way to read in all the reviews from one file, sort them, and then delimit them as negative or positive. I did this by sorting things rated 1,2,3 as negative, and 3.5,4,5 as positive. To identify something as a 3.5 rating I searched for the "3.5" inside of the review and moved it accordingly.
To run this code, run the file yelp_import_convert.py and answer the prompts.
I'm going to be trying all the same features and tests as in my last section.

Results

Baseline:

	BoW	NB	SVM	LR	DT
Baseline Accuracy	83%	59%	76%	92.5%	80.6%

With Parameters from movie review dataset:

	BoW	NB	SVM	LR	DT
Movie review parameters	75%	82.4%	87.5%	90.5%	89%

At this point I discovered just how long the code takes to run and decided to stop testing with the stopwords feature, as that took up too much computation time.

However, I ran the code with different features enabled, and here are my results for what I was able to have finish computing.

	BoW + punctuation removal	BoW+PR+ limit to insignfi	BoW+PR +alpha+ insignif	BoW+PR +A+inisgnif+Negation	BoW+PR +A+insig+Neg +Weighting
BoW	77.6%	77.6%	75.9%	76%	50%
NB	57.6%	64%	63.7%	63.7%	63%
SVM	83%	82%	85%	65.1%	81%
LR	92.7%	92.5%	92.3%	85.7%	86%
DT	85.7%	88.7%	88.7%	87.1%	87%

I found these results extremely surprising as for this dataset, it suggests that removing the non alpha characters as well as the stopwords causes a decrease in the overall accuracy of the classifiers.

I also need to test out the features with respect to size and the threshold. For this dataset, I found a threshold of 1.6, although not providing the overall most accurate measurement, provided the best results. I also found that the feature size after 100 didn't really make a statistical difference in the accuracy rates. My overall best accuracy rates were provided by the Regularised Logistical Regression between 92-93% depending on the parameters.

Discussions and Conclusions

Although for the original review dataset, I found that both the stopwords and the limitation of tokens to alphabetical words only really helped the accuracy, for the Yelp dataset, these features hindered the analysis. The alphatoken one makes sense, as if a review has a 1 or a 2 in it they're probably talking about the number of stars and those both classify as negative. The stopwords feature hindering the Yelp dataset made less sense to me, but some googling led me to an article that suggests that this is a hotly debated topic in computational linguistics as some people suggest that the numbers and positions of stopwords can actually convey sentiment-related information.

I found it interesting that on the Yelp dataset the classifiers were significantly more accurate then on the review dataset. This is most probably because of the wealth of additional training data provided (literally a factor or 10x) and the fact that the numbers help out a decent bit.

It makes a lot of sense that the Regularized logistical regression would be the most accurate, because as mentioned earlier in this report, its able to really weight correlations to help it make a decision. Most probably it found a few really accurate correlations and stuck with them.

Works Cited

Pak, A., & Paroubek, P. (2010, May). Twitter as a corpus for sentiment analysis and opinion mining. In LREc (Vol. 10, No. 2010).

Kouloumpis, E., Wilson, T., & Moore, J. D. (2011). Twitter sentiment analysis: The good the bad and the omg!. Icwsm, 11(538-541), 164.

Pang, Bo, and Lillian Lee. "A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts." Proceedings of the 42nd annual meeting on Association for Computational Linguistics. Association for Computational Linguistics, 2004.

Wiegand, M., Balahur, A., Roth, B., Klakow, D., & Montoyo, A. (2010, July). A survey on the role of negation in sentiment analysis. In *Proceedings of the workshop on negation and speculation in natural language processing* (pp. 60-68). Association for Computational Linguistics.