Natural Language Processing, Project Report

UID: 1690550

March 16, 2017

1 Introduction

Sentiment classification is an ongoing area of research in data mining and natural language processing areas. It has a wide range of applications, varying from opinion gauging to marketing. Its potential is constantly growing together with the development of related areas, including machine learning and neural networks.

The notion of sentiment classification has appeared in the early 2000s, and has proven to be a challenge that still fuels research. It has practical uses as mentioned before, but the difficulty of the task itself encourages researchers as well.

In this group project, we are interested specifically in the sentiment classification of tweets, both as a whole and as phrases taken out of this corpus.

2 Preprocessing

2.1 Tokenization

Tokenization of tweets is a challenge in itself, as the corpus contains various features not present in texts coming from different sources. They include hashtags (#), tags (@), urls, emoticons, and unconventional abbreviations.

Fortunately, multiple libraries exist to tokenize tweets. The one we have chosen is Natural Language Toolkit [1]. It is capable of much more than just parsing tweets, however we used it only for this purpose.

The output of it is the text split into a list of tokens, which then can be pre-processed.

2.2 Negations

Given a tweet, we have chosen to consider every word that follows a negation to have the opposite sentiment than usual. That is, after a word such as "not", "isn't", "wasn't", etc, and up to and including a token that is followed by punctuation, all tokens have reversed sentiment. More precisely, we construct a regex to capture the negations:

```
regex_negation = "^(not)|[a-zA-Z]+n't$"
```

Each of the words that follow negation then gets prepended with "NOT_" identifier. However, we do not want to prepend the identifier to words such as "but" or to urls, smileys or tags. Those are picked by regex:

```
regex_exception = "^(but|(\W.*)|(http:.*))$"
```

The relation between a word and its negation is such that the weight of the negation is the negative of the weight of the word.

This has to be done obviously before classifying urls, tags, etc., as we replace them with "URLLINK", "USERTAG", or similar identifiers.

2.3 Stemming

To ensure that different form of a word are considered as one, we should to perform stemming on our tokenized tweets. To do this, we attempted to use the Stemming library created by Matt Chaput [2].

It is based on the Porter2 algorithm for stemming English language. The full description of the algorithm can be found here: `http://snowball.tartarus.org/algorithms/english/stemmer.html`

While working on this part, there was a concern that the algorithm was overfitting and truncating too much, and after comparing the scores, there was no significant difference. We decided to leave it out.

2.4 Reading data

Finally we can scan the training files to count how many times each word occurs with each sentiment. The outcome of the preprocessing is a dictionary of words which appear in the corpus, each one associated with an array:

`{'word' : [positive count, neutral count, negative count]}`

3 Classification

The challenge was to find a way of converting the aforementioned dictionary of words and counts for each sentiment, to a dictionary of words and weights.

The aim was to assign a score s to each word, with $s \in [-1, 1]$. From the score we can read off the sentiment as follows:

Given a fixed $\epsilon > 0$, if:

- $s < -\epsilon$ the sentiment is negative
- $-\epsilon < s < \epsilon$ the sentiment is neutral
- $s > \epsilon$ the sentiment is positive

The first approach was a naive mean weight calculated by the formula:

$$\text{weight} = \frac{\text{positive count} - \text{negative count}}{\text{total count}}$$

Where total count means the number of times the word appears in the corpus.

We have also experimented with different weight functions,such as

$$\text{weight} = \frac{(\text{positive count} - \text{negtive count}) \times (\text{total} - \text{neutral count})}{\text{total}^2}$$

However all of the functions we tried gave results very similar (differing by less than 1%) to the results given by the naive formula. Therefore we have chosen to stick to the naive one.

We also had to adjust the interval for neutral sentiment, determined by the value of ϵ.

To obtain the scores, we have used SciKitLearn [3]. It is a necessary dependency, without which the file `classifier.py` will not work.

3.1 Results

After training the program on the cleansed training files and running on dev files, we obtained the following accuracy for different values of epsilon, as shown in Table 1.

In this case accuracy is calculated as the percentage of tweets for which our predicted sentiment agreed with the given sentiment in the dev corpus.

As it is clear from the table, the models are more accurate the tighter the neutral interval is. We have therefore chosen the value of $\epsilon = 0$ for the final version.

ϵ	Training set A	Training set B
0	63.9%	36.6%
0.02	63.3%	27.6%
0.04	62.1%	11.9%
0.06	61.5%	3.4%
0.08	60.6%	1%
0.1	59.3%	0.0%

Table 1: Comparison of scores for A and B for a different interval for neutral sentiment

4 Conclusion

We have created a program that learns from a training set of tweets, and based on that predicts the sentiment of tweets not seen before.

The accuracy is a little disappointing. The program is more accurate with shorter phrases than whole tweets. In the former case we achieve the probability of $\frac{2}{3}$ of getting the sentiment correct. In the latter, the probability of a correct score is almost the same as that of guessing it - we have three possible sentiments, so the probability of a correct guess is $\frac{1}{3}$.

This of course can be extended to a larger project. Given the opportunity, I would like to experiment further with different weight functions, attempting to skew them towards away from 0.

References

[1] S. Bird et al., *Natural Language Processing with Python*, O'Reilly Media Inc, Sebastopol, CA, 2009

[2] M. Chupat, stemming-1.0, available: `https://pypi.python.org/pypi/stemming/1.0`

[3] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python" in *Journal of Machine Learnin*, vol. 21, 2011, pp 2825-2830