

Twitter Sentiment Analysis

TASK:

The goal of the project was to do twitter sentiment Analysis. Microblogging sites such as Twitter have evolved to become the source of varied information .This is due to nature of microblogs on which people post real time messages about their opinions on a variety of topics, discuss current issues, complain, and express positive sentiment for products they use in daily life. Many companies are interested in mining sentiments about their products from such microblogging websites. Thus our challenge is to

Build technology to detect and summarize an overall sentiment of Twitter feeds

PROBLEM:

Sentiment analysis has been handled as a Natural Language Processing task at many levels of granularity. Starting from being a document level classification task (Turney, 2002; Pang and Lee, 2004), it has been handled at the sentence level (Hu and Liu, 2004; Kim and Hovy, 2004) and more recently at the phrase level (Wilson et al., 2005; Agarwal et al., 2009). Our approach in this project mimics the one taken by Agarwal et al. 2011

The paper introduces the concept of POS-specific prior polarity features and explores the use of a tree kernel to obviate the need for tedious feature engineering.

DESCRIPTION OF APPROACH

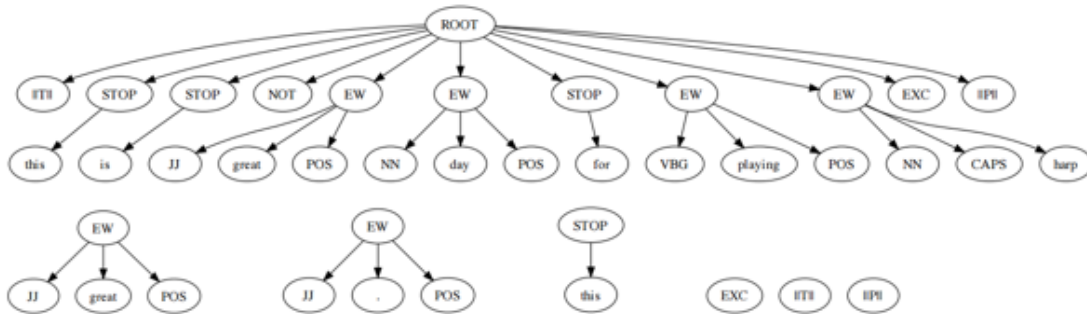
In addition to the popular unigram or Bag of words model for mining sentiments we introduce certain more features depending upon the polarity of the words. These features can be divided into following categories

N	Polar	POS	# of (+/-) POS (JJ, RB, VB, NN)	f_1
		Other	# of negation words, positive words, negative words	f_2
			# of extremely-pos., extremely-neg., positive, negative emoticons	f_3
			# of (+/-) hashtags, capitalized words, exclamation words	f_4
	Non-Polar	POS	# of JJ, RB, VB, NN	f_5
		Other	# of slangs, latin alphabets, dictionary words, words	f_6
			# of hashtags, URLs, targets, newlines	f_7
R	Polar	POS	For POS JJ, RB, VB, NN, \sum prior pol. scores of words of that POS	f_8
		Other	\sum prior polarity scores of all words	f_9
	Non-Polar	Other	percentage of capitalized text	f_{10}
B	Non-Polar	Other	exclamation, capitalized text	f_{11}

Our features can be divided into three broad categories: ones that are primarily counts of various features and therefore the value of the feature is a natural number $\in \mathbb{N}$. Second, features whose value is a real number $\in \mathbb{R}$. These are primarily features that capture the score retrieved from sentiwordnet. Thirdly, features whose values are boolean $\in \mathbb{B}$. These are bag of words, presence of exclamation marks and capitalized text. Each of these broad categories is divided into two subcategories: Polar features and Non-polar features. We refer to a feature as polar if we calculate its prior polarity either by looking it up in Sentiwordnet or in the emoticon dictionary. All other features which are not associated with any prior polarity fall in the Nonpolar category. Each of Polar and Non-polar features is further subdivided into two categories: POS and Other. POS refers to features that capture statistics about parts-of-speech of words and Other refers to all other types of feature.

These features go a long way in improving the accuracy of the unigram baseline model. We used two separate kernels in our approach the traditional linear and the one based on Tree Kernels.

We design a tree representation of tweets to combine many categories of features in one succinct convenient representation.



DETAILED APPROACH AND SPECIFIC IMPLEMENTATION

DATA :

I used data from one kaggle problem("Partly Sunny with plenty of Hashtags). The data contains tweets about weather which has been classified in the following categories

S3 : Neutral or 0

S4 : Positive or 1

S3 : Negative or -1

The data contains some other classification but we prune it to the above mentioned columns. Also since each tweet is reviewed by many users, the classification is not mutually exclusive so we had to take the maximum value of the three as the true label for a certain training set

The datasets contains 77496 lines of tweet which is very huge . Storing it on file took my entire disk space so the file provided in the submission correspond to 10000 lines only

DATA PREPROCESSING And REPRESENTATION

we designed two new resources for pre-processing twitter data:

- 1) an emoticon dictionary and
- 2) an acronym dictionary.

We prepare the emoticon dictionary by labeling 170 emoticons listed on Wikipedia with their emotional state. For example, "🙂" is labeled as positive whereas "😞" is labeled as negative. I was tempted to use the AFINN emoticons but their list is very limited. We assign each emoticon a label from the following set of labels: Extremely-positive, Extremely-negative, Positive, Negative, and Neutral.

We compile an acronym dictionary from an online resource(noslang.com) The dictionary has translations for 5,184 acronyms.

We pre-process all the tweets as follows:

- a) replace all the emoticons with a their sentiment polarity by looking up the emoticon dictionary,
- b) replace all URLs with a tag ||U||,
- c) replace targets (e.g. "@John") with tag ||T||,
- d) replace all negations (e.g. not, no, never, n't, cannot) by tag "NOT", and
- e) replace a sequence of repeated characters by three characters, for example, convert cooooooooool to cool.

After preprocessing the data we use the Stanford Tagger to do part of speech tagging of tweets. Since Java provides more flexibility in using the MaxentTagger such as tweaking the matching braces options of PTB tokenizer so that the emoticons like (:) are preserved and not converted in -RRB and :-). For this reason exactly instead of Stanford Parser we used nltk tweet tokenizer

We then built a tree representation of the tweets using the following rules

Initialize the main tree to be "ROOT". Then tokenize each tweet and for each token:

- a) if the token is a target, emoticon, exclamation mark, other punctuation mark, or a negation word, add a leaf node to the "ROOT" with the corresponding tag
- b) if the token is a stop word, we simply add the subtree "(STOP ('stop-word'))" to "ROOT".
- c) if the token is an English language word, we map the word to its part-of-speech tag, calculate the prior polarity of the word and add the subtree (EW ('POS' 'word' 'prior polarity')) to the "ROOT".
- d) For any other token we add subtree "(NE ())" to the "ROOT". "NE" refers to non-English
- e) For any token which has repeated characters like cool we add a tag EMP (emphasis) to the root

A number of our features are based on prior polarity of words. For this we use SentiWordnet and extend it using WordNet. SentiWordNet is a lexical resource for opinion mining. SentiWordNet assigns to each synset of WordNet three sentiment scores: positivity, negativity, objectivity. If a word is not directly found in the sentiwordnet we retrieve all synonyms from Wordnet. We then look for each of the synonyms in sentiwordnet. If any synonym is found in sentiwordnet, we assign the original word the same pleasantness score as its synonym. If none of the synonyms is present in sentiwordnet, the word is not associated with any prior polarity.

For our linear classification we use Support Vector Machines (SVM) and report averaged 5-fold cross-validation test results

But for Tree kernel Based method we used the free software provided by Alessandro Moschitti (<http://disi.unitn.it/moschitti/Tree-Kernel.htm>)

Evaluation data, metric and experimental Results

Model	Accuracy
Unigram	54.34
Unigram +features described(f1-f11)	59.23
Unigram+features described(f1-f11) Tree Kernel	60.43

We use results from 5 cross validation

```
X_train, X_test, y_train, y_test =  
train_test_split(feature_SVM_Vector['feature_vector'],feature_SVM_Vector['labels'],test_size=0.  
20,random_state=0)
```

CONCLUSION:

We investigated two kinds of models: tree kernel and feature based models and demonstrate that both these models outperform the unigram baseline. For our feature-based approach, we do feature analysis which reveals that the most important features are those that combine the prior polarity of words and their parts-of-speech tags. We tentatively conclude that sentiment analysis for Twitter data is not that different from sentiment analysis for other genres.

REFERENCES

B. Pang and L. Lee. 2004. A sentimental education: Sentiment analysis using subjectivity analysis using subjectivity summarization based on minimum cuts. ACL

Agarwal, Apoorv; Xie, Boyi; Vovsha, Ilia; Rambow, Owen; Passonneau, Rebecca; Sentiment analysis of twitter data, Proceedings of the workshop on languages in social media,,30-38,2011, Association for Computational Linguistics

"Moschitti, Alessandro; ", Making Tree Kernels Practical for Natural Language Learning., EACL, 113, 120, 24, 2006,

Alessandro Moschitti. 2006. Efficient convolution kernels for dependency and constituent syntactic trees. In Proceedings of the 17th European Conference on Machine Learning.

Programming Language Detection

Abstract:

Programming Language Detector is an application extension which can classify the language of the highlighted code snippet and execute it either locally or on an online compiler.

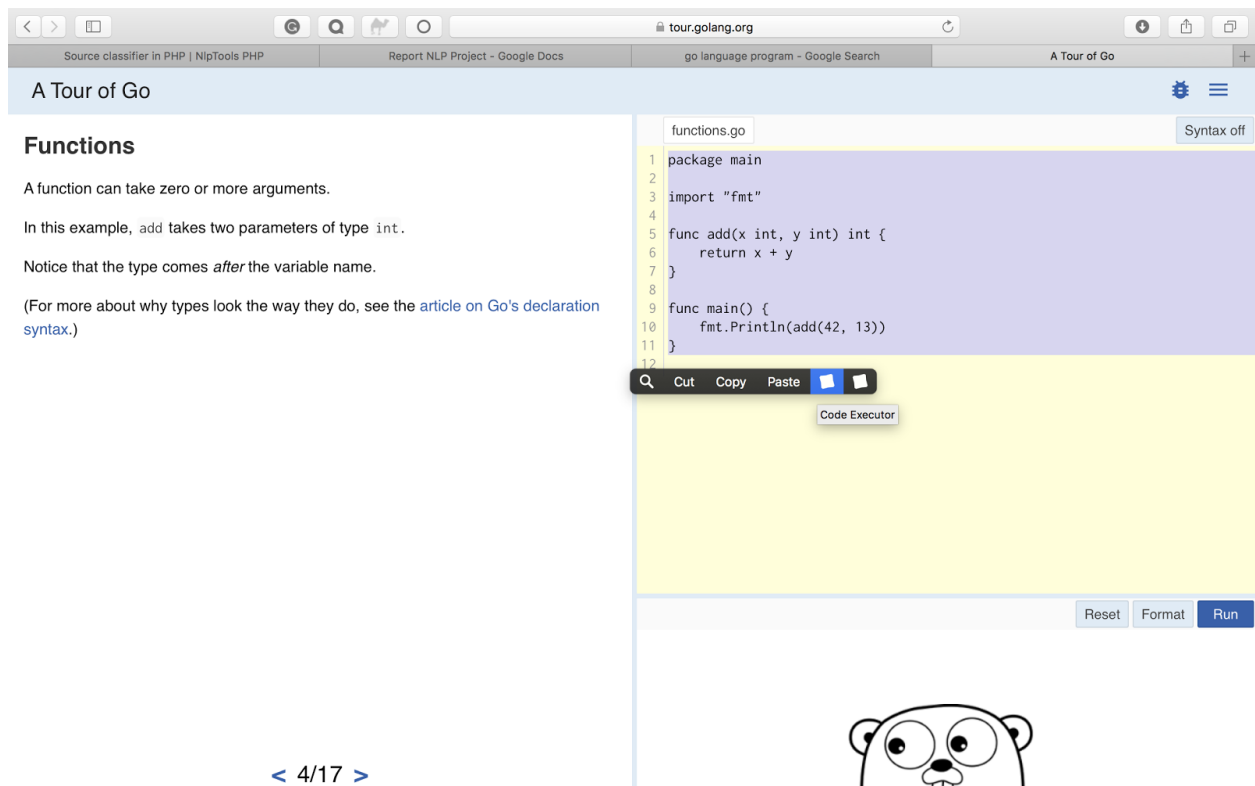
Components:

- a. **PopClip:** PopClip is a popular Mac productivity application that consistently ranks among the top apps in the app store. PopClip gets activated on highlighting a text anywhere on a Mac. It has support to add extensions to add new functionality and these extensions are open source.
- a. **Programming Language Classifier:** Programming Language classifier is an extension to the PopClip app. The classifier uses a probabilistic learning method, namely the multinomial naive bayes classifier. The classifier is trained on a dataset consisting of solutions posted on various online coding competitions.
- a. **Script and XML file:** The script and xml files are responsible for things like storing the highlighted text, executing and running the program based on the programming language detected, etc. It is written using AppleScript programming language.

Methodology:

The highlighted text is received by PopClip and is passed on to the programming language detection module, which runs a multinomial naive bayes classifier. The source is separated into tokens according to whitespace and punctuations, and is trained using the classifier. Frequency is taken as the keyword count and various thresholds we tested. Currently, it is set to 5.

Results:



The screenshot shows a web browser window with the URL `tour.golang.org`. The page title is "A Tour of Go". The left sidebar contains the heading "Functions" and the following text:

A function can take zero or more arguments.

In this example, `add` takes two parameters of type `int`.

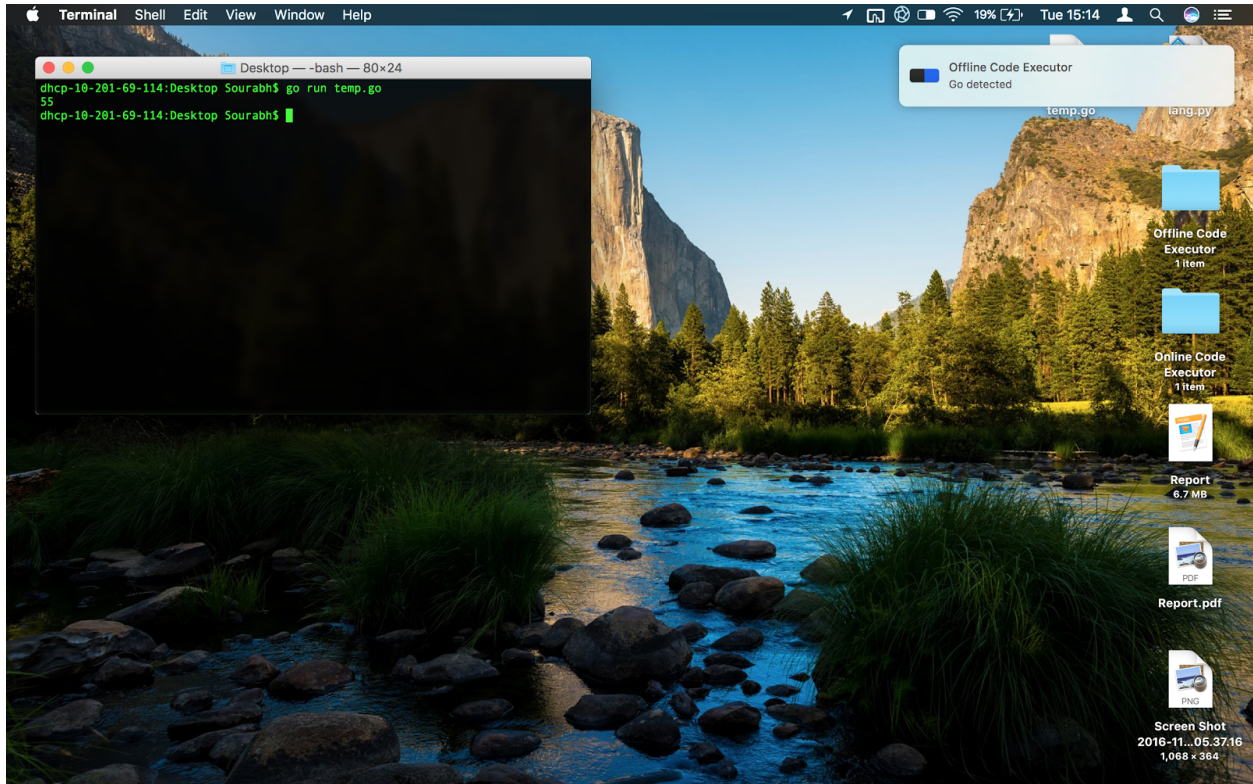
Notice that the type comes *after* the variable name.

(For more about why types look the way they do, see the [article on Go's declaration syntax](#).)

The main content area displays a code editor with the filename `functions.go`. The code is as follows:

```
1 package main
2
3 import "fmt"
4
5 func add(x int, y int) int {
6     return x + y
7 }
8
9 func main() {
10     fmt.Println(add(42, 13))
11 }
12
```

Below the code editor is a "Code Executor" section with buttons for "Reset", "Format", and "Run". At the bottom of the page, there is a navigation bar with the text "< 4/17 >" and a cartoon dog head icon.



Youtube demo of the project: https://www.youtube.com/watch?v=JgJxWGs_B5o

Problems Faced:

- a. Absence of a standard way to determine page load status in Safari. Currently it is being solved by using a delay, but the code can be made more efficient.

Future Work:

- a. The results can be improved by using different features related to the languages such as the kind of parenthesis used. Since we already separate the tokens using whitespace and punctuation tokenizer, we can use the parenthesis token and make better predictions.
- a. Identify code snippets that are placed separately in the page but belong to the same program.

References

[1] NLPTools, <https://github.com/angeloskath/php-nlp-tools/tree/master/src/NlpTools>

[2] <https://github.com/chrislo/sourceclassifier>

