WS Project 2

Sentiment mining on LEGO product reviews

Anonymous Author
Computer Science, University of Copenhagen
DIKU, KU
Copenhagen, Denmark
anonymous@ku.dk

***Abstract*—This document presents two sentiment mining approaches of LEGO product reviews. It will assess the overall performance and accuracy of each method by comparing the outputted results with the "ground truth" that has been provided for our analysis.**

Keywords—web science, sentiment mining, sentiment analysis, natural language processing, machine learning, Core NLP, sentiment classification

I. Introduction

The project, I had to carry out, is to conduct two sentiment mining experiments by using an automatic sentiment classification method and a deep learning method by using a natural-language-processing library named CoreNLP developed at Stanford University.

Sentiment analysis (also known as opinion mining) refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials. Sentiment analysis is widely applied to reviews and social media for a variety of applications, ranging from marketing to customer service. Sentiment analysis aims to determine the attitude of a speaker or a writer with respect to some topic or the overall contextual polarity of a document.

The sentiment analysis would show some insight on how positive, neutral or negative the review is towards the product. And most importantly I will show how efficient these two approaches are by comparing our classifiers with the real world results.

II. Methodology

A. Corpus of data and preprocessing

The data provided was in the form of a .csv file that I had to download from a Google Docs link noted in the footnote[1]. Python 2.7 was the language of choice in preprocessing the data. Each line consisted of the following information: the student's name that contributed in the review, sentiment score (1 for positive, 0 for neutral, -1 for negative), a link to the aforementioned post, and the post's text. The data wasn't fully homogenous and therefore I had to apply some basic preprocessing to remove entries that had incomplete scores. Others just had an ambiguous format (e.g.: "... Seriously? ...0?").

I did not remove the stop words from the text file, because words like "not", "but", "very" etc. will impact the overall sentiment classification. Pairs like "not good" have a negative sentiment value to our review, otherwise we would correlate the single word "good" with a negative impact in our reviews.

B. Automatic sentiment classification

We have been given some out of the box APIs at our disposal for carrying out this task. From the list given, I decided to go with uClassify's solution[2] for sentiment mining. Not much information is given on which type of classifiers models the web service is using, but after some thorough searching through uClassify's website I learned that at it's core is a Naive Bayesian[3] classifier which is a popular (baseline) method for text categorization. It tries to solve the problem of judging documents as belonging to one category or the other with word frequencies as the features.

All naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable.

I use 3-fold validation when testing our data set. Each of the 3981 review have been split into 3 separate sets. I used 2 parts for training the classifier and the remaining set is used for testing, on 3 different iterations. The data isn't randomized and it might impact the performance of our classifier to certain extent, because the reviews are in chronological order and tied to certain LEGO products. If the product is objectively good overall, most of the reviews might be positive as well. But in order to replicate the experiment I decided not to randomize the 3-fold validation indexes.

After I get the results back from uClassify I compare my results with the entries that have been inputted as ground truth by the students. I will discuss and compare the results in the next chapter.

---

[1] https://docs.google.com/spreadsheets/d/1Bfq2fPRiwwCGS6JgpyrVP4OSyPuhuU_sO77xnFKWSoQ/edit - gid=0

[2] https://www.uclassify.com/browse/uclassify/sentiment

[3] https://www.uclassify.com/docs/technical

C. *Deep learning with CoreNLP*

Working with Stanford's CoreNLP framework proved to be a little tricky due to the lack of online documentation and not a lot of commented code. It has a very detailed set of features but these are not explained in the documentation.

The objective of training is to find a set of network weights such that the summed squared error is minimized. This is done by incrementally changing the weights along the direction of the error gradient with regards to weights.

Weights are computed after presenting all training samples. One such pass through all samples is called an epoch. Before the first epoch, weights are initialized, typically to small random numbers. A variant is incremental learning, where weights are changed after presentation of individual training samples. Before I could do any training for our sentiment miner, I had to preprocess the reviews into machine readable form:

1. To do just that, I split the reviews into sentences, and each sentence has been assigned the overall review score.
2. After that has been done I binarize each sentence into a tree form assigning each term the label of the entire review. This process is done by executing the *BuildTrainingSet.java* class.
3. The CoreNLP application uses 5 labels each scored from 0 to 4 respectively.
    a. 0 = Very Negative
    b. 1 = Negative
    c. 2 = Neutral
    d. 3 = Positive
    e. 4 = Very Positive
4. Each sentiment score for our reviews must be converted by using the following rules:

TABLE I. CONVERSION RULES

	Negative	Neutral	Positive
Current notation	-1	0	1
CoreNLP notation	1	2	3

5. I saved these abstract syntax trees into their respective files and use them to train our opinion miner.
6. I managed to train each sentiment model with the following parameters:
    a. **-epochs 10** – which represents the number of passes the trainer does on the training set
    b. **-trainPath binary_train_X** – file path to the binarized trained data for fold X
    c. **-devPath dev.txt** – we need to increment and decrement our weights, the file is used to test the training data in intermediary steps
    d. **-nthreads 8 –** a performance improvement for using the maximum amount of threads supported on the host machine
    e. **-model sentiment_model_0.ser.gz –** output file for our sentiment model.

D. *Setting up the CoreNLP pipeline*

After I have trained the 3 sentiment models, I took in each test set and split it up in chunks of reviews. Each chunk consisted of the overall review score and the review text separated by new lines.

I set up the pipeline with the following properties:

```
Properties props = new Properties();
props.put("annotators", "tokenize, ssplit, pos, lemma, ner,
parse, dcoref, sentiment");
if (sentimentModel != null) {
    props.setProperty("sentiment.model", sentimentModel);
}
```

and instantiate each review as an annotation. An annotation can contain multiple sentences and I analyze each sentence's sentiment score. Because I need the overall review score, I create a zero vector of 5. This vector represents how many *very negative, negative, neutral, positive and very positive* sentences the review has.

I assume that a *very negative and very positive sentence* has double the weight of their sentiment. For example, one *very negative* sentence is treated as two *negative* independent sentences.

Also I assume that the *very negative* and *very positive* cancel themselves out. The same goes for the *positive and negative* sentences. The neutral sentence score does not affect the overall sentiment score. The overall review score is averaged out by the number of sentences in each review. After I have the average review score, I assign a label that we can compare to the "ground truth" in the test set. The following piece of code is the direct snippet from the *LEGOClassifier.java* class:

```
// calculate the overall review score
float avgReviewScore = (scoreVector[0] * (-2) + scoreVector[1] * (-1) +
        scoreVector[2] * (0) +
        scoreVector[3] * (1)+ scoreVector[4] * (2)) / sentenceCount;

if (avgReviewScore <= -0.5f){
    computedReviewScore = 1;
} else if (avgReviewScore > -0.5f && avgReviewScore < 0.5f){
    computedReviewScore = 2;
} else {
    computedReviewScore = 3;
}
```

E. *Issues along the way*

The problem is that when dealing with 18600 trees even a small parsing issues can destroy our sentiment trainer. And that is exactly what happened. I tried looking at some options to resolve the issue at hand.

One way of handling the error is by striping away all unnecessary special characters such as non UTF-8 characters that might impact the generation of the binarized dataset. But even then, we are not certain that this is causing the issue.

Furthermore, striping away symbols like ":-D", ":-)" etc. will impact our opinion predictor's decision making and might not be the issue on why the training set could not be completed. The only thing I know for sure is that the problem is in the first binarized set.

It turned out that the problem was generated by the word "Compare" in the last part of the first part of the training set of the first fold. I removed the appropriate reviews and managed to train the other two folds.

Because we had to construct our own dataset I tried labeling the sentences with the overall review score. This means that each leaf node and sub tree would have the same review score. In the end, this method would produce poor results because there is difference between the terms "not good" and "good", for example.

Due to the lack of time, I was unable to carry out constructing the dev trees, because this has to be done manually. Instead I decided to use the default *dev trees* provided by the CoreNLP framework. Event thought the dev trees file contains values higher and lower than ours (*0 for very negative, 4 for very positive*), this does not significantly affect our training model**.** A more robust solution might be applying for Amazon's Mechanical Turk service[4], but this in return adds additional fees to this project.

III. Findings

I compared the outputted value with the "ground truth" to test the accuracy of our classifiers. For each correct entry I increased a counter and after iterating through the test set I divided it by total number of reviews in the test set.

A. Automatic sentiment classification

The accuracy for my Naïve Bayesian classifier can be seen in Table 1:

TABLE II. Naïve Bayesian Classifier Results

	Fold 1	Fold 2	Fold 3	Average
Accuracy	0.562	0.587	0.589	0.579

As we can see, this classifier has an average accuracy of 0.579 out of 1 (1 being 100% accurate). As a comparison, humans are can only be roughly 80% accurate[5] [1] in providing a clear answer on the opinion of a sentence. This inaccuracy of human assessment defines the term "good enough" accuracy for opinion mining.

I suspect that this classifier would do better if I had more time to experiment with:

- removing unnecessary stop words like "the", "and", "it" etc.
- removing punctuation marks, because I don't know if these punctuation marks are treated by the online classifier.
- removing words that are not in the English dictionary. (e.g. usernames, dates, e-mails etc.)
- building a set of words with typos (because people might make some mistake in writing the reviews); this will also add a lot of overhead so I assumed that the word are written correctly.

B. Deep learning method using Recurrent Neural Networks

After conducting the 3-fold cross validation on the CoreNLP framework, we get the following results:

TABLE III. Recurrent Neural Network (CoreNLP) Results

	Fold 1	Fold 2	Fold 3	Average
Accuracy	0.552	0.562	0.579	0.564

As we can see these results are almost on par with the ones found using the Naïve Bayesian Classifier method. These results show that, without optimization done to the training data (assigning different sentiment labels to phrases, negating phrases, removing non essential stop words, etc.), it can produce the very similar results to the simple Bayesian classifier.

Currently the deep learning method that I had implemented has these following major drawbacks that impact the precision of the classifier significantly:

1. The labels assigned to the words are inconsistent; For example, in some situations a phrase like "worst set ever" will have both positive and negative labels depending on the context. This is due to the fact that I'm assigning all tree branches the same label as the entire review, making the weight discovery inconsistent.
2. During the training phase I don't assign an optimized "dev tree" file with real sentiment scores and phrases for LEGO reviews. Instead, I use the "general" dev trees provided by the framework that might not contain specific LEGO product words or phrases.
3. The phrases are not negated. For example, having the phrase "would recommend it" and "would not recommend it" might have the same sentiment score, because it only depends on the entire review label.
4. Because I have a lot of stop words, these would produce an effect on the results as well. Stop words like "the", "to", "and" etc. contribute to the overall score, which is not desirable.

[4] https://requester.mturk.com/create/sentiment/about

[5] http://www.informationweek.com/software/information-management/expert-analysis-is-sentiment-analysis-an-80--solution/d/d-id/1087919?

IV. Conclusion

The scope of the second project was to compare the accuracy of different sentiment mining methods. From the experiments that I had carried out, I concluded that these two approaches are similar. From analyzing the results, it's safe to assume that the data that I used to train these to methods can be significantly improved, which in turn will improve the performance of both classifiers.

Also the documentation for the CoreNLP leaves much to be desired and can be improved significantly. Too much time was wasted on figuring out how to start using the software.

V. References

[1] W. &. H. Wilson, "Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis", Univ. of Pittsburgh, Pitsburgh, 2005.