UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des Sciences
Département d'Informatique

Natural Language Processing
Assignment 3

Chatri Alaaedine

Année académique 2016 - 2017

Contents

1 Introduction

In this assignment, we will implement our own sentiment analyzer to predict the polarity(positive/negative) of the sentiment that is expressed by individual consumers in their online reviews of products they bought.Reviews consist of a short evaluative text and a product score(1 to 5) that are assumed to express the same underlying sentiment. Our sentiment analyzer will learn to predict the score of the reviewer from the written text, so that afterwards,the system can also detect the sentiment in reviews without explicit scores.

There are 4 parts in this project:

1. Part 1: Weight of this assignment
2. Part 2: Selected Data Set
3. Part 3: Design choice
4. Part 4: Discussion about results

2 Part1 - Weight of this assignment

This project took me a lot of time but it came during the exam period so I could not get involved as I wanted. Yet the material is super interesting and the fact of having total freedom in the choice of implementation and design allows us to explore more possibilities.

So I chose to put a weight of 2 on this project because I am not satisfied with the work delivered.

3 Part 2 - Selected Data Set

For his research, Dr. Julian McAuleyat the University of California (San Diego) has collected product reviews from Amazon, covering 24 different product categories. He has publicly released samples of the datasets on his website.[1]

3.1 Sample review


```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "helpful": [2, 3],
  "reviewText": "I bought this for my husband who plays the piano.
  He is having a wonderful time playing these old hymns.
  The music  is at times hard to read because we think the book was published
  for singing from more than playing from.
  Great purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}
```


Figure 1: Sample of one review

Where

- **reviewerID** - ID of the reviewer
- **asin** - ID of the product
- **reviewerName** - name of the reviewer
- **helpful** - helpfulness rating of the review
- **reviewText** - text of the review
- **overall** - rating of the product
- **summary** - summary of the review
- **unixReviewTime** - time of the review (unix time)
- **reviewTime** - time of the review (raw)

[1] http://jmcauley.ucsd.edu/data/amazon/

3.2 My choice

I chose as product category: Video games.
My training set and my dev set have 10000 lines and my test set has 3000 lines.

4 Part3 - Design Choice

4.1 Implementinga probabilistic sentiment analyser

In this part, there will be the choices and the implementation of the project.

4.1.1 Pre-Pocessing

Score and Text: We have in our sample 2 points to extract: text and overall.(see fig.1)
I put the overalls in 3 sentiment categories:

- positive - overall equals 4 or 5
- neutral - overall equals 3
- negative - overall equals

I chose these categories because when I ran through my sample, I noticed that the overall ranged from 1 to 5 without decimals.
For the text, I handle the problem of negation and use the tokenize and POS tags.

4.1.2 Training

We have to choose a supervised machine learning algorithm we would like to use for training our sentiment analysis.
I have chosen the Naïve Bayes classifier we discussed in class. I didn't implement the algorithm, I took a publicly available python library: **NLTK**.
We have to Use the training to train and fine-tune at least 2 different models to predict sentiment categories from text.

1. Model 1 use unigram as features
2. Model 2 use features derived from a sentiment lexicon (**SentiWord**)

Model 1 & 2. For the 2 models, I transformed each text into words in order to work on it because the classifier waits for a dictionary of words.
For model 1, after retrieving the words, the words are put in lowercase and the negation is handled. And before creating the dictionary, the punctuations are removed too. For model 2, word retrieval and lowercase is the same as model 1. Negation is handled later.

- Model 1: The dictionary of type $< key, value >$ with key equal to (Word: Boolean) and the value is equal to one of the categories of feeling. The Boolean value is true, to avoid duplicates in the dictionary. This dictionary that represents the training set is passed the algorithm of Naïve Bayes. This algorithm will return a classifier that will be used for the test file. After launching the test with the classifier, accuracy, precision, recall and f-measure are calculated.

- Model 2: I used the SentiWord API. This API works in this way[2]:

```
>>> breakdown = swn.senti_synset('breakdown.n.03')
>>> print(breakdown)
<breakdown.n.03: PosScore=0.0 NegScore=0.25>
>>> breakdown.pos_score()
0.0
>>> breakdown.neg_score()
0.25
>>> breakdown.obj_score()
0.75
```

  You give it a word he returns you a positive score and a negative score for the word. But for the same word there may be several possible answers depending on the tag of the word.
  There are therefore 5 possible tags[2]:

SentiWordNet Tags	POS tags
n - NOUN	JJ, JJR, JJS
v - VERB	NN, NNS, NNP, NNPS
a - ADJECTIVE	VB, VBD, VBG, VBN, VBP, VBZ
s - ADJECTIVE SATELLITE	/
r - ADVERB	RB, RBR, RBS

Table 1: Table of 5 SentiWordNet Tags

  I retrieve the table of words and the tag of each word. The list of words (by text) will be managed with negation. Depending on the tag and the word the positive and negative score will be recovered. The positive scores for each word of the text will be added together and the same for the negatives. If a word was found with negation (_NEG) the positive and negative scores are exchanged. The dictionary of type $< key, value >$ with key equal to (pos_score,neg_score) and the value is equal to one of the categories of feeling.

[2]http://www.nltk.org/howto/sentiwordnet.html

5 Discussion about results

5.1 Evaluation

For both models the following 4 points are evaluated[1]:

1. Accuracy: The equation for accuracy, which asks what percentage of all the observations (for the spam or pie examples that means all emails or tweets) our system labeled correctly. Although accuracy might seem a natural metric, we generally don't use it. That's because accuracy doesn't work well when the classes are unbalanced (as indeed they are with spam, which is a large majority of email, or with tweets, which are mainly not about pie).

2. Precision: Precision measures the percentage of the items that the system detected (i.e., the system labeled as positive) that are in fact positive (i.e., are positive according to the human gold labels).

$$Precision = \frac{truepositives}{truepositives + falsepositives} \tag{1}$$

3. Recall: Recall measures the percentage of items actually present in the input that were correctly identified by the system. Recall is defined as

$$Recall = \frac{truepositives}{truepositives + falsenegative} \tag{2}$$

4. F-measure:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{3}$$

The β parameter differentially weights the importance of recall and precision, based perhaps on the needs of an application. Values of $\beta > 1$ favor recall, while values of $\beta < 1$ favor precision. When $\beta = 1$, precision and recall are equally balanced; this is the most frequently used metric, and is called $F_\beta = 1$ or just F_1:

$$F_1 = \frac{2PR}{P + R} \tag{4}$$

5.2 My results

Discussion. The values are not really representative because:

- In the sample, positive overalls are more represented. I tried with a more balanced file and I get better results.

- We take into account the neutral values, whereas in a large majority of the examples seen there are only the positive and negative values. I also tried by removing the neutral values and the results increase.

- I used unigram. With a bigram or trigram model I get bigger results.

```
--MODEL UNIGRAM--
Accuracy Naive Bayes for Unigram Model :  0.399
Precision    Pos: 0.931973 - Neg: 0.421053
Recall   Pos: 0.324953 - Neg: 0.377528
F-Mesure Pos: 0.481885 - Neg: 0.398104
--Lexicon Model--
Accuracy Naive Bayes for Lexicon Model :  0.681
Precision    Pos: 0.711245 - Neg: 0.202614
Recall   Pos: 0.951139 - Neg: 0.069663
F-Mesure Pos: 0.813883 - Neg: 0.103679
```

Figure 2: Screen of evaluation

With the first tests I also had results that were not very conclusive, I modified some points to improve them:

- Use bigram and trigram instead of unigram
- Eliminate Numerals, Punctuation
- Use lowercase

5.3 Comparison between models

As said in the section 5.1, the accuracy does not give a good representation when the training files are not of "good quality", in my case I have a set of training that is not balanced (more d Overall positive). However, it can be observed in fig.2 that there is a better precision in the model 1. The time of execution is much longer for the model 2. Because it is necessary to arrange the training and the test set to make the classification. You also need to find the tag and get it from the SentiWord API.

5.4 Error Analysis

Line	obs	pre1	pre2
2	pos	neutre	pos
3	pos	neutre	pos
4	pos	neutre	pos
23	neg	neutre	pos
28	pos	neutre	pos
29	pos	neutre	pos
80	neutre	neutre	neg
81	neutre	neutre	pos
123	neg	neg	pos
161	neg	neg	pos

Figure 3: 10 misclassifications

There are several reasons for having an incorrect classification:

- Not enough training
- Ambiguity on words
- Choose the right tags
- Unigram does not give enough information

5.5 Improvement

- Try a Different Classification Algorithm
- Reduce words through Lemma Reduction
- Diversify our Training set (and more data's)
- Eliminate Low Quality Features

References

[1] James H. Martin Dan Jurafsky. *Speech and Language Processing (3rd ed. draft).* Springer Science and Business Media, 2017.

[2] Jyotsna Kumar Mandal Durga Prasad Mohapatra Lakhmi C. Jain, H.S. Behera. *Computational Intelligence in Data Mining - Volume 1.* Springer India, 2015.