NAME: Jérôme Charrière
Student ID: 260123

Sentiment Analyzer

Table of content:

1.Presentation:

1. Project description

This project implements a sentiment analyzer capable of automatically evaluating the emotional polarities (good/bad) of a serie of comments, contained within a text file.

2. Usage

Once in the project directory, to interact with the sentiment analyzer, it is necessary to call the python script '`main.py`'

1) Building the XML corpus.

Before doing anything related to sentiment analysis, we first need to build the XML corpus from the text document file.

The python script '`main.py`' must then be called with the option '`-buildXML`', just as follows:

```
python main.py -buildXML
```

2) Sentiment analysis

After the creation of the XML corpus we then can get to the evaluation of the product comments.

To do so, the python script '`main.py`' must be called followed by the option '`-analyzeXML`'.

Without providing any other options, the script is going to evaluate the emotional polarity of each of the comments, and output the global level of accuracy of the results produced (the number of correct guesses over the total count of comments).

It is however possible to provide other options (following the `-analyzeXML` option) in order to observe some of the data produced by the sentiment analyzer:

- If we add the option '`-top20`', the script will output the first twenty words holding the highest good/bad scores:

Example:

```
python main.py -analyzeXML -top20
```

- The option `-get-lexicon` will output the entire scored lexicon built during the unsupervised learning procedure.

  Example:

  ```
  python main.py -analyzeXML -get-lexicon
  ```

- The option `-get-infos` will output relative informations regarding a comment whose id will have to be supplied as argument to the option.

  The output of the command will display:
  _ the comment's content.
  _ the comment's computed emotional polarity score.
  _ the comment's referential (hand-gathered) sentiment value (see section 2.1).

  Example:

  ```
  python main.py -analyzeXML -get-infos 2
  ```

- Finally, the option `-get-all-infos` will do the same operation as the one stated before. Except that here, the program will output informations regarding all the comments inside the XML corpus.

  Example:

  ```
  python main.py -analyzeXML -get-all-infos
  ```

2. Technical documentation:

1. The referential text document file.

The textual corpus, named “comments” in the project directory, is supposed to contain all of the product reviews that we want the sentiment analyzer to evaluate.

The referential text document file must follow a certain format where each line must contain two elements regarding a comment made on a product:

- A character that can be either 'p' or 'n' which is a 'hand-inputted' data describing the overall sentiment of the comment (positive or negative). (IMPORTANT: This data is only here to help setting up statistics regarding the output of the engineered solution and does not intervene with the sentiment analysis).

- The comment itself, in textual format, without any newline characters.

2. Building the XML (structured) corpus.

To proceed to the emotional evaluation, all of the data contained within the text corpus document file will have to be preprocessed and introduced to an XML file.

The structure of the XML file is as follow:

For each of the comments inside the textual corpus, a main node will be added to the XML file.
Within that node, multiple other nodes will be added in order to document certain properties regarding the comment. Most importantly, one of the nodes will contain the list of the selected words that has been returned by the pre-processing layer, described in the next section.

3. Tokenization & Pre-processing.

In order to ease the sentiment analysis process, it is necessary to proceed to a selection and pre-processing of the comments.

During this process, the pre-processing layer will take care of breaking each comments into a set of carefully selected words. This process is done rather easily as the software uses the Python NLTK module for both tokenizing the input and setting up the selection. Indeed, the NLTK module is able to break a chunk of text into a stream of words paired with their grammatical types. Those values associated with the words are relevant as grammar related informations happen to precisely be the type of data that we wish to base the selection on.

We then only have to rely on the values returned by the NLTK module to filter out irrelevant words. In consequence, only words belonging to certain grammatical types will be taken account of. We choose to eliminate data which we know for sure won't hold any significant informations, and might even add a level of noise to the analysis, like pronouns or conjunctions.

The pre-processing stage also try take care of one of the peculiarities of the language: the negation.
The sentiment analyzer is going to analyze each words independently of the context in which they appear in. We then take the risk of jeopardizing results as negations can completely switch the semantic orientation of a word.

As a way to respond to this issue, the software will attempt to search for negation operators in the provided comment. When one is found, then every tokens that are part of a certain range will be marked as to be interpreted as 'negated'.
This interval is special, it's limits are defined with the identified negation operator and a clausal form.
While imperfect, this method can correctly tackle cases like: ”This product isn't even great or interesting.”.

4. The sentiment analysis.

Mainly, the sentiment analyzer make use of an algorithm called SO-LSA[1] (Semantic Orientation - Latent Semantic Analysis).

The process of the algorithm can be described as follow:

First, each of the tokens inside the XML corpus is going to be inputted to the sentiment

analyzer algorithm. Based on that data, the algorithm is going to construct a matrix where each row will represent a word and each column a product review. The data at position X_{ij} in the matrix will state the weight of a word in a specific comment.

The weight of a word in a comment is given by computing the tf-idf score (term frequency – inverse document frequency). This value will indicate how significant the word is in the comment by taking into account it's number of occurrences compared to the total count of words in the comment as well as giving a bonus if the word is not featured in many comments overall.

As a second step, after the matrix built, the algorithm is going to review all of the words that it has seen so far in order to assign a score to each of them.

This score will give out the semantic orientation of a certain word. This will indicate if a word tends to show positive or negative emotions as well as giving out it's strength of belief (the scale). To assign such score, the program relies on two small predesigned sets holding words that can most likely be respectively found in comments of either very negative or very positive nature.

With these two sets, a word's semantic orientation can be calculated by measuring the similarity levels between itself and the two predefined word sets:

$$SO(w) = \sum_{w' \in S_1} cosim(w, w') - \sum_{w' \in S_2} cosim(w, w')$$

The sets S_1 and S_2 refers to the row vectors (retrieved from the matrix) identified as being words part of the predefined word sets of respectively highly negative and highly positive nature. w is the row vector (retrieved from the matrix) that refers to the target word that we would like to determine the semantic orientation value of.

As for the calculation of the similarity levels: the SO() function make use of another function called cosim(). This latter function can compute the similarity value by measuring how often a word appeared in the same context of one of the word in one of the predefined set:

$$cosim(A, B) = \frac{\sum_{i=1}^{n} (A_i * B_i)}{\sqrt{\sum_{i=1}^{n} A_i^2} * \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Now, after the assignment of the semantic orientation scores it is then possible to proceed the evaluation of all the comments inside the XML corpus.

This process is rather simple:
To determine the emotional polarity of a comment, the program only have to calculate the score average of all the comment's scored tokenized words.

3. Demonstrations:

1. Data used.

The textual corpus associated (by default) with this project is made up of about 730 user reviews made on the video game « Call of Duty: Black Ops II ». All of those reviews were gathered directly from the website www.amazon.com.

As for the referential word sets, it included the following words:

Positive word set:
$S_1 = \{\text{good}, \text{great}, \text{amazing}, \text{love}\}$

Negative word set:
$S_2 = \{\text{bad}, \text{horrible}, \text{worst}, \text{hate}\}$

2. Results.

After building up the XML file (with the command 'main.py -buildXML'), we can then proceed to the sentiment analysis part:

```
commode@commode-VirtualBox:~/Downloads/SentimentAnalyzer$ python main.py
-analyzeXML
--------------------------------
building lexicon..
computing semantic orientation scores..
evaluating comments..
accuarcy: 79.15%
correct guesses: 577
total number of comments: 729
```

Here the program outputted statistics related to the number of times the sentiment analyzer was found to have made correct guesses.

We can see that we obtained a fairly good success rate as the program was able to make the correct evaluation of about 80% of all the comments.

3. Analysis of the lexicon built.

Now, let us launch the program with the option -analyzeXML -top20 in order to examine the semantic orientation scores assigned to each of the words by the SO-LSA algorithm:

```
commode@commode-VirtualBox:~/Downloads/SentimentAnalyzer$ python main.py
-analyzeXML -top20
--------------------------------
building lexicon..
computing semantic orientation scores..
best positive words:

         1.11 - great
```

```
         1.07 - love
         1.02 - amazing
         1.00 - good
         0.81 - recomend
         0.81 - incredible.
         0.81 - compelling
         0.50 - story
         0.44 - boyz
         0.43 - game
         0.39 - son
         0.36 - Worked
         0.32 - was
         0.32 - Very
         0.31 - packed
         0.31 - everybody.
         0.31 - Intense
         0.31 - dramatic
         0.31 - condition
         0.30 - say

best negative words:

        -1.16 - worst
        -1.14 - horrible
        -1.05 - hate
        -1.03 - bad
        -0.62 - Going
        -0.62 - product.
        -0.62 - thinks
        -0.56 - made.
        -0.56 - blops
        -0.56 - buy._neg
        -0.56 - support
        -0.55 - crap
        -0.50 - repeat
        -0.50 - kinda
        -0.49 - Dumb
        -0.49 - comment
        -0.49 - require
        -0.48 - beginning
        -0.48 - dropped
        -0.48 - sucked
```

Here, the program outputted the 20 best negative and the 20 best positive words identified during the training.

We can see that the program was able to assign high positive/negative scores to words that do seem likely to be found in reviews showing either strong appraisal or strong criticism. As an example, we can see that words such as “incredible”, “compelling” and “dramatic” were given high positive scores while words such as “crap”, “dumb” and “sucked” were given high negative scores.

But despite that fact, we can also see that a significant number of words, that do not seem to hold any significant information in terms of emotional value, were assigned high positive or high negative scores. This is one of the drawbacks of the method used: here, the algorithm 'wrongfully' assigned high scores to those words because they tended to

strongly appear in the same contexts as the words inside one of the two predefined word set.

As a personal note, I've noticed that this effect tended to be reduced as the size of the corpus grew.
The accuracy of the program would then be significantly improved with a corpus of a larger scale.

4. Bibliography:

[1] Turney, Peter D. and Michael L. Littman. 2003. Measuring praise and criticism: inference of semantic orientation from association. ACM Transactions on Information Systems 21: 315-346.