Computational Linguistics and Analysis

Sentiment Analysis using SPR

Ganeshram Iyer

Motivation

The motivation behind this project is to make the use of syntactic rules to better convey the underlying sentiment in a sentence. There are mainly two methods for sentiment analysis, using natural language processing techniques and using machine learning techniques. This approach uses the natural language processing technique and incorporates dependency parse trees into sentiment analysis. Dependency parse trees help us identify the structure of the sentence along with the dependency relations between each word. The naive approach to lexicon based approach is to use a sentiment lexicon and calculate the polarity of individual words in a sentence. Having identified the polarity of each sentence we try to gauge the sentiment of the sentence as a whole by summing up individual polarities. As you can see this does not capture any of the important features of an English sentence such as

1. Inversions
2. Modifiers
    - Verbs
    - Prepositions
    - Adverbs
    - Nouns

Each of these features affect the polarity of the word it governs in a different way. Inversions reverse the polarity of the word. This feature is captured under the negation modifier dependency of Stanford Parser. In a similar way the other features too have a Stanford Dependency representation which can be used to identify them and reflect the impact of these modifiers on the polarity of the dependent in an appropriate manner. This way we are using Syntatic Rule Propagation to accurately determine the polarity of individual words.

The main motivation for this approach was obtained from a paper on Sentiment Analysis via Dependency Parsing [1]. The paper discusses the importance of propagating the values of polarity

to the other words based on the dependency tree structure. For example, let us take the case of a adverbially modifier-

The movie was really bad.

In this example the polarity of the word really by itself is 0 and the polarity of the word bad is -0.5. The structure obtained from the dependency parse tree advmod(bad, really). This indicates that the polarity of word should be changed based on the structure. Thus the polarity of really is propagated to bad. Similarly, there are other dependency structure which need the polarity of its constituents to be changed. General structure of a dependency

Type(Governor, Dependent)

For example,

neg(bad, not)

- Type: Negation
- Governor:

Implementation

The code for this project is implemented using Java as the main programming language. A number of API's and supporting documents are used to successfully demonstrate syntactic rule propagation.

1. Specification: The program takes a review as an input and generates the sentiment based on the words and the polarity calculated after applying the syntactic rules. The input is split into sentences and given to a parser which emits a list of dependencies that exists in the sentence. These dependencies are then passed to rule engine which computes the polarity of the key word in the sentence, mostly noun in case of reviews. On applying all the rules on the dependencies we get the final result associated with the key aspect of the sentence. The logic applied on calculating the sentiment of a review can be applied to any product. The output of the program is the sentiment value for each aspect of product mentioned in the review.

The phone is really bad but the camera is surprisingly good.

On passing this sentence to the program the following key output is obtained: -

Sentiment value for phone: -0.5135765997885184
Sentiment value for camera: 0.5703868978414685

As you can see the program captures the structure of the sentence and generates the sentiment for each aspect reviewed by the user. This concept is extremely useful while analyzing the reviews in detail to gain more knowledge of reviews on individual aspects of the product. In this case as we can see the overall review for the product is bad, but the phone has got a really good sentiment rating for its camera which is a positive thing.

Note: - There are other aspects of the input which are displayed but not shown in this output.

The main limitation of this program is that it depends on the polarity of the words to determine the sentiment, it cannot determine the difference on how these words are used.

The laptop is an improvement from its previous version.
The laptop needs an improvement to its keyboard.

Both these reviews will be considered as positive by the program. Also the process of identifying aspects is not efficient, all nouns in the sentence are considered as aspects. One way to mitigate the same is to compare the nouns with a list of aspects for which reviews are needed. This way only those nouns which are aspects get shown up in the final sentiment display.

2. Description: The program is split up to different classes for different functions. The Stanford Parser API's are used to generate the list of TypedDepency objects. Each TypedDependency object consists of the following elements –
    a. Type of dependency
    b. Governor
    c. Dependent

Along with these elements it also stores the part of speech for each of its constituents. In this way we obtain the part of speech tags of words in the sentence. It is essential to have these part of speech tags as it plays an important role in determining the polarity of the words. Polarity of the words is stored in a file called SentiWordnet which stores the values

in structured manner. The polarity is mapped to the word and the part of speech. The parser needs to be loaded from a a serialized file. This file is different for different languages and thus we use the English version of the file.

edu/stanford/nlp/models/lexparser/englishPCFG.ser.gz

Along with that to successfully run the program we also used slf4j API's as a logging interface used by our parser.

Java Classes: -

All the classes are under a package named SentimentAnalysis.

a. DependencyTree
   This class takes the sentence as an input and generates the dependency tree for the same using the functions given in Stanford Parser. The main function inside this class which does this operation is called generateTree. generateTree takes a String as an input and does the following operations to get a list of TypedDependency for the sentence.
   - Tokenize the sentence
   - Generate a list of CoreLabel
   - Use apply function of the LexicalizedParser to generate a Tree
   - Generate TreebankLanguage pack from the Tree
   - Obtain GrammaticalStructure from the Language pack
   - Obtain list of TypeDependency from the Grammatical Structure

   The generateTree function is called for each sentence and the output is fed to identifyRules function of SytnaticEngine class.

b. SyntaticEngine
   The class where syntactic rules are applied on the sentence. This class has two functions, a function to generate dependencies for the sentence by calling the generateTree function and another to apply the rules on these dependencies and generate the sentiment value for the aspects.

   run – This function seperates the input based on full stops and fees to the generateTree function. The list obtained is then given to the identifyRules function of the same class to apply the rules.

identifyRules – This function does the actual job of Syntatic Rule Propagation. All the dependencies are given to the function as a form of list. The function does the following operations

- Store the words in a map. Key – Word with tag Value – Polarity
- Iterate through the list to find the already defined dependencies and change the polarity of the words accordingly
- Iterate through the map to find nouns and display their polarity values as sentiment of the aspect.

The key part in this operation is to define the dependencies which require us to change the polarity of either the governor or the dependent. In this program we have defined certain rules and the way the polarity gets changed.

Function names:

- Inversion
  - Invert the polarity of the word
  - value of governor gets changed to either 1 – original value or original value-1 based on whether the word was positive or negative.
- Tuner
  - Adverbial modifiers which by themselves do not have any polarity but intensify its governors
  - value = value *(1+coefficent)
  - For all cases the value of the coefficient is arbitrarily determined. We can use statistics to determine and change these values as time goes. But at present this is not implemented.
- Modifier
  - Modifiers, Verbs and Prepositions which affect the polarity of its governor
  - value = value + (coefficient *polarity of dependent)
- DirectObject
  - Cases where there are direct objects in the sentence
  - The polarity of the dependent changes in the following way
  - value = polarity of governor
- NominalSubject
  - Cases dealing with nominal subjects are present as dependencies
  - The value of the dependent changes in the following way
  - value = polarity of depenent * coefficient;

c. WordPolarity – This class is used to calculate the polarity of a word by parsing through SetniWordNet document. The extract function takes the Indexed word as an input converts it into word and its tag and the parses through the list of words to generate its polarity
d. SentimentMain – Used to call the run function for the String input.

3. Conclusion: The program and the rules identify and propagate polarities for the common cases. The sentiments calculated are for different aspect of the product captured in reviews. The output obtained from the program is as follows:
   - Dependency List
   - Polarity of words before applying rules
   - Polarity of words after applying rules
   - Polarity of aspects

   However, the sentiment is highly reliant on the polarity of the word and the structure rather than the semantics of the sentence. This an area which needs further research and change to the approach used to determine the sentiment of the reviews. The process of determining coefficients should depend on the intensity of the word. Thus there can be different coefficients for different words. This way the output will be more accurate.

   As a future work Input can be given to the run function by reading a json file which contains reviews from a well known website. This well help us evaluate the performance of the engine using real reviews.

References:

http://www.academia.edu/2559547/Sentiment_analysis_via_dependency_parsing
http://www.sciencedirect.com/science/article/pii/S2090447914000550
http://nlp.stanford.edu/software/dependencies_manual.pdf