ANLP-MAI

Final project

negDetection

Author:

Andrés Lamilla

18-05-2016

Problem:

NegEx is an algorithm that identify negatives in textual medical records. This reports are critical for process patient information and they should be treated with extreme precaution.

The idea of this project is to improve the performance of NegEx program.

Files:

I am using negEx as a base for my implementation. This library is found in the **lib** folder.
The principal files used in this project are:
negDetection.py : This is the main file and execute functions or class from negDetection_classes or negDetection_functions. In this file I load the data file and split it in training and test sets. Then I train the classifier and get the results of the test set.
negDetection_classes.py : In this file I have all the implementation of the negDetection system. The tagger and the classifier lives here.
negDetection_functions.py : Here I have simple functions that read the rules file, the data file and print the results in the display.
Annotations-1-120.txt : File with data for training and test.
negex_triggers.txt : File with all the rules needed by negEx.

Description:

Basically I saw this problem as a classification problem. What I did was to use the tags provided by the NegEx program to create features to train a Naive Bayes Classifier. My intuition was that the negation terms and their position relative to the evaluated phrase can predict if the phrase was negated or not. Besides of the used tags in NegEx I add some others tags.

The tags used in negEx are:

+ [**PREN**] - Prenegation rule tag
+ [**POST**] - Postnegation rule tag
+ [**PREP**] - Pre possible negation tag
+ [**POSP**] - Post possible negation tag
+ [**PSEU**] - Pseudo negation tag
+ [**CONJ**] - Conjunction tag
+ [**PHRASE**] - Term is rcognized from the term list

My additional tags were:

- **POINT**: This is a tag for '.'
- **COMMA**: tag for ','
- **OR**: tag for 'or'
- **AND**: tag for 'and'
- **WORDS**: tag for all the words that hasn't a previous tag

I removed all the **WORDS** tags that were contiguous. So for example if I have a tagged sentence like this: *WORDS WORDS WORDS COMMA WORDS PHRASE POINT*
I converted it to *WORDS COMMA WORDS PHRASE POINT*
I did this because It think that multiple consecutive **WORDS** doesn't have additional information but they can add noise to some of the used features.

The features that I used were the followings.

- Feature1: The previous and next tag to the PHRASE
- Feature2: The two previous and next tags to the PHRASE
- Feature3: The three previous and next tags to the PHRASE
- Feature4: If there was a PREN tag before PHRASE and there wasn't a POINT between them.

Then when I have this features I just use Naive Bayes to find the most probable class.

So for example if I have a sentence "*No murmurs, GALLOPS, or rubs.*" the tagger process gives me a list like this ["PREN", "WORDS", "COMMA", "PHRASE", "COMMA", "OR", "WORDS", "POINT"] and then I can extract all the features from it. The obtained features would be:

- Feature1_pre: "COMMA"
- Feature1_next: "COMMA"
- Feature2_pre: "COMMA_WORDS"
- Feature2_next: "COMMA_OR"
- Feature3_pre: "COMMA_WORDS_PREN"
- Feature3_next: "COMMA_OR_WORDS"
- Feature4: True

Then I just have to find the probability that the PHRASE is negated using this features and all the training examples, and then return the class with the maximum probability.

I decide to use a classifier because it looks to be the most intuitive way to get a good performance using the negEx library without adding a lot of rules and without changing anything in the library.

Results:

I run the program to obtained the results of the training and test sets and obtained the following results (Results are the results obtained using the naive bayes classifier and NegEx results are the one obtained with negEx):

Train results

- Total examples: 2115
- Results:
    - Correct: 2071

    - Negative Correct: 426
    - Negative Incorrect: 20
    - Positive Correct: 1645
    - Positive Incorrect: 24
- NegEx results:
    - Correct: 2056
    - Negative Correct: 406
    - Negative Incorrect: 40
    - Positive Correct: 1650
    - Positive Incorrect: 19

Test results

NegEx

- Total examples: 235
- Results:
    - Correct: 229
    - Negative Correct: 42
    - Negative Incorrect: 2
    - Positive Correct: 187
    - Positive Incorrect: 4

Feature 1

- Total examples: 235
- Results:
    - Correct: 224
    - Negative Correct: 35
    - Negative Incorrect: 9
    - Positive Correct: 189
    - Positive Incorrect: 2

Features 1,2

- Total examples: 235
- Results:
    - Correct: 229
    - Negative Correct: 42
    - Negative Incorrect: 2
    - Positive Correct: 187

    - Positive Incorrect: 4

Features 1,2,3

- Total examples: 235
- Results:
    - Correct: 230
    - Negative Correct: 43
    - Negative Incorrect: 1
    - Positive Correct: 187
    - Positive Incorrect: 4

Features 1,2,3,4

- Total examples: 235
- Results:
    - Correct: 232
    - Negative Correct: 44
    - Negative Incorrect: 0
    - Positive Correct: 188
    - Positive Incorrect: 3

Feature 4

- Total examples: 235
- Results:
    - Correct: 230
    - Negative Correct: 43
    - Negative Incorrect: 1
    - Positive Correct: 187
    - Positive Incorrect: 4

The accuracy obtained in the training set were:
negEx: .9721
classifier (all features): .9791

The accuracy obtained in the test set were:
negEx: .9744
classifier (all features): .9872

Improvements:

The current performance it is pretty good but I think that it could be improved in many ways. For example adding more features. In the results I saw that every one of the features improve a little the performance of the program, so thinking in add more features looks to be a good way to improve the overall performance.
I am using **COMMA**, **OR** and **AND** tags in this program because I thought that they could add some information to the classifier but I didn’t check this completely. Maybe the classifier could be improved adding manually more useful tags or removing some useless tags.