NLP Final Project
Professor: Adam Meyers
Team: Arjun Madgavkar, Han Zhang, Jaeduk Choi, Nicholas Ang, Ricardo Nunes
Project: Information Retrieval With Sentiments

Information Retrieval/Sentiment Weights in Emotional-Solution Entries

Abstract/Purpose:

This project intends to perform information retrieval on emotional text entries. Emotional entries are paragraphs of text which describes an individual's state of mind. These state of minds include depression, loss, anger, frustration etc.. The program match's an emotional state entry to a emotional solution which aims at providing emotional relief, comfort and advice for that particular state of mind. Our approach combines a search component based on TF-IDF matching with the subsequent use of emotional sentiment weights to find the best matching pair of entries. Our experiments are on a dataset which we have manually extracted from online forums and posts.

Data:

Our project uses data which we have manually gathered from online databases and websites. These online resources are typically forums where individuals express their current emotional state and discussions take place or advice given. We have manually scraped these websites and saved an individual's emotional entry as a "problem" in our *entries.txt* file. Conversely, we have saved responses to these emotional problems as "solutions" in our *response.txt* file.

Entries.txt* -> *Text file consisting of saved emotional entries from online forums and databases (Problems)

E.g "Anyone have a relative living with dementia?: Dementia touches a lot of people's lives, so is there anyone who wishes to talk about it on here? Just a thought"

Response.txt* -> *Text file consisting of saved emotional solutions and advice from online forums and databases (Solutions)

E.g "The best way to deal with your anxiety is to make sure to use deep breathing techniques. You can look them up on Google."

***Positive.txt* ->** *List of positive words*

E.g "authentic, awesome, elated, gracefully etc.."

***Negative.txt* ->** *List of negative words*

E.g "bitter, cheater, depression, malicious etc.."

***Resources* ->** *Websites that we used to extract emotional data*

https://www.mentalhealthforum.net/forum/ |
https://www.actualized.org/forum/forum/6-serious-emotional-problems/ |
https://www.enotalone.com/forum/index.php |
https://www.facebook.com/NYUSecrets/ |
https://www.facebook.com/BerkeleySecrets |

Data Modification
Data for our database was not taken as is and modified before running it through our algorithm. We noticed that majority of potential emotional solutions did not contain key words from the respective emotional problems they were addressing. For example, a particular emotional problem would mention and use the word "depression". However, its corresponding solution would not use the word "depression" or "depressed". This lack of keywords affected the accuracy of our TF-IDF algorithm as it relies on the term frequency of common words to calculate its score. Hence, we modified each individual solution with the keywords found in its respective problem.

> E.g. "Sorry to hear that I have suffered with it now since my **anxiety** started it's an horrible obsession, I don't know what to do I'm not on meds but I know suffer with **anxiety** and **depression**, it's the worst **obsession** to have but we are still here it's our stupid minds".

Above as an example of a modified emotional solution with the keywords that we manually added in bold. The addition of keywords improved our TF-IDF scores and also provides us with and avenue to use them as filters for better data classification in the future.

Algorithm
The algorithm relies on the TF-IDF score and Cosine similarity score to determine the relevance of each problem to its solution. The program also uses positive and negative words to determine the emotional sentiments of each solution relative to the given problem. The algorithm finally spits out a list of problems and their matched solutions along with their associated sentiment analysis score.

Our Github: https://github.com/rds493/NLPemo

Algorithm Steps :

1. Entries from the entries.txt and response.txt files are extracted from their respective files into the algorithm
2. The algorithm cleans up each entry by removing unnecessary characters such as stop words, symbols and numbers from the paragraph
3. Next the algorithm calculates the TF-IDF scores for the sets of problems and solutions. It then creates vectors using the remaining relevant words in both sets
4. The algorithm calculates the Cosine similarity score between each problem and the entire set of possible solutions. It returns the top three solutions which have the highest Cosine score
5. Sentiment analysis is then applied on the top three solutions and the problem involved. Percentage weights of positive and negative sentiments are assigned to each of the these entries
6. The negative weight of the problem is compared to the positive weights of the three potential solutions. The solution which has a positive weight with the

smallest deviation from the negative weight of the problem is chosen as the answer

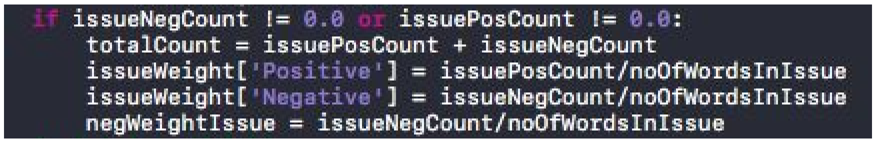
7. Finally, the algorithm spits out the user's problem and the chosen answer. The algorithm also creates a scoring file *answerCheck.txt* and a file which contains a list of every emotional entry in *Entries.txt* along with its associated answer from *Response.txt*

Our algorithm used NLTK packages for stemming and tokenization. Our list of positive and negative words were taken from:
https://github.com/jeffreybreen/twitter-sentiment-analysis-tutorial-201107
Despite incorporating these other programs, we implemented the TF-IDF, Cosine scoring, Sentiment analysis evaluation portions ourselves.

Sentiment
As previously mentioned, our program performs sentiment analysis on the top three solutions with the highest Cosine scores that the TF-IDF algorithm retrieves. We initially calculated sentiment scores of an entry in relation to only itself. As external sentiment analysis packages tend to perform sentiment analysis on texts in relation to itself, we decided to implement our own sentiment scoring algorithm. By implementing our own scoring system, we are able to assign potential solutions a sentiment score in relation to the negativity of a problem.

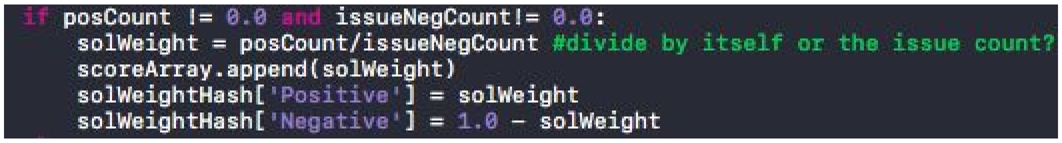
Code Snapshot:

-Problem sentiment score:



```
if issueNegCount != 0.0 or issuePosCount != 0.0:
    totalCount = issuePosCount + issueNegCount
    issueWeight['Positive'] = issuePosCount/noOfWordsInIssue
    issueWeight['Negative'] = issueNegCount/noOfWordsInIssue
    negWeightIssue = issueNegCount/noOfWordsInIssue
```

-Solution sentiment score:



```
if posCount != 0.0 and issueNegCount!= 0.0:
    solWeight = posCount/issueNegCount #divide by itself or the issue count?
    scoreArray.append(solWeight)
    solWeightHash['Positive'] = solWeight
    solWeightHash['Negative'] = 1.0 - solWeight
```

Effects of change in sentiment scoring technique:

We discovered that our evaluation (Precision, Recall and F-Measure) scores improved when we changed our sentiment scoring technique to take into consideration the solutions sentiment weights in relation to the negative sentiment weight of a problem.

-Sentiment of solutions:

E.g. [SENTIMENT:S] : {'Positive': 1.8, 'Negative': -0.8}
Sentiment weights of solutions have the possibility of > 1.0 as we are dividing by the total number of negative words in the problem. Hence, the

number of positive words in a given solution might exceed the number of negative words in the problem. This reflects the degree of effectiveness of a solution for a problem. The solutions negative weighted score can be negative. However, we ignore the negative score as we did not make use of it in our analysis.

The sentiment scoring for a problem calculates the sentiment weights of the problem in relation to itself. We score a problem by taking the total number of positive and negative word counts and divide it by the total number of words (tokens) in the problem. However, instead we calculated the sentiment weights for solutions in relation to the number of negative words in the problem that it aims to address. By doing so, the sentiment weights of solutions take into consideration the degree of negativity of problems and the algorithm is able to choose the best emotional response from the top three solutions returned by the TF-IDF algorithm.

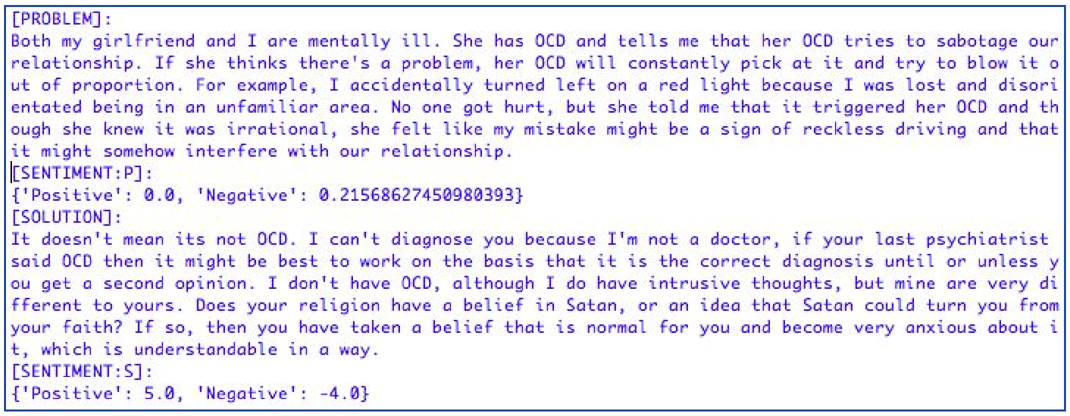
Sample Input and Output Data
The following data is sample input and output entries for our algorithm. The user first enters an emotional entry/problem and the algorithm spits out a solution along with all the associated sentiment weights.

Running the Program:

Algo.py : Simply run the *Algo.py* file from the command line or the the Python IDE. The algorithm will spit out a file *finalAnswers.txt* where you can view all the problem in our database their assigned solutions and sentiment weights

Note: For the purpose of successfully using the grading algorithm. When running Algo.py and prompted to enter a possible problem, kindly hit the return/enter key else the scoring algo will not pass.

Sample:



```
[PROBLEM]:
Both my girlfriend and I are mentally ill. She has OCD and tells me that her OCD tries to sabotage our
relationship. If she thinks there's a problem, her OCD will constantly pick at it and try to blow it o
ut of proportion. For example, I accidentally turned left on a red light because I was lost and disori
entated being in an unfamiliar area. No one got hurt, but she told me that it triggered her OCD and th
ough she knew it was irrational, she felt like my mistake might be a sign of reckless driving and that
it might somehow interfere with our relationship.
[SENTIMENT:P]:
{'Positive': 0.0, 'Negative': 0.21568627450980393}
[SOLUTION]:
It doesn't mean its not OCD. I can't diagnose you because I'm not a doctor, if your last psychiatrist
said OCD then it might be best to work on the basis that it is the correct diagnosis until or unless y
ou get a second opinion. I don't have OCD, although I do have intrusive thoughts, but mine are very di
fferent to yours. Does your religion have a belief in Satan, or an idea that Satan could turn you from
your faith? If so, then you have taken a belief that is normal for you and become very anxious about i
t, which is understandable in a way.
[SENTIMENT:S]:
{'Positive': 5.0, 'Negative': -4.0}
```

Stemming:
In addition to tokenization and the removal of stop words, we decided to add stemming capabilities to our algorithm which ultimately proved our evaluation scores. We stemmed words

in all entries, responses, positive and negative words. We used NLTK's stemmer: nltk.stem.snowball import SnowballStemmer to handle stemming.

Evaluation
We evaluated our program using Precision, Recall and F-Measure scores. Our answer set *answer.txt* was created as we scraped data from online forums and databases. The answer set *answer.txt* contains entries which contain a problem along with the top three solutions from users responses/comments that were within the same problem's post. Hence, these solutions were what individuals online thought were the best responses. The top three solutions were users responses which we thought did best at addressing the problem. Our scoring algorithm *scoringAlgo.py* takes the files *answerCheck.txt and answers.txt* and calculates the scores:

Initial Score (Baseline):

System Evaluation:
Precision: 0.185714285714
Recall: 0.353741496599
F-Measure: 0.24355971897

Our initial score reflects a version of our algorithm which did not include our custom sentiment scoring system, stemming and was run on a smaller dataset. Here, recall was the highest among subsequent systems evaluations because of the smaller size of our initial dataset. As our initial dataset was smaller, the algorithm had a smaller pool of potential solutions to choose from. Thus, due to this limitation it matches problems and solutions more accurately as the possibility of find a better solution than the ones we assigned in our answer set is lower.

After changing sentiment weights scoring technique:

System Evaluation:
Precision: 0.261904761905
Recall: 0.299319727891
F-Measure: 0.279365079365

Our second score reflects a version of our algorithm which included our updated sentiment scoring technique. With this change we saw a slight drop in recall. However, we saw improvements in precision and a slight improvement in our overall f-measure. Recall decreased as we were now taking into consideration the negative weight of a problem when we calculated the positive sentiment weights (relatively) of our solutions. This resulted in our algorithm matching solutions which were better at addressing the emotional state of an entry than the solutions associated with the problem in the answer set. Due to the relative sentiment weighting, precision increased as the top three solutions chosen by our algorithm contained more correct answers.

After adding more data:

System Evaluation:
Precision: 0.179245283019
Recall: 0.191275167785
F-Measure: 0.185064935065

Our third score reflects the version of our algorithm which processed a much larger data set of problems and solutions. With a larger data set all evaluation scores for our program drops. Despite this, we discovered that the systems score drops as the algorithm was actually suggesting better solutions for a given problem than the ones associated with the problem in the answer set. This happens because as more data is added, the algorithm benefits from a wider range of solutions it could choose from. The algorithm indeed chooses solutions for a problem from other websites and databases which did not exist in the answer set for that problem (Answers were created by simply looking at the comments immediately following an online emotional/problem post).

After adding Stemming:

System Evaluation:
Precision: 0.22641509434
Recall: 0.241610738255
F-Measure: 0.233766233766

Finally, our forth score reflects the version of our algorithm which includes stemming. By stemming words, we remove the inconsistency between words and their morphological form such as “Depression”, “Depressed”, “Depressing” etc. Hence, getting to a words morphological root form. Stemming helped to increase our evaluation scores.

Future Improvements

The following are improvements to our program which we believe will increase our evaluation scores in the future.

1. Continue modifying the solutions in our *response.txt* file to include more keywords and summarize each solution to reduce its overall size and increase the generability of solutions. This will increase the TF-IDF scores for keywords and increase the total number of applicable solutions for each problem.
2. Manually reorganize our answer set by reading through each problem and assigning the top three solutions from our entire solution set instead of assuming that the “best” answers/solutions are found within the same post for each problem.
3. Create a new segmentation algorithm which first categorizes problems and solutions into different categories before performing TF-IDF. Categories will include: Depression, Family, Friends, Work, Academics and Monetary. Our original algorithm will first classify a problem under one of these categories and next access the database of potential solutions for the category. This will reduce the total time taken to search for possible solutions and also increase the accuracy of solutions suggested.

4. Redefine our method of evaluation. To increase the accuracy and evaluation method, we can create a new *answerSheet.txt* set. Similar to the old *answers.txt*. This new answer sheet will contain a list of problems the top three solutions. However, the solutions are instead matched using attributes that psychologists deem important in an “ideal” solution. We would then compare the answer set of our algorithm *answerCheck.txt* and the answer set scraped from online forums *answers.txt* to this new *answerSheet.txt* file. This way we are able to compare the success of our algorithm at giving better solutions than what individuals online could.

Conclusion

Our program implements TF-IDF and includes a layer of sentiment analysis. Using our algorithm we achieved decent f-measure scores of ~0.25 to ~0.28. With further refinement of our algorithm and a better evaluation system these scores would improve significantly.

We would like to pre-category responses into distinctive categories as suggested above. This would reduce the probability of the algorithm drawing a solution that has little relation to a problem given. Thus, this will improve our scores significantly and also reduce the time taken for the algorithms to process information. This will require us to code a new algorithm that first reads our dataset of solutions and categorize them into different categories. This new algorithm will fall under “Document Classification” in the list of possible programing project topics.

Due to the subjectiveness of emotional entries and responses it was challenging determining the appropriate answers. We also felt that some solutions were too specific to particular type of problem or circumstance. Hence, we would need larger sets of data under each category and also circumstances to give better solutions. Regarding, sentiment analysis we found negative and positive sentiments for words can be applied to the same word depending on the context it is used in. These are usually descriptive words such as large or small. These type of words decrease the accuracy of our sentiment scorer overall. A combination of sentiment analysis systems are needed to better assign sentiment analysis scores and better calculate a sentence sentiment score in relation to another sentence.