Project 14

Parsa Sharmila, Jonada Ferracaku, Ciprian Florea

Analysis of Hate Speech 3

https://unioulu-my.sharepoint.com/personal/psharmil18\_student\_oulu\_fi/\_layouts/15/onedrive.aspx?slrid=5dacad9e-c0c1-7000-a42b-4523701ffccc&FolderCTID=0x01200017379E84D4AA2D40A43C7552892D8B71&id=%2Fpersonal%2Fpsharmil18\_student\_oulu\_fi%2FDocuments%2FNLP\_Shared%2FProject14\_AllPythonFiles

Contents

[Introduction 3](#_Toc534613879)

[Problem description 4](#_Toc534613880)

[Dataset description 5](#_Toc534613881)

[General methodology 6](#_Toc534613882)

[Detailed methodology 7](#_Toc534613883)

[Results and discussions 15](#_Toc534613884)

[Conclusion and perspectives 17](#_Toc534613885)

[References 17](#_Toc534613886)

# Introduction

The subject of this project is analysis of hate speech. The aim of this work was to implement programs that can identify hate speech, compute the hate-sentiment of a given document, and quantify the hate intensity of various collected datasets.

For this purpose, we will describe step by step how the programs were implemented. We will explain how a simplified corpus of hate speech was constructed following the approach of Kim and Hovy’s, Determining the sentiment of opinions, how certain heuristic were applied to assign intensity to each word of the vocabulary, how we computed the hate-sentiment of a given document, test the variation of the hate-sentiment intensity in various collected datasets in reddit.com, and run our hate-sentiment program in order to quantify the global hate intensity for different groups.

The related literature surveyed was mainly Kim and Hovy’s, Determining the sentiment of opinions, given the similarities between these two works. Thus, in this paper, the authors aim to determine the sentiment of opinions. Their goal was to identify sentiments and present a system that automatically finds the people who hold opinions about that topic and the sentiment of each opinion. They do this by determining word sentiment and combining sentiments within a sentence. Our project is similar in that sense, we aimed to identify hate speech in documents and collected datasets and quantify the hate intensity for certain groups of people. In addition, we also determine word sentiment intensity and combine these in texts. We followed the same approach as in the aforementioned literature to firstly, construct the required corpus of hate speech, and secondly, to compute the hate-sentiment of a given document.

Compared to this surveyed work, we believe the novelty of our work relates to running these programs in various collected datasets in reddit.com, where we aim to identify discussion threats associated to hate-speech populated by male author, and another one populated by female author, and a set of hate related discussion threats populated by various age population (teenager, professional and elderly) and quantify the hate intensity of each group. The challenge of this work was that we had to employ our own methodology to collect each dataset. Given that we cannot access information about users such as gender and age, some assumptions were required. For instance, the majority of users posting in the AskMen subreddit, a place to ask random strangers for advice from the male perspective, are male.

# Problem description

The problem description translates to firstly, how to assign intensity to the extended vocabulary obtained starting from the small amount of seed words gathered by hand, that refer to hate-speech. The extended vocabulary was obtained by implementing a program that uses WordNet to identify a larger number of synonyms and hyponyms. The problem in this matter was that the defined heuristic for assigning intensity must keep in mind the relationship between words. For example, a direct hyponym of a seed word would be assigned less intensity than the actual seed word, but more intensity than a secondary hyponym. The heuristic for this matter will be explain in the methodology chapter.

Another problem was how to compute the hate-sentiment of a given document. To this end, we used the approach described in Kim and Hovy’s paper. Firstly, a string-matching algorithm that identifies any of the speech words in the used vocabulary in an inputted document was required. Then a Treebank implementation was needed to identify modifiers pertaining to each of the SS words (e.g., negation, less, very, ...) and update the hate intensity according to the modifier. For instance, in case of presence of negation, we expect that the hate speech intensity vanishes. While “very” would increase the intensity of the hate sentiment by a fixed factor that you can setup. The problem description was how to compute the total hate-sentiment of the text as the sum of the hate sentiment of individual identified SS words after considering the modifiers.

Thirdly, a problem raised by this project was how to test the variation of the hate-sentiment intensity in various collected datasets in reddit.com. This was in particular more challenging as certain heuristic needed to be defined. Thus, the methodology employed to collect each dataset had to be defined. We used our own keyword lists to identify discussion threats associated to hate-speech populated by male author, and another one populated by female author, and a set of hate related discussion threats populated by various age population (teenager, professional and elderly). Problems raised by this task were how to implement the hate-sentiment program in order to quantify the global hate intensity for each group and also use the Tree-bank to identify the most frequent named-entities and named-phrase associated to each group.

We believe we tackled these problems quite successfully.

# Dataset description

Firstly, we assembled a small amount of seed words by hand that refer to hate-speech:

Hate, attack, disparage, violence, prejudice, intimidate, discrimination, bully, cyber-bullying, misogyny, dislike, censure, criticize, invade, tense, grave, rotten, wicked, disrespect.

Secondly, we identified a few examples of hate speech words in each category. Three categories were defined (high intensity, medium and moderate intensity):

High: rotten, wicked, violence

Medium: censure, intimidate, prejudice

Moderate: tense, dislike, disrespect.

The vocabulary was then extended to include more words that refer to hate-speech. We called this obtained vocabulary “S” and it contains, in addition to the seed words, synonym words for each manually identified word, first and second hyponyms, synonyms for each first and second hyponym, and first and second hyponyms of each synonym. Thus, the vocabulary “S” reached a word count of 1338 words that refer to hate-speech.

In addition, this dataset also contains intensity weight for each word. The intensity for each word was assigned using the heuristic that will be described in the methodology chapter.

For the string matching algorithm and intensity calculation for a given text based on literals/modifiers, a text file hateSpeech.txt, two dictionary files negative.yml, positive.yml was used along with the vocabulary mentioned above.

To further expand the dataset, various subreddits according to a specific target group were collected in order to test the variation of the hate-sentiment intensity. Based on our own heuristic, the selected subreddits are:

Askmen - assuming that the target group that uses this subreddits are male

Askwoman – assuming that the target group that are active in this subreddit are women

Teenagers – assuming that the target group that are active in this subreddit are teenagers between 13 – 19 years old

ITdept – assuming that the target group that are active in this subreddit are professionals

1920s – assuming that the target group that are active in this subreddit are elders

# General methodology

The overall methodology and steps taken to solve the problem. We started by constructing a simplified corpus of hate speech. We started from seed words gathered by hand and assigned them to one of the three hate-speech categories.

Next, we identified more speech words that relate to these seed words using WordNet. The result was a vocabulary “S” containing 1338 words. This was done by writing a program that iterates through the list of seed words and, using WordNet, identifies synonyms for each word, first and second hyponyms, their synonyms, and first and second hyponyms of each synonym. The program is a Python program. The final vocabulary “S” can be defined as:

S = words + synonyms + hyponyms + synonyms\_for\_hyponyms + hyponyms\_of\_synonyms​

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Category Intensity** | Vocabulary | Synonyms | Hyponyms | Synonyms for hyponyms | Hyponyms of synonyms |
| High | 3 |  |  |  |  |
| Medium | 3 |  |  |  |  |
| Low | 3 |  |  |  |  |

Next, intensity was assigned for each of the words in vocabulary “S”. We followed a simple heuristic that considers the relationship between words. The ProgramIntensity.py program defines a Word class with word and intensity properties. The words are categorized and iterated through to assign the right weight/intensity to the right type of word.

Both programs, Program.py and ProgramIntensity.py, were written using the Python programming language.

For string matching algorithm implementation, simple python program was written that takes hateSpeech.txt file as input and finds out the words that matches with SS. For absolute match, re module of python was used.

For the intensity calculation of the text based on the modifiers/literals, first a treebank is created by tokenizing the text. Then the literals are matched from the dictionary in the text and are labelled with a range from -3 to 3 based on the tags for the literals in the dictionary (i.e. for negLow return -1, posLow return 1 etc.). The total tags are calculated to get the sum of the intensity of the text.

To test the variation of the hate-sentiment intensity in reddit.com and the selected subreddits, the Python Reddit API Wrapper (PRAW) was used. It is a python package that allows for simple access to reddit’s API. The purpose of using reddit API is to collect comments on the specified subreddits (askman, askwoman, teenagers, ITdept, 1920’s) and to identify the hate-sentiment intensity in each group.

In order to quantify the global hate intensity for each group, ProgramIntensity.py was used. The result of applying the reddit.com dataset in the created program, was the identification of hate words based on the SS vocabulary and their intensity given by our own heuristic.

To further expand the program, the Tree-bank approach was used to identify the most frequent named-entities and named-phrased associated to each group of subreddit. All functionalities can be found on redditTreebank.py file.

# Detailed methodology

We followed the approach of Kim and Hovy’s, Determining the sentiment of opinions in order to construct a simplified corpus of hate speech. For this purpose, we assembled a small amount of seed words by hand that refer to hate-speech. These words are:

Hate, attack, disparage, violence, prejudice, intimidate, discrimination, bully, cyber-bullying, misogyny, dislike, censure, criticize, invade, tense, grave, rotten, wicked, disrespect.

Typically, one requires to have three categories of hate speech (high intensity, medium and moderate intensity). We manually identify few examples of hate speech words in each category.

*High: rotten, wicked, violence.   
Medium: censure, intimidate, prejudice.   
Moderate: tense, dislike, disrespect.*

The next step of the project was to design and implement a program that uses WordNet to identify a list of synonym words for each of our manually identified word, a list of first and second hyponyms, their synonyms and first / second hyponyms of each synonym. We called the obtained vocabulary “S”.

To this end we created our first program, Program.py, and added our words to a list:

*words = ['rotten', 'wicked', 'violence', 'censure', 'intimidate', 'prejudice', 'tense', 'dislike', 'disrespect']*

Then, we created secondary lists for each of the requirements:

*S = []  
synonyms = []  
hyponyms = []  
synonyms\_of\_hyponyms = []  
hyponyms\_of\_hyponyms = []*

The following code iterates through each seed word, and for each lemma of each synset, it appends the lemma name to the list of synonyms. For each synset we also take the first and second hyponyms and append them to the hyponyms list:

*for word in words:*

*for synset in wn.synsets(word):*

*## print (synset, "for ", word)*

*for lemma in synset.lemmas():*

*## print (lemma, "for ", synset)*

*synonyms.append(lemma.name())*

*for hyponym in synset.hyponyms()[:2]:*

*for hyponym\_lemma in hyponym.lemmas():*

*hyponyms.append(hyponym\_lemma.name())*

The same process is repeated for each word in the hyponyms list to obtain the other requirements:

*for word in hyponyms:*

*for synset in wn.synsets(word):*

*for lemma in synset.lemmas():*

*synonyms\_of\_hyponyms.append(lemma.name())*

*for hyponym in synset.hyponyms()[:2]:*

*for hyponym\_lemma in hyponym.lemmas():*

*hyponyms\_of\_hyponyms.append(hyponym\_lemma.name())*

The vocabulary named “S” is made from all the obtained sub lists:

*S = words + synonyms + hyponyms + synonyms\_for\_hyponyms + hyponyms\_of\_synonyms​*

The word count at this point is 1338.

The project then required to asign intensity to each word. We followed the example heuristic and assigned intensity using the following heuristic: words in category 1 (high intensity hate speech words) will be assigned weight 5, their synonyms will be assigned weight 5 as well. Their direct hyponyms will be assigned weight 4.7 (as well as their synonyms), while the second hyponyms will be assigned weight 4.5. Etc.  Words belonging to Category 2 (medium hate speech) will be assigned 4 and reduce by 0.2 and 0.3 as in previous case for direct and second hyponyms if any. While words of Category 3 (moderate hate speech) will be assigned weight 3.

To do this, we wrote a different Python program called ProgramIntensity.py. This program defines a class Word with word and intensity properties:

*class Word:*

*def \_\_init\_\_(self, word, intensity):*

*self.word = word*

*self.intensity = intensity*

The lists are defined in the beginning by intensity category and an empty list for all words with intensity is declared:

*all\_words\_with\_intensity = []*

*high\_intensity\_words = ['rotten', 'wicked', 'violence']*

*medium\_intensity\_words = ['censure', 'intimidate', 'prejudice']*

*moderate\_intensity\_words = ['tense', 'dislike', 'disrespect']*

Moving forward, we iterated through each word of each category, and appended the word to the list and set the intensity by creating new Word objects:

*a = Word(word, 5)*

*all\_words\_with\_intensity.append(a)*

Following the heuristic defined above, we exemplify the code methodology for words in high intensity words category:

*for word in high\_intensity\_words:*

*a = Word(word, 5)*

*all\_words\_with\_intensity.append(a)*

*## their synonyms will be assigned weight 5 as well*

*for synset in wn.synsets(word):*

*for lemma in synset.lemmas():*

*b = Word(lemma.name(), 5)*

*all\_words\_with\_intensity.append(b)*

*## their direct hyponyms will be assigned weight 4.7 (as well as their synonyms)*

*i = 0*

*for hyponym in synset.hyponyms():*

*if (i == 0):*

*for hyponym\_lemma in hyponym.lemmas():*

*c = Word(hyponym\_lemma.name(), 4.7)*

*all\_words\_with\_intensity.append(c)*

*for hyponym\_synset in wn.synsets(hyponym\_lemma.name()):*

*for synset\_lemma in hyponym\_synset.lemmas():*

*d = Word(synset\_lemma.name(), 4.7)*

*all\_words\_with\_intensity.append(d)*

*## the second hyponyms will be assigned weight 4.5. Etc.*

*if (i == 1):*

*for hyponym\_lemma in hyponym.lemmas():*

*e = Word(hyponym\_lemma.name(), 4.5)*

*all\_words\_with\_intensity.append(e)*

*for hyponym\_synset in wn.synsets(hyponym\_lemma.name()):*

*for synset\_lemma in hyponym\_synset.lemmas():*

*f = Word(synset\_lemma.name(), 4.5)*

*all\_words\_with\_intensity.append(f)*

*i += 1*

We follow the same approach for medium and moderate intensity hate speech words but starting with weight 4 and 3.

The next task was to implement a simple string-matching algorithm that will identify the hate words from an inputted text listed in vocabulary SS mentioned above. For implementing this the method was to simply search the words from SS in each line of the text file. We have used regular expression module of Python to match the absolute text. Snippet of the code is below –

*filename = "hateSpeech.txt"*

*infile = open(filename, 'r')*

*lines = infile.readlines()*

*hatearray = []*

*for line in lines:*

*for hate in SS:*

*if re.search(r'\b%s\b' % hate, line):*

*#print(hate)*

*hatearray.append(hate)*

*tkinter.messagebox.showinfo("Results", hatearray)*

Next, a treebank is to be implemented to identify modifiers pertaining to each word of SS. The approach should be, to update the hate intensity of an inputted text if there is any word mentioned in the listed modifiers (very, less, severe etc.) followed by the hate word mentioned in SS. Eventually, the total hate intensity has to be calculated based on the modifiers.

Firstly, the words are split by each sentence of the text and are pos tagged.

*class Splitter(object):*

*def \_\_init\_\_(self):*

*self.nltk\_splitter = nltk.data.load('tokenizers/punkt/english.pickle')*

*self.nltk\_tokenizer = nltk.tokenize.TreebankWordTokenizer()*

*def split(self, text):*

*sentences = self.nltk\_splitter.tokenize(text)*

*tokenized\_sentences = [self.nltk\_tokenizer.tokenize(sent) for sent in sentences]*

*return tokenized\_sentences*

*class POSTagger(object):*

*def \_\_init\_\_(self):*

*pass*

*def pos\_tag(self, sentences):*

*pos = [nltk.pos\_tag(sentence) for sentence in sentences]*

*# adapt format*

*pos = [[(word, word, [postag]) for (word, postag) in sentence] for sentence in pos]*

*return pos*

A class DictionaryTagger is declared to invoke the literals/modifiers and match with the text. For this, two .yml files are used to define the intensity of the literals/modifiers.

***negative.yml:***

*severe: [negLow]​*

*extreme: [negMedium]​*

*intense: [negHigh]*

***positive.yml:***

*moderate: [posLow]​*

*less: [posMedium]​*

*little: [posHigh]​*

***none: [None]***

The tag “none:[None]” in the dictionary is for the case when the modifier is found in the text but the next word is not found in SS. Then the modifier will not be tagged with any intensity.

*def \_\_init\_\_(self, dictionary\_paths):*

*files = [open(path, 'r') for path in dictionary\_paths]*

*dictionaries = [yaml.load(dict\_file) for dict\_file in files]*

*map(lambda x: x.close(), files)*

*self.dictionary = {}*

*self.max\_key\_size = 0*

*for curr\_dict in dictionaries:*

*for key in curr\_dict:*

*if key in self.dictionary:*

*self.dictionary[key].extend(curr\_dict[key])*

*else:*

*self.dictionary[key] = curr\_dict[key]*

*self.max\_key\_size = max(self.max\_key\_size, len(key))*

The modifiers/literals are matched from the dictionary with the text, if the literal is found in the text, check the next word, if the next word is in SS, tag the literal with the intensity defined in the code as per the tag in the dictionary (posHigh, negLow etc.)

*if literal in self.dictionary:*

*print(literal)*

*filename = "hateSpeech.txt"*

*infile = open(filename, 'r')*

*lines = infile.readlines()*

*for line in lines:*

*list\_of\_words = line.split()*

*next\_word = list\_of\_words[list\_of\_words.index(literal) + 1]*

*#print(next\_word)*

*if next\_word in SS:*

*#for hate in SS:*

*#if re.search(r'\b%s\b' % hate, next\_word):*

*#for hate in SS:*

*# if re.search(r'\b%s\b' % hate, line):*

*#self.logger.debug("found: %s" % literal)*

*is\_single\_token = j - i == 1*

*original\_position = i*

*i = j*

*taggings = [tag for tag in self.dictionary[literal]]*

*tagged\_expression = (expression\_form, expression\_lemma, taggings)*

*if is\_single\_token: #if the tagged literal is a single token, conserve its previous taggings:*

*original\_token\_tagging = sentence[original\_position][2]*

*tagged\_expression[2].extend(original\_token\_tagging)*

*tag\_sentence.append(tagged\_expression)*

*tagged = True*

*else:*

*is\_single\_token = j - i == 1*

*original\_position = i*

*i = j*

*taggings = ['None']*

*tagged\_expression = (expression\_form, expression\_lemma, taggings)*

*if is\_single\_token: # if the tagged literal is a single token, conserve its previous taggings:*

*original\_token\_tagging = sentence[original\_position][2]*

*tagged\_expression[2].extend(original\_token\_tagging)*

*tag\_sentence.append(tagged\_expression)*

*tagged = True*

*else:*

*j = j - 1*

*if not tagged:*

*tag\_sentence.append(sentence[i])*

*i += 1*

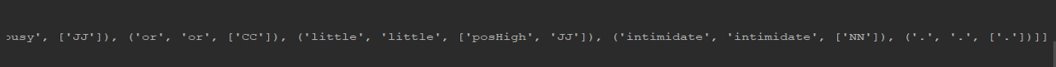
*return tag\_sentence*

*dicttagger = DictionaryTagger([ 'positive.yml', 'negative.yml'])*

*dict\_tagged\_sentences = dicttagger.tag(pos\_tagged\_sentences)*

*print(dict\_tagged\_sentences)*

The snippet of intensity tagging –



*def value\_of(sentiment):*

*if sentiment == 'negLow': return -1*

*if sentiment == 'negMedium': return -2*

*if sentiment == 'negHigh': return -3*

*if sentiment == 'posLow': return 1*

*if sentiment == 'posMedium': return 2*

*if sentiment == 'posHigh': return 3*

*if sentiment == 'None': return 0*

*return 0*

The total hate intensity for the modifiers are calculated by below code as per the sum of above conditions –

*def sentiment\_score(review):*

*return sum ([value\_of(tag) for sentence in dict\_tagged\_sentences for token in sentence for tag in token[2]])*

To get the hate intensity of subreddits, PRAW was used. Prerequisites of using and applying Python Reddit API Wrapper were: reddit account to access Reddit’s API and client ID & client secret in order to access Reddit’s API as a scrip application.

To create a read-only Reddit instance the following code was executed:

*Import praw*

*reddit = praw.Reddit(user\_agent='Comment Extraction (by /u/USERNAME)',*

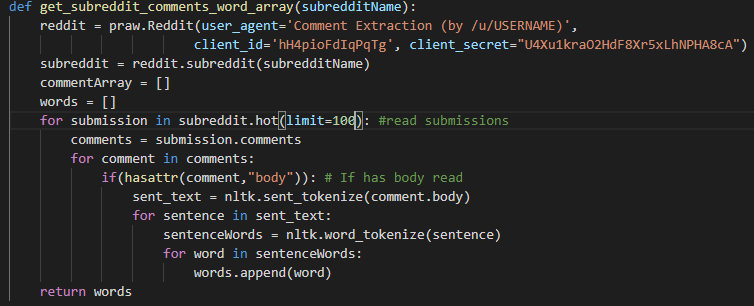
*client\_id='hH4pioFdIqPqTg', client\_secret="U4Xu1kraO2HdF8Xr5xLhNPHA8cA")*

The created instance was used in each of the selected subreddits. The following outputs the titles of the 100 hottest submissions in each of subreddit groups:

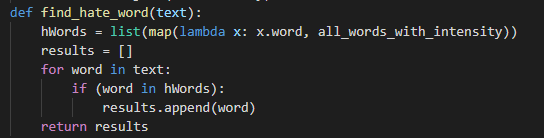
*For submission in subreddit.hot(limit=100):*

*Comments = submission.comments*

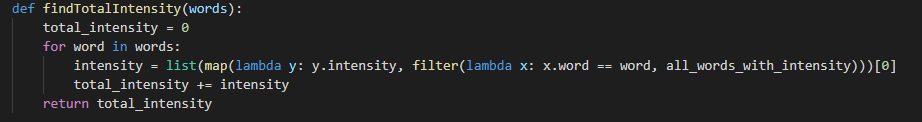
A list of comments is displayed and tokenized in order to get the words to be analyzed.



After having a list of words used in each of the subreddits, they are analyzed to find the hate words used in every group according to our vocabulary.

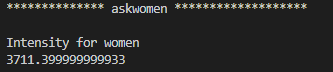


For each of the words the hate-intensity of the word is calculated using ProgramIntensity.py

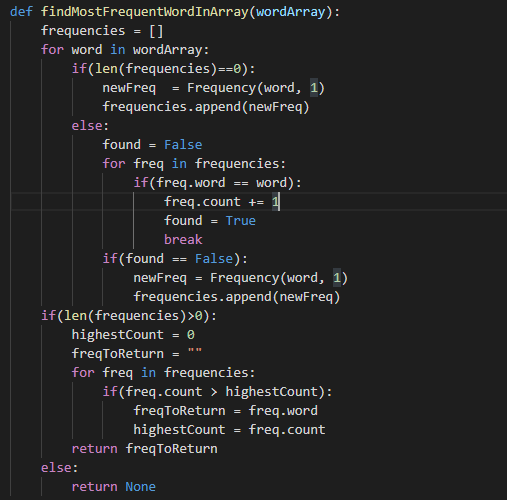


The three above functions are located in analyzeRedditHateSpeech.py file. They are implemented in each of the subreddits displaying a numeric hate-intensity result calculated by adding the hate intensity of each hate word in the assigned subreddit. By looking at the results, we can determine which subreddit has the highest intensity of hate-words used while commenting in the specified subreddit.

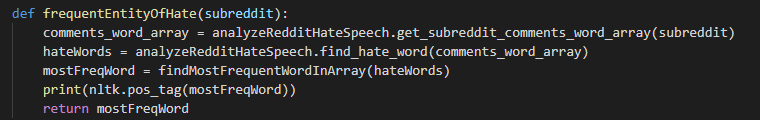
The current result displays the subreddit that has the highest intensity of hate words used.



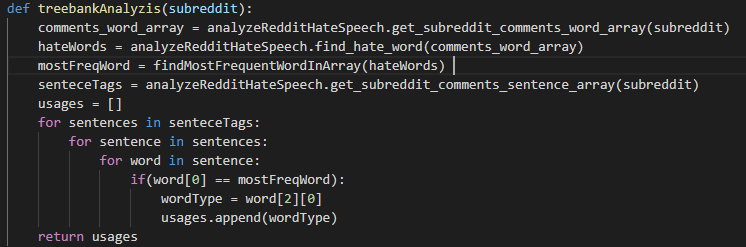
The most frequent word was found using the following functions. It returns the number of usage for each word in a word array given.



The following function find the most frequent hate word based on our vocabulary for each subreddit.



The following function, using treebank, analyses the part of speech for the usage of the most frequent word. Giving the fact that one specific word may be used In different context of speech, it returns a list of usages.



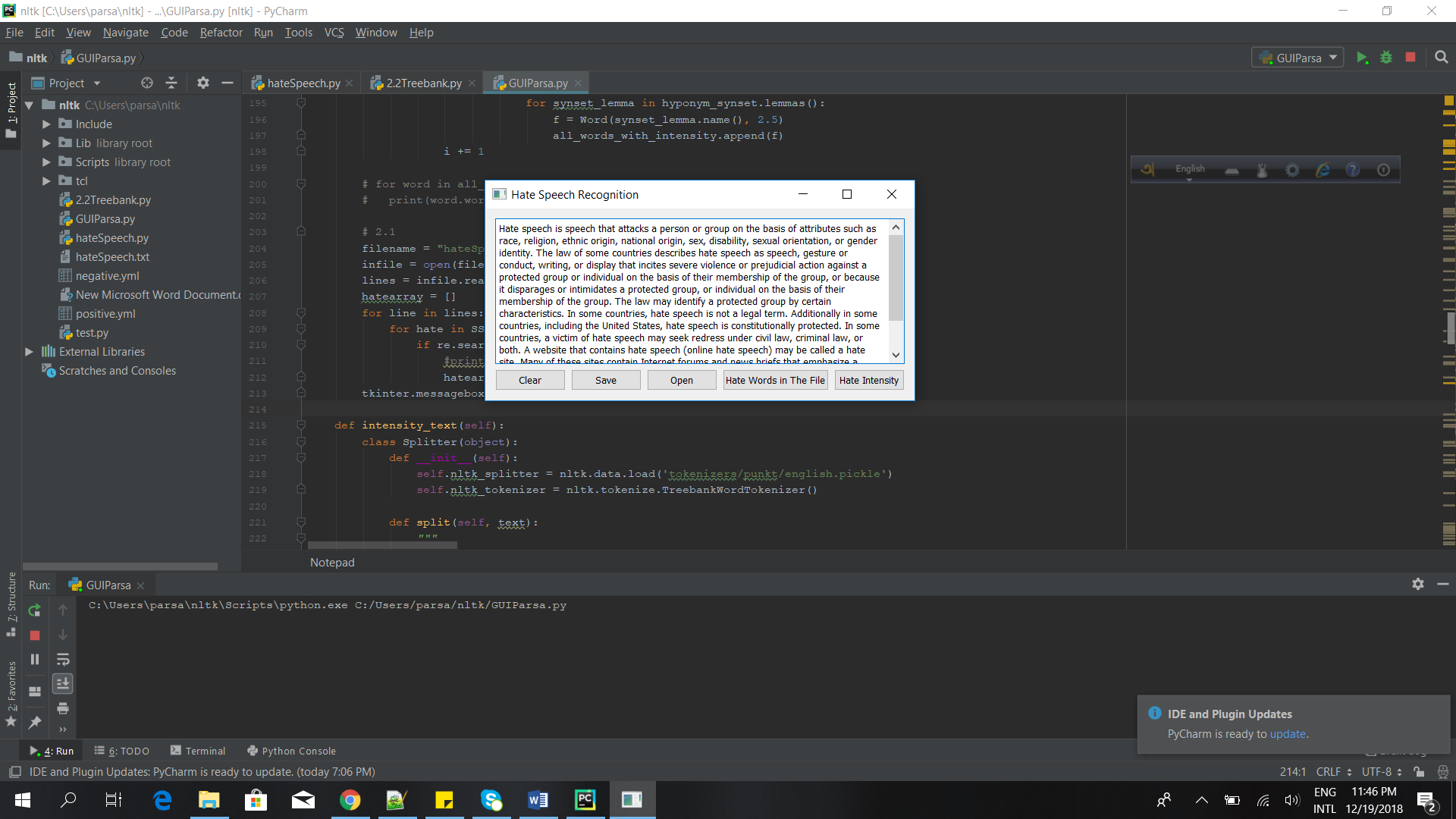
For example, the following word in ‘askwoman’ was used in the following context:



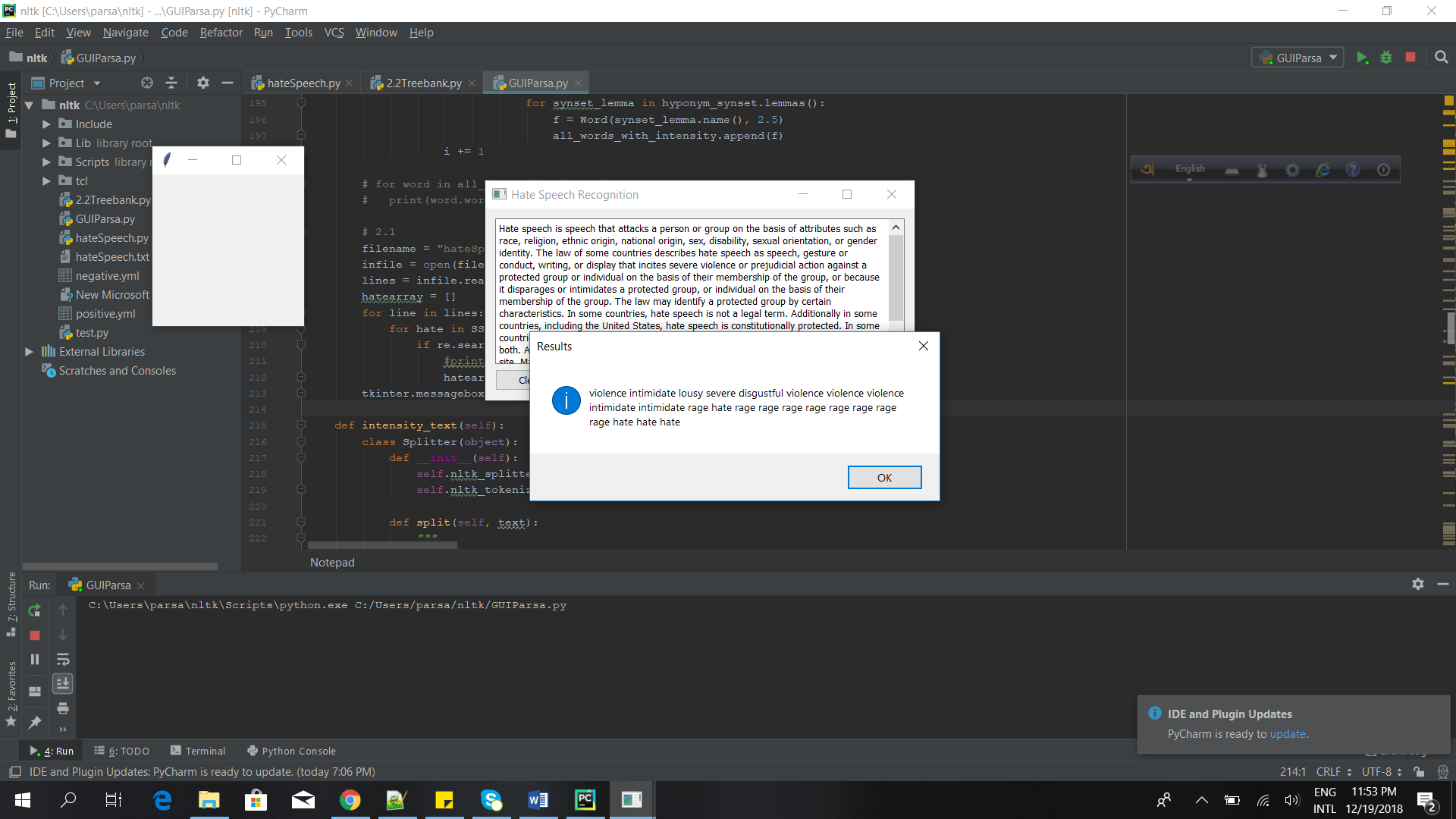
# Results and discussions

Regarding our vocabulary “S” and “SS”, we were satisfied with being able to construct a corpus of speech that contains a significant amount of words and be able to use it for the latter analysis of given documents and reddit comments purposes, starting with a handful of words, using WordNet and implementing our programs. The intensity assignment heuristic was also a good place to start when working with hyponyms and synonyms, however, in an ideal case, we would have to perform several tests and compare them to see how different weights affect the results.

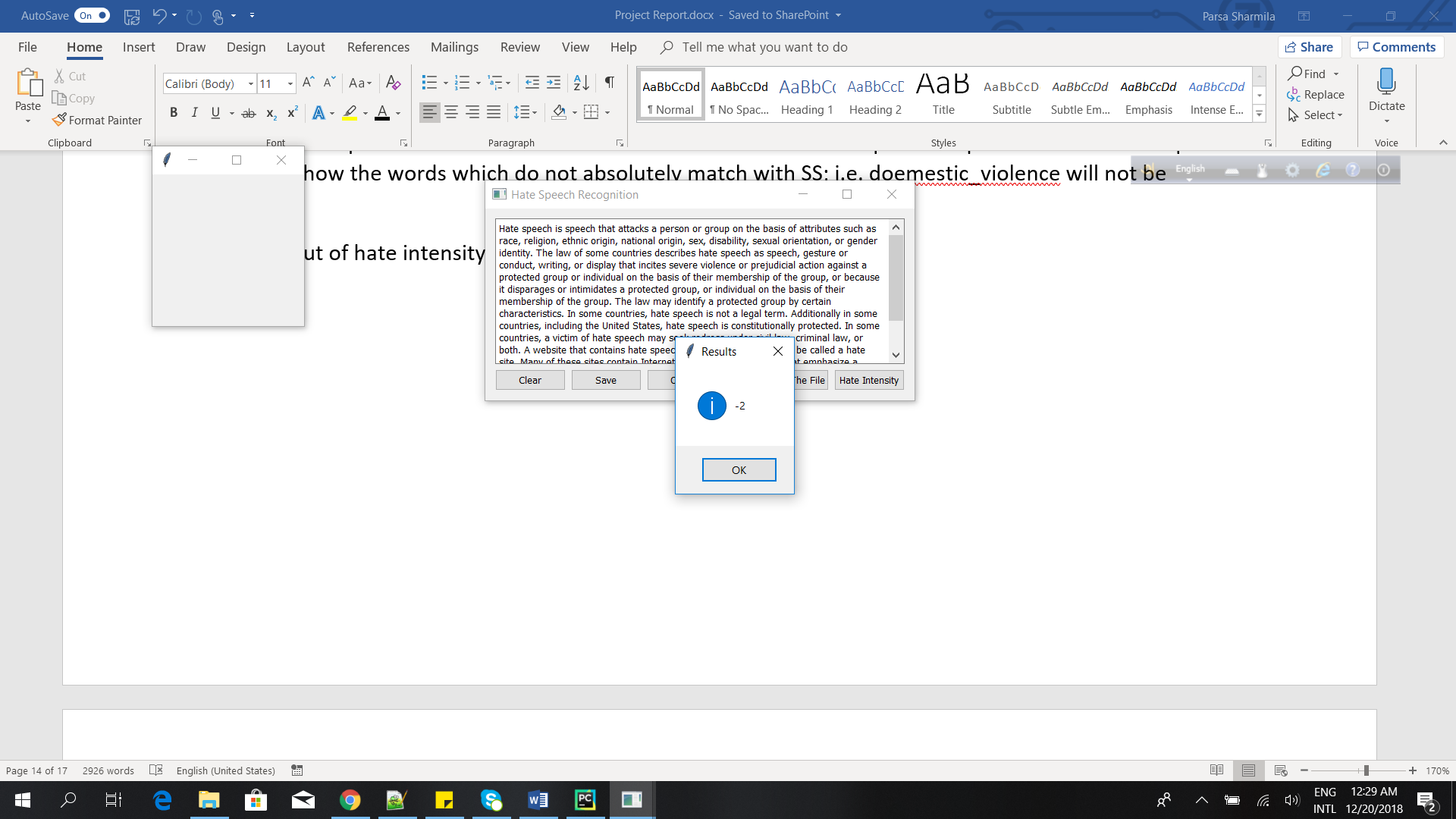
The string-matching algorithm looks like below with a simple GUI developed with PyQt5 (GUIHateSpeech.py) –



The Open button allows to modify the hateSpeech.txt file as user’s choice and the user has to click save after making the change. To find the words which are present in the vocabulary SS from this text, the user has to click “Hate Words in The File” button. The output looks like below –

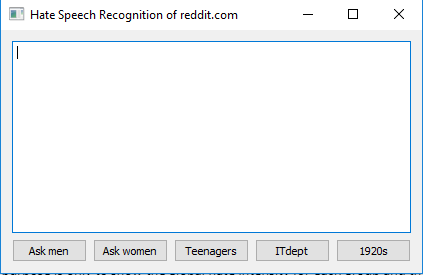


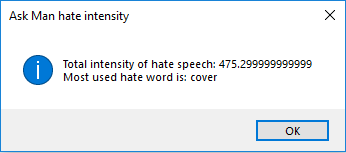
There is multiple result for the same word because the word is multiple times present in SS. The output will not show the words which do not absolutely match with SS; i.e. doemestic\_violence will not be shown.

The output of hate intensity calculation looks like below –

To display the results of the various subreddits that were used as a dataset, a GUI was created. The purpose is only to show the global hate intensity for each group and the most used frequent named-entity & name-phrased associated.

By selecting one of the five categories, a message box is displayed containing all the findings based on reddit.com dataset.





# Conclusion and perspectives

To conclude, we believe that the project was interesting and challenging enough, and that we managed to tackle the problems that it raised. The subject and related problems are real ones, especially in online mediums, such as reddit.com. We feel this project offered a good opportunity to better understand the applications associated with the natural language processing subject.

Regarding the latter part of the project, hate-speech analysis is a challenging and difficult part of understanding online forum users. As our results show, good results can be obtained even with relatively simple methodologies and only a small amount of manual seeding effort.

# References

- Kim, S. M., & Hovy, E. (2004, August). Determining the sentiment of opinions. In *Proceedings of the 20th international conference on Computational Linguistics* (p. 1367). Association for Computational Linguistics.

- http://fjavieralba.com/basic-sentiment-analysis-with-python.html​