Department of Computer Science
Technical University of Cluj-Napoca

Intelligent Systems

Laboratory activity 2018-2019

Project title: Knowledge representation
Tool: ConceptNet

Name: Romina-Ioana Băilă
Group: 30433
Email: romina.baila16@gmail.com

Assoc. Prof. dr. eng. Adrian Groza
Adrian.Groza@cs.utcluj.ro

Contents

Chapter 1

Installing the tool

The tool I will be using is ConceptNet - my primary tool - the one that gave me the idea for a natural language processing based project. The other tool I will be using is Vader from NLTK. As I found out, just by using ConceptNet I cannot achieve a reliable result for a sentiment analysis. So I will use help of Vader, which will give me reliable scores when analyzing sentences. ConceptNet doesn't require any installation. A simple import in Python will do the trick. The same goes for Vader, some simple import statements are all I need.

Chapter 2

Running and understanding examples

To get used to the tools, some examples were run. The examples were provided by the respective tools. I also worked a bit with the examples, to get used to them, understand how they work and how I can use them.

2.1 ConceptNet example

The ConceptNet example I ran was the one provided on its API GitHub page:

```
import requests
obj = requests.get('http://api.conceptnet.io/c/en/example').json()
obj.keys()
len(obj['edges'])
```

This example provided by ConceptNet is for the word "example. The input is the html address of the word "example" stored in their database. The output is the json associated with "example". The example uses python to run the code and the requests library. There, we can see the fields that the word "example" internally stores, such as the context in which it is used, the id of this word and the edges going from and coming into this node.

2.2 Vader example

Here is the example I used for the Vader tool:

```
tricky_sentences = [
...     "Most automated sentiment analysis tools are shit.",
...     "VADER sentiment analysis is the shit.",
...     "Sentiment analysis has never been good.",
...     "Sentiment analysis with VADER has never been this good.",
...     "Warren Beatty has never been so entertaining.",
...     "I won't say that the movie is astounding and I wouldn't claim that \
...     the movie is too banal either.",
...     "I like to hate Michael Bay films, but I couldn't fault this one",
...     "It's one thing to watch an Uwe Boll film, but another thing entirely
...     to pay for it",
...     "The movie was too good",
...     "This movie was actually neither that funny, nor super witty.",
...     "This movie doesn't care about cleverness, wit or any other kind of \
...     intelligent humor.",
```

```
...     "Those who find ugly meanings in beautiful things are corrupt without
...     being charming.",
...     "There are slow and repetitive parts, BUT it has just enough spice to
...     keep it interesting.",
...     "The script is not fantastic, but the acting is decent and the cinemat
...     is EXCELLENT!",
...     "Roger Dodger is one of the most compelling variations on this theme."
...     "Roger Dodger is one of the least compelling variations on this theme.
...     "Roger Dodger is at least compelling as a variation on the theme.",
...     "they fall in love with the product",
...     "but then it breaks",
...     "usually around the time the 90 day warranty expires",
...     "the twin towers collapsed today",
...     "However, Mr. Carter solemnly argues, his client carried out the kidna
...     under orders and in the ''least offensive way possible.''"
... ]
sentences.extend(tricky_sentences)
sid = SentimentIntensityAnalyzer()
for sentence in sentences:
    print(sentence)
    ss = sid.polarity_scores(sentence)
    for k in sorted(ss):
        print('{0}: {1}, '.format(k, ss[k]), end='')
    print()
```

My algorithm implies combining the Vader tool provided by NLTK and ConceptNet to make a real life application: a generator of fake reviews for some electronic products. This might not be the most noble thing to do; this kind of generator would basically lie to the possible buyers. But nevertheless, this is a real life application, which is actually demanded.

Chapter 3

Project description

3.1 Narative description

This artificial intelligence project runs in the field of natural language processing.
The system will generate fake reviews based on some existing ones. Of course, this fake reviews need some kind of validation, so we can decide whether or not a generated review is reliable.
The input of the system is a dataset containing actual product reviews, extracted from Amazon.
The output of the system will be a new set, containing only valid fake reviews.
To accomplish this goal, these steps will be followed, for each review from the original set:

- Depending on the user input, a fake review will be generated, either by replacing some words with their synonyms, antonyms, or related words.
- Again, depending on the user input, either the nouns, adjectives, adverbs, verbs or all words will be replaced.
- After making these decisions, the system will search in the review for the specified part of speech. Those corresponding words will be put in a list.
- For every item in that list, the system will search its ConceptNet correspondence, based on the relation chosen by the user.
- ConceptNet will give a json response. This response needs to be parsed, so that only the new word will be appended to a list.
- Of course, there will be cases when ConceptNet will not be able to give a response, so the we will put a flag in the list containing the new words. This flag will have the form of a string containing "exceptie".
- Next, we will proceed to the replacing part. We traverse the list with the new words. If we come across the word "exceptie", it means we cannot replace a word in the original review. So we go to the next word in the list.
- If "exceptie" was not found, we test the new word to see if it contians _ . If ConceptNet gives a response consisting on multiple words, it will append them with _ .So we need to remove those.
- After removing any existing underscores, we can finally create the fake review, by replacing the original words, with the new ones.

  - When it comes to verbs, things get a little complicated. Using ConceptNet for retrieving synonyms or antonyms is not very reliable. So we need a different method. Also, we deal with verbs only if we look for antonyms.
  - The part is well commented in the code, but the basic idea is that we check every word against different cases. We look for negative words, to know how to negate the verbs, we look for modal verbs, etc.

- For the validation part, we check the average sensitivity score of the original review (computed by Vader), with the average sensitivity score of the newly created review. If the difference between the 2 scores is bigger than 0.1, the review is not valid.

- Finally, we create the fake reviews set. Each fake review will be printed in a .txt file, on a new line.

- The length of the original reviews set and fake reviews set will be printed, so they can be further analyzed and used to compute graphics.

The only assumption I made was that in the original dataset, every review (not sentence) is on a separate line in the .txt file.

Chapter 4

Preliminary results

As for the preliminary results, I took every piece of code and tested it separately. I found out that, because I could not use ConceptNet on verbs, it was quite difficult to get some code working on this part of speech.
The algorithms worked, but needed some refinement.
Another interesting for me was to find out that Vader does not work as good as I thought. For example, in the sentence "It doesn't last long", the word "last" was tagged as an adjective, not as a verb.
So, it was obvious from the beginning that working with just these two tools will not yield perfect results.

Chapter 5

Implementation details

5.1 Relevant code

- Sentiment analysis
  This piece of code computes the average sensitivity score of a review, using Vader.

```
def analyzeSentence(sentence):
        sid = SentimentIntensityAnalyzer()
        ss = sid.polarity_scores(sentence)
        return ss
```

- POS tagging
  POS tagging means associating every word with its correspondent part of speech. This is also done by Vader.

```
def identifyPOS(sentence):
      sentence = nltk.word_tokenize(sentence)
      pos = nltk.pos_tag(sentence)
      return pos
```

- Selecting desired word base on POS
  Selecting the words to be replaced, based on their POS was a rather easy task. This is an example for selecting adjectives from a review. We deal similarly with nouns, adverbs and verbs.

```
for (word, pos) in posList:
            if pos == 'JJ':
                  listaAdj.append(word)
```

- Getting the json response using the ConceptNet API
  This was the most expensive task (from the resources point of view), as it took very long to get the responses using the ConceptNet API

```
for adj in listaAdj:
        response = requests.get('http://api.conceptnet.io/
        query?start=/c/en/'+adj+'&rel=/r/Synonym&
        end=/c/en&limit=1').json()
        responseList.append(response)
```

This gives the synonyms of all adjectives in a review. We can easily change it to give us antonyms or related words, by replacing

```
rel=/r/Synonym
```

with the appropiate relation.

- Parsing the ConceptNet response and dealing with exceptions
  The ConceptNet response is a json containing all sorts of data. We just need the word representing the synonym, antonym or related word, so we parse it. We also deal with with the case when ConceptNet was not able to find a suitable replacing word and we signal this using the word "exceptie".

```
for resp in responseList:
        try:
                temp = resp['edges'][0]['@id']
                temp = temp.split("/en/")
                temp = temp[2].split("/")
                synonym = temp[0]
                synonyms.append(synonym)
        except IndexError:
                synonyms.append("exceptie")
```

- Creating the fake review
  Creating the fake review means replacing the original words with new words. Here we will also get rid of the underscores that ConceptNet will place between the words forming a single response.

```
for i in range(0, len(synonyms)):
        if synonyms[i] == "exceptie":
                continue
        else:
                if synonyms[i].find("_") != -1:
                        synonyms[i] = synonyms[i].replace("_", " ")
                fakeReview = fakeReview.replace(listaAdj[i],synonyms[i])
```

- Validating the review
  Validating the review means comparing the overall score of both the original and newly created review. If the difference between the two is bigger than 0.1, the fake review is considered invalid.
  Note: the value 0.1 was arbitrary chosen by me, after running multiple example through the Vader tool. And finally, we append the newly created review to the list of fake reviews.

```
review = synonyms.getAdvSyn(sentence)
                initScore = sa.analyzeSentence(sentence)['compound']
                fakeScore = sa.analyzeSentence(review)['compound']
                delta = initScore - fakeScore
                if delta < -0.1 or delta > 0.1:
                        print("The generated review is not valid")
                else:
                        synonyms.fakeReviews.append(review)
```

Chapter 6

Tool expressivity

6.1 Vader by NLTK

NLTK is the abbreviation on Natural Language Toolkit. It is widely used in Python when dealing with problems from the field of Natural Language Processing.
Vader is a Python module developed by NLTK, which deals very well with the POS tagging task, as well as the sentiment analysis task. It also stands out, because in the case of sentiment analysis, it takes into account the negations, punctuation and orthography.

6.2 ConceptNet

ConceptNet is the primary tool used and the one that gave me the idea of concentrating this project around the NLP field. However, it is a rather new tool, which does not come with much support for processing its dataset. It is still hard to extract only the information you want from their data base. Also, while it is recognized for its outstanding results in contextual language analysis, it did not yield outstanding results for my task, which is by far not as difficult as other tasks from the NLP field.

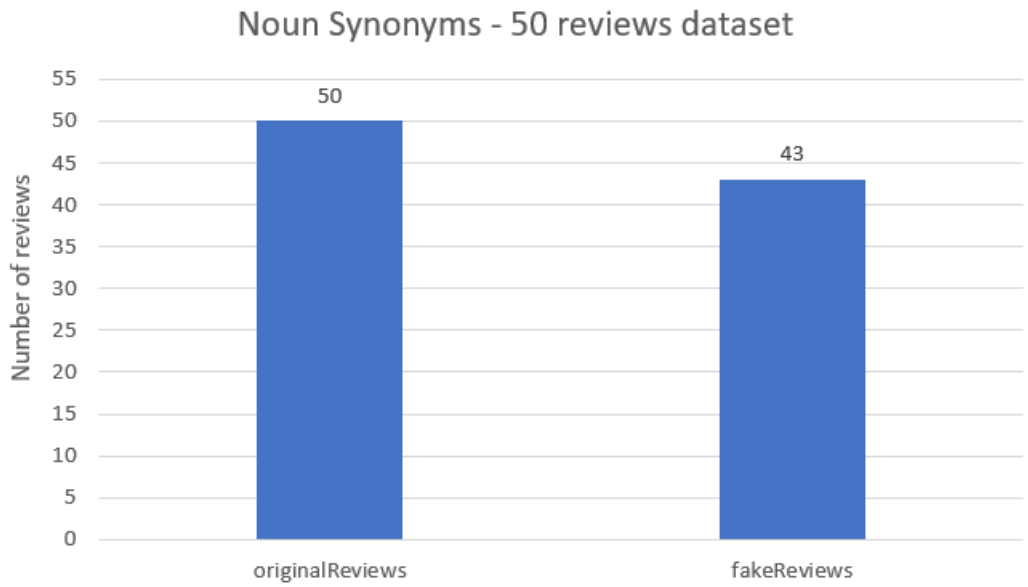
Chapter 7

Graphs and experiments

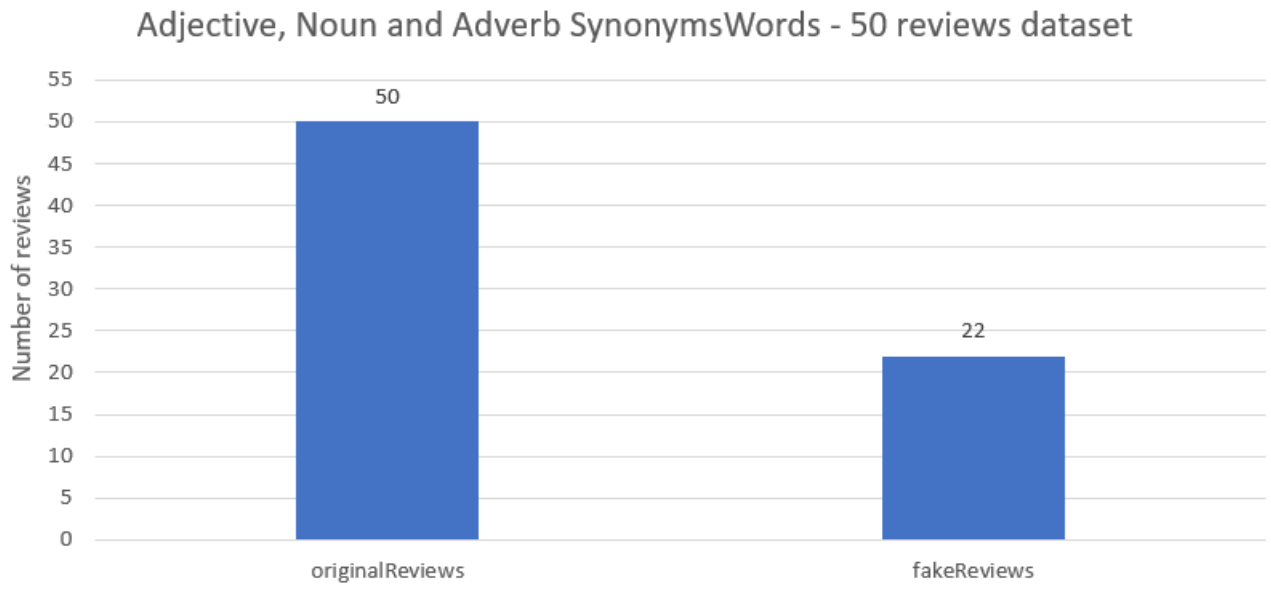
In this chapter I will display visual representations of the results, to make it easier to evaluate the obtained results.
I took into consideration 2 cases. One when the original reviews are positive (5 stars on Amazon) and one when the original reviews are negative (1 star on Amazon).
This approach would lead to several case analyses with their corresponding graphs, so I will show only the best and the worst case. **Original *Positive* Reviews**

7.1 Synonyms

7.1.1 Best Case – Noun Synonyms for 50 Reviews

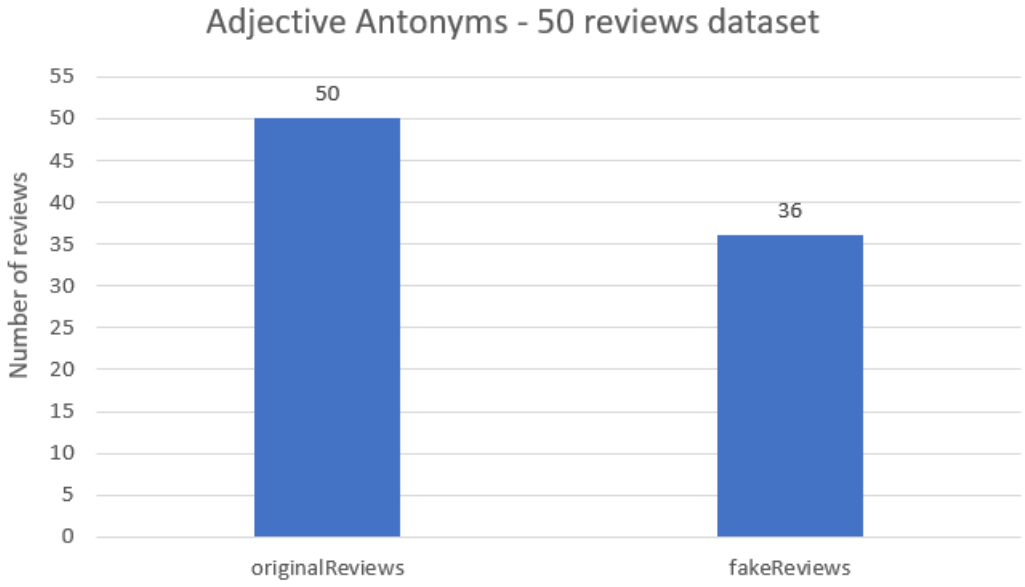


7.1.2 Worst Case – Adjective, Noun and Adverb Synonyms for 50 Reviews

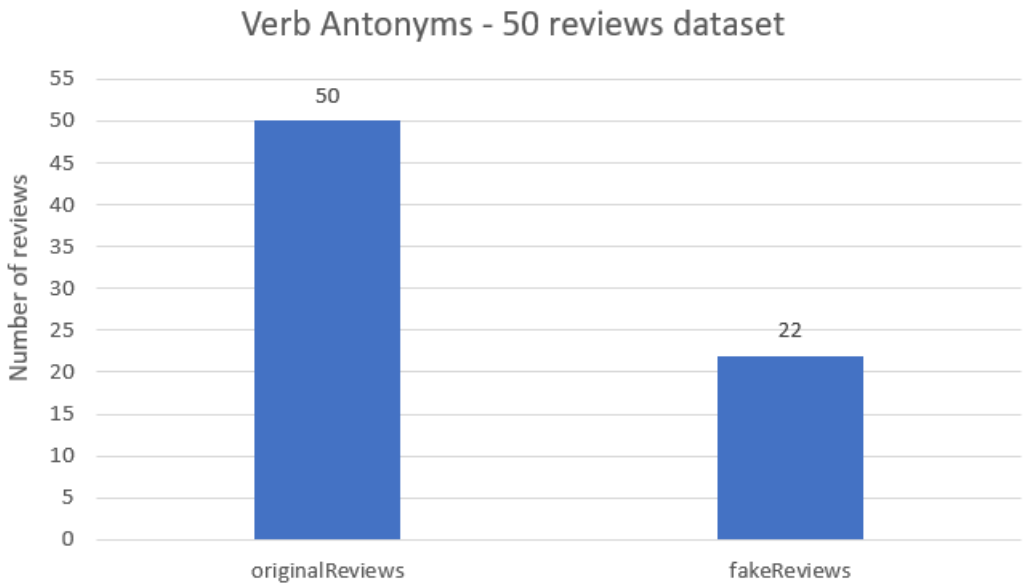


7.2 Antonyms

7.2.1 Best Case – Adjective Antonyms for 50 Reviews

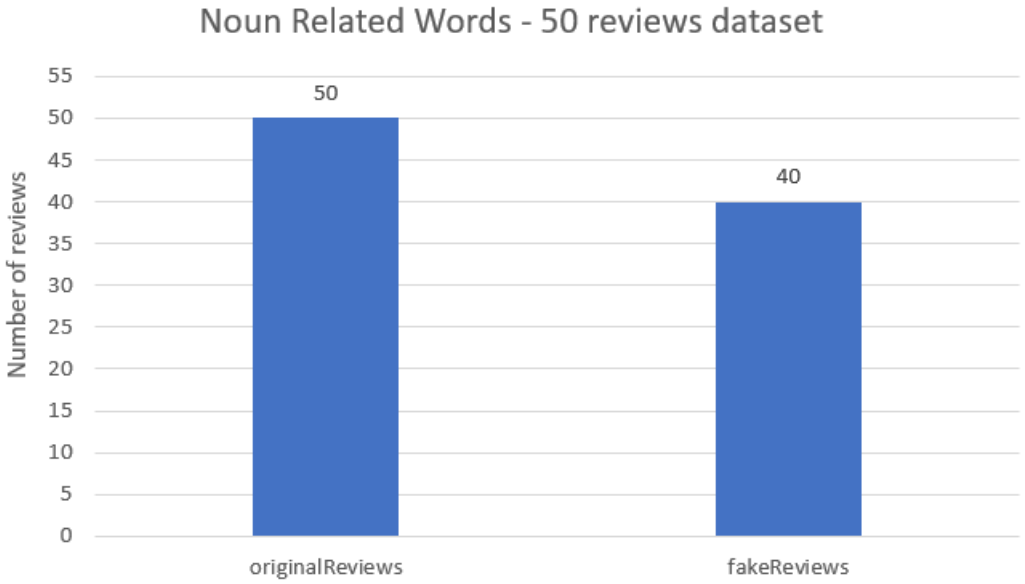


7.2.2 Worst Case – Verb Antonyms for 50 Reviews

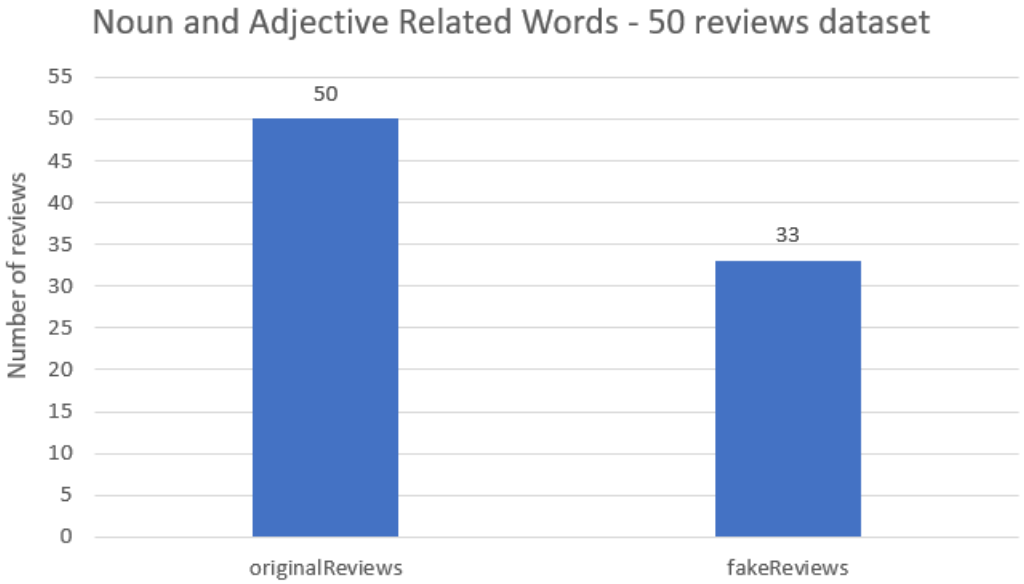


7.3 Related Words

7.3.1 Best Case – Noun Related Words for 50 Reviews



7.3.2 Worst Case – Noun and Adjective Related Words for 50 Reviews



Chapter 8

Documentation

8.1 Advantages and limitations of your solution

Some of the main **advantages** of this project are:

- It has a real life application. Nowadays, people relate more and more on the reviews they read online when buying something. And big companies know this. That is, some may want to increase their popularity, by uploading some positive, but auto-generated reviews. Some may even want to sabotage their competitors,by uploading for some products negative, but also fake, reviews.
- It makes use of 2 of the most powerful tools in the NLP field.
- It is not expensive when talking about storage space.

Some of the main **disadvantages** of this project are:

- it uses an internet connection. If the internet connection is not good, the application will take forever to run.
- It combines 2 tools, which are not specifically created to interact with one another. Especially if Vader does not work correct, because it is used in the first part of the project, it will really affect the correctness of the ConceptNet responses.
- It is very expensive from the duration(time) point of view. Retrieving data using the ConceptNet API takes sometimes way too long.
- It does not deal with special cases of words, such as past verbs, comparative and superlative adjectives and adverbs, etc.
- Even though from a computational point of view it, some fake reviews were classified as valid, a human being would be able to pick out some more invalid fake reviews from the final generated dataset. This is primary because of ConceptNet not giving appropriate words as responses, but sometimes it can be also Vader's fault.

8.2 Possible extensions of the current work

This project could be further extended with some more validation rules to make the fake reviews more believable.
A significant improvement would be also dealing with the special cases discussed above:

- verbs in 3rd person
- verbs at past tenses
- adjectives and adverbs in comparative form
- adjectives and adverbs in superlative form
- nouns in plural forms
- proper nouns in singular and plural form

Chapter 9

Conclusions

After developing this project and analyzing the results, I came up with the following conclusions:

- ConceptNet is not necessary the best tool for finding new words, like synonyms, antonyms or related words.
- Because this application is dependent of an internet connection, it can be really, really slow.
- Independent of the kind of words we want to generate (synonyms, related words), the worst case is always the one when all the words (adjectives, nouns (and adverbs)) were changed.
- Changing the nouns always yields the best results.
- Getting the antonyms of words gives the worst results.
- Working on positive reviews yields better results than working on negative reviews.

Chapter 10

Bibliography

1. ConceptNet site
2. ConceptNet API guide
3. NLTK Vader Sentiment Analysis
4. StackOverflow - for Python related questions
5. Tokenization and Parts of Speech(POS) Tagging in Python's NLTK library
6. Learning POS Tagging Chunking in NLP

Appendix A

Your original code

This section should contain only code developed by you, without any line re-used from other sources. This section helps me to correctly evaluate your amount of work and results obtained. Including in this section any line of code taken from someone else leads to failure of IS class this year. Failing or forgetting to add your code in this appendix leads to grade 1. Don't remove the above lines.

This section does not include all the code I have written. It shows the most important parts of code, as for the rest of the code I just changed some variables and some strings.

```
from sentimentAnalyzer import sentimentAnalyzer as sa
import requests

class synonyms:

    fakeReviews = []

    def __init__(self):
            pass

    @staticmethod
    def getAdjSyn(initSentence):
            listaAdj = []
            synonyms = []
            responseList= []
            fakeReview = initSentence[:]

            posList = sa.identifyPOS(initSentence)

            for (word, pos) in posList:
                if pos == 'JJ':
                    listaAdj.append(word)

        for adj in listaAdj:
                response = requests.get('http://api.conceptnet.io/query?
                start=/c/en/'+adj+'&rel=/r/Synonym&end=/c/en&
                limit=1').json()
                responseList.append(response)
```

```
        for resp in responseList:
                try:
                        temp = resp['edges'][0]['@id']
                        temp = temp.split("/en/")
                        temp = temp[2].split("/")
                        synonym = temp[0]
                        synonyms.append(synonym)
                except IndexError:
                        synonyms.append("exceptie")

        for i in range(0, len(synonyms)):
                if synonyms[i] == "exceptie":
                    continue
                else:
                    if synonyms[i].find("_") != -1:
                        synonyms[i] = synonyms[i].replace("_", " ")
                        fakeReview = fakeReview.replace(listaAdj[i],
                        synonyms[i])

                return fakeReview


@staticmethod
def validateReview(sentences, choice):
        for sentence in sentences:
            if choice == "adv":
                review = synonyms.getAdvSyn(sentence)
                initScore = sa.analyzeSentence(sentence)['compound']
                fakeScore = sa.analyzeSentence(review)['compound']
                delta = initScore - fakeScore

                if delta < -0.1 or delta > 0.1:
                    print("The generated review is not valid")
                else:
                    synonyms.fakeReviews.append(review)
return synonyms.fakeReviews


@staticmethod
def getVbAnt(initSentence):
    listaVb = []
    listaVbFlag = []
    negVerbs = []
    negationWords = ["not", "don't", "doesn't", "n't"]
    negationFlag = 0
    fakeReview = initSentence[:]

    posList = sa.identifyPOS(initSentence)
```

```
    for (word, pos) in posList:
        if word in negationWords:
            negationFlag = 1

        if negationFlag == 1 and word in negationWords:
            fakeReview = fakeReview.replace(word, "")

        if negationFlag == 0:
            if pos == 'MD':
                listaVbFlag.append((word, 1))
                listaVb.append(word)
            elif word == "is":
                listaVbFlag.append((word, 1))
                listaVb.append(word)
            elif pos == 'VBP':
                listaVbFlag.append((word, 0))
                listaVb.append(word)

        if negationFlag == 0:
            for (verb, flag) in listaVbFlag:
                if flag == 1:
                    newVerb = verb + " not"
                elif flag == 0:
                    newVerb = "don't " + verb
                negVerbs.append(newVerb)

        elif negationFlag == 1:
            for (word, pos) in posList:
                if word == "wo":
                    newVerb = "will"
                    fakeReview = fakeReview.replace(word, newVerb)
                elif word == "ca":
                    newVerb = "can"
                    fakeReview = fakeReview.replace(word, newVerb)
                elif word == "sha":
                    newVerb = "shall"
                    fakeReview = fakeReview.replace(word, newVerb)

        for i in range(0, len(negVerbs)):
            fakeReview = fakeReview.replace(listaVb[i], negVerbs[i])

    return fakeReview
```

Main code

```
from related import related as r
from antonyms import antonyms as a
from synonyms import synonyms as s
from sentimentAnalyzer import sentimentAnalyzer as sa
import requests
```

```
import os

originalReviewsSet = []
fakeReviewSet = []


folder = "E:/Romina/Facultate/An III/Sem I/IAI/FakeReviewsGenerator"
os.chdir(folder)

with open("originalPosRevs.txt", "r") as originalReviewsFile:
    originalReviewsSet = originalReviewsFile.read().splitlines()

print("Welcome! This is the Fake Review Generator (FKG) v1.0.")
print("You can choose to chage the words from a review with their SYNONYMS,
ANTONYMS or RELATED WORDS.")
print("For SYNONYMS, type s; for ANTONYMS type a; for RELATED WORDS type r")
typeOfRev = input("Please enter your choice: ")

if typeOfRev == "s":
    print("Now, you can choose to substitute only the ADJECTIVES, NOUNS,
    ADVERBS or ALL of them.")
    print("For ADJECTIVES type 'adj'; for NOUNS type 'noun';
    for ADVERBS type 'adv'; for ALL of them type 'all'")
    choice = input("Please enter your choice: ")
    fakeReviewSet = s.validateReview(originalReviewsSet, choice)
    print("Finish")
elif typeOfRev == "a":
    print("Now, you can choose to substitute only the ADJECTIVES or VERBS.")
    print("For ADJECTIVES type 'adj'; for VERBS type 'vb'")
    choice = input("Please enter your choice: ")
    fakeReviewSet = a.validateReview(originalReviewsSet, choice)
    print("Finish")
elif typeOfRev == "r":
    print("Now, you can choose to substitute only the ADJECTIVES, NOUNS
    or ALL of them.")
    print("For ADJECTIVES type 'adj'; for NOUNS type 'noun';
    for ALL of them type 'all'")
    choice = input("Please enter your choice: ")
    fakeReviewSet = r.validateReview(originalReviewsSet, choice)
    print("Finish")
else:
    print("Your choice is invalid")
    exit

with open("E:/Romina/Facultate/An III/Sem I/IAI/FakeReviewsGenerator/
fakeSynAll.txt", "w") as fakeRevsFile:
    for review in fakeReviewSet:
        fakeRevsFile.write("%s\n" % review)
print("Lengtth of original file is: ", len(originalReviewsSet))
print("Length of fake reviews file is: ", len(fakeReviewSet))
```

Appendix B

Quick technical guide for running your project

Step by step technical manual

1. In the main.py file, change the folder variable by writing the path to the folder where you have downloaded the projecy.
2. Run the main.py file and follow the instructions printed on the screen.