Universidad Autónoma de Madrid

Escuela Politécnica Superior

Máster Universitario en Ciencia de Datos

Procesamiento del Lenguaje Natural 2022-23

Lab assignments: Aspect opinion extraction

Starting date: Wednesday, 8th February 2023

Submission deadline: Tuesday, 22nd March 2023

The goal of this set of lab assignments is the extraction of **opinions about item aspects** from text reviews; specifically, positive and negative opinions about aspects of hotels existing in a collection of Yelp[1] reviews.

The following are examples aspects that may be considered along with some terms that can be used to refer to such aspects:

- **Amenities**: amenities, services, …
- **Atmosphere**: atmosphere, ambiance, …
- **Bedrooms**: room, bedroom, suite …
- **Facilities**: facilities, equipment, …
- **Location**: location, environment, neighborhood, …
- **Pool**: pool, swimming pool, …
- **Restrooms**: restrooms, toilets, …
- **Shopping**: shops, boutiques, stores, …
- **Staff**: staff, employees, receptionist, …

The example below shows a review where four (three positive and one negative) opinions about the shops, pool, bar and lobby of a hotel have been identified.

Review

Great hotel in Central Phoenix for a stay-cation, but not necessarily a place to stay out of town and without a car. Not much around the area, and unless you're familiar with downtown, I would rather have a guest stay in Old Town Scottsdale, etc. BUT if you do stay here, it's awesome. **Great** boutique rooms. **Awesome** pool that's happening in the summer. A **GREAT** rooftop patio bar, and a very **very** busy lobby with Gallo Blanco attached. A great place to stay, but have a car!

Polarity	*Aspect*	*Adjective*	*Word*
1.0	SHOPPING	great	boutique rooms
1.0	SWIMMING_POOL	awesame	pool
1.0	BAR	great	rooftop patio bar
-1.0	LOBBY	very busy	lobby

We will pursuit the extraction of such opinions by addressing **three main tasks**:

1. Generating a vocabulary with words (in principle, nouns) describing the considered aspects. The vocabulary could be created manually or semi-automatically, e.g., by retrieving synonyms from WordNet, a lexical database for the English language.
2. Performing a syntactic analysis of review sentences in order to identify the adjectives complementing nouns that belong to the aspect vocabulary. Such analysis, e.g. based on POS tagging and dependency parsing, could be conducted using the NLTK and CoreNLP tools.
3. Assigning a positive/negative polarity to the identified adjectives. This will be done exploiting opinion lexicons with [adjective, polarity] pairs.

[1] Yelp crowd-sourced reviews about businesses, https://www.yelp.com/

Goal

You are requested to perform the above tasks by means of a program in **Python** that uses the NLTK toolkit, https://www.nltk.org, https://www.nltk.org/howto, http://www.nltk.org/nltk_data.

The final output of the program must be a summary of the opinions made about a particular hotel by users from a given review dataset. You are free to decide the elements and representation of such summary, but at least it should contain the number of positive and negative opinions associated to each aspect of the hotel.

Assignments

Assignment 1: Review datasets

You are provided with a JSON file `yelp_hotels.json` containing 5,034 reviews generated by 4,148 Yelp users about 284 hotels.

You also have two additional JSON files, `yelp_beauty_spas.json` and `yelp_restaurants.json`, which contain Yelp reviews about beauty/spa resorts and restaurants.

Each review (JSON record) has the following fields:

- **reviewerID**: the identifier of the user who wrote the review
- **asin**: the identifier of the reviewed hotel
- **reviewText**: the text of the user's review about the hotel
- **overall**: the 1-5 Likert scale rating assigned by the user to the hotel

The following tasks are proposed:

- Task 1.1 - mandatory: loading all the hotel reviews from the Yelp hotel reviews file. See Appendix A
- Task 1.2 - optional [low difficulty]: loading line by line* the reviews from the Yelp beauty/spa resorts and restaurants reviews files
- Task 1.3 - optional [medium/high difficulty]: loading line by line* reviews on other domains (e.g., movies, books, phones, digital music, CDs and videogames) from McAuley's Amazon dataset[2]

* *Remark*: for assignments 1.2 and 1.3, it may be the case that not all the reviews can be loaded into memory. The JSON files should be read and processed line by line.

Assignment 2: Aspect vocabularies

You are provided with a CSV file `aspects_hotels.csv` containing examples of hotel aspects and associated terms[3].

You also have CSV files that contain aspect vocabularies for other domains: spa/beauty resorts, restaurants, movies, books, phones, digital music, CDs and videogames.

The following tasks are proposed:

- Task 2.1 - mandatory: loading (and printing on screen) the vocabulary of the `aspects_hotels.csv` file, and directly using it to identify aspect references in the reviews. In particular, the aspects terms could be mapped by *exact matching* with nouns appearing in the reviews. See Appendix B
- Task 2.2 - optional [low difficulty]: generating or extending the lists of terms of each aspect with synonyms extracted from WordNet. See Appendix E
- Task 2.3 - optional [low difficulty]: managing vocabularies for additional Yelp or Amazon domains. See assignments 1.2 and 1.3
- Task 2.4 - optional [high difficulty]: identifying hidden/implicit aspect references in reviews. For instance, the example review of page 1 has references to the hotel's `location` and `transportation` aspects, since there is "not much around the area" and going by car to the hotel is recommendable

[2] McAuley's Amazon product reviews dataset, https://jmcauley.ucsd.edu/data/amazon/
[3] You are free to add/remove/change the aspects and associated terms of the provided vocabulary.

Assignment 3: Opinion lexicons

Opinion lexicons have diverse information about the polarity of words, commonly used to express subjective opinions.

- good, helpful, kind, … (–)
- bad, unfavorable, boring, … (+)

In general:

- The lexicon words are adjectives, but can also include nouns, verbs, and adverbs
- The considered polarities take discrete values (e.g., *negative*, *neutral* or *positive*), but can also be measured by means of numeric vales (e.g., in the [-1,+1] range)

Advanced lexicons may consider more complex opinion dimensions (e.g., *arousal/intensity* – from calm/low to excited/high– and *valence/pleasantness level*), sentiment categories (e.g., *fear*, *anger*, *joy*, *sadness*, *acceptance*, *disgust*, *anticipation*, and *surprise*) and emotion models.

Opinion expressions may be positively/negatively intensified through modifiers:

- "The hotel staff was **quite** *friendly*" (moderate intensification)
- "The hotel staff was **very** *friendly*" (high intensification)
- "The hotel staff was **too** *friendly*" (low/moderate intensification + polarity modification)
- "The hotel staff was **much less** *friendly*" (high intensification + polarity modification)

The polarity of opinions could be (not) modified by:

- The negation of the lexicon words: "The hotel staff was **not** *friendly*" (polarity modification)
- The negation of sentences: "I **do not** think the hotel staff was *friendly*" (polarity modification)
  "I **do not** think the hotel staff was **not** *friendly*" (no polarity modification)

There may exist opinion words whose polarities are aspect- or domain-dependent:

- "*small* room" (–) vs. "*small* weight" (+) in the hotel and phone domains, respectively
- "*low* resolution" (–) vs. "*low* energy consumption" (+) in the camera domain

Moreover, there may be omitted or implicit references to aspects in subjective sentences:

- "The hotel was *expensive*" implies a "*high* price"
- "There is not much to do at the hotel" implies "*scarce* services"

The following tasks are proposed:

- Task 3.1 - mandatory: loading Liu's opinion lexicon composed of positive and negative words, accessible as an NLKT corpus, and exploiting it to assign the polarity values to aspect opinions in assignment 4. Instead of this lexicon, you are allowed to use others, such as SentiWordNet. See Appendix F
- Task 3.2 - optional [low/medium difficulty]: considering modifiers to adjust the polarity values of the aspect opinions in Assignment 4. The modifiers to use could be those provided with the NLTK Sentiment Analyzer (see Appendix G) and/or those given in `modifiers.csv`
- Task 3.3 - optional [medium/high difficulty]: considering negation of opinion words and negation of sentences to adjust the polarity values of the aspect opinions in Assignment 4

Assignment 4: Aspect opinions

Once the aspect vocabulary and opinion lexicons are loaded, the opinions about aspects have to be extracted from the reviews. For this purpose, POS tagging, constituency and dependency parsing could be used.

- POS tagging would allow identifying the adjectives in the sentences.
- Constituency and dependency parsing would allow extracting the relations between nouns and adjectives and adverbs.

See Appendix C and Appendix D for code examples.

The following tasks are proposed:

- Task 4.1 - mandatory: extracting the [aspect, aspect term, opinion word, polarity] tuples from the input reviews
- Task 4.2 - optional [low/medium difficulty]: extracting the [aspect, aspect term, opinion word, modifier, polarity] tuples from the input reviews, taking the modifiers of assignment 3.2
- Task 4.3 - optional [low/medium difficulty]: extracting the [aspect, aspect term, opinion word, isNegated, polarity] tuples from the input reviews, taking the modifiers of assignment 3.3

Assignment 5: Opinion summarization

To validate and evaluate the solutions implemented in previous tasks, you are finally proposed the following tasks:

- Task 5.1 - mandatory: visualizing on screen the aspect opinions (tuples) of a given review
- Task 5.2 - mandatory: visualizing on screen a summary of the aspect opinions of a given item. Among other issues, the total number of positive/negative opinions for each aspect of the item could be visualized
- Task 5.3 - optional [low]: conducting and reporting a manual evaluation of the implemented aspect opinion approach
    - Precision can be computed by checking the correctness of extracted aspect opinion tuples
    - Recall can be computed by checking whether real aspect opinions were not extracted by the implemented approach

---

Deliverable

The deliverable of the lab assignments consists of the following files:

- **`nlp2021-lab-XX.ipynb`**: a Jupyter notebook with the developed Python code. This code must be commented to ease its comprehension
- **`nlp2021-lab-XX.pdf`**: a short report listing the assignments done, discussing main issues of interest, and (if applicable) reporting evaluation results

Submit these files via Moodle within a zip file named **`nlp2021-lab-XX.zip`**, where XX has to be replaced accordingly with the team id, e.g., 01, 02, ...

Grade

The grade of the lab assignments will be computed as follows:

- Mandatory tasks: up to 7 points
    - 4 points will be assigned to tasks 1.1, 2.1, 3.1 and 4.1
    - 3 points will be assigned to tasks 5.1 and 5.2
- Optional tasks: up to 3 points
    - Depending on their difficulty, 2 or 3 tasks will be enough to achieve the maximum grade

Appendix A: Reading JSON files

The following code allows you to load in memory the Yelp hotel reviews:

```
import json

with open('yelp_dataset/yelp_hotels.json', encoding='utf-8') as f:
   reviews = json.load(f)
f.close()

numReviews = len(reviews)
print(numReviews, 'reviews loaded')

print(reviews[0])
print(reviews[0].get('reviewerID'))
```

```
{'reviewerID': 'qLCpuCWCyPb4G2vN-WZz-Q', 'asin': '8ZwO9VuLDWJOXmtAdc7LXQ', 'summary': 'summary',
'reviewText': "Great hotel in Central Phoenix for a stay-cation, but not necessarily a place to
stay out of town and without a car. Not much around the area, and unless you're familiar with
downtown, I would rather have a guest stay in Old Town Scottsdale, etc. BUT if you do stay here,
it's awesome. Great boutique rooms. Awesome pool that's happening in the summer. A GREAT rooftop
patio bar, and a very very busy lobby with Gallo Blanco attached. A great place to stay, but have
a car!", 'overall': 4.0}
qLCpuCWCyPb4G2vN-WZz-Q
5034 reviews loaded
```

You are free to use or adapt the above code for your solution.

If you want to load each review separately, you could use the function `f.readline()` to read each line from the file and the function `json.loads(line)` to parse the JSON content from a string. In this case, be careful with the "[" and "]" symbols at the beginning and end of the file, and the commas "," at the end of each line.

Appendix B: Reading CSV files

The following code allows you to load in memory an aspect-terms vocabulary:

```
f = open("aspects/aspects_hotels.csv", "r")
for l in f:
   tokens = l.rstrip('\n').split(',')
   print(tokens[0], tokens[1])
```

```
amenities amenity
amenities amenities
amenities services
atmosphere atmosphere
atmosphere atmospheres
atmosphere ambiance
atmosphere ambiances
...
```

You are free to use or adapt the above code for your solution.

Appendix C: POS tagging of sentences

The following code allows you to extract the POS tags for a given text (review):

```
import nltk

def pos_tagging(text):
   sentences = nltk.sent_tokenize(text)
   sentences = [nltk.word_tokenize(s) for s in sentences]
   sentences = [nltk.pos_tag(s) for s in sentences]
   return sentences

postagged_sentences = pos_tagging(reviews[0].get("reviewText"))

print(postagged_sentences[0])
```

```
[('Great', 'JJ'), ('hotel', 'NN'), ('in', 'IN'), ('Central', 'NNP'), ('Phoenix', 'NNP'), ('for',
'IN'), ('a', 'DT'), ('stay-cation', 'NN'), (',', ','), ('but', 'CC'), ('not', 'RB'), ('necessarily',
'RB'), ('a', 'DT'), ('place', 'NN'), ('to', 'TO'), ('stay', 'VB'), ('out', 'IN'), ('of', 'IN'),
('town', 'NN'), ('and', 'CC'), ('without', 'IN'), ('a', 'DT'), ('car', 'NN'), ('.', '.')]
```

You are free to use or adapt the above code for your solution.

You can find documentation in Chapter 5 'Categorizing and tagging words' of the 'Natural Language Processing with Python' book, https://www.nltk.org/book/ch05.html

---

Appendix D: Syntactic analysis of sentences

To extract related adjective and nouns that could refer to aspect opinions, you may use the grammar-based constituency parser provided in NLTK.

The following code shows an example of how to use such parser:

```
grammar = r"""
  JJNN: {<JJ>*<NN>+}   # chunk adjective and sequences noun
        {<JJ>*<NNP>+}  # chunk sequences of proper nouns
"""
cp = nltk.RegexpParser(grammar)

chunk_parse = cp.parse(pos_tagging('The tall green tree is in the wonderful Central Park')[0])

print(chunk_parse)

for child in chunk_parse:
   if isinstance(child, nltk.Tree):
      if child.label() == 'JJNN':
         print('child: ', child)
         for i in range(len(child)):
               print('\t', i, ':', child[i])
               for j in range(len(child[i])):
                  print('\t\t', j, ':', child[i][j])
```

```
(S
  The/DT
  (JJNN red/JJ tree/NN)
  is/VBZ
  the/DT
  (JJNN great/JJ Central/NNP Park/NNP))
child:  (JJNN tall/JJ green/JJ tree/NN)
    0 : ('tall', 'JJ')
       0 : tall
       1 : JJ
    1 : ('green', 'JJ')
       0 : green
       1 : JJ
    2 : ('tree', 'NN')
       0 : tree
       1 : NN
child:  (JJNN wonderful/JJ Central/NNP Park/NNP)
    0 : ('wonderful', 'JJ')
       0 : wonderful
       1 : JJ
    1 : ('Central', 'NNP')
       0 : Central
       1 : NNP
    2 : ('Park', 'NNP')
       0 : Park
       1 : NNP
```

You are free to use or adapt the above code for your solution. Please note that the grammar of the example is very simple, and may be not able to capture more complex dependencies between adjectives and nouns.

You can find documentation in Section 7.2 'Chunking' of the 'Natural Language Processing with Python' book, https://www.nltk.org/book/ch07.html

An alternative approach to find adjective-noun relations that correspond to aspect opinions can be implemented with the dependency parser of the Stanford CoreNLP Java toolkit.

The NLKT library allows accessing such parser by establishing a connection with the CoreNLP server, as follows:

```
dependency_parser = CoreNLPDependencyParser(url='http://localhost:9000')
```

For such purpose, you have to first download the server from the toolkit webpage:

https://stanfordnlp.github.io/CoreNLP/download.html

http://nlp.stanford.edu/software/stanford-corenlp-latest.zip

Once downloaded, you can run the service from the command console as follows:

```
stanford-corenlp-4.2.0>java -mx4g -cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer -port 9000 -timeout 15000
```

The following code shows an example of how to use such parser:

```
review = reviews[0].get("reviewText")

from nltk.parse.corenlp import CoreNLPDependencyParser

dependency_parser = CoreNLPDependencyParser(url='http://localhost:9000')

sentences = nltk.sent_tokenize(review)
for s in sentences:
   print(s)
   result, = dependency_parser.raw_parse(s)
   for head, relation, dependent in result.triples():
      print(head, relation, dependent)
```

```
Great hotel in Central Phoenix for a stay-cation, but not necessarily a place to stay out of town and without a car.
('hotel', 'NN') amod ('Great', 'JJ')
('hotel', 'NN') nmod ('Phoenix', 'NNP')
('Phoenix', 'NNP') case ('in', 'IN')
('Phoenix', 'NNP') compound ('Central', 'NNP')
('hotel', 'NN') nmod ('cation', 'NN')
('cation', 'NN') case ('for', 'IN')
('cation', 'NN') det ('a', 'DT')
('cation', 'NN') compound ('stay', 'NN')
('cation', 'NN') punct ('-', 'HYPH')
('hotel', 'NN') punct (',', ',')
('hotel', 'NN') conj ('place', 'NN')
('place', 'NN') cc ('but', 'CC')
('place', 'NN') advmod ('not', 'RB')
('place', 'NN') advmod ('necessarily', 'RB')
('place', 'NN') det ('a', 'DT')
('place', 'NN') dep ('stay', 'VB')
('stay', 'VB') mark ('to', 'TO')
('stay', 'VB') obl ('town', 'NN')
...
A GREAT rooftop patio bar, and a very very busy lobby with Gallo Blanco attached.
('attached', 'VBN') nsubj ('bar', 'NN')
('bar', 'NN') det ('A', 'DT')
('bar', 'NN') amod ('GREAT', 'JJ')
('bar', 'NN') compound ('rooftop', 'NN')
('bar', 'NN') compound ('patio', 'NN')
('bar', 'NN') punct (',', ',')
('bar', 'NN') conj ('lobby', 'NN')
('lobby', 'NN') cc ('and', 'CC')
('lobby', 'NN') det ('a', 'DT')
('lobby', 'NN') amod ('busy', 'JJ')
('busy', 'JJ') advmod ('very', 'RB')
...
```

You are free to use or adapt the above code for your solution.

You can find more information in the NLTK documentation:

https://www.nltk.org/api/nltk.parse.html#module-nltk.parse.stanford

Appendix E: WordNet

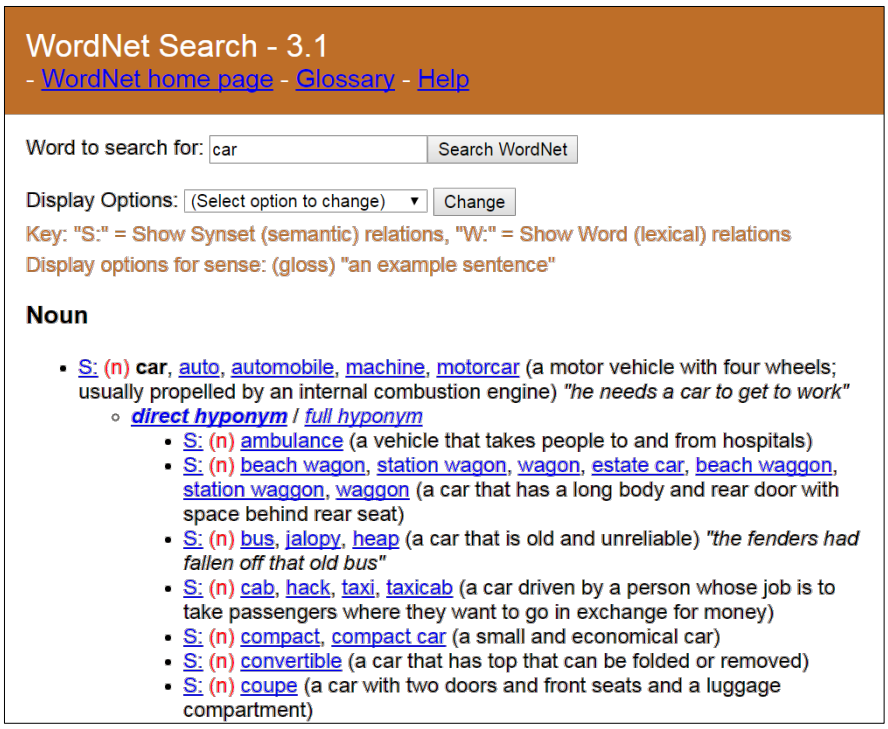
WordNet (http://wordnet.princeton.edu) is a large lexical database of English, which can be downloaded from:

https://wordnet.princeton.edu/

In WordNet, nouns, verbs, adjectives and adverbs are grouped into sets of synonyms (called *synsets*), each expressing a distinct concept/meaning. Synsets are interlinked by means of semantic and lexical relations, such as hypernym, hyponym and meronym. The resulting network of meaningfully related concepts makes it a useful tool for computational linguistics and natural language processing.

WordNet can be queried online:

http://wordnetweb.princeton.edu/perl/webwn



WordNet superficially resembles a thesaurus, in that it groups words together based on their meanings. However, there are some important distinctions. First, WordNet interlinks not just word forms - strings of letters - but specific senses of words. As a result, words that are found in close proximity to one another in the network are semantically disambiguated. Second, WordNet labels the semantic relations among words, whereas the groupings of words in a thesaurus do not follow any explicit pattern other than meaning similarity.

The main relation among words in WordNet is synonymy, as between the words *car* and *automobile*. Synonyms –words that denote the same concept and are interchangeable in many contexts– are grouped into sets called "synsets". Each of WordNet's synsets is linked to other synsets by means of a small number of "conceptual relations." Additionally, a synset contains a brief definition and, in most cases, one or more short sentences illustrating the usage examples of the synset members. Word forms with several distinct meanings are represented in as many distinct synsets. Thus, each form-meaning pair in WordNet is unique.

The most frequently encoded relation among synsets is the super-subordinate relation (also called as hyperonymy, hyponymy and is-a relation). It links more general synsets like {furniture, piece of furniture} to increasingly specific ones like {bed} and {bunkbed}. Thus, WordNet states that the category *furniture* includes *bed*, which in turn includes *bunkbed*; conversely, concepts like *bed* and *bunkbed* make up the category *furniture*. All noun hierarchies ultimately go up the root node {entity}. Hyponymy relation is transitive: if an *armchair* is a kind of *chair*, and if a *chair* is a kind of *furniture*, then an *armchair* is a kind of *furniture*.

WordNet distinguishes among "types" (common nouns) and "instances" (specific persons, countries and geographic entities). Thus, *armchair* is a type of *chair*, *Barack Obama* is an instance of a *president*. Instances are always leaf (terminal) nodes in their hierarchies.

Meronymy, the part-whole relation holds between synsets like {chair} and {back, backrest}, {seat} and {leg}. Parts are inherited from their superordinates: if a chair has legs, then an armchair has legs as well. Parts are not inherited "upward" as they may be characteristic only of specific kinds of things rather than the class as a whole: chairs and kinds of chairs have legs, but not all kinds of furniture have legs.

The following code shows an example of how to use WordNet with NLTK:

```
from nltk.corpus import wordnet as wn

synsets = wn.synsets('car')
for s in synsets:
   print(s.name())
   print('\t', s.definition())
   print('\t', s.examples())
   print('\t', s.lemma_names())
   print('\tHyponyms:')
   hyponyms = s.hyponyms()
   for h in hyponyms:
      print('\t\t', h.name(), ' => ', h.lemma_names())
   print('\tHyperyms:')
   hypernyms = s.hypernyms()
   for h in hypernyms:
      print('\t\t', h.name(), ' => ', h.lemma_names())
```

```
car.n.01
    a motor vehicle with four wheels; usually propelled by an internal combustion engine
    ['he needs a car to get to work']
    ['car', 'auto', 'automobile', 'machine', 'motorcar']
   Hyponyms:
       ambulance.n.01  =>  ['ambulance']
       beach_wagon.n.01  =>  ['beach_wagon', 'station_wagon', 'wagon', 'estate_car',...]
       bus.n.04  =>  ['bus', 'jalopy', 'heap']
       cab.n.03  =>  ['cab', 'hack', 'taxi', 'taxicab']
       compact.n.03  =>  ['compact', 'compact_car']
       coupe.n.01  =>  ['coupe']
       cruiser.n.01  =>  ['cruiser', 'police_cruiser', 'patrol_car', 'police_car',...]
       electric.n.01  =>  ['electric', 'electric_automobile', 'electric_car']
       jeep.n.01  =>  ['jeep', 'landrover']
       limousine.n.01  =>  ['limousine', 'limo']
       minicar.n.01  =>  ['minicar']
       pace_car.n.01  =>  ['pace_car']
       racer.n.02  =>  ['racer', 'race_car', 'racing_car']
       roadster.n.01  =>  ['roadster', 'runabout', 'two-seater']
       sedan.n.01  =>  ['sedan', 'saloon']
       sport_utility.n.01  =>  ['sport_utility', 'sport_utility_vehicle', 'S.U.V.', 'SUV']
       sports_car.n.01  =>  ['sports_car', 'sport_car']
       ...
   Hyperyms:
       motor_vehicle.n.01  =>  ['motor_vehicle', 'automotive_vehicle']
car.n.02
    a wheeled vehicle adapted to the rails of railroad
    ['three cars had jumped the rails']
    ['car', 'railcar', 'railway_car', 'railroad_car']
   Hyponyms:
       baggage_car.n.01  =>  ['baggage_car', 'luggage_van']
       cabin_car.n.01  =>  ['cabin_car', 'caboose']
       club_car.n.01  =>  ['club_car', 'lounge_car']
       handcar.n.01  =>  ['handcar']
       mail_car.n.01  =>  ['mail_car']
       passenger_car.n.01  =>  ['passenger_car', 'coach', 'carriage']
       van.n.03  =>  ['van']
   Hyperyms:
       wheeled_vehicle.n.01  =>  ['wheeled_vehicle']
car.n.03
    ...
```

You can find documentation in Section 2.5 'WordNet' of the 'Natural Language Processing with Python' book, https://www.nltk.org/book/ch02.html

Appendix F: Opinion lexicon

There exist several opinion lexicons that can be used to establish the polarity of subjective expressions.

Liu's lexicon[4] is composed of a list of positive words and a list of negative words. The following code shows an example of how to access such lists in NLTK:

```
from nltk.corpus import opinion_lexicon

negativeWords = opinion_lexicon.negative()
positiveWords = opinion_lexicon.positive()

print(negativeWords)
print(len(negativeWords))
print(positiveWords)
print(len(positiveWords))
```

```
['2-faced', '2-faces', 'abnormal', 'abolish', ...]
4783
['a+', 'abound', 'abounds', 'abundance', 'abundant', ...]
2006
```

You can also use other opinion lexicons, such as SentiWordNet[5], which is also accessible through NLTK.

http://www.nltk.org/howto/sentiwordnet.html

```
from nltk.corpus import sentiwordnet as swn

print(list(swn.senti_synsets('happy')))

polarity = swn.senti_synset('happy.a.01')
print('pos', polarity.pos_score(), 'neg', polarity.neg_score())
```

```
[SentiSynset('happy.a.01'),       SentiSynset('felicitous.s.02'),       SentiSynset('glad.s.02'),
SentiSynset('happy.s.04')]
pos 0.875 neg 0.0
```

Moreover, NLTK includes the Vader Opinion lexicon, which includes emojis, abbreviations and acronyms.

https://www.nltk.org/_modules/nltk/sentiment/vader.html

The following code allows accessing to the [word, polarity] tuples of the lexicon

```
f = nltk.data.load("vader_lexicon/vader_lexicon.txt")
lexicon = {}
for line in f.split("\n"):
   (word, polarity) = line.strip().split("\t")[0:2]
   lexicon[word] = float(polarity)
   print(word, lexicon[word])
```

```
$: -1.5
%) -0.4
%-) -1.5
&-: -0.4
&: -0.7
('-: 2.2
(': 2.3
((-: 2.1
...
aas 2.5
aayf 2.7
afu -2.9
alol 2.8
ambw 2.9
...
```

[4] Liu's opinion lexicon, https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html
[5] SentiWordNet, https://github.com/aesuli/SentiWordNet

Appendix G: Opinion polarity modifiers

There are words (mainly adverbs) that modify the polarity of subjective expressions. They can increase, decrease or even negate the polarity scores.

You are provided with a file `modifiers.csv` that contains examples of the above "modifier" words. Each of these modifiers has associated a numeric weight (-1, 0.5, 2) that could be used to change the polarity of related opinions.

```
absolutely,2   profoundly,2   amazingly,2    greatly,2      heavily,2      highly,2
hugely,2       immensely,2    ...
slightly,0.5   smoothly,0.5   solely,0.5     some,0.5       somehow,0.5    vaguely,0.5
virtually,0.5  weakly,0.5     ...
dubiously,-1   erroneously,-1 excessively,-1  inadequately,-1  insufficiently,-1  ...
```

You can use all/some of these modifiers for your solution.

You can also utilize the modifiers ('boosters') given in the NTLK Vader sentiment opinion module.

The module also has a method aimed to compute the polarity ('compound' score) of a sentence:

```
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.sentiment.vader import VaderConstants

constants = VaderConstants()
print(constants.BOOSTER_DICT)

analyzer = SentimentIntensityAnalyzer()
s = 'a funny joke'
print(s, ' => ', analyzer.polarity_scores(s))
s = 'a quite funny joke'
print(s, ' => ', analyzer.polarity_scores(s)) # could it be better annotated?
s = 'a very funny joke'
print(s, ' => ', analyzer.polarity_scores(s))
s = 'a not very funny joke'
print(s, ' => ', analyzer.polarity_scores(s))
s = 'a not funny joke'
print(s, ' => ', analyzer.polarity_scores(s))
s = 'it is a funny joke'
print(s, ' => ', analyzer.polarity_scores(s))
s = 'it is not a funny joke'
print(s, ' => ', analyzer.polarity_scores(s))
s = 'i do not think it is a funny joke'
print(s, ' => ', analyzer.polarity_scores(s)) # wrongly annotated
s = 'a too funny joke'
print(s, ' => ', analyzer.polarity_scores(s)) # is it wrongly annotated?
```

```
{'absolutely': 0.293, 'amazingly': 0.293, 'awfully': 0.293, 'completely': 0.293, 'considerably':
0.293, 'decidedly': 0.293, 'deeply': 0.293, 'effing': 0.293, 'enormously': 0.293, 'entirely': 0.293,
'especially': 0.293, 'exceptionally': 0.293, 'extremely': 0.293, 'fabulously': 0.293, 'flipping':
0.293, 'flippin': 0.293, 'fricking': 0.293, 'frickin': 0.293, 'frigging': 0.293, 'friggin': 0.293,
'fully': 0.293, 'fucking': 0.293, 'greatly': 0.293, 'hella': 0.293, 'highly': 0.293, 'hugely':
0.293, 'incredibly': 0.293, 'intensely': 0.293, 'majorly': 0.293, 'more': 0.293, 'most': 0.293,
'particularly': 0.293, 'purely': 0.293, 'quite': 0.293, 'really': 0.293, 'remarkably': 0.293, 'so':
0.293, 'substantially': 0.293, 'thoroughly': 0.293, 'totally': 0.293, 'tremendously': 0.293, 'uber':
0.293, 'unbelievably': 0.293, 'unusually': 0.293, 'utterly': 0.293, 'very': 0.293, 'almost': -0.293,
'barely': -0.293, 'hardly': -0.293, 'just enough': -0.293, 'kind of': -0.293, 'kinda': -0.293,
'kindof': -0.293, 'kind-of': -0.293, 'less': -0.293, 'little': -0.293, 'marginally': -0.293,
'occasionally': -0.293, 'partly': -0.293, 'scarcely': -0.293, 'slightly': -0.293, 'somewhat': -
0.293, 'sort of': -0.293, 'sorta': -0.293, 'sortof': -0.293}
a funny joke  =>  {'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': 0.6249}
a quite funny joke  =>  {'neg': 0.0, 'neu': 0.15, 'pos': 0.85, 'compound': 0.688}
a very funny joke  =>  {'neg': 0.0, 'neu': 0.15, 'pos': 0.85, 'compound': 0.688}
a not very funny joke  =>  {'neg': 0.702, 'neu': 0.298, 'pos': 0.0, 'compound': -0.5743}
a not funny joke  =>  {'neg': 0.811, 'neu': 0.189, 'pos': 0.0, 'compound': -0.5096}
it is a funny joke  =>  {'neg': 0.0, 'neu': 0.282, 'pos': 0.718, 'compound': 0.6249}
it is not a funny joke  =>  {'neg': 0.589, 'neu': 0.411, 'pos': 0.0, 'compound': -0.5096}
i do not think it is a funny joke  =>  {'neg': 0.0, 'neu': 0.495, 'pos': 0.505, 'compound': 0.6249}
a too funny joke  =>  {'neg': 0.0, 'neu': 0.164, 'pos': 0.836, 'compound': 0.6249}
```