Задание №1:
Анализ близости слов автоматическими методами.

Скачав файл “WordSim 363 Goldstandard” и воспользовавшись пакетом nltk доступа к WordNet, я составил программы подсчета близости слов методами lch, wup и jcn. Суть этих методов описана на сайте NLTK, я же буду последовательно приводить их ниже.

1. Метод lch:

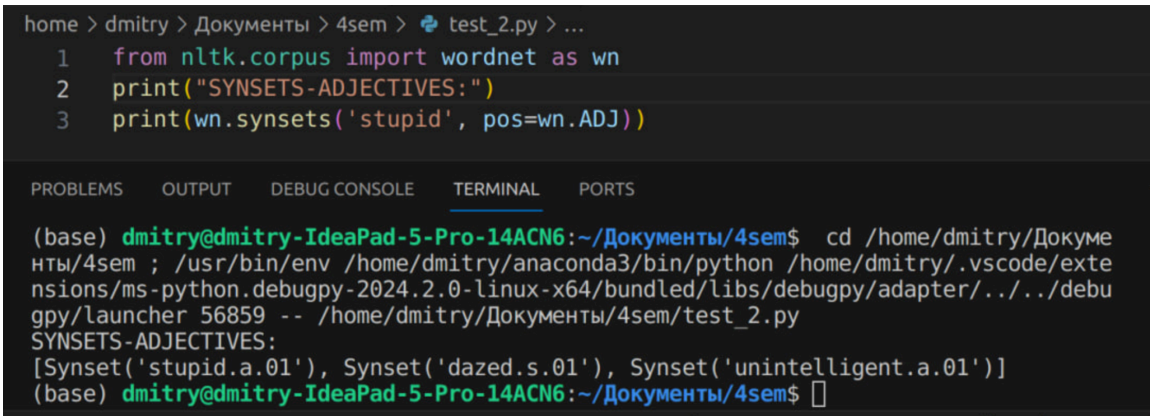
> `synset1.lch_similarity(synset2):` Leacock-Chodorow Similarity: Return a score denoting how similar two word senses are, based on the shortest path that connects the senses (as above) and the maximum depth of the taxonomy in which the senses occur. The relationship is given as -log(p/2d) where p is the shortest path length and d the taxonomy depth.

Суть программы для анализа слов из вышеупомянутого файла и сама программа приведены ниже (см. файл “test_lch.py”).

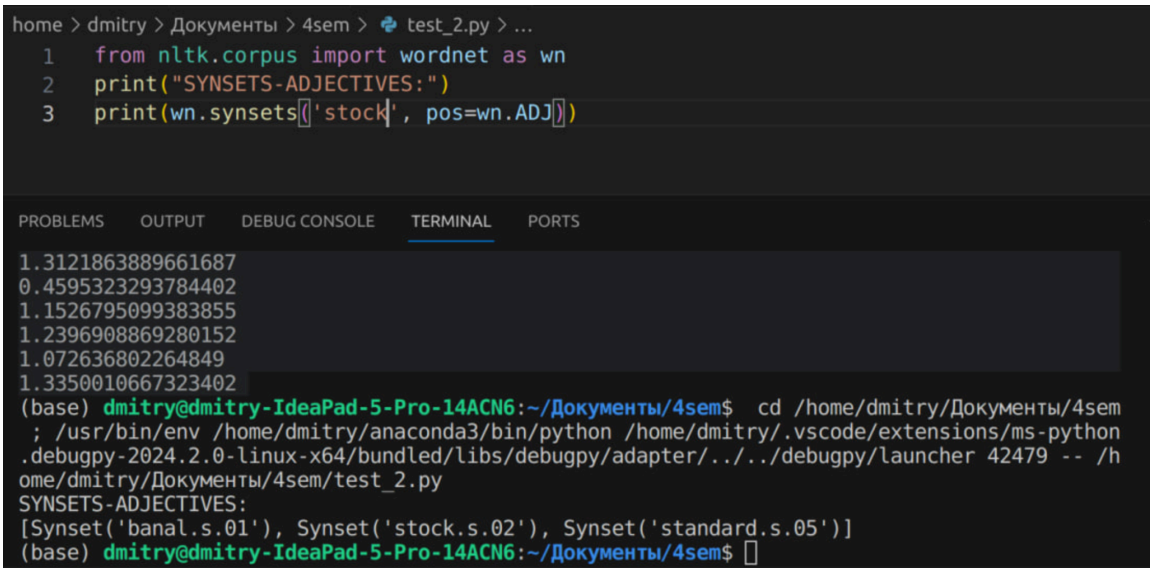
home > dmitry > Документы > 4sem > test.py > ...

```python
from nltk.corpus import wordnet as wn                    #импортируем данные

#цикл прохода по всем словам
for i in range(1,204):
    word1, word2 = input("").split()                     #вводим пары слов, разделяя их через пробел

    #получаем синсеты для каждого слова в паре двух слов - здесь рассматриваем часть речи NOUN
    synsets1 = wn.synsets(word1, pos=wn.NOUN)
    synsets2 = wn.synsets(word2, pos=wn.NOUN)
    #создаем список из чисел-близости пар слов для этих двух синсетов
    my_list = [synset1.lch_similarity(synset2) for synset1 in synsets1 for synset2 in synsets2]
    #вычисляем максимальное число близости этих двух слов (для части речи NOUN)
    if len(my_list) != 0:
        max_similarity1 = max(my_list)
    else:
        max_similarity1 = 0

    #аналогичные действия для части речи VERB
    synsets1 = wn.synsets(word1, pos=wn.VERB)
    synsets2 = wn.synsets(word2, pos=wn.VERB)
    my_list = [synset1.lch_similarity(synset2) for synset1 in synsets1 for synset2 in synsets2]
    if len(my_list) != 0:
        max_similarity2 = max(my_list)
    else:
        max_similarity2 = 0

    #аналогичные действия для части речи ADJECTIVE
    synsets1 = wn.synsets(word1, pos=wn.ADJ)
    synsets2 = wn.synsets(word2, pos=wn.ADJ)
    my_list = [synset1.lch_similarity(synset2) for synset1 in synsets1 for synset2 in synsets2]
    if len(my_list) != 0:
        max_similarity3 = max(my_list)
    else:
        max_similarity3 = 0
    #вычисляем максимальное число близости этих двух для всех частей речи в целом
    max_similarity = max(max_similarity1, max_similarity2, max_similarity3)
    #печать результата для каждой пары
    print(max_similarity)
```

Программа рабочая, однако с некоторыми словами возникала следующая проблема. При рассмотрении некоторых слов выяснилось, что в базе данных synset-ов с частью речи ADJECTIVE для этих слов находятся слова, у которых часть речи “не определена”. Обычно слова в базе данных хранятся в следующей форме: *(слово).(1-я буква части речи).(число)*. Соответственно, при рассмотрении частей речи ADJECTIVE второй аргумент должен быть “a”, однако в базе данных встречались слова, для которых 1-я буква соответствующей части речи была “s”, которая не позволяет определить слово ни к какой части речи. Ниже привожу примеры таких несоответствий.



Здесь часть речи слова “dazed” определена как “s” вместо “a”, из-за чего основная программа аварийно завершается.
Ниже еще один пример возможной некорректности в базе данных.



С учетом вышесказанного, было принято решение не рассматривать для некоторых слов synset-ы с частью речи ADJECTIVE. На общий результат для сравнения это практически не повлияло, т.к. таких “некорректных” слов оказалось около 5 из 203 пар слов, находящихся в файле “WordSim 363 Goldstandard”. Отмечу, что с оставшимися для анализа двумя методами возникала такая же проблема.

Итак, подав на вход основной программе все 203 пары слов, получаем 203 числа, которые мы запишем в таблицу, речь о которой пойдет ниже (см. пункт 4).

2. Метод wup:

> `synset1.wup_similarity(synset2):` Wu-Palmer Similarity: Return a score denoting how similar two word senses are, based on the depth of the two senses in the taxonomy and that of their Least Common Subsumer (most specific ancestor node). Note that at this time the scores given do **not** always agree with those given by Pedersen’s Perl implementation of Wordnet Similarity.
>
> The LCS does not necessarily feature in the shortest path connecting the two senses, as it is by definition the common ancestor deepest in the taxonomy, not closest to the two senses. Typically, however, it will so feature. Where multiple candidates for the LCS exist, that whose shortest path to the root node is the longest will be selected. Where the LCS has multiple paths to the root, the longer path is used for the purposes of the calculation.

Суть программы и сама программа приведены ниже (см. файл “test_wup.py”).

home > dmitry > Документы > 4sem > test_wup.py > ...

```python
from nltk.corpus import wordnet as wn                #импортируем данные

#цикл прохода по всем словам
for i in range(1,204):
    word1, word2 = input("").split()                 #вводим пары слов, разделяя их через пробел

    #получаем синсеты для каждого слова в паре двух слов - здесь можно рассматривать все части речи сразу
    synsets1 = wn.synsets(word1)
    synsets2 = wn.synsets(word2)

    #создаем список из чисел-близости пар слов для этих двух синсетов
    my_list = [synset1.wup_similarity(synset2) for synset1 in synsets1 for synset2 in synsets2]

    #вычисляем максимальное число близости этих двух слов
    if len(my_list) != 0:
        max_similarity = max(my_list)
    else:
        max_similarity = 0

    #печать результата для каждой пары
    print(max_similarity)

```

Подав на вход программе данные, получаем результат и записываем его в таблицу.

3. Метод jcn:

`synset1.jcn_similarity(synset2, ic):` Jiang-Conrath Similarity Return a score denoting how similar two word senses are, based on the Information Content (IC) of the Least Common Subsumer (most specific ancestor node) and that of the two input Synsets. The relationship is given by the equation 1 / (IC(s1) + IC(s2) - 2 * IC(lcs)).

Суть программы и сама программа приведены ниже (см. файл “test_jcn.py”).

home > dmitry > Документы > 4sem > test_jcn.py > ...

```
#импортируем данные
from nltk.corpus import wordnet as wn
from nltk.corpus import wordnet_ic
brown_ic = wordnet_ic.ic('ic-brown.dat')

#цикл прохода по всем словам
for i in range(1,204):
    word1, word2 = input("").split()            #вводим пары слов, разделяя их через пробел
    #получаем синсеты для каждого слова в паре двух слов - здесь рассматриваем часть речи NOUN
    synsets1 = wn.synsets(word1, pos=wn.NOUN)
    synsets2 = wn.synsets(word2, pos=wn.NOUN)
    #создаем список из чисел-близости пар слов для этих двух синсетов
    my_list = [synset1.jcn_similarity(synset2, brown_ic) for synset1 in synsets1 for synset2 in synsets2]
    #вычисляем максимальное число близости этих двух слов (для части речи NOUN)
    if len(my_list) != 0:
        max_similarity1 = max(my_list)
    else:
        max_similarity1 = 0
    #аналогичные действия для части речи VERB
    synsets1 = wn.synsets(word1, pos=wn.VERB)
    synsets2 = wn.synsets(word2, pos=wn.VERB)
    my_list = [synset1.jcn_similarity(synset2, brown_ic) for synset1 in synsets1 for synset2 in synsets2]
    if len(my_list) != 0:
        max_similarity2 = max(my_list)
    else:
        max_similarity2 = 0
    #аналогичные действия для части речи ADJECTIVE
    synsets1 = wn.synsets(word1, pos=wn.ADJ)
    synsets2 = wn.synsets(word2, pos=wn.ADJ)
    my_list = [synset1.jcn_similarity(synset2, brown_ic) for synset1 in synsets1 for synset2 in synsets2]
    if len(my_list) != 0:
        max_similarity3 = max(my_list)
    else:
        max_similarity3 = 0
    #вычисляем максимальное число близости этих двух для всех частей речи в целом
    max_similarity = max(max_similarity1, max_similarity2, max_similarity3)
    #печать результата для каждой пары
    print(max_similarity)
```

Подав на вход программе данные, получаем результат и записываем его в таблицу.

4. Рассмотрим сформированную таблицу (см. таблицу по ссылке https://docs.google.com/spreadsheets/d/1pU1Fwt-eCbOsgluK4GI57NEiOuJ0EHoWi_-5_JIx7CM/edit?usp=sharing):
    - В столбцах A и B находятся соответствующие пары слов;
    - В столбце C находятся значения близости слов (similarity), определенные т.н. стандартом в файле “WordSim 363 Goldstandard” (similarity);
    - В столбцах D, E, F находятся значения близости слов, полученные методами lch, wup, jcn соответственно;

- Получим меру Спирмена (несложная идея взята с сайта https://www.codecamp.ru/blog/spearman-rank-correlation-google-sheets/?ysclid=ltyz1y076k177455617). В столбцах G, H, I, J находятся значения, полученные при вычислении спец.функции =РАНГ.СР() для стандарта, методов lch, wup, jcn соответственно. Далее, в отдельных ячейках строки 8 и столбцов L, M, N находится полученная мера Спирмена для методов lch, wup, jcn соответственно с использованием спец.функции =КОРРЕЛ().
- Отметим, что коэффициент Спирмена, может принимать значение от -1 до +1, т.е. от идеальной отрицательной связи между значениями до идеальной связи.

5. Анализ полученных данных.

   Данный вопрос касается проблемы семантического анализа слов и их схожести с точки зрения человека и автоматизированных методов, таких как lch, wup и jcn. Эти методы основываются на структуре таксономии, такой как WordNet, для определения степени близости между словами.

   Итак, рассмотрим причины, почему некоторые близкие по значению слова для человека оказались далекими от автоматического определения методами.

   Одно слово может иметь несколько значений в зависимости от контекста, и автоматические методы могут не учитывать этот фактор. Автоматические методы, как правило, основываются на статической структуре таксономии, которая, к тому же, является ограниченной. Помимо этого, некоторые слова могут быть близкими по значению для человека из-за различных идиом, которые автоматические методы могут не учесть. Также могут иметь место ситуации, когда слова морфологически близки, но имеют разные значения, что также приводит к неточностям.

   Для наиболее точного определения близости слов следует использовать сложные подходы, такие как нейронные сети и глубокое обучение, которые могут учитывать контекст и нюансы значений слов.