Анализ данных

Хашин С.И.
http://math.ivanovo.ac.ru/dalgebra/Khashin/index.html

Ивановский университет

.

Обработка текстов
Natural Language Processing (NLP)

.

Иваново-2023

План

Словари

Словарь — неупорядоченная структура данных, которая позволяет хранить пары «ключ — значение».

```
d = {'в': 1177, 'и': 980, 'на': 572, 'не': 365, }
print(d)
d['и'] += 1    # изменить значение
del d['в']     # удалить гнездо
d['q'] += 1    # ошибка!
d[word] = d.get(word, 0) + 1
for key in d.keys(): print(key)
for key, value in d.items():
    print(key, '->', value)
L = list(d.items())     # список пар (ключ,значение)
L.sort(key=lambda el: el[1]) # сортировка по значению
```

Словари. Задание 0

Дан словарь вида (число $\rightarrow$ число).
Напишите функцию f_00(d), которая по словарю d находит сумму всех положительных значений.

Словари. Задание 1

Пусть X — вектор из целых чисел, например:
`X = np.random.randint(0,10**4, size=10**5)`
Напишите функцию f_01(X), которая по вектору X возвращает словарь:
число $\rightarrow$ сколько раз оно встретилось в векторе X

Словари. Задание 2

Напишите функцию f_02(X1, X2), которая «объединяет» два словаря:
Если ключ n есть только в одном словаре, он входит в ответ.
Если ключ n есть в обоих словарях, он входит в ответ с суммарным значением.
В Python предусмотрено объединение словарей:

```
merged_dict = dict1 | dict2
```

Однако если в словарях есть одинаковые ключи, ключу в объединенном словаре будет присвоено значение из второго словаря.

Словари. Задание 2a

Напишите функцию f_02a(d), которая по словарю d (ключ: значение) строит «обратный» словарь: (значение:ключ)

Частотный словарь

Дана строка s="'<title> Дмитрий Геннадьевич Брашнов Удивительная астрономия </title><section><title> Предисловие </title> Звездное небо привлекало внимание людей с незапамятных времен. Сколько легенд и мифов, сколько суеверий и пророчеств связано с ним! "'

Напишите функцию f_02b(s), которая по строке строит частотный словарь (символ $\rightarrow$ частота_появления).

Сортировка частотного словаря

Дан словарь d вида (слово $\rightarrow$ частота_появления).
Напишите функцию f_02c(d), которая по словарю d такого вида строит словарь (слово $\rightarrow$ число), где «число» — номер слова по убыванию частоты его появления.

Словари. Ценность буквы

В некоторой игре каждой русской букве задана ценность:

```
points_ru = {1:'АВЕИНОРСТ',
        2:'ДКЛМПУ',
        3:'БГЁЬЯ',
        4:'ЙЫ',
        5:'ЖЗХЦЧ',
        8:'ШЭЮ',
        10:'ФЩЪ'}
```

Напишите функцию f_03(d), которая по словарю такого вида строит «обратный» словарь:
буква $\rightarrow$ её ценность.
`points: {'А':1, 'Б':3, ... }`

Словари. Ценность слова

По словарю points из предыдущей задачи найдите ценность каждого слова. То есть надо написать функцию f_04(d, w), которая по словарю в такого вида и слову w находит «ценность» слова.

Словари, укладка рюкзака

Задание 5. Собираем рюкзак для похода. Даны вещи и их вес в граммах в словаре things:

```
things = {'зажигалка': 20, 'компас': 100, 'фрукты': 500,
   'рубашка': 300, 'термос': 1000, 'аптечка': 200,
   'куртка': 600, 'бинокль': 400, 'удочка': 1200,
   'салфетки': 40, 'бутерброды': 820, 'палатка': 5500,
   'спальный мешок': 2250, 'жвачка': 10}

ves = 10000  # Предельный вес рюкзака
```

Напишите функцию f_05(things, weight), которая по словарю такого вида и предельному весу рюкзака строит список вещей, которые в него поместятся.

Словари, права доступа

Задание 6. Над каждым файлом можно производить определенные действия:
запись – W;
чтение – R;
запуск – X.
Параметры файлов заданы в виде словаря:

```
files = {'python.exe':'X', 'book.txt':'RW',
    'notebook.exe':'RWX')
```

Запросы к файлам заданы в виде списка пар (файл,операция):

```
operations = (('read', 'python.exe'),
('read','book.txt'), ('write','notebook.exe'),
('execute','notebook.exe'), ('write','book.txt'))
```

Словари, права доступа

Задание 6. Напишите функцию f_06(files, operations), которая по словарю files и списку операций operations возвращает список True/False, возможно ли выполнение операции. В примере:
[False, True, True, True, True]

Словари, продажи

Задание 7. Дан список продаж онлайн-магагина в виде списка троек:
(покупатель, товар, количество), например:

```
pokupki= [
[('Сергей', 'Карандаш', 3),
[('Андрей', 'Тетрадь', 5),
[('Юлия', 'Линейка', 1),
[('Сергей', 'Ручка', 2),
[('Юлия', 'Книга', 4),
```

Напишите функцию f_07(pokupki), которая по списку покупок pokupki возвращает словарь со списком покупок каждым покупателем. В примере:

```
{'Андрей':[('Тетрадь',5)],
 'Сергей':[('Карандаш', 3), ('Ручка',2)],
 'Юлия': [('Книга', 4), ('Линейка', 1)] }
```

Русские буквы

```
import re
text = '</p></title><section><title><p>Ё Предисловие</p></t
print(text)
text = text.lower()  # маленькие буквы
text = text.replace('ё', 'е') # замена ё на е
print(text)
text = re.sub("[^а-я]", " ", text)
print('->', text, '<--')  # небуквы заменили на пробелы
words = text.split()      # разрезали текст на слова
for word in words: print(word) # одно слово в строку
```

Читаем файл

```
import re
with open('astro.fb2', 'r', encoding='utf-8') as f:
    text = f.read().lower()
print('len=', len(text))
text = text.replace('ё', 'е')
print(text[1200:1400])
text = re.sub("[^а-я]", " ", text)
words = text.split()
for word in words[:10]: print(word)
```

Частотный словарь

Уже имеем список слов words.

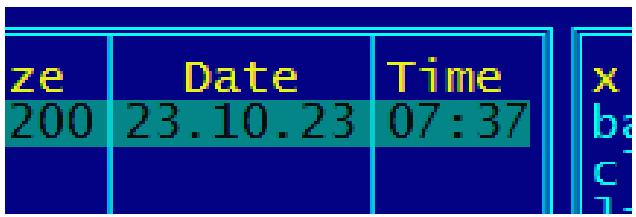
```
d = {} # пустой словарь: слово->частота
for word in words:
    d[word] = d.get(word, 0) + 1
# частотный словарь готов. print(d) осторожно
L = list(d.items())  # список пар (слово, частота)
> [('дмитрий', 2), ('геннадьевич', 2), ('брашнов', 2), ...
print('L init:', L[:10])
# сортировка словаря по убыванию частоты
L.sort(key = lambda x: -x[1])
for word, freq in L[:10]:  # первые 10 слов
    print(f'{freq:10d} {word}')
```

Частотный словарь

Частота	слово
1177	в
990	и
572	на
365	не
362	что
306	с
276	из
203	от
201	по
200	а

Кодировка текстов 866, 1251

cp-866(русская),OEM,DOS	cp-1251 (WIN)
0x80: АБВГДЕЖЗИЙКЛМНОП	**********
0x90: РСТУФХЦЧШЩЪЫЬЭЮЯ	**********
0xa0: абвгдежзийклмноп	********Ё*******
0xb0: **********	********ё*******
0xc0: **********	АБВГДЕЖЗИЙКЛМНОП
0xd0: **********	РСТУФХЦЧШЩЪЫЬЭЮЯ
0xe0: рстуфхцчшщъыьэюя	абвгдежзийклмноп
0xf0: Ёё**********	рстуфхцчшщъыьэюя



Кодировка koi-8r

```
cp-20866, koi-8r (Unix)

0x80: ****************
0x90: ****************
0xa0: ***ё************
0xb0: ***Ё************
0xc0: юабцдефгхийклмно
0xd0: пярстужвьызшэщчъ
0xe0: ЮАБЦДЕФГХИЙКЛМНО
0xf0: ПЯРСТУЖВЬЫЗШЭЩЧЪ
```

utf-8

- 0xxxxxxx [0-127] — один байт, 128
- 110xxxxx [192-223] — два байта, $32 * 64 = 2048$
- 1110xxxx [224-239] — 3 байта, $16 * 64^2 = 2^{16}$
- 11110xxx [240-247] — 4 байта, $8 * 64^3 = 2^{21}$
- 111110xx [248-251] — 5 байт, $4 * 64^4 = 2^{26}$

Все байты, кроме начального имеют вид: 10xxxxxx, то есть в десятичном виде от 128 до 191.

Частоты русских букв

а	б	в	г	д	е	ж	з
5095,	1380,	3509,	1224,	2477,	5259,	833,	1375,
и	**й**	**к**	**л**	**м**	**н**	**о**	**п**
3290,	286,	2746,	3310,	1851,	4925,	6699,	2261,
р	**с**	**т**	**у**	**ф**	**х**	**ц**	**ч**
3654,	3746,	4425,	1668,	157,	496,	311,	1188,
ш	**щ**	**ъ**	**ы**	**ь**	**э**	**ю**	**я**
616,	243,	22,	765,	429,	286,	219,	791

Здесь сумма частот $65536 = 2^{16}$.

Распознавание кодировок

Читаем всё содержимое файла в байтовую строку:

```
open('fname.ext', 'rb') as fh:
    content = fh.read()
```

Предположим, что кодировка cp-866, cp-1251, koi8-r, utf-8. Для каждой из них находим вектор частот букв.
Выбираем наиболее подходящую кодировку.
Например, выбираем ту, у которой скалярное произведение с таблицей частот наибольшее.

Читаем файл в разных кодировках

```
try:
  with open('astro_1251.fb2','r',encoding='utf-8') as f:
    txt = f.read()
    print('utf8:', txt[1200:1300])
except UnicodeError:
  print('not utf-8 file')

with open('astro_1251.fb2','r',encoding='cp1251') as f:
  txt_1251 = f.read()
print('1251:', txt_1251[1200:1300])
```

Читаем файл

Пусть мы уже знаем, что файл в кодировке utf-8.

```
import re
with open('astro.fb2', 'r', encoding='utf-8') as f:
    text = f.read().lower()
print('len=', len(text))
print(text[1200:1400])
text = text.lower()
text = text.replace('ё', 'е')
print(text[1200:1400])
text = re.sub("[^а-я]", " ", text)
words = text.split()
for word in words[:10]: print(word)
```

Удалить C-комментарии

В одну строку:

```
import re
s = ''' 123545665 // фдлвоы
asdfsadffd
  1242354 // фжлдыова
'''
s1 = re.sub(r'//.*', ' ', s)
#              1123
print(s1)
```

```
1 - символ '/'
2 - . любой символ
3 - * - повторитель предыдущего любое количество раз
23- .* любой символ много раз
```

Удалить многострочные C-комментарии

```
s = '''1234 /* asdf
  asdk;fj
  asdf */ 7890'''
s1 = re.sub(r'/\*.*?\*/', ' ', s, flags=re.DOTALL)
#              1 2345 67
print(s1)
1 - символ '/'
2 - \* символ '*'
3 - . любой символ
4 - * - повторитель предыдущего любое количество раз
34- .* любой символ много раз
5 - ? лекарство от жадности
6 - \* символ '*'
7 - символ '/'
flags=re.DOTALL - строки многострочные
```

Задание

1. Прочитать все C-файлы из папки.
2. Удалить из каждого комментарии.
3. Оставить от программы только идентификаторы разделённые пробелами.
4. Постройте частотный словарь: сколько раз какой идентификатор появился в программе.
5. Упорядочить по убыванию частоты.
6. Построить график: X: номер по убыванию частоты, Y: частота появления.

NLP(Natural language processing)

Основные задачи:

- определение темы/автора/настроения,тональности
- машинный перевод
- аннотирование
- чат-бот, ответы на вопросы
- справочная / экспертная система
- описание изображения

Некоторые проблемы

- Избыточность текста.
- Синонимы, опечатки, коверканье языка.
- Многозначность / омонимы. «Эти типы стали есть в цехе»
- Новые слова, сленг.
- Устойчивые выражения, профессиональный сленг, контекст. «бабье лето»

Стемминг Портера

Стемминг (англ. stemming — находить происхождение) — это процесс нахождения основы слова для заданного исходного слова. Алгоритм не использует список основ слов, а лишь, применяет последовательно ряд правил, отсекает окончания и суффиксы, основываясь на особенностях языка.Поэтому работает быстро, но не всегда верно.
На Питоне: Porter.py

```
from Porter import Porter
print(Porter.stem('устойчивость')) # устойчив
print(Porter.stem('Абажуры'))      # абажур
```

Задание. Найдите 10 различных слов, у которых одинаковые основы.

Частотный словарь

```
import re
from Porter import Porter
with open('astro.fb2', 'r', encoding='utf-8') as f:
    text = f.read().lower()
text = text.replace('ё', 'е')
text = re.sub("[^а-я]", " ", text)
words = text.split() # список слов малыми буквами
for i, word in enumerate(words):
    words[i] = Porter.stem(word) # только основы слов
d = {} # пустой словарь: слово->частота
for word in words:
  d[word] = d.get(word, 0) + 1
L = list(d.items())
L.sort(key = lambda x: -x[1])
for word, freq in L[:10]: # первые 10 слов
    print(f'{freq:10d} {word}')
```

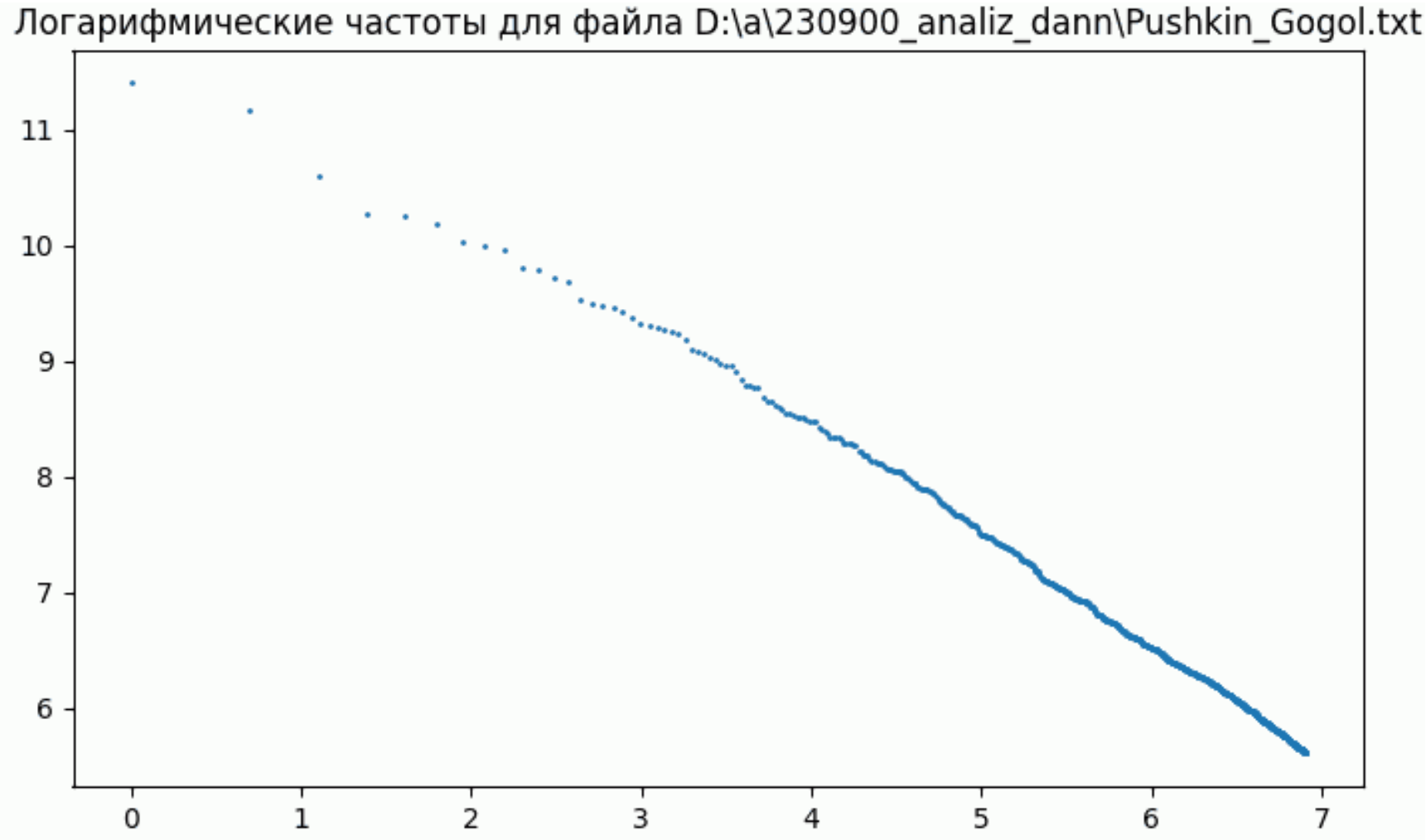
Задание

1. Найдите 30 самых частых основ слов (через стемминг Портера) и их частоты для файл astro.fb2
2. Найдите 30 самых редких основ слов (через стемминг Портера) и их частоты для файл astro.fb2

Частотный словарь

```
Всего слов  34598 , различных  4615
k:   30, fr:  117, sum= 8738, 25.26%
k:  100, fr:   52, sum=14108, 40.78%
k:  300, fr:   21, sum=20641, 59.66%
k: 1000, fr:    6, sum=28041, 81.05%
k: 1500, fr:    3, sum=30171, 87.20%
k: 2000, fr:    2, sum=31467, 90.95%
k: 3000, fr:    1, sum=32983, 95.33%
```


Логарифмические частоты для файла astro.fb2
7
6
5
4
3
2
0
1
2
3
4
5
6
7


Логарифмические частоты для файла D:\a\230900_analiz_dann\Pushkin_Gogol.txt
11
10
9
8
7
6
0
1
2
3
4
5
6
7

pymorphy2

pymorphy2 - морфологический анализатор для русского языка, умеет приводить слова к нормальной форме, т.е. к такой, как оно записано в словаре:
для глаголов это будет неопределённая форма;
для существительных — ед.число, именительный падеж;
для прилагательных — ед.число, им.падеж, мужской род.
Korobov M.: Morphological Analyzer and Generator for Russian and Ukrainian Languages // Analysis of Images, Social Networks and Texts, pp 320-332 (2015).
Установка:

```
pip install pymorphy2
```

Работа:

```
import pymorphy2
morph = pymorphy2.MorphAnalyzer()
```

pymorphy2

```
import pymorphy2
morph = pymorphy2.MorphAnalyzer()
...
def dict_add(d, s):
  # в словарь d добавить слова из строки s
  ww = s.split(' ')
  for w in ww:
    # w1 - нормализованное слово
    w1 = morph.parse(w)[0].normal_form
    d[w1] = d.get(w1, 0) + 1
```

Нерусские килилические тексты

Кириллицей, помимо русских текстов написаны тексты на белорусском, сербском, болгарском , украинском, македонском языках, причем частота появления отдельных букв сравнительно близка к частоте букв в русских текстах
Для того, чтобы отличить такие тексты от русских, возьмём частоты самых распространённых русских слов. Небольшая проверка показывает, что в русских текстах следующие слова имеют такие частоты (*1000):

№	Слова	мин.	макс
0	и	10	80
1	в	10	60
2	не	0	40
3	он	0	23
4	я	0	33
6	что	1	35
7	с	5	31

LibRusEc

Возьмем любой достаточно большой набор русских текстов, например fb2-файлы из библиотеки LibRusEc.
Всего в библиотеке оказалось 7.8 млрд. слов, среди которых 6 млн. различных. Слова, встречающиеся лишь по 1-2-3 раза следует рассматривать как «шум» являющийся результатом опечаток и других ошибок. Пример слов, встречающихся по 1 разу:

пгростительн	абабахв	ababaслопат	яюыюцхэ
пгруз	яюыюцхэш	предшествеши	пгростодушн
ababaджаня	ababaгрушапр	ababaгаламаж	ababaгаламаг

LibRusEc

Пример слов, встречающихся по 6700 раз (№30000 по убыванию частоты):

```
пожухлый барыга чёрненький бурда гаечный чтец болеслав
проветрить чувствительно друза воланд огородный
спецподразделение побелить дознаться гормональный
люковый кровопийца индоевропейский углубить
сикорский диссонанс учредитель карамазов
```

Неплохую аппроксимацию для слова №k в списке по убыванию частоты дает формула (при $k < 30\,000$)

$$Freq[k] \approx \frac{0.1}{k}.$$

Самые частые слова

и	в	не	он	я	на	что	с	быть
3.45	2.81	2.07	1.98	1.71	1.61	1.40	1.24	1.21

N	$Freq$	$Dolya$
15	$1/160$	22.3%
54	$1/500$	35.8%
163	$1/1700$	48.5%
5115	$1/51K$	83.1%
24646	$1/510K$	95.0%
75658	$1/5M$	98.3%

Например, слово №163 («никогда») встречается с частотой $1/1700$ и слова с номерами 0..163 составляют 48.5% от общего количества слов.

Первые 30 тысяч слов составляют около 97% от общего количества слов.

Частотный словарь

Результатом первого этапа является построение частотного словаря (размера N) и алгоритма приведения произвольного слова к номеру в словаре.
При этом словарь должен начинаться с двух специальных слов:
№0: «unknown» — неопознанное слово.
№1: «point» — точка, конец предложения.
«Алгоритм»: приводим слово к нормальной форме и достаем его номер из словаря. Нескольким словам может соответствовать один и тот же номер.
«Базовые слова».

Предсказание следующего слова

Как оценить уровень понимания текста нашей программой?
Вполне разумный показатель — вероятность правильного предсказания следующего слова в тексте.
Даже если предсказывать будет человек, — вероятность будет намного меньше 100%.

word2vec

Как преобразовать текст в числа?
Простейший метод: занумеруем все слова и вместо текста получим последовательность чисел.
Плохо: близкие числа не означают близкие слова.
Алгоритм word2vec, предложенный Томашем Миколовым в 2013 году.
Мы хотим научиться выражать понятие смысла слова математически. А именно: сопоставить каждому слову вектор в многомерном пространстве так чтобы близким по смыслу словам соответствовали геометрически близкие вектора, а далеким словам — далекие.
Идея состоит в том, что смысл слова определяется контекстом, в котором оно часто встречается.

word2vec

Каждому слову сопоставим некоторый вектор:

```
w0('и')    -> v0 = {v00, v01, v02, ...}
w1('не')   -> v1 = {v10, v11, v12, ...}
w2('он')   -> v2 = {v20, v21, v22, ...}
w3('я')    -> v3 = {v30, v31, v32, ...}
w4('на')   -> v4 = {v40, v41, v42, ...}
w5('что') -> v5 = {v50, v51, v52, ...}
w6('с')    -> v6 = {v60, v61, v62, ...}
...
```

Все вектора v_i объединим в одну матрицу A.

word2vec

Если мы имеем «фразу», то есть цепочку слов:

```
w1, w22, w77, w1, w2, ...
```

то ей тоже можно сопоставить вектор

```
v1 + v22 + v77 + w1 + w2 + ...
```

Каждому появления слова в тексте тоже сопоставим «вектор контекста» с заданным размахом K. Например, при $K = 2$:

```
w1, [w2, v3, w77, w4, w5], v6, ...
```

```
w77 -> v2 + v3 + v4 + v5
```

Если слово входит в текст несколько раз, то векторы контекста, конечно же будет разными.

word2vec, цель

Цель: подобрать матрицу A так, чтобы различные векторы контекстов для одного слова были бы как можно меньше.

Локальный контекст

Пусть $W = (w_0, \ldots, w_{N-1})$ — базовые слова, $N = 32768$.
Обозначим через L векторное пространство с базисом W размерности N.
Локальным контекстом слова w будем называть вектор из L показывающий, с какой частотой появляются слова из W в ближайшей окрестности слова w. Более точно, пусть T — некоторый нормализованный текст $T = (t_1, \ldots, t_s)$, где $t_i \in W$.
Контекстом слова t_i из текста T с весами $R(x)$ будем называть вектор из L:

$$H_0(t_i) = t_{i-e} + \cdots + t_i + \cdots + t_{i+e}.$$

Локальный контекст

Например, для участка текста:

$$\ldots, t_{i-5}, t_{i-4}, t_{i-3}, t_{i-2}, t_{i-1}, \ t_i, \ t_{i+1}, t_{i+2}, t_{i+3}, t_{i+4}, t_{i+5}, \ldots.$$

$$\ldots, 100, \ 2, \ 300, \ 1, \ 800, \ 777, \ 1000, \ 2, \ 1, \ 101, \ 102, \ldots.$$

вектор локального контекста будет:

$$H_0(t_i) = t_{i-5} + t_{i-4} + t_{i-3} + t_{i-2} + t_{i-1} +$$

$$+ t_{i+1} + t_{i+2} + t_{i+3} + t_{i+4} + t_{i+5} =$$

$$= w_{100} + w_2 + w_{300} + w_1 + w_{800} + w_{1000} + w_2 + w_1 + w_{101} + w_{102} =$$

$$= 2w_1 + 2w_2 + w_{100} + w_{101} + w_{102} + w_{300} + w_{800} + w_{1000}.$$

Локальный контекст

Для каждого базового слова w_i сложим локальные контексты всех его появлений в тексте. Полученный вектор после нормализации будем называть вектором локального контекста слова w_i.

Все эти вектора объединим в одну матрицу A — матрица локального контекста.

Основная идея: будем считать, что слова близки по смыслу, если близки их локальные контексты.

Вторая идея: Вектора длиной в несколько тысяч неудобны. Надо сократить размерность.

Сокращение размерности

Рассмотрим матрицу $A \cdot A^t$:

```
AAt = A.dot(A.T)
```

Её собственные значения и вектора:

```
e, v = np.linalg.eigh(AAt)
```

Возьмем последние K значений и векторов в обратном порядке:

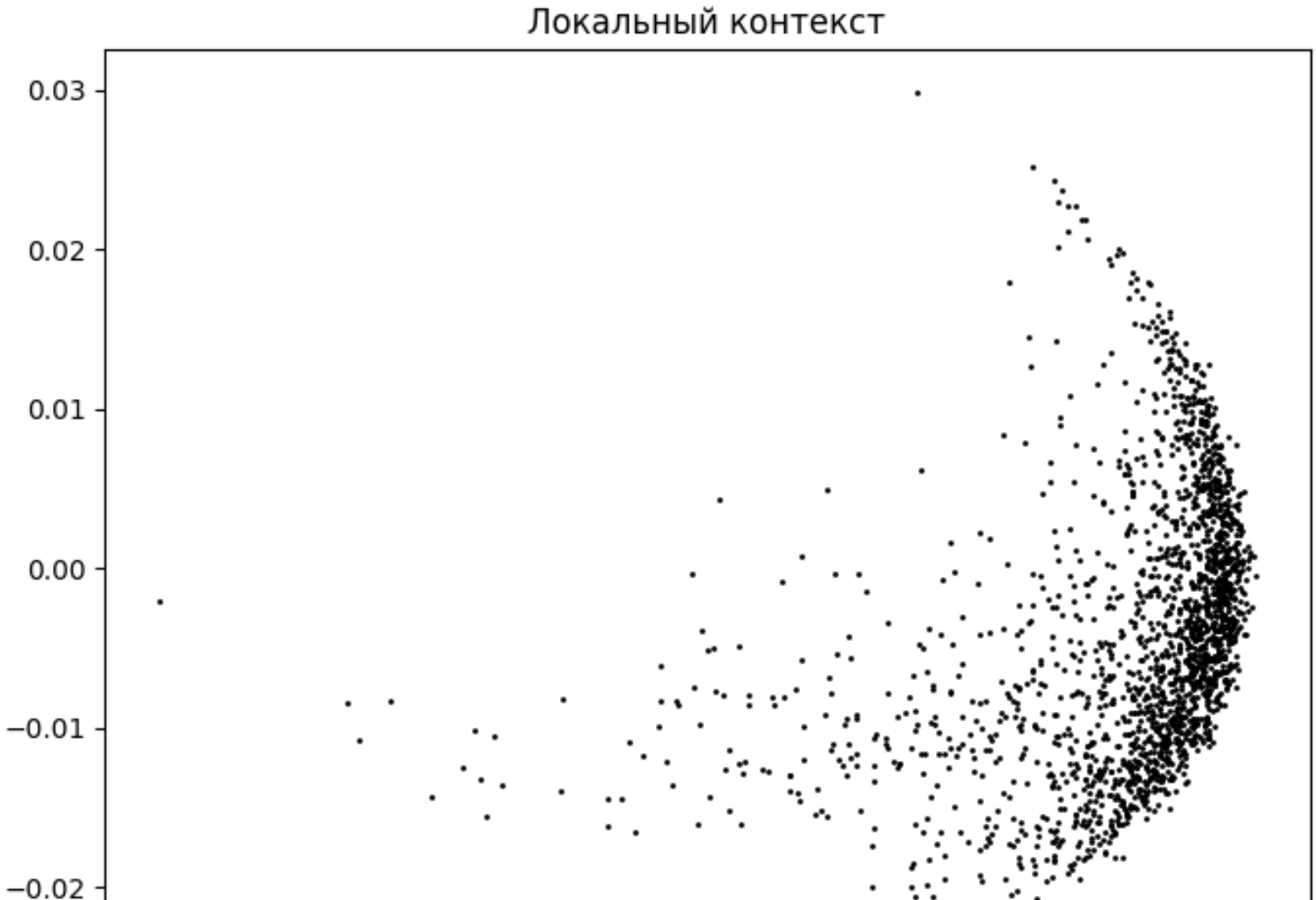
```
K = 2
e = e[   -1:-1-K:-1] # последние K значений
v = v[:, -1:-1-K:-1] # последние K столбцов
```

Сокращение размерности

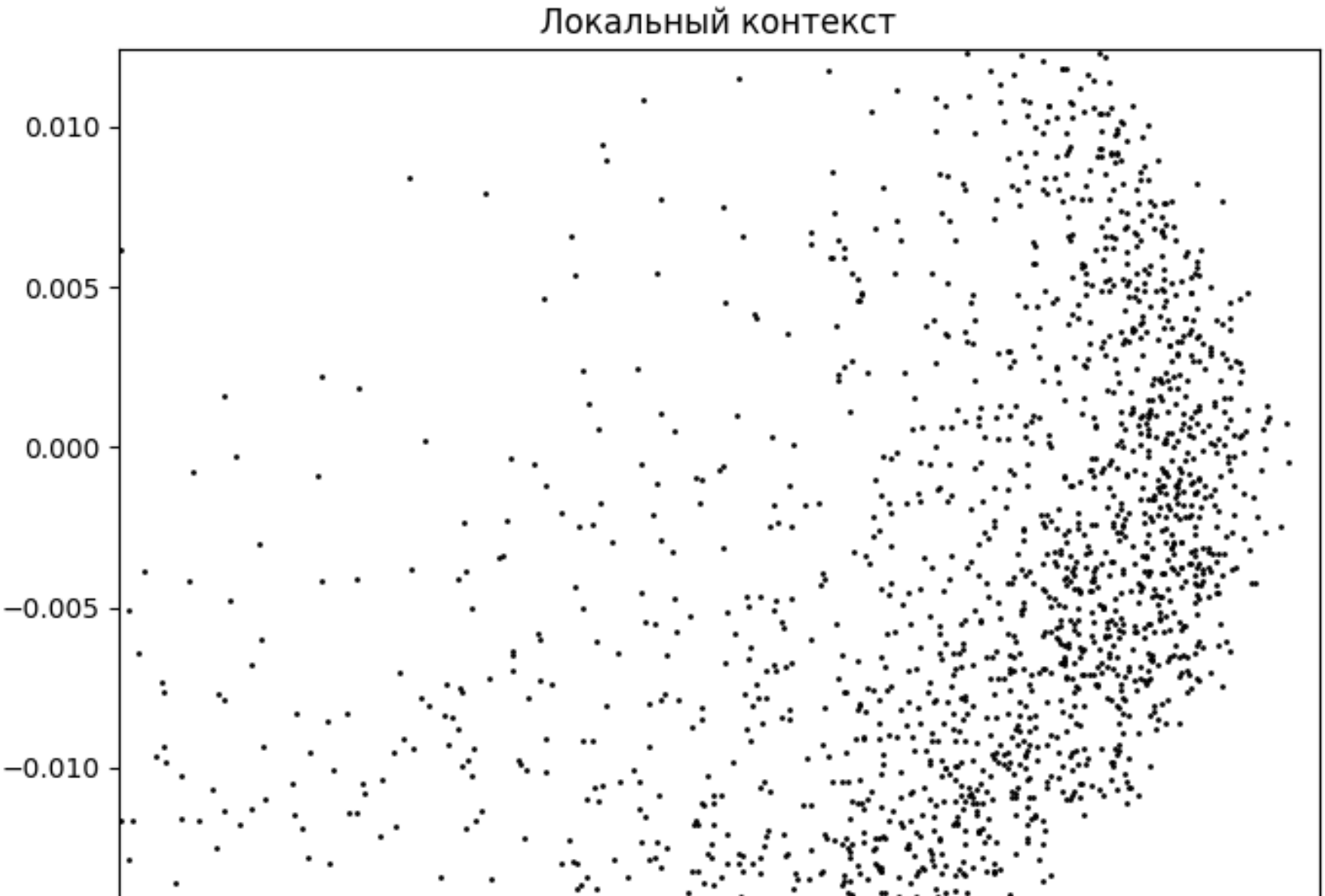
В результате получаем отображение всех слов в $K = 2$-мерное пространство:

```
From file astro.fb2 2 coords
в       , 0.0322748, 0.0004425
и       , 0.0321491, 0.0006880
на      , 0.0321198, 0.0064374
планет  , 0.0319014, 0.0062099
не      , 0.0316115, 0.0082834
что     , 0.0316297,-0.0014420
звезд   , 0.0309660,-0.0044937
так     , 0.0315639, 0.0047473
котор   , 0.0320103,-0.0037287
астроном, 0.0313108,-0.0017843
```

Отображение на плоскость



Отображение на плоскость-2



ToDo

Вторичное представление.