



# Информационные ресурсы в финансовом мониторинге

---

НИЯУ МИФИ, КАФЕДРА ФИНАНСОВОГО МОНИТОРИНГА

КУРС ЛЕКЦИЙ

В.Ю. РАДЫГИН. ЛЕКЦИЯ 5

# Применение современных технологий в задачах обработки текста

---

# Основные группы задач

---

Наиболее частые задачи, связанные с интеллектуальной машинной обработкой текста, можно поделить на следующие группы:

1. Поддержка ввода текста (автоматический перенос и проверка орфографии).
2. Машинный перевод.
3. Информационный поиск (в том числе, нечёткий и полнотекстовый), включая задачу сопоставления текстов и поиска плагиата.
4. Компрессия текста.
5. Классификация текстов.
6. Извлечение фактов и знаний.
7. Вопросно-ответные системы.
8. Диалог человек-машина.
9. Другие задачи.

# Часть 1

---

ПОДДЕРЖКА ВВОДА ТЕКСТА

# Автоматический перенос

---

Для человека задача расстановки переносов в его родном языке довольно проста. Школьники знакомятся с ней в начальных классах. Но для автоматического решения на первый взгляд она кажется нерешаемой. Тем не менее, алгоритмы известны уже давно. Все они преимущественно базируются на создании словарей переносов. Самый популярный из них – алгоритм Ляна-Кнута.

# Алгоритм Ляна-Кнута

---

Алгоритм Ляна-Кнута заключается в описании набора правил переноса в виде небольших шаблонов, состоящих из кусочков слов и цифр в местах переносов. Нечётные цифры разрешают перенос в данном месте. Их значение означает приоритет. Чётные цифры означают запрет переноса. Точки в конце или начале шаблона означают конец или начало слова.

Пример правил:

.че2с1к

.юс1

4а3а

аа2п

аа2р

Пример готового словаря шаблонов можно рассмотреть словарь TeX [3].

# Пример

---

Рассмотрим переносы слова *алгоритм*.

Правила: лго1 1г о1ри и1т и2тм тм2

а л г о р и т м

л г о1

1г

о1р и

и1т

и2т м

т м2

а л1г о1р и2т м2. В итоге получаем: ал-го-ритм.

# Недостатки алгоритма Ляна-Кнута

---

Недостатком данного алгоритма является необходимость для эффективной работы держать весь словарь переносов в оперативной памяти. Данный факт может вызвать проблемы на устройствах с маленьким объёмом памяти. Например, на смартфонах и ридерах. В таких ситуациях обычно используют алгоритм П. Христова в модификации Дымченко и Варсанофьева.

Кроме того, данный алгоритм очень чувствителен к правильности словаря. Тем не менее, найти правильный словарь шаблонов переносов сегодня не составляет особого труда.



# Исключения

---

Часто к шаблонам добавляют набор слов исключений, для которых правила переноса определяются уникальным образом.

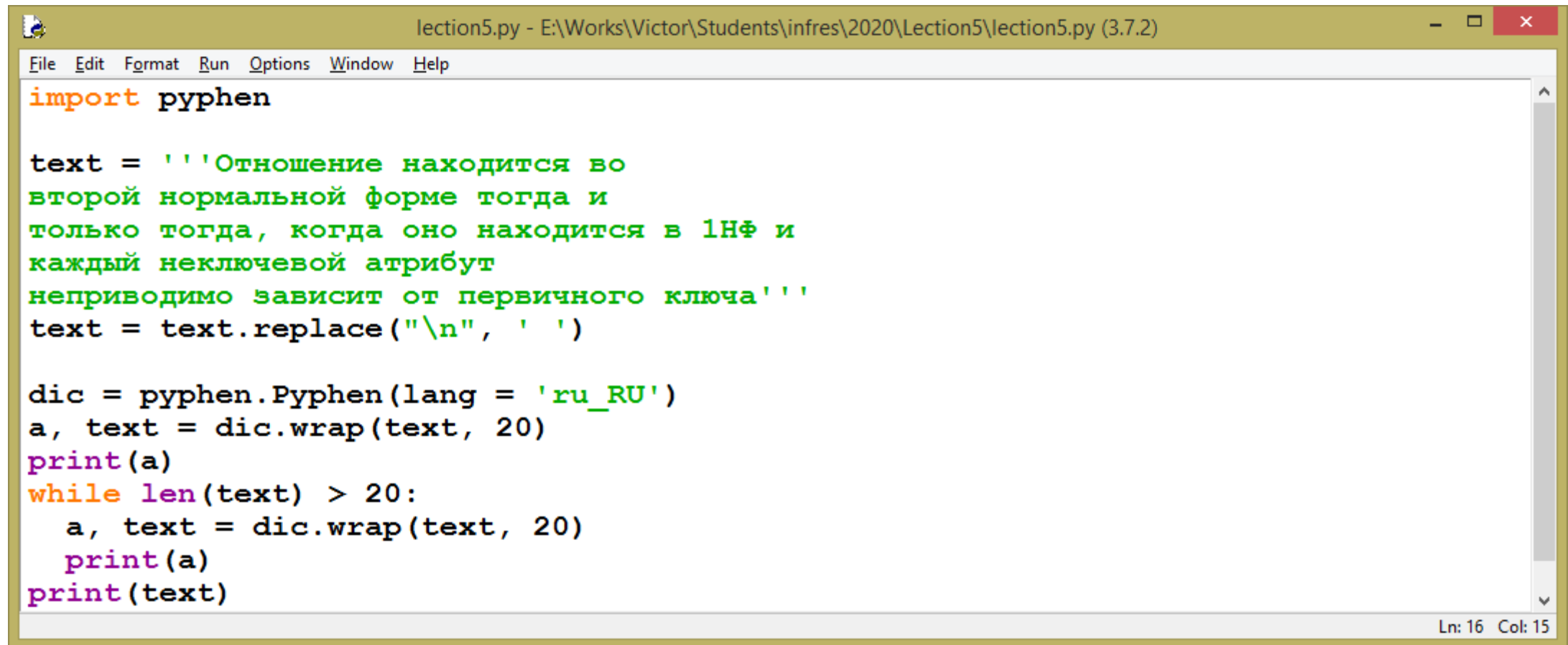
# Есть готовые решения

---

Во многих языках программирования есть готовые решения для реализации переноса слов. Например, в Python – это библиотека Pyphen [1, 2]. Она устанавливается следующей командой:

```
python.exe -m pip install pyphen
```

# Пример использования pyphen



```
lection5.py - E:\Works\Victor\Students\infres\2020\Lecture5\lecture5.py (3.7.2)
File Edit Format Run Options Window Help
import pyphen

text = '''Отношение находится во
второй нормальной форме тогда и
только тогда, когда оно находится в 1НФ и
каждый неключевой атрибут
неприводимо зависит от первичного ключа'''
text = text.replace("\n", ' ')

dic = pyphen.Pyphen(lang = 'ru_RU')
a, text = dic.wrap(text, 20)
print(a)
while len(text) > 20:
    a, text = dic.wrap(text, 20)
    print(a)
print(text)
```

Ln: 16 Col: 15

# Пример использования pyphen (текстом)

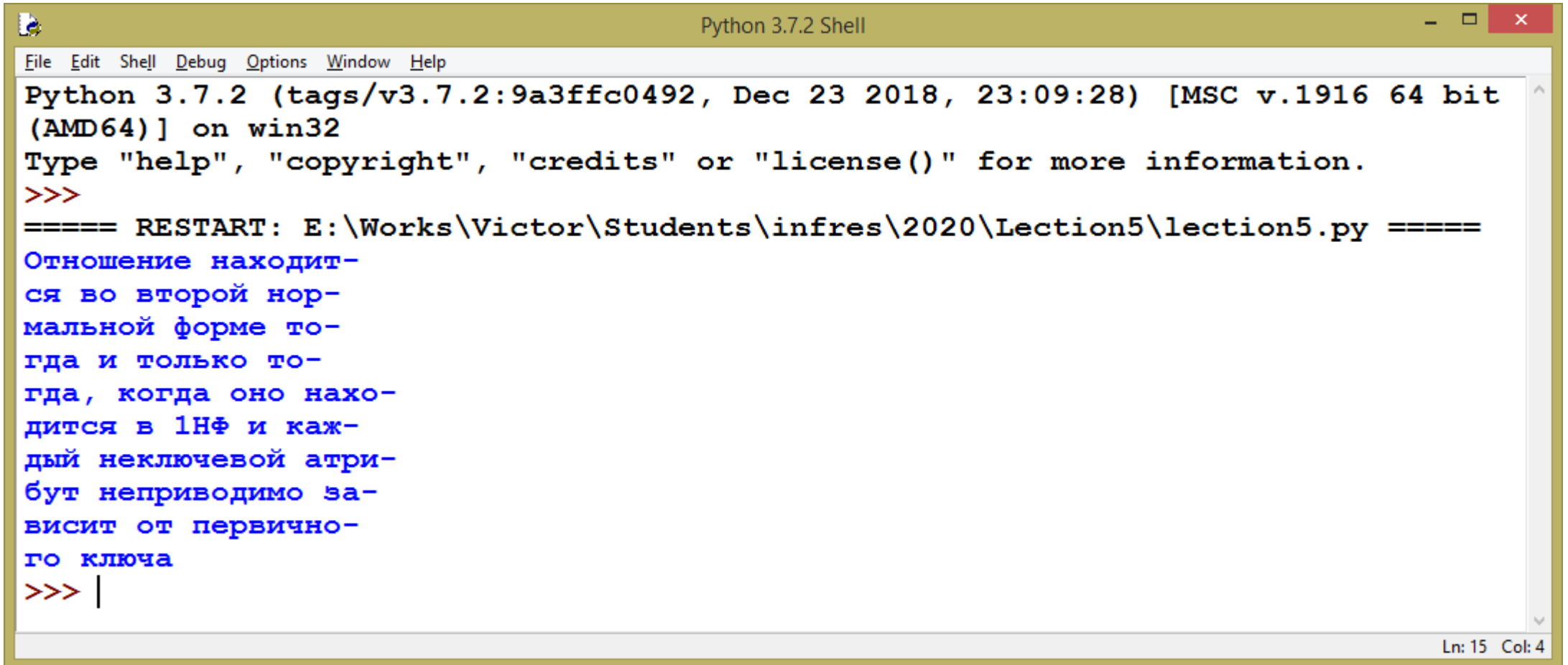
---

```
import pyphen
```

```
text = '''Отношение находится во  
второй нормальной форме тогда и  
только тогда, когда оно находится в 1НФ и  
каждый неключевой атрибут  
неприводимо зависит от первичного ключа'''  
text = text.replace("\n", ' ')
```

```
dic = pyphen.Pyphen(lang = 'ru_RU')  
a, text = dic.wrap(text, 20)  
print(a)  
while len(text) > 20:  
    a, text = dic.wrap(text, 20)  
    print(a)  
print(text)
```

# Результат

A screenshot of a Python 3.7.2 Shell window. The window has a title bar "Python 3.7.2 Shell" and a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main text area shows the following output:

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: E:\Works\Victor\Students\infres\2020\Lecture5\lecture5.py =====
Отношение находится во второй нормальной форме тогда и только тогда, когда оно находится в 1НФ и каждый неключевой атрибут неприводимо зависит от первичного ключа
>>> |
```

The status bar at the bottom right indicates "Ln: 15 Col: 4".

# Исправление ошибок

---

Одной из важных задач помимо расстановки переносов является проверка орфографии текста и исправление ошибок. Вопрос полноценной проверки орфографии довольно сложный и не будет рассматриваться в данной лекции. Но вот задача поиска возможных исправлений для неправильно написанного слова не так сложна.

Предположим, что у нас есть некоторый словарь слов:

```
words = ['Машина', 'Марина', 'Марьино', 'Вершина', 'Машинист', 'Машенька']
```

И есть слово, написанное с ошибкой:

```
test_word = 'Марьина'
```

Как ответить на вопрос (без участия человека) к какому из слов данное ошибочное слово ближе? Именно такую задачу решают системы исправления ошибок каждый раз, когда предлагают пользователю варианты коррекции.

# Расстояние Хэмминга

---

Расстояние Хэмминга [4] — это количество различающихся позиций для строк с одинаковой длиной.

Например, для строк 'Машина', 'Марина' расстояние Хэмминга равно 1, а для строк 'Машинист', 'Машенька' — 4.

Недостатком кода Хэмминга является требование равенства длины сравниваемых строк и отсутствие учёта положения строк относительно друг друга. Поясним последнее. Расстояние Хэмминга для слов 'Железнодорожник', 'Кжелезнодорожни' равно 15, хотя разница между ними только в перестановке одного символа из конца в начало.

Преимущество расстояния Хэмминга — простота вычисления.

Этот простой пример показывает, почему расстояние Хэмминга не очень хорошо использовать для поиска вариантов исправления ошибки написания слов.

# Расстояние Левенштейна

---

Расстояние Левенштейна [5] — (редакционное расстояние, дистанция редактирования) — минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.

Например, для строк 'Машина', 'Марина' расстояние Левенштейна равно 1, а для строк 'Машинист', 'Машенька' — 4. При этом для слов 'Железнодорожник', 'Кжелезнодорожни' расстояние Левенштейна равно 3.

Преимуществом расстояния Левенштейна являются его применимость к словам разной длины, нечувствительность к сдвигу символов одной строки по отношению к другой и относительная простота вычисления.



# Расстояние Дамерау-Левенштейна

---

Фредерик Дамерау показал, что наиболее часто совершаемые ошибки – это перестановка двух соседних символов слова (транспозиция). По этой причине он предложил усовершенствовать метрику, определяемую расстоянием Левенштейна введя в неё данную новую операцию. Новая метрика получила название Расстояние Дамерау-Левенштейна [6].

# Мера сходства (расстояние) слов Джаро

---

Расстояние Джаро для двух слов  $s_1$  и  $s_2$  вычисляется следующим образом:

$$d_j = \begin{cases} 0, & \text{если } m = 0 \\ \frac{1}{3} \left( \frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m - t}{m} \right) & \text{иначе} \end{cases}$$

Здесь:  $m$  – число совпадающих символов в пределах длины  $L$ ,  $|s_i|$  – длина  $i$ -ой строки,  $t$  – половина числа транспозиций, не превышающих длины  $L$ . Причём

$$L = \left\lfloor \frac{\max(|s_1|, |s_2|)}{2} \right\rfloor - 1.$$

# Пример

---

$s_1 = \text{'Машинист'}, s_2 = \text{'Машисинт'}, s_3 = \text{'Баринист'}$ .

Для пары  $s_1$  и  $s_2$  получаем:  $L = 8/2 - 1 = 3, m = 8, t = 1$ .

Тогда  $d_j = 1/3 (8/8 + 8/8 + 7/8) = 23/24 = 0,9583$

Для пары  $s_1$  и  $s_3$  получаем:  $L = 8/2 - 1 = 3, m = 6, t = 0$ .

Тогда  $d_j = 1/3 (6/8 + 6/8 + 6/6) = 20/24 = 0,83$

При этом расстояния Хэмминга и Левенштейна для данных пар слов будут одинаковы (2). Таким образом, для исправления ошибок расстояние Джаро может быть эффективнее более простых мер.

Тем не менее, у расстояния Джаро есть один недостаток. Оно не даёт преимущества строкам, совпадающим от начала до некоторой длины. Этот недостаток устраняет расстояние Джаро-Винклера.

# Расстояние Джаро-Винклера

---

Расстояние Джаро-Винклера для двух слов  $s_1$  и  $s_2$  вычисляется следующим образом:

$$d_w = d_j + \left( lp(1 - d_j) \right).$$

Здесь:  $d_j$  – расстояние Джаро,  $l$  – длина общего префикса (одинаковой в обоих словах подстроки) от начала слов до максимум 4-х символов,  $p$  – эмпирический коэффициент масштабирования ( $p \in [0; 0,25]$ ). Обычно  $p = 0,1$ . Очень частый префиксный бонус «включается» только, если  $d_j > b_t$ . Обычно  $b_t = 0,7$ .

Расстояние Джаро-Винклера – это не метрика в математическом понимании (не подчиняется неравенству треугольника и из нулевой метрики не следует равенство слов)!

# Пример

---

$s_1 = \text{'Машинист'}, s_2 = \text{'Машисинт'}, s_3 = \text{'Манишист'}$ .

Для пары  $s_1$  и  $s_2$  получаем:  $d_j = 0,9583, l = 4$ .

Тогда  $d_w = 0,9583 + 4 * 0,1 * (1 - 0,9583) = 0,975$

Для пары  $s_1$  и  $s_3$  получаем:  $d_j = 0,9583, l = 2$ .

Тогда  $d_w = 0,9583 + 2 * 0,1 * (1 - 0,9583) = 0,9667$

При этом расстояния Хэмминга и Левенштейна для данных пар слов будут одинаковы (2). Таким образом, расстояние Джаро-Винклера отдаёт предпочтение вариантам, обладающим начальным сходством.

# Библиотека python-Levenshtein

---

Для расчёта большинства из рассмотренных мер (расстояний) в современных языках есть готовые реализации. Для языка Python одной из наиболее эффективных реализаций является библиотека python-Levenshtein, реализованная на языке C [7].

Для её установки предварительно потребуется установить компилятор языка C. Инструкция по установке компилятора для Windows доступна на сайте Microsoft [8].

После установке компилятора языка C библиотека python-Levenshtein устанавливается командой

```
python.exe -m pip install python-Levenshtein
```

# Пример использования метрик

```
lecture5.py - E:\Works\Victor\Students\infres\2020\Lecture5\lecture5.py (3.7.2)
File Edit Format Run Options Window Help

words = ['Машина', 'Марина', 'Марьино',
         'Вершина', 'Машинист', 'Машенька']
test_word = 'Марьино'
print('Расстояние Хэмминга')
# Нельзя вычислить для слов другой длины
word = 'Марьино'
print("%s и %s: %f" % (test_word, word, hamming(test_word, word)))
word = 'Вершина'
print("%s и %s: %f" % (test_word, word, hamming(test_word, word)))
print('Расстояние Левенштейна')
for word in words:
    print("%s и %s: %f" % (test_word, word, distance(test_word, word)))
print('Сходство Джаро')
for word in words:
    print("%s и %s: %f" % (test_word, word, jaro(test_word, word)))
print('Сходство Джаро — Винклера')
for word in words:
    print("%s и %s: %f" % (test_word, word, jaro_winkler(test_word, word)))
```

Ln: 38 Col: 0

# Пример использования метрик (текстом)

```
from Levenshtein import *
words = ['Машина', 'Марина', 'Марьино',
         'Вершина', 'Машинист', 'Машенька']
test_word = 'Марьино'
print('Расстояние Хэмминга')
# Нельзя вычислить для слов другой длины
word = 'Марьино'
print("%s и %s: %f" % (test_word,
                        word, hamming(test_word, word)))
word = 'Вершина'
print("%s и %s: %f" % (test_word,
                        word, hamming(test_word, word)))
```

```
print('Расстояние Левенштейна')
for word in words:
    print("%s и %s: %f" % (test_word,
                           word, distance(test_word, word)))
print('Сходство Джаро')
for word in words:
    print("%s и %s: %f" % (test_word,
                           word, jaro(test_word, word)))
print('Сходство Джаро — Винклера')
for word in words:
    print("%s и %s: %f" % (test_word,
                           word, jaro_winkler(test_word, word)))
```



# Результат

```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
010.py
Расстояние Хэмминга
Марина и Марино: 1.000000
Марина и Вершина: 3.000000
Расстояние Левенштейна
Марина и Машина: 2.000000
Марина и Марина: 1.000000
Марина и Марино: 1.000000
Марина и Вершина: 3.000000
Марина и Машинист: 5.000000
Марина и Машенька: 5.000000
Сходство Джаро
Марина и Машина: 0.849206
Марина и Марина: 0.952381
Марина и Марино: 0.904762
Марина и Вершина: 0.714286
Марина и Машинист: 0.690476
Марина и Машенька: 0.713095
Сходство Джаро — Винклера
Марина и Машина: 0.879365
Марина и Марина: 0.966667
Марина и Марино: 0.961905
Марина и Вершина: 0.714286
Марина и Машинист: 0.752381
Марина и Машенька: 0.770476
>>>
```

Ln: 193 Col: 4

# N-граммы (N-gram)

---

N-граммами называются все возможные комбинации из N подряд идущих символов слова.

Например, слово машина подразумевает:

пять 2-грамм (ма, аш, ши, ин, на);

четыре 3-грамм (маш, аши, шин, ина);

три 4-граммы (маши, ашин, шина) и т.д.

Понятие N-грамм применяется во многих алгоритмах анализа текста.

# Расстояние в N-граммах

---

Расстояние в N-граммах – это количество совпадающих по числу вхождений N-грамм определённой длины в двух словах.

# Пример

---

$s_1 = \text{'Машинист'}$ ,  $s_2 = \text{'Машисинт'}$ ,  $s_3 = \text{'Манишист'}$ .

Слово	ма	аш	ши	ин	ни	ис	ст	си	нт	ан	иш
Машинист	1	1	1	1	1	1	1	0	0	0	0
Машисинт	1	1	1	1	0	1	0	1	1	0	0
Манишист	1	0	1	0	1	1	1	0	0	1	1

$ngram(s_1, s_2, 2) = 5$ ,  $ngram(s_1, s_3, 2) = 5$ ,  $ngram(s_2, s_3, 2) = 3$

# Мера Жаккара

---

Мера Жаккара позволяет оценить степень сходства двух слов по их компонентному составу. Мету Жаккара можно применять как для сравнения слов по набору букв, так и для сравнения по набору N-грам.

Формула для расчёта меры Жаккара следующая:

$$d_{jc} = \frac{|A \cap B|}{|A \cup B|},$$

где  $A \cap B$  – это пересечение множеств символов (или N-грам) двух слов (без повторений),  
 $A \cup B$  – это объединение множеств символов (или N-грам) двух слов (без повторений)

# Пример

---

$s_1 = \text{'машинист'}$ ,  $s_2 = \text{'машисинт'}$ .

Слово	ма	аш	ши	ин	ни	ис	ст	си	нт
машинист	1	1	1	1	1	1	1	0	0
машисинт	1	1	1	1	0	1	0	1	1

$d_{jc} = 5 / 9 = 0,5556$  для биграмм (2-грамм)

$d_{jc} = 9 / 9 = 1$  для символов

# Пример 2

---

$s_1 = \text{'мама'}$ ,  $s_2 = \text{'маша'}$ .

Слово	ма	ам	аш	ша
мама	2	1	0	0
маша	1	0	1	1

$d_{jc} = 1 / 4 = 0,25$  для биграм (2-грам)

$d_{jc} = 2 / 3 = 0,67$  для символов

# Библиотека nltk

---

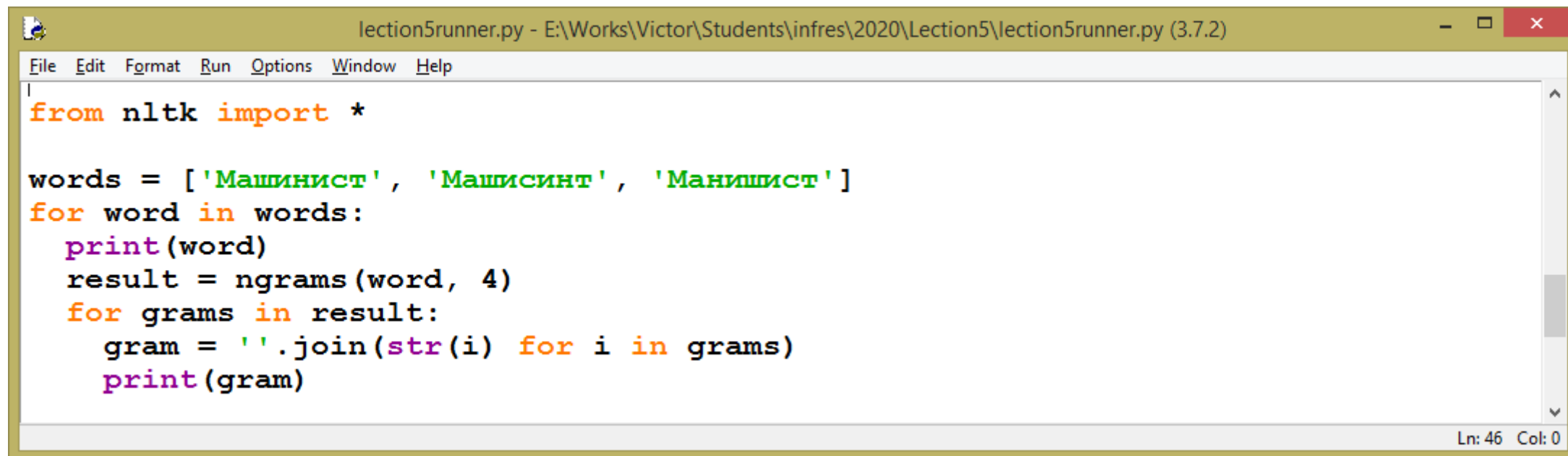
Для языка Python одной из наиболее популярных библиотек для работы с естественными языками является библиотека nltk [9].

Для её установки необходимо выполнить команду:

```
python.exe -m pip install nltk
```



# Пример. Расчёт 4-грам



```
lection5runner.py - E:\Works\Victor\Students\infres\2020\Lecture5\lection5runner.py (3.7.2)
File Edit Format Run Options Window Help
from nltk import *

words = ['Машинист', 'Машисинт', 'Манишист']
for word in words:
    print(word)
    result = ngrams(word, 4)
    for grams in result:
        gram = ''.join(str(i) for i in grams)
        print(gram)
```

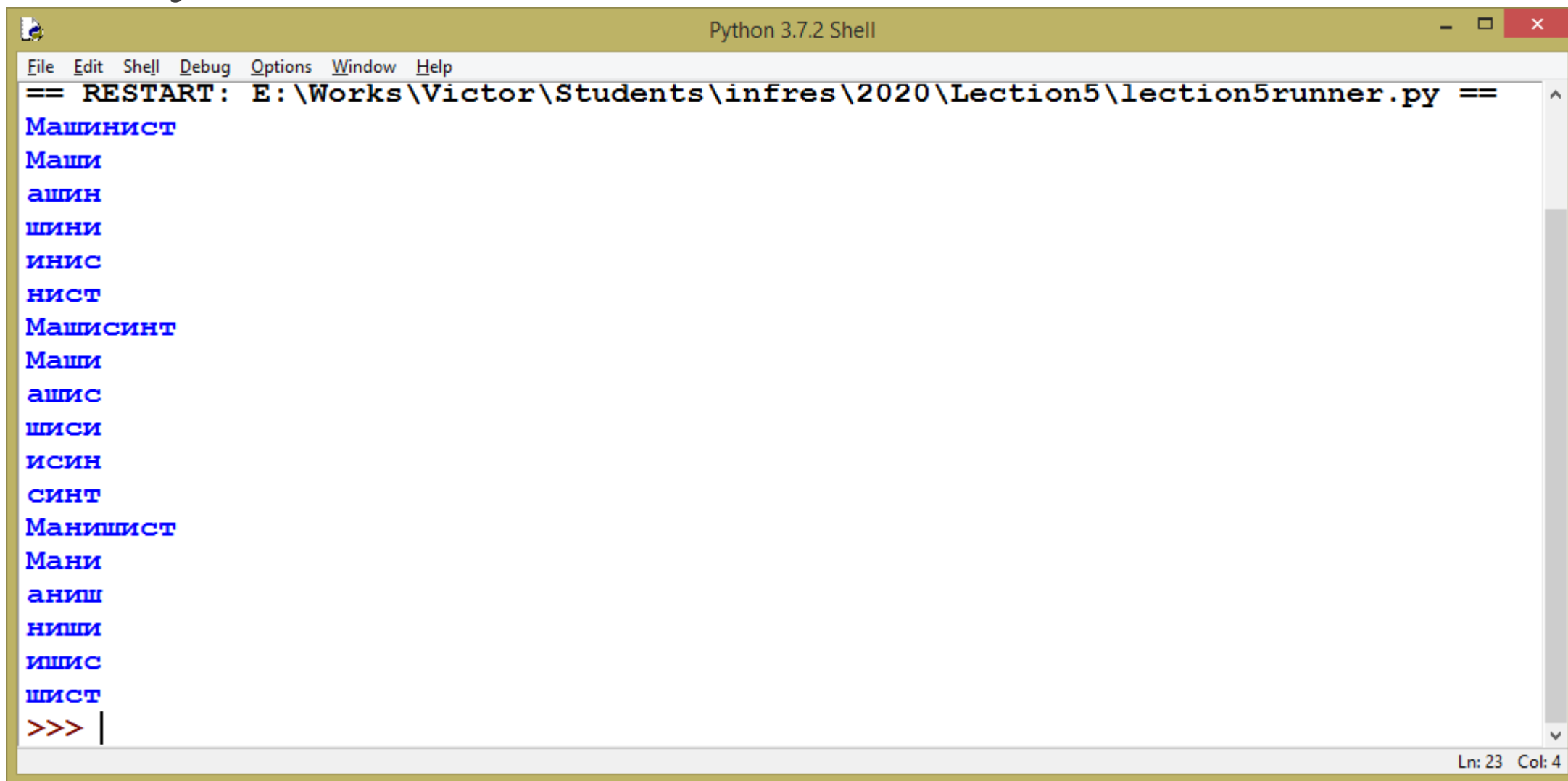
Ln: 46 Col: 0

# Пример. Расчёт 4-грам (текстом)

---

```
from nltk import *  
  
words = ['Машинист', 'Машисинт', 'Манишист']  
for word in words:  
    print(word)  
    result = ngrams(word, 4)  
    for grams in result:  
        gram = ''.join(str(i) for i in grams)  
        print(gram)
```

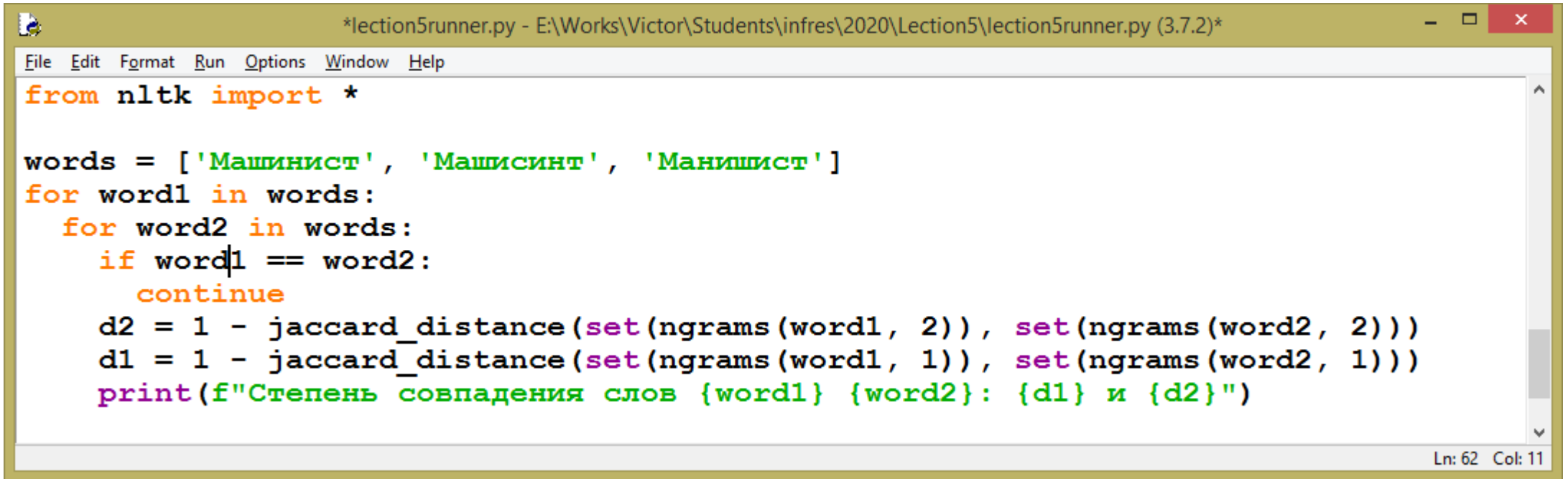
# Результат



```
Python 3.7.2 Shell
File Edit Shell Debug Options Window Help
== RESTART: E:\Works\Victor\Students\infres\2020\Lecture5\lecture5runner.py ==
Машинист
Маши
ашин
шини
инис
нист
Машисинт
Маши
ашис
шиси
исин
синт
Манишист
Мани
аниш
ниши
ишис
шист
>>> |
```

Ln: 23 Col: 4

# Пример. Расчёт меры Жаккара



```
*lection5runner.py - E:\Works\Victor\Students\infres\2020\Lesson5\lection5runner.py (3.7.2)*
File Edit Format Run Options Window Help
from nltk import *

words = ['Машинист', 'Машисинт', 'Манишист']
for word1 in words:
    for word2 in words:
        if word1 == word2:
            continue
        d2 = 1 - jaccard_distance(set(ngrams(word1, 2)), set(ngrams(word2, 2)))
        d1 = 1 - jaccard_distance(set(ngrams(word1, 1)), set(ngrams(word2, 1)))
        print(f"Степень совпадения слов {word1} {word2}: {d1} и {d2}")
```

Ln: 62 Col: 11

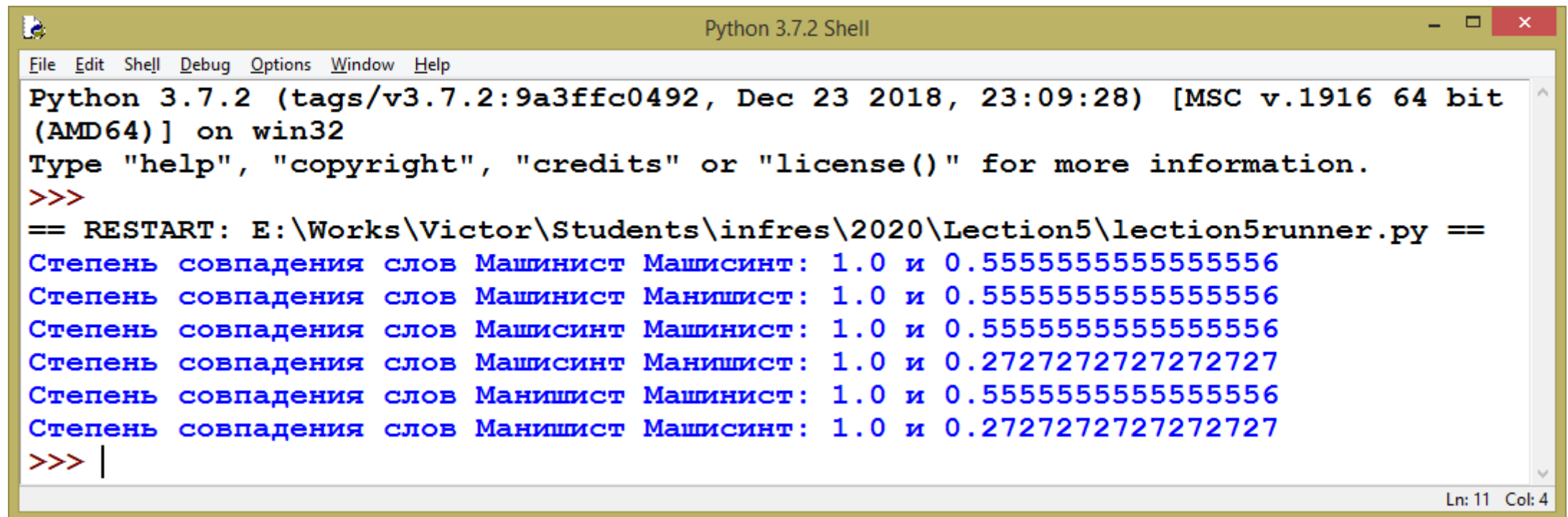
# Пример. Расчёт меры Жаккара

---

```
from nltk import *

words = ['Машинист', 'Машисинт', 'Манишист']
for word1 in words:
    for word2 in words:
        if word1 == word2:
            continue
        d2 = 1 - jaccard_distance(set(ngrams(word1, 2)), set(ngrams(word2, 2)))
        d1 = 1 - jaccard_distance(set(ngrams(word1, 1)), set(ngrams(word2, 1)))
        print(f"Степень совпадения слов {word1} {word2}: {d1} и {d2}")
```

# Результат



```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
== RESTART: E:\Works\Victor\Students\infres\2020\Lecture5\lecture5runner.py ==
Степень совпадения слов Машинист Машисинт: 1.0 и 0.5555555555555556
Степень совпадения слов Машинист Манишист: 1.0 и 0.5555555555555556
Степень совпадения слов Машисинт Машинист: 1.0 и 0.5555555555555556
Степень совпадения слов Машисинт Манишист: 1.0 и 0.2727272727272727
Степень совпадения слов Манишист Машинист: 1.0 и 0.5555555555555556
Степень совпадения слов Манишист Машисинт: 1.0 и 0.2727272727272727
>>> |
```

Ln: 11 Col: 4

# Часть 2

---

ИНФОРМАЦИОННЫЙ ПОИСК

# Задачи

---

Основной задачей информационного поиска является нахождение слов или фраз в тексте, даже, если они написаны в другом падеже, форме или склонении.

Например, поиск слова «идти» во фразе «Шла Саша по шоссе».

Обычно данная задача называется полнотекстовый поиск.

Различают два основных направления решения данной задачи:

- полнотекстовый поиск на основе модели естественного языка (корпус язык);
- полнотекстовый нечёткий поиск на основе различных метрик.



# Полнотекстовый поиск

---

Полнотекстовый поиск на основе модели естественного языка является сложной задачей, требующей серьёзного анализа натурального языка. Основным фактором успешности данного поиска является качество корпуса языка.

Большинство современных СУБД (Oracle, MySQL, PostgreSQL и т.д.) имеют встроенные механизмы полнотекстового поиска. Например, в Oracle данную задачу решает компонента Oracle Text [11].

К сожалению, большинство бесплатных реализаций полнотекстового поиска для русского языка значительно уступают коммерческим продуктам.

# Нечёткий поиск

---

Полнотекстовый нечёткий поиск использует математические подходы для оценки схожести искомого слова (или строки) в исследуемом тексте. Различают два основных подхода:

- онлайн методы, выполняющие непосредственное сравнение текста и искомой цепочки в момент поиска;
- оффлайн методы, базирующиеся на построении индексной структуры анализируемого текста (поиск происходит в индексе, а не в самом тексте) и/или словаря .

Некоторые из оффлайн методов также хорошо подходят для решения задачи поиска плагиата.

# Наиболее известные алгоритмы

---

Онлайн методы:

- прямое сравнение на основе расстояния Левенштейна или других метрик;
- различные модификации Bitap алгоритма (Baeza-Yates-Gonnet, Wu-Manber и т.д.).

Оффлайн методы:

- алгоритм расширения выборки;
- метод N-грамм;
- хеширование по сигнатуре;
- Деревья Burkhard-Keller (БК-деревья).

# Часть 3

---

## КЛАССИФИКАЦИЯ ТЕКСТОВ

# Закон Ципфа

---

Данный закон был сформулирован в 1949 году Джорджем Ципфом на основе анализа статистики частоты слов в текстах на многих языках. Его можно определить следующим образом:

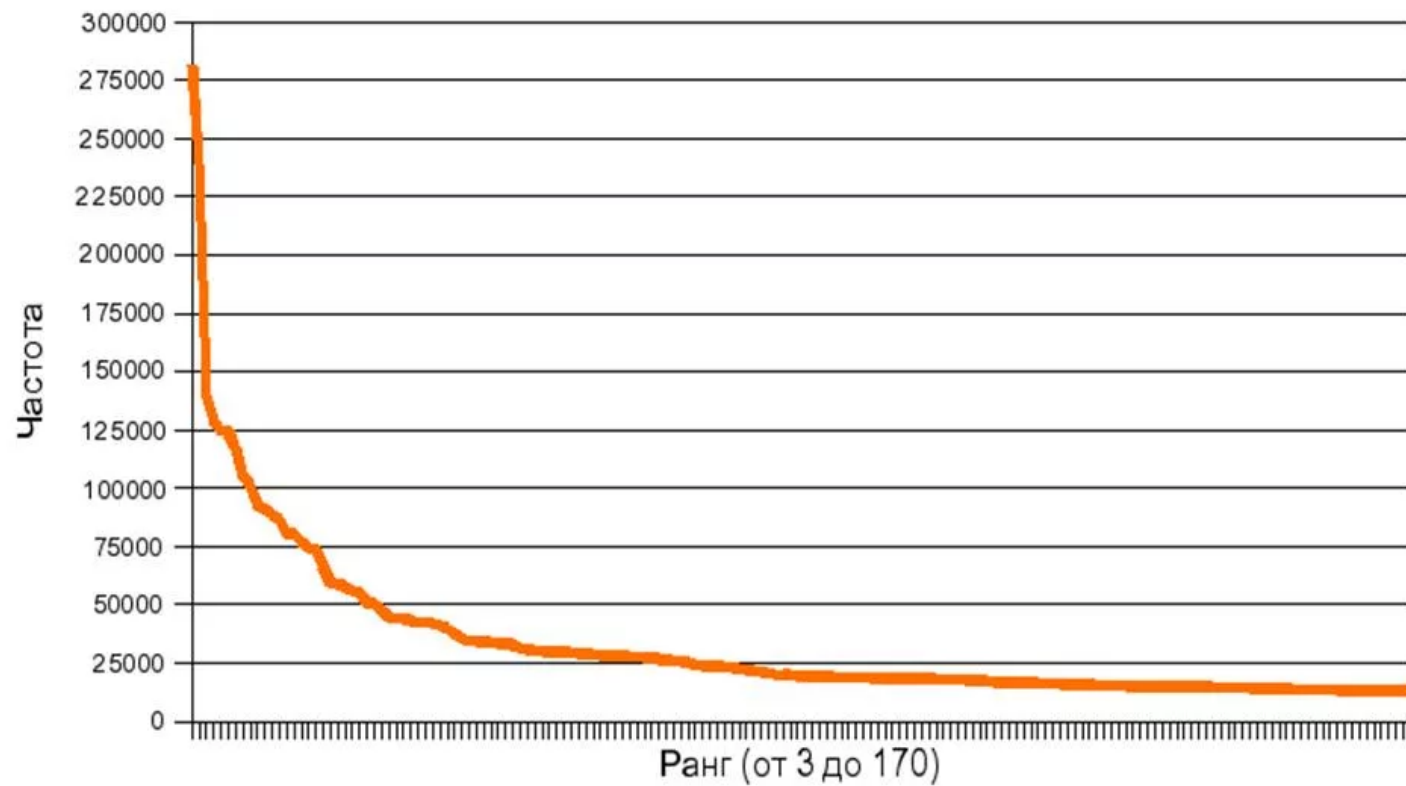
*Если все слова достаточно длинного текста упорядочить по убыванию частоты их использования, то частота  $n$ -го слова в данной последовательности будет обратнопропорциональна (приблизительно) его порядковому номеру  $n$ .*

Число  $n$  принято называть рангом слова. Например, второе по используемости слово встречается примерно в два раза реже, чем первое, третье — в три раза реже, чем первое и т.д.

Формула:  $F \times R = C$ ,  $R$  — ранг слова,  $F$  — количество слов в тексте,  $C$  — некоторая эмпирическая константа (для русского языка - 0,06-0,07, для английского - 0,1).

# Закон Ципфа для википедии

---



# Использование закона Ципфа

---

1. Определение «естественности» текста.
2. Для выявления ключевых слов в тексте.
3. Для определения авторства текста.
4. Для других целей.

# Выявление ключевых слов в тексте

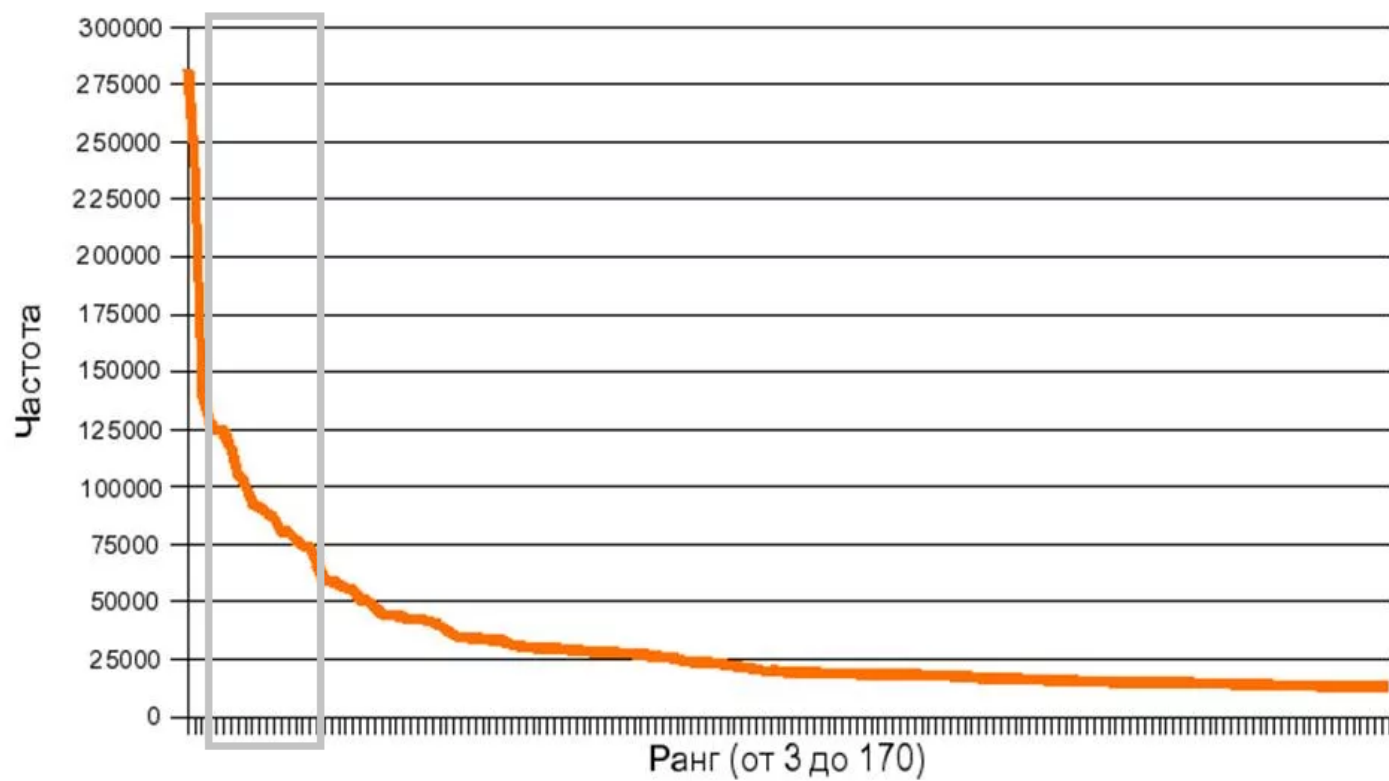
---

Исследования различных авторов показывают, что наиболее значимые слова текста соответствуют средней части графика закона Ципфа. Причина проста. Слова, которые попадают слишком часто, в основном оказываются предлогами или местоимениями. Редко встречающиеся слова тоже, в большинстве случаев, не имеют решающего смыслового значения.



# Наиболее значимые слова

---



# Выбор диапазона

---

От установки ширины полосы значимых слов зависит качество их отделения. Если установить большую ширину диапазона, то в ключевые слова будут попадать вспомогательные слова. Если же наоборот установить узкий диапазон, то будут потеряны смысловые термины. Каждый текст или научная область направления текстов требует собственных эвристических подходов.

Одним из общих приёмов при этом является исключение, так называемых, стоп-слов. Для русского текста стоп-словами могут являться все предлоги, частицы, личные местоимения.

Один из вариантов наборов стоп слов приведён в работе [13].

# Закон Хипса

---

Закон Хипса (H.S. Hears) устанавливает связь между объемом документа и объемом словаря уникальных слов, входящих в него. По логике вещей кажется, что словарь уникальных слов должен насыщаться, а его объем стабилизироваться при увеличении объемов текста. Хипс эмпирически показал, что это не так! Для всех известных сегодня текстов в соответствии с законом Хипса, эти значения связаны соотношением:

$$V = \alpha n^{\beta},$$

где  $V$  – объём словаря текста,  $n$  – число слов в тексте,  $\alpha$  и  $\beta$  – некоторые эмпирические параметры. Для европейских языков  $\alpha \in [10; 100]$ ,  $\beta \in [0,4; 0,6]$ .

При больших объемах текста становится видно, что график закона Хипса идет не плавно, а ступенчато. Такая особенность связана с тем, что с некоторого момента тексты, относящиеся к какой-то узкой предметной области, заканчиваются. Следующий текст относится уже к другой предметной области, а для нее характерна другая лексика.

# Использование закона Хипса

---

Закон Хипса может быть использован как первая ступень проверки на наличие плагиата. Если при добавлении в коллекцию нового текста наблюдаются отклонения от закона Хипса, например, число слов в словаре не возрастает, это может свидетельствовать о наличии плагиата.

# Векторное представление текста

---

Одним из способов интеллектуального исследования текста является переход к представлению текста в виде многомерных числовых векторов.

Наиболее популярным алгоритмом является алгоритм векторизации текста на основе меры TF-IDF.

# Мера TF-IDF

---

**TF** (*term frequency*) — частота слова

$$TF = \frac{\text{число вхождений слова в документ}}{\text{общее число слов в документе}}$$

**IDF** (*inverse document frequency*) — обратная частота документа

$$IDF = \log \left( \frac{\text{число документов в коллекции}}{\text{число документов коллекции, в которых есть слово}} \right)$$

$$TF-IDF = TF \times IDF$$

# Пример решения реальной задачи

---

Возьмем 21 последний твит Дональда Трампа. 20 для обучения, один для проверки. Попробуем разделить данные твиты на два кластера и выделить в них наиболее важные слова.

# ТВИТЫ

1. The golden era of American energy is now underway!
2. With incredible grit, skill, and pride, the 7,000 workers here at Sempra Energy are helping lead the American Energy Revolution. They are not only making our nation WEALTHIER but they are making America SAFER by building a future of American Energy INDEPENDENCE!
3. It is great to be here in Hackberry, Louisiana with the incredible men and women who are making America into the energy superpower of the world!
4. Today marks the one-year anniversary of the opening of the United States Embassy in Jerusalem, Israel. Our beautiful embassy stands as a proud reminder of our strong relationship with Israel and of the importance of keeping a promise and standing for the truth.
5. China will be pumping money into their system and probably reducing interest rates, as always, in order to make up for the business they are, and will be, losing. If the Federal Reserve ever did a "match," it would be game over, we win! In any event, China wants a deal!
6. When Prime Minister @AbeShinzo of Japan visited with me in the @WhiteHouse two weeks ago, I told him that I would be going to the G20 in Osaka, Japan. I look forward to being with him and other World Leaders!
7. Our great Senator (and Star) from the State of Arkansas, @TomCottonAR, has just completed a wonderful book, "Sacred Duty," about Arlington National Cemetery and the men and women who serve with such love and devotion. On sale today, make it big!
8. ....This money will come from the massive Tariffs being paid to the United States for allowing China, and others, to do business with us. The Farmers have been "forgotten" for many years. Their time is now!
9. Our great Patriot Farmers will be one of the biggest beneficiaries of what is happening now. Hopefully China will do us the honor of continuing to buy our great farm product, the best, but if not your Country will be making up the difference based on a very high China buy.....
10. ....of the tremendous ground we have lost to China on Trade since the ridiculous one sided formation of the WTO. It will all happen, and much faster than people think!
11. When the time is right we will make a deal with China. My respect and friendship with President Xi is unlimited but, as I have told him many times before, this must be a great deal for the United States or it just doesn't make any sense. We have to be allowed to make up some.....
12. Billions of Dollars, and moving jobs back to the USA where they belong. Other countries are already negotiating with us because they don't want this to happen to them. They must be a part of USA action. This should have been done by our leaders many years ago. Enjoy!
13. We can make a deal with China tomorrow, before their companies start leaving so as not to lose USA business, but the last time we were close they wanted to renegotiate the deal. No way! We are in a much better position now than any deal we could have made. Will be taking in.....
14. ....so that they will be more competitive for USA buyers. We are now a much bigger economy than China, and have substantially increased in size since the great 2016 Election. We are the "piggy bank" that everyone wants to raid and take advantage of. NO MORE!
15. China buys MUCH less from us than we buy from them, by almost 500 Billion Dollars, so we are in a fantastic position. Make your product at home in the USA and there is no Tariff. You can also buy from a non-Tariffed country instead of China. Many companies are leaving China.....
16. In one year Tariffs have rebuilt our Steel Industry - it is booming! We placed a 25% Tariff on "dumped" steel from China & other countries, and we now have a big and growing industry. We had to save Steel for our defense and auto industries, both of which are coming back strong!
17. I met Marine Sgt. John Peck, a quadruple amputee who has received a double arm transplant, at Walter Reed in 2017. Today, it was my honor to welcome John (HERO) to the Oval, with his wonderful wife Jessica. He also wrote a book that I highly recommend, "Rebuilding Sergeant Peck."
18. Wishing former President Jimmy Carter a speedy recovery from his hip surgery earlier today. He was in such good spirits when we spoke last month - he will be fine!
19. Great to welcome Chairman Shin from Lotte Group to the WH. They just invested \$3.1 BILLION into Louisiana-biggest investment in U.S. EVER from a South Korean company, & thousands more jobs for Americans. Great partners like ROK know the U.S. economy is running stronger than ever!
20. Today, I officially updated my budget to include \$18 million for our GREAT @SpecialOlympics, whose athletes inspire us and make our Nation so PROUD!
21. ..There will be nobody left in China to do business with. Very bad for China, very good for USA! But China has taken so advantage of the U.S. for so many years, that they are way ahead (Our Presidents did not do the job). Therefore, China should not retaliate-will only get worse!



# Подготовка 20 твитов

---

```
import re

file_path = "tw.txt"
with open(file_path, "r") as file_obj:
    rows = file_obj.readlines()
orig_rows = rows
rows = [re.split('\s+', re.sub(r"[^A-z\s]", "", row.strip()))
for row in rows]
twits = [' '.join([word.lower() for word in row]) for row in rows]
```

# Подготовка 21-го твита

---

```
string = "..There will be nobody left in China to do business with. Very bad for China, very good  
for USA! But China has taken so advantage of the U.S. for so many years, that they are way  
ahead (Our Presidents did not do the job). Therefore, China should not retaliate-will only get  
worse!"
```

```
stringp = ' '.join([word.lower() for word in  
re.split('\s+', re.sub(r"^[A-z\s]", "", string.strip()))])
```

# Обучение

---

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
vectorizer = TfidfVectorizer(stop_words = 'english')
X = vectorizer.fit_transform(twits)
knum = 2
model = KMeans(n_clusters = knum, init = 'k-means++', max_iter = 100, n_init = 1)
model.fit(X)
```

# Результат – разбиение по кластерам

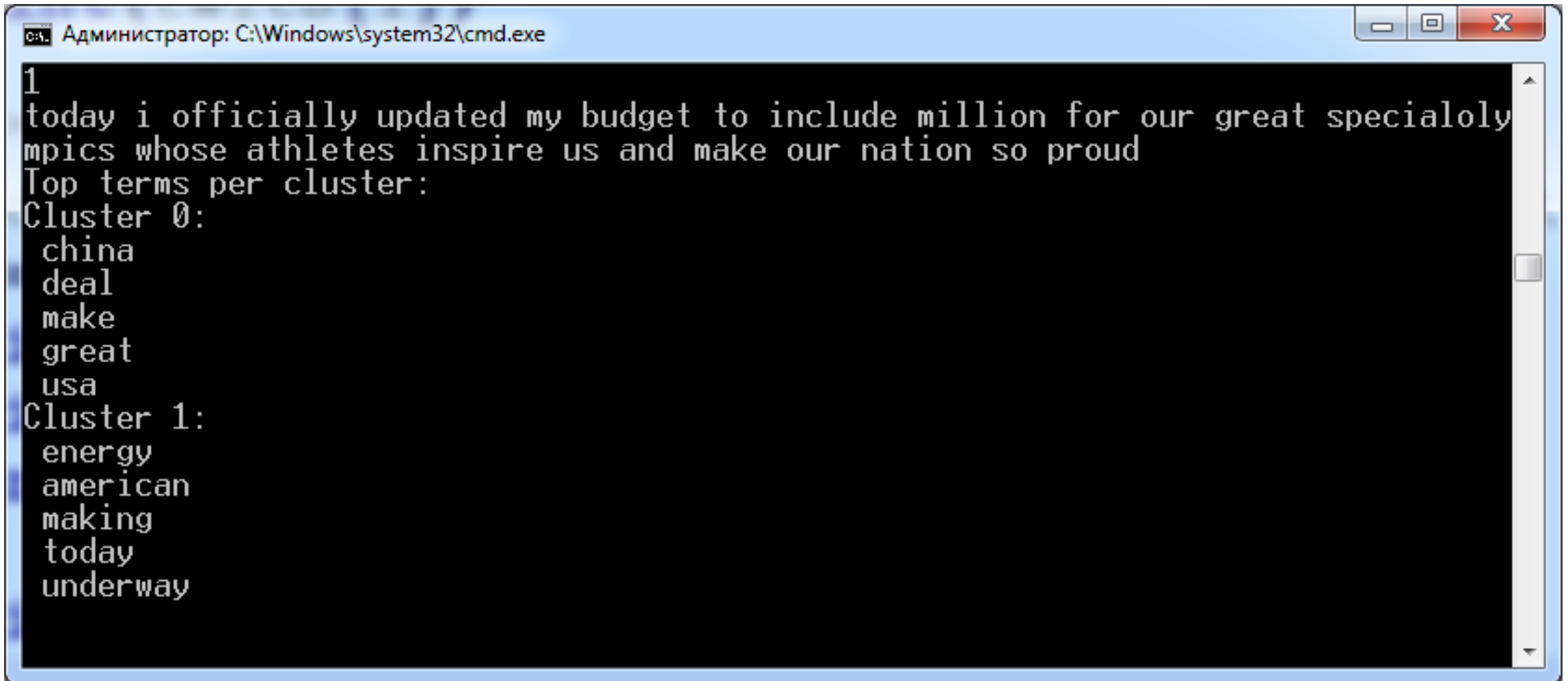
```
Администратор: C:\Windows\system32\cmd.exe
C:\Users\User\Desktop\Students\InfRes\IR6>C:\Users\User\AppData\Local\Programs\Python\Python37\python.exe tw_example_r.py
1
the golden era of american energy is now underway
1
with incredible grit skill and pride the workers here at sempra energy are helping lead the american energy revolution they are not only making our nation wealthier but they are making america safer by building a future of american energy independence
1
it is great to be here in hackberry louisiana with the incredible men and women who are making america into the energy superpower of the world
1
today marks the oneyear anniversary of the opening of the united states embassy in jerusalem israel our beautiful embassy stands as a proud reminder of our strong relationship with israel and of the importance of keeping a promise and standing for the truth
0
china will be pumping money into their system and probably reducing interest rates as always in order to make up for the business they are and will be losing if
```

# Ключевые слова

---

```
print("Top terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()
for i in range(knum):
    print("Cluster %d:" % i),
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind]),
    print
```

# Результат – ключевые слова

A screenshot of a Windows command prompt window. The title bar is light blue and contains the text "Администратор: C:\Windows\system32\cmd.exe" on the left and standard window control buttons (minimize, maximize, close) on the right. The main area is black with white text. The text displayed is: "1", "today i officially updated my budget to include million for our great specialoly", "mpics whose athletes inspire us and make our nation so proud", "Top terms per cluster:", "Cluster 0:", "china", "deal", "make", "great", "usa", "Cluster 1:", "energy", "american", "making", "today", "underway".

```
Администратор: C:\Windows\system32\cmd.exe

1
today i officially updated my budget to include million for our great specialoly
mpics whose athletes inspire us and make our nation so proud
Top terms per cluster:
Cluster 0:
china
deal
make
great
usa
Cluster 1:
energy
american
making
today
underway
```

# Предсказание

---

```
print("Prediction")
```

```
Y = vectorizer.transform([stringp])
```

```
prediction = model.predict(Y)
```

```
print(prediction)
```



The screenshot shows a Windows command prompt window titled "Администратор: C:\Windows\system32\cmd.exe". The window has a black background with white text. The output of the Python script is displayed as follows:

```
Prediction  
[0]  
C:\Users\User\Desktop\Students\InfRes\IR6>
```

# Полезные ссылки

---

1. <https://pypi.org/project/Pyphen/> – страница библиотеки Pyphen на сайте pypi.
2. <https://pyphen.org/> – официальная страница библиотеки Pyphen.
3. <https://github.com/Protagores/hyphen-ru/blob/master/ruhyphal.tex> – TeX словарь шаблонов для переносов русского текста.
4. <https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5%D0%A5%D1%8D%D0%BC%D0%BC%D0%B8%D0%BD%D0%B3%D0%B0> – Википедия о расстоянии Хэмминга.
5. <https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5%D0%9B%D0%B5%D0%B2%D0%B5%D0%BD%D1%88%D1%82%D0%B5%D0%B9%D0%BD%D0%B0> – Википедия о расстоянии Левенштейна.
6. <https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5%D0%94%D0%B0%D0%BC%D0%B5%D1%80%D0%B0%D1%83%E2%80%94%D0%9B%D0%B5%D0%B2%D0%B5%D0%BD%D1%88%D1%82%D0%B5%D0%B9%D0%BD%D0%B0> – Википедия о расстоянии Дамерау-Левенштейна.



# Полезные ссылки

---

7. <https://pypi.org/project/python-Levenshtein/> – страница библиотеки python-Levenshtein на сайте pypi.
8. <https://docs.microsoft.com/en-us/cpp/build/vscpp-step-0-installation?view=vs-2019> – инструкция по установке компилятора языка C.
9. <https://floppy.ru/2016/05/19/text-tanimoto/> – страница, описывающая меру Жаккара.
10. <http://www.nltk.org/> – официальная страница библиотеки nltk.
11. <http://oraclestart.blogspot.com/2013/08/oracle-text.html> – полнотекстовый поиск в Oracle.
12. [https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D0%BE%D0%BD\\_%D0%A6%D0%B8%D0%BF%D1%84%D0%B0](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D0%BE%D0%BD_%D0%A6%D0%B8%D0%BF%D1%84%D0%B0) – Википедия о законе Ципфа.

# Полезные ссылки

---

13. <https://github.com/stopwords-iso/stopwords-ru/blob/master/stopwords-ru.txt> – коллекция стоп-слов Русского языка.
14. [https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D0%BE%D0%BD\\_%D0%A5%D0%B8%D0%BF%D1%81%D0%B0](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D0%BE%D0%BD_%D0%A5%D0%B8%D0%BF%D1%81%D0%B0) – Википедия о законе Хипса.