Базовые методы обработки данных с использованием Python

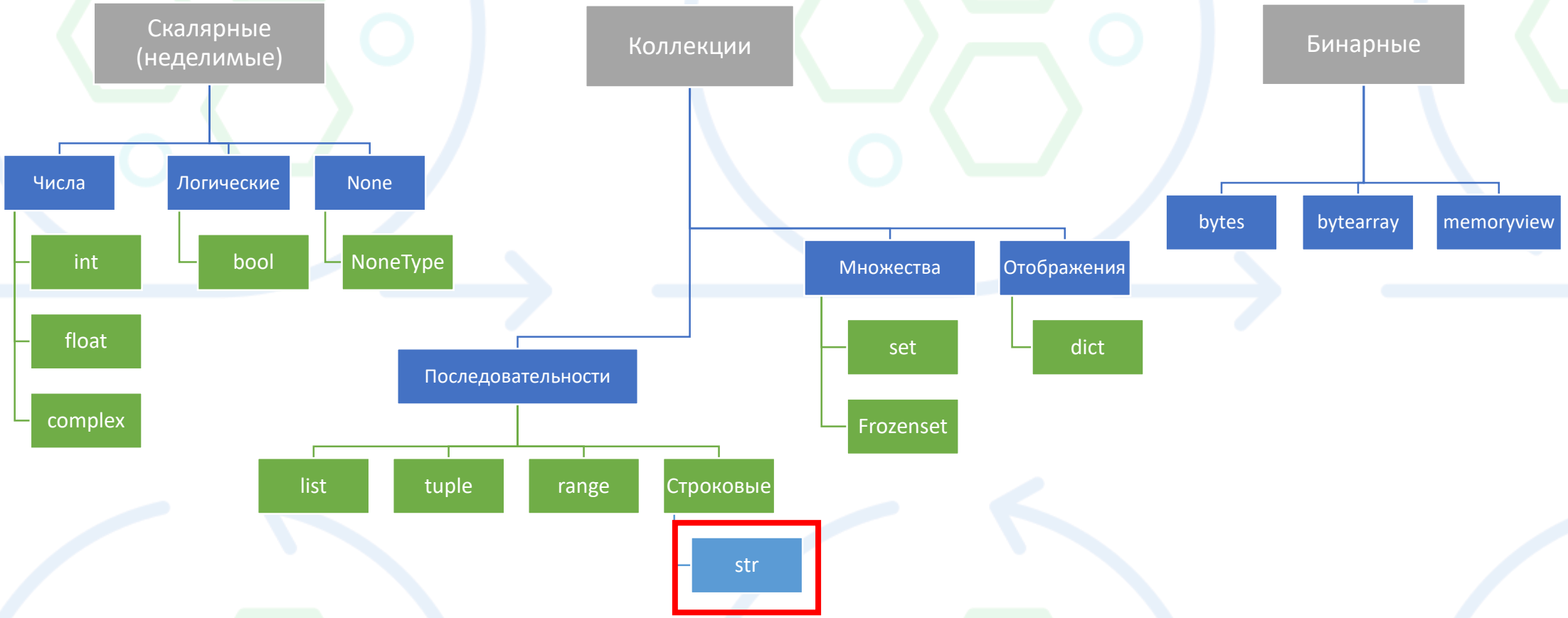
Лекция 5. Обработка текстовых данных в Python. Строковые методы. Регулярные выражения. Основы токенизации и векторизации текстов

Киреев Василий Сергеевич

к.т.н., доцент

Москва, 2024

Встроенные типы данных в Python



Строки

В Python строки - это упорядоченная последовательность символов Unicode, , используемые для хранения и представления текстовой информации, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме.

Каждый символ в строке имеет уникальный индекс в последовательности. Индекс начинается с 0. Первый символ в строке имеет позиционный индекс 0. Индекс продолжает увеличиваться по направлению к концу строки.

Строки. Экранирование

Экранированная последовательность	**Назначение**
\n	Перевод строки
\a	Звонок
\b	Забой
\f	Перевод страницы
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\N{id}	Идентификатор ID базы данных Юникода
\uhhhh	16-битовый символ Юникода в 16-ричном представлении
\Uhhhh…	32-битовый символ Юникода в 32-ричном представлении
\xhh	16-ричное значение символа
\ooo	8-ричное значение символа
\0	Символ Null (не является признаком конца строки)

Если перед открывающей кавычкой стоит символ 'r' (в любом регистре), то механизм экранирования отключается.

Строки. Методы

Методы	Описание
S = 'str'; S = "str"; S = '''str'''; S = """str"""	Литералы строк
S = "s\np\ta\nbbb"	Экранированные последовательности
S = r"C:\temp\new"	Неформатированные строки (подавляют экранирование)
S = b"byte"	Строка байтов
S1 + S2	Конкатенация (сложение строк)
S1 * 3	Повторение строки
S[i]	Обращение по индексу
S[i:j:step]	Извлечение среза
len(S)	Длина строки
S.find(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.rfind(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или -1

Строки. Методы

Методы	Описание
S.index(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
S.rindex(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError
S.replace(шаблон, замена[, maxcount])	Замена шаблона на замену. maxcount ограничивает количество замен
S.split(символ)	Разбиение строки по разделителю
S.isdigit()	Состоит ли строка из цифр
S.isalpha()	Состоит ли строка из букв
S.isalnum()	Состоит ли строка из цифр или букв
S.islower()	Состоит ли строка из символов в нижнем регистре
S.isupper()	Состоит ли строка из символов в верхнем регистре
S.index(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
S.rindex(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError

Строки. Методы

Методы	Описание
S.isspace()	Состоит ли строка из неотображаемых символов (пробел, символ перевода страницы ('\f'), "новая строка" ('\n'), "перевод каретки" ('\r'), "горизонтальная табуляция" ('\t') и "вертикальная табуляция" ('\v'))
S.istitle()	Начинаются ли слова в строке с заглавной буквы
S.upper()	Преобразование строки к верхнему регистру
S.lower()	Преобразование строки к нижнему регистру
S.startswith(str)	Начинается ли строка S с шаблона str
S.endswith(str)	Заканчивается ли строка S шаблоном str
S.join(список)	Сборка строки из списка с разделителем S
ord(символ)	Символ в его код ASCII
chr(число)	Код ASCII в символ
S.capitalize()	Переводит первый символ строки в верхний регистр, а все остальные в нижний

Строки. Методы

Методы	Описание
S.center(width, [fill])	Возвращает отцентрованную строку, по краям которой стоит символ fill (пробел по умолчанию)
S.count(str, [start],[end])	Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию)
S.expandtabs([tabsize])	Возвращает копию строки, в которой все символы табуляции заменяются одним или несколькими пробелами, в зависимости от текущего столбца. Если TabSize не указан, размер табуляции полагается равным 8 пробелам
S.lstrip([chars])	Удаление пробельных символов в начале строки
S.rstrip([chars])	Удаление пробельных символов в конце строки
S.strip([chars])	Удаление пробельных символов в начале и в конце строки
S.partition(шаблон)	Возвращает кортеж, содержащий часть перед первым шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий саму строку, а затем две пустых строки
S.rpartition(sep)	Возвращает кортеж, содержащий часть перед последним шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий две пустых строки, а затем саму строку
S.center(width, [fill])	Возвращает отцентрованную строку, по краям которой стоит символ fill (пробел по умолчанию)
S.count(str, [start],[end])	Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию)
S.expandtabs([tabsize])	Возвращает копию строки, в которой все символы табуляции заменяются одним или несколькими пробелами, в зависимости от текущего столбца. Если TabSize не указан, размер табуляции полагается равным 8 пробелам

Строки. Методы

Методы	Описание
S.swapcase()	Переводит символы нижнего регистра в верхний, а верхнего – в нижний
S.title()	Первую букву каждого слова переводит в верхний регистр, а все остальные в нижний
S.zfill(width)	Делает длину строки не меньшей width, по необходимости заполняя первые символы нулями
S.ljust(width, fillchar=" ")	Делает длину строки не меньшей width, по необходимости заполняя последние символы символом fillchar
S.rjust(width, fillchar=" ")	Делает длину строки не меньшей width, по необходимости заполняя первые символы символом fillchar
S.format(*args, **kwargs)	Форматирование строки
S.swapcase()	Переводит символы нижнего регистра в верхний, а верхнего – в нижний
S.title()	Первую букву каждого слова переводит в верхний регистр, а все остальные в нижний
S.zfill(width)	Делает длину строки не меньшей width, по необходимости заполняя первые символы нулями
S.ljust(width, fillchar=" ")	Делает длину строки не меньшей width, по необходимости заполняя последние символы символом fillchar
S.rjust(width, fillchar=" ")	Делает длину строки не меньшей width, по необходимости заполняя первые символы символом fillchar

Строки. Доступ к символам

Python позволяет обращаться к любому отдельному символу из строки по его индексу. В данном случае 0 - это нижняя граница, а 11 - верхняя граница строки. Таким образом, var[0] возвращает H, var[6] - P. Если индекс в квадратных скобках превышает верхнюю границу, Python выдает ошибку IndexError.

```
var="HELLO PYTHON"
print(var[0], var[7], var[-1])
```

```
H Y N
```

Строки. Изменение

В Python строка (объект класса str) имеет неизменяемый тип. Неизменяемый объект - это объект, который может быть изменен на месте, созданный в памяти. Поэтому, в отличие от списка, любой символ в последовательности нельзя перезаписать, вставить или добавить в нее, если только не использовать определенный метод string, который возвращает новый объект string.

Поскольку объекты string и list являются последовательностями, они взаимопревращаемы. Следовательно, если привести строковый объект к списку, модифицировать список методами insert(), append() или remove() и преобразовать список обратно в строку, то мы получим модифицированную версию

```
s1="СЛОВО"
print ("исходная строка:", s1)
l1=list(s1)
l1[-1]='A'
print ("Измененная строка:", ''.join(l1))
```

```
исходная строка: СЛОВО
Измененная строка: СЛОВА
```

Строки. Форматирование

Форматирование строк - это процесс динамического построения строкового представления путем вставки значений числовых выражений в уже существующую строку.

Оператор конкатенации строк в Python не принимает операнд, не являющийся строкой. Поэтому Python предлагает следующие приемы форматирования строк

- Использование оператора % для подстановки
- Использование метода format() класса str
- Использование синтаксиса f-строк
- Использование класса String Template

Строки. Форматирование. Оператор %

Одной из самых замечательных возможностей Python является оператор формата строк %. Этот оператор уникален для строк и компенсирует отсутствие функций из семейства printf() языка C. Символы спецификации формата (%d %c %f %s и т.д.), используемые в языке C, используются в строке в качестве заполнителей.

```
имя=" Rajesh "
возраст=30
print ("Меня зовут %s, а мой возраст
составляет %d лет" % (имя, возраст))
```

```
Меня зовут Rajesh, а мой возраст составляет
30 лет
```

Строки. Методы

Формат	Символ и преобразование
%c	символ
%s	преобразование строки с помощью функции str() перед форматированием
%i	знаковое десятичное целое число
%d	целое знаковое десятичное число
%u	беззнаковое десятичное целое число
%o	восьмеричное целое число
%x	шестнадцатеричное целое число (строчные буквы)
%X	шестнадцатеричное целое число (заглавные буквы)
%e	экспоненциальная нотация (со строчными буквами 'e')
%E	экспоненциальная нотация (с верхним регистром 'E')
%f	вещественное число с плавающей запятой

Строки. Форматирование. Метод f-строк

В версии 3.6 в Python появился новый метод форматирования строк - f-strings или Literal String Interpolation. С помощью этого метода форматирования можно использовать встроенные выражения Python внутри строковых констант.

Python f-строки являются более быстрыми, читабельными, лаконичными и менее подверженными ошибкам. Строка начинается с префикса 'f', в нее вставляется один или несколько держателей, значение которых заполняется динамически.

```
имя = 'Rajesh'
возраст = 30
fstring = f'Меня зовут {имя} и мне {возраст}
лет'
print (fstring)
```

```
Меня зовут Rajesh и мне 30 лет
```

Строки. Форматирование. Класс Template

В стандартной библиотеке Python имеется модуль string. Он предоставляет функциональные возможности для выполнения различных операций со строками. Класс Template в модуле string предоставляет альтернативный метод динамического форматирования строк. Одним из преимуществ класса Template является возможность настраивать правила форматирования.

Реализация класса Template использует регулярные выражения для поиска общего шаблона допустимых шаблонных строк. Правильная строка шаблона, или placeholder, состоит из двух частей: Символ $, за которым следует допустимый идентификатор Python. Необходимо создать объект класса Template и использовать шаблонную строку в качестве аргумента конструктора. Далее следует вызвать метод substitute() класса Template. Он подставляет вместо шаблонных строк значения, указанные в качестве параметров.

```
from string import Template
temp_str = "Меня зовут $name и мне $age лет"
tempobj = Template(temp_str)
ret = tempobj.substitute(name='Rajesh', age=30)
print (ret)
```

```
Меня зовут Rajesh и мне 30 лет
```

Регулярные выражения

Регулярное выражение - это последовательность символов, которая формирует шаблон поиска. Регулярное выражение можно использовать для проверки того, содержит ли строка указанный шаблон поиска, например:

- поиска телефонного номера или email-адреса
- для разбивки строк на подстроки
- для поиска, замены и извлечения символов
- для быстрого выполнения нетривиальных операций

Регулярные выражения. Наборы символов

Регулярное выражение	Результат
[abc]	a, b, или c (простой набор)
[^abc]	Любой символ кроме a, b, или c (исключение)
[a-zA-Z]	Любые символы латинского алфавита, от a до z и от A до Z включительно

Регулярные выражения. Наборы символов. Пример

Выражение	**Строка**	**Результат**
[abc]	a	+
	ac	++
	Hey Jude	-

Регулярные выражения. Предустановленные наборы символов

Регулярное выражение	Результат
.	Любой символ (кроме символов конца строки)
\d	Цифра: [0-9]
\D	Не цифра: [^0-9]
\s	Любой пробельный символ: [\t\n\x0B\f\r]
\S	Любой не пробельный символ: [^\s]
\w	Любой буквенный или цифровой символ, а также знак подчёркивания: [a-zA-Z_0-9]
\W	Любой символ кроме буквенного и цифрового, а также знака подчёркивания: [^\w]

Регулярные выражения. Предустановленные наборы символов. Пример

Выражение	Строка	Результат
[0-38]	a	-
	9	-
	d01	++

Регулярные выражения. Границы

Регулярное выражение	Результат
^	Начало строки
$	Конец строки
\b	Граница слова
\B	Не граница слова

Регулярные выражения. Границы. Пример

Выражение	Строка	Результат
\bfoo	football	+
	a football	+
	afootball	-

Регулярные выражения. Жадные кванторы

Регулярное выражение	Результат
X?	X, один раз или ни разу
X*	X, ноль или более раз
X+	X, один или более раз

Регулярные выражения. Жадные кванторы. Пример

Выражение	Строка	Результат
ma+n	mn	-
	man	+
	maaan	+

Регулярные выражения. Ленивые кванторы

Регулярное выражение	Результат
X?	X, один раз или ни разу
X*	X, ноль или более раз
X+	X, один или более раз

Регулярные выражения. Логические операторы

Регулярное выражение	Результат
XY	X, за которым идёт Y
X|Y	Либо X, либо Y
(XY)	XY как отдельная группа

Регулярные выражения. Логические операторы. Пример

Выражение	Строка	Результат
a|b	cde	-
	ade	+
	acdbea	+++

Регулярные выражения. Модуль re в Python

Python имеет встроенный пакет с именем re, который можно использовать для работы с регулярными выражениями. Список часто используемых функций:

Функция	Назначение
findall()	Возвращает список, содержащий все совпадения
search()	Возвращает объект Match, если где-либо в строке есть совпадение
split()	Возвращает список, в котором строка была разделена при каждом совпадении
sub()	Заменяет одно или несколько совпадений строкой
subn()	Делает то же самое, что и sub(), но возвращает новую строку и количество замен
match()	Ищет совпадение с начала строки
compile()	Компилирует regular expression, на выходе получаем объект, к которому затем можно применять все перечисленные функции

Регулярные выражения. Модуль re в Python. Функция search

```python
import re

txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

```
['ai', 'ai']
```

Регулярные выражения. Модуль re в Python. Функция findall()

```
1 import re
2
3 s = 'geeks.forgeeks'
4
5 match = re.search(r'.', s)
6 print(match)
```

```
<re.Match object; span=(0, 1), match='g'>
```

Регулярные выражения. Модуль re в Python. Функция split()

```
1 import re
2
3 txt = "The rain in Spain"
4 x = re.split("\s", txt)
5 print(x)
```

```
['The', 'rain', 'in', 'Spain']
```

Регулярные выражения. Модуль re в Python. Функция sub()

```python
import re

txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

```
The9rain9in9Spain
```

Регулярные выражения. Модуль re в Python. Функция match()

```python
import re

txt = "The rain in Spain"
x = re.search("ai", txt)
print(x)
```

```
<re.Match object; span=(5, 7), match='ai'>
```

Регулярные выражения. Модуль re в Python. Функция compile

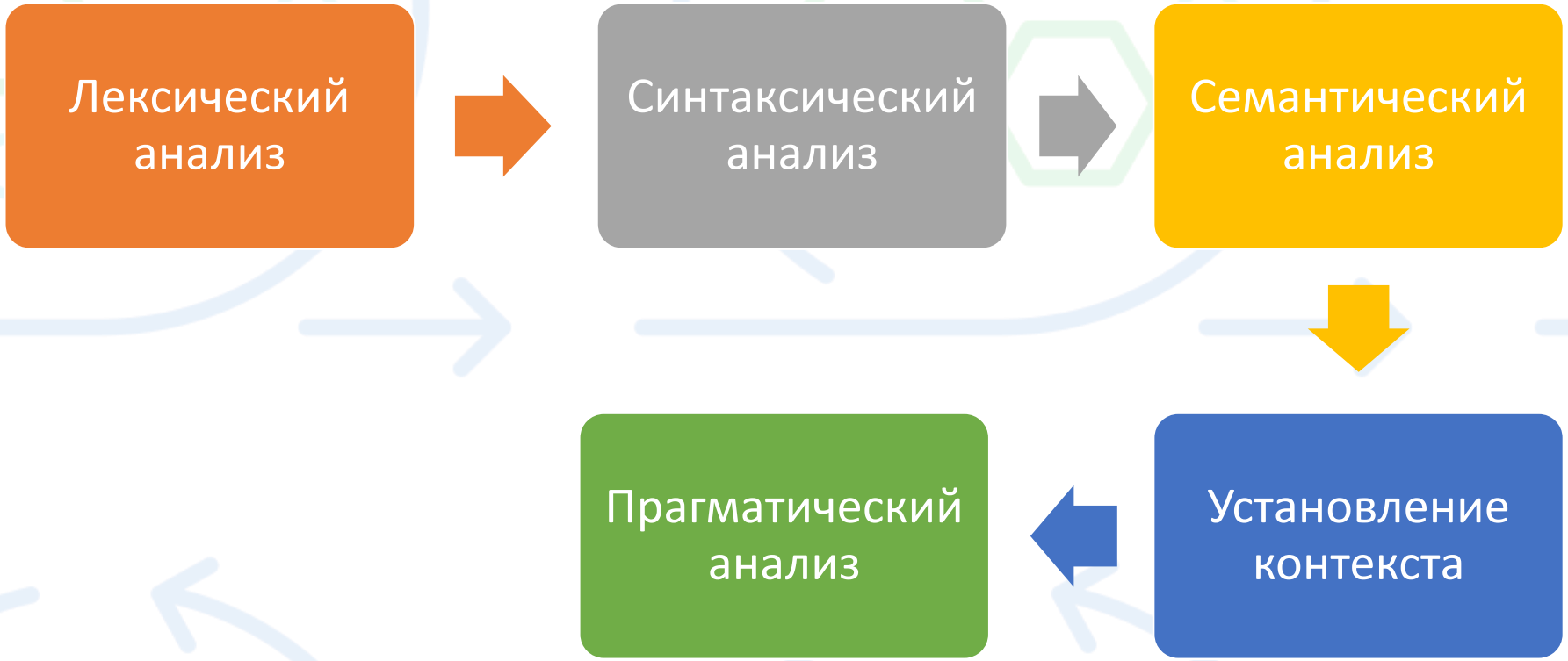
```python
import re

s = "Emma's luck numbers are 251 761 231 451"

pattern = r"\d{3}"
regex_pattern = re.compile(pattern)
print(type(regex_pattern))
result = regex_pattern.findall(s)
print(result)
```

```
<class 're.Pattern'>
['251', '761', '231', '451']
```

Этапы обработки текстов на естественном языке



Векторизация текста

Векторизация текста включает в себя преобразование нечисловых данных, таких как текст, в числовые векторы. Для этого есть две причины: модели машинного обучения требуют числового ввода, и поэтому сначала нужно преобразовать нечисловые данные (например, текст и категории) в векторы. Вторая причина кроется в возможности визуализации текстовых данных после их векторизации.

Векторизация текста. Понятия

Текст разбивается на текстовые единицы (токены), например, символы, слова, словосочетания, предложения, абзацы и т.д. Чаще всего токен соответствует слову.

Токены образуют словарь, который может быть отсортирован по алфавиту.

Документ – это совокупность токенов, которые принадлежат одной смысловой единице. В качестве документа может выступать предложение, комментарий или пост пользователя.

Корпус – это генеральная совокупность всех документов.

Векторизация текста (создание эмбеддингов – векторных представлений слов)

Создание векторных представлений слов имеет основополагающее значение в НЛП по нескольким причинам:

- Снижение размерности : они представляют слова в непрерывном векторном пространстве меньшей размерности, что позволяет эффективно обрабатывать обширные словари.
- Семантическое сходство : вложения слов кодируют семантические отношения, позволяя алгоритмам понимать синонимы, антонимы и связанные значения.
- Контекстная информация: фиксируя контекст из окружающих слов, встраивания помогают моделям понимать значение слова в контексте, что имеет решающее значение для таких задач, как анализ настроений и распознавание именованных сущностей.
- Обобщение: Они хорошо обобщают новые для модели слова, изучая свойства распределения слов в обучающих данных.
- Представление признаков: векторные представления служат в качестве представлений признаков для моделей машинного обучения, позволяя применять различные методы к задачам обработки естественного языка.
- Эффективное обучение: модели, обученные с использованием векторных представлений слов, сходятся быстрее и часто работают лучше, чем модели, использующие разреженные представления.
- Трансферное обучение: предварительно обученные эмбеддинги, такие как Word2Vec или GloVe, позволяют моделям использовать знания из больших корпусов даже при ограниченном объеме данных, специфичных для конкретной задачи.

Векторизация текста. Методы

Прямое (горячее) кодирование

Мешок слов

Мешок слов (Tf-Idf)

Векторные представления (эмбеддинги)

Векторизация текста. Прямое кодирование

Прямое кодирование (one-hot encoding) считается самым простым способом преобразования токенов в тензоры и выполняется следующим образом:

- каждый токен представляет бинарный вектор (значения 0 или 1);
- единица ставится тому элементу, который соответствует номеру токена в словаре.

Векторизация текста. Прямое кодирование. Пример

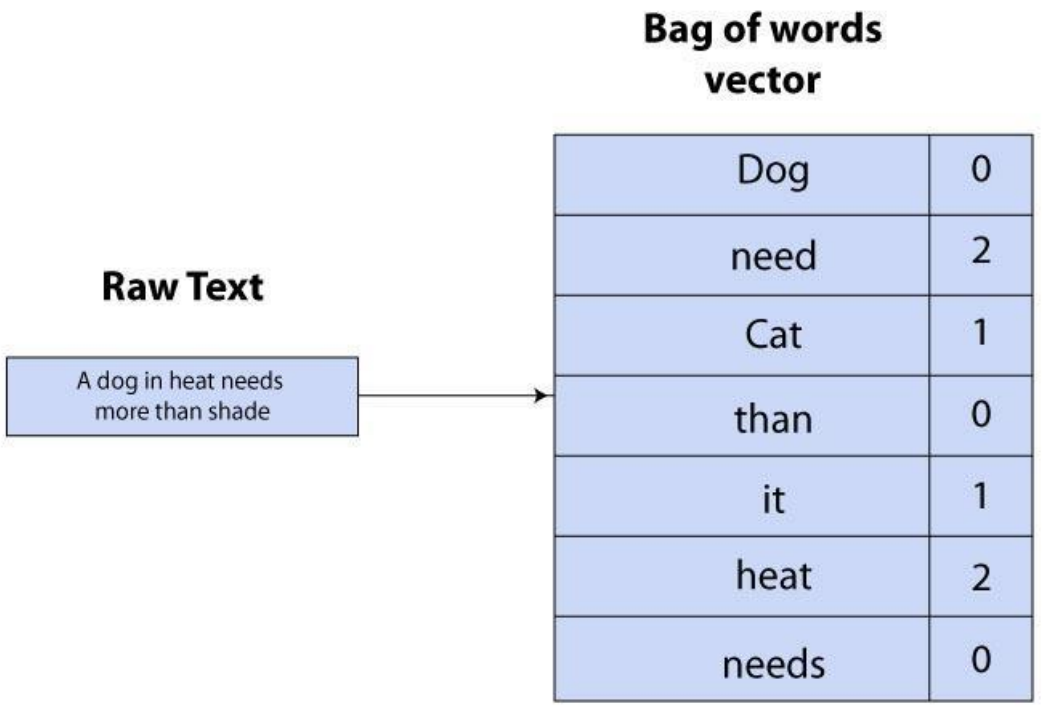
```
1 text='Это первый документ.'
2
3 tokens = set(text.lower().split())
4 length = len(tokens)
5 index_map = {x:index for x,index in zip(tokens,range(length))}
6 ohe_matrix = []
7
8 for token in tokens:
9             ohe = np.zeros(length)
10            ohe[index_map[token]] = 1
11            print(token,ohe)
12            ohe_matrix.append(ohe)
```

```
это [1. 0. 0.]
первый [0. 1. 0.]
документ. [0. 0. 1.]
```

Векторизация текста. Мешок слов

Мешок слов (Bag of words) выделяет вектору весь документ, и каждый элемент кодируется 1 по порядку следования слов в словаре. Bag of words решает проблему размерности по одной оси. Количество строк определяется количеством документов. Однако, этот метод не учитывает важность того или иного токена, ведь одно слово может повторятся по несколько раз.

Векторизация текста. Мешок слов. Визуальное представление



Векторизация текста. Мешок слов. Пример

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 corpus = ['Это первый документ.',
3  'Этот документ является вторым документом.',
4  'А это третий документ.',
5 'Это первый документ?',
6  ]
7 vectorizer = CountVectorizer()
8 X = vectorizer.fit_transform(corpus)
9 vectorizer.get_feature_names_out()
```

```
array(['вторым', 'документ', 'документом', 'первый', 'третий', 'это',
       'этот', 'является'], dtype=object)
```

Векторизация текста. TF-IDF

Схема TF-IDF представляет собой разновидность подхода bag words, при котором вместо нулей и единиц в вектор вложения добавляются плавающие числа, содержащие больше полезной информации по сравнению с нулями и единицами.

Идея схемы TF-IDF заключается в том, что слова, имеющие высокую частоту встречаемости в одном документе и меньшую частоту встречаемости во всех остальных документах, являются более важными для классификации.

Векторизация текста. Мешок слов (Tf-Idf)

TF-IDF состоит из двух компонентов: Term Frequency (частотность слова в документе) и Inverse Document Frequency (инверсия частоты документа). В TF-IDF редкие слова и слова, которые встречаются во всех документах, несут мало информации.

$$TF\text{-}IDF = TF \times IDF$$

Векторизация текста. Мешок слов (Tf-Idf). Tf

Частота употребления термина (Term Frequency) - это просто количество слов, присутствующих в предложении. TF в основном отражает важность слова независимо от длины документа.

$$TF_{token_i} = \frac{n_i}{N_i},$$

n_i — сколько раз встречается токен в i-ом документе,
N_i — общее количество токенов в i-ом документе,

Векторизация текста. Мешок слов (Tf-Idf). Idf

IDF каждого слова - это логарифм отношения общего количества строк к количеству строк в конкретном документе, в котором присутствует это слово. IDF будет измерять редкость того или иного термина. Такие слова, как “я” и “и”, встречаются во всех документах корпуса, но редкие слова есть не во всех документах.

$$IDF_{token} = \log \frac{p}{P},$$

p — количество документов, в которых встречается токен,
P — общее количеств документов.

Векторизация текста. Мешок слов (Tf-Idf). Пример

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
2 corpus = ['Это первый документ.',
3  'Этот документ является вторым документом.',
4  'А это третий документ.',
5 'Это первый документ?',
6  ]
7 vectorizer = TfidfVectorizer()
8 X = vectorizer.fit_transform(corpus)
9 vectorizer.get_feature_names_out()
```

```
array(['вторым', 'документ', 'документом', 'первый', 'третий', 'это',
       'этот', 'является'], dtype=object)
```

Спасибо за внимание!