Занятие 9: Обработка текстов в Python

Практикум на ЭВМ 2020/2021

Александр Чернышёв

МГУ имени М. В. Ломоносова, факультет ВМК, кафедра ММП

26 октября 2020 г.

Задачи обработки текстов

Основная цель обработки текстов: улучшение качества работы алгоритмов машинного обучения

- Токенизация
  I'm — один токен или два?
- Определение границ предложения
  Mr. Bing — одно предложение или два?
- Нормализация (стемминг и лемматизация)
  Красивый, красивая, красивое — разные токены?
- Отбор признаков (токенов)
  Нужны ли признаки для слов то, либо, нибудь?
- Выделение коллокаций (n-грамм)
  Метод опорных векторов — коллокация

Модель Bag of words

$D = \{d_1, d_2 \dots d_N\}$ — обучающая коллекция документов

Каждый документ состоит из токенов из словаря W:

$d_i = (w_1, w_2, \dots w_{n_d}), \quad n_d$ — длина документа d

Модель мешка слов (bag of words):
Порядок слов в документе не важен, важно число вхождений каждого токена в документ
Каждый документ представляется вектором длины |W|:

$tf(w, d)$ — сколько раз w встречался в d

$v(d) = [tf(w, d)]_{w \in W}$

Варианты выбора tf

$tf(w, d)$ — term frequency weight

Есть разные определения $tf(w, d)$:

$$tf(w, d) = \sum_{w' \in d} \mathbb{I}[w = w'] = n_{wd}$$

$$tf(w, d) = \mathbb{I}[w \in d]$$

$$tf(w, d) = \frac{n_{wd}}{n_d}$$

$$tf(w, d) = 1 + \log(n_{wd})$$

Модель TF-IDF

$idf(w)$ — обратная документная частота

Есть несколько определений, в scikit-learn по умолчанию используется:

$$idf(w) = \log\left(\frac{N}{\sum_{i=1}^{N} \mathbb{I}[w \in d_i]}\right) + 1$$

Модель TF-IDF:
Каждый документ представляется вектором длины $|W|$

$$v(d) = [tf(w,d) \cdot idf(w)]_{w \in W}$$

Реализации моделей в scikit-learn

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> s = ['my name is',
...      'your name are',
...      'my father is']
>>> vectorizer = CountVectorizer()
>>> vectorizer.fit_transform(s).toarray()
array([[0, 0, 1, 1, 1, 0],
       [1, 0, 0, 0, 1, 1],
       [0, 1, 1, 1, 0, 0]], dtype=int64)
>>> vectorizer.get_feature_names()
['are', 'father', 'is', 'my', 'name', 'your']
>>> vectorizer = TfidfVectorizer()
>>> vectorizer.fit_transform(s).toarray()
array([[ 0.,  0.,  0.57735,  0.57735,  0.57735,  0.],
       [ 0.62276,  0.,  0.,  0.,  0.47362 ,  0.622766,
       [ 0.,  0.68091,  0.51785,  0.51785,  0.,  0.]])
```

BOW и TF-IDF в сравнении с современными моделями

Model	AG news	DBpedia	Yelp15
BoW*	88.8	96.6	-
ngrams*	92.0	98.6	-
ngrams TFIDF*	92.4	98.7	-
char-CNN*	87.2	98.3	-
char-CRNN⋆	91.4	98.6	-
VDCNN◇	91.3	98.7	-
SVM+TF†	-	-	62.4
CNN†	-	-	61.5
Conv-GRNN†	-	-	66.0
LSTM-GRNN†	-	-	67.6
fastText (ngrams=1)‡	91.5	98.1	**62.2
StarSpace (ngrams=1)	91.6	98.3	62.4
fastText (ngrams=2)‡	92.5	98.6	-
StarSpace (ngrams=2)	92.7	98.6	-
fastText (ngrams=5)‡	-	-	66.6
StarSpace (ngrams=5)	-	-	65.3

Table 2: Text classification test accuracy. * indicates models from (Zhang and LeCun 2015); ⋆ from (Xiao and Cho 2016); ◇ from (Conneau et al. 2016); † from (Tang, Qin, and Liu 2015); ‡ from (Joulin et al. 2016); ** we ran ourselves.

Свойства признаковых представлений

Два свойства рассмотренных представлений:

- Огромная размерность
  (полный словарь одного языка $\approx 5 * 10^5$ слов)
- Сильная разреженность

Как бороться с огромной размерностью?

Свойства признаковых представлений

Два свойства рассмотренных представлений:

- Огромная размерность
  (полный словарь одного языка $\approx 5 * 10^5$ слов)
- Сильная разреженность

Как бороться с огромной размерностью?

- Методы понижения размерности (PCA)
- Предобработка данных
- Использовать другие представления (word embeddings, topic modeling)

Удаление лишних символов

Обычно, не хотим различать слова с заглавной и строчной буквами ⇒ перед работой приводим строки в нижний регистр

```
>>> s.lower()
```

Обычно, не хотим использовать не буквы и не цифры ⇒ удалим все лишние символы
Воспользуемся библиотекой `re` для работы с регулярными выражениями:

```
>>> import re
>>> s = 'mothe23242rsss мамау224чк2а'
>>> re.sub('[^a-zа-яё ]', '', s)
'mothersss мамаучка'
```

re.search как str.find

re.search может работать как str.find:

```
>>> import re
>>> long_line = 'Oh my god it is very hard'


>>> long_line.find('is very')
13
>>> re.search('is very', long_line)
<_sre.SRE_Match object; span=(13, 20), match='is very'>
```

re.search как str.startwith

re.search может работать как str.startwith:

```
>>> import re
>>> long_line = 'Oh my god it is very hard'


>>> long_line.startswith('is very')
False
>>> re.search('^is very', long_line) # результат None
>>> re.search('^Oh', long_line)
<_sre.SRE_Match object; span=(0, 2), match='Oh'>
```

Регулярные выражения в `re`

- `^` — начало строки
- `$` — конец строки
- `.` — любой символ
- `\s` — любой пробельный символ
- `\S` — любой НЕ пробельный символ
- `*` — любая последовательность из символа перед звёздочкой (в том числе и нулевой длины)
- `+` — любая ненулевая последовательность из символа перед плюсом

```
>>> long_line = 'Oh myyy          god it is very hard'
>>> re.search('Oh my*\s*', long_line)
<_sre.SRE_Match object; span=(0, 15), match='Oh myyy          '>
```

Регулярные выражения в re

- [abcd] — любой символ из a, b, c, d
- [a-z] — любой символ с a по z
- [^xy] — любой символ, не совпадающий с символами x, y

```
>>> long_line = 'Oh myyy god it is 12213'
>>> re.search('[^a-zA-Z ]+', long_line)
<_sre.SRE_Match object; span=(18, 23), match='12213'>
```

Функции re

`re.compile(pattern)` — строковое представление выражения преобразуется в программное.

```
>>> regex = re.compile(u'[^a-zа-яё ]+')
```

`re.sub(pattern, repl, string, count=0)` — заменить count вхождений в string, удовлетворяющих pattern, на repl

```
>>> re.sub('[^0-9]', '!', '1995 year was...', count=4)
'1995!!!!r was...' # если count=0 заменяет все подходящие
```

`re.split(pattern, string, maxsplit=0)` — split по вхождениям, удовлетворяющим pattern

```
>>> re.split('[^a-z ]', "i look for. for many years. in")
['i look for', ' for many years', ' in']
```

Пример использования регулярных выражений

Регулярные выражения, служащие для определения заголовков публикаций, быстро теряющих актуальность:

```
.*онлайн-|-трансляция.*
.*в эт(у?и?) минут.*

.*в эт((от)?и?) час.*

.*в этот день.*

.*[0-9]{2}[0-9]{2}( .*|:*|:.*|;.*|)

.*[0-9]{1,2}[0-9]{2}([0-9]{2}|[0-9]{4})(

.*|:*|:.*|;.*|)
.*(от|за|на) [0-9]{1,2} (январ|феврал|март|апрел

|ию(н|л)|август|сентябр|ноябр|октябр|декабр|ма(я|й)).*
.*(от|за|на) [0-9]{1,2}[0-9]{2}( .*|:*|:.*|;.*|)
```

Токенизация

Токенизация - разделение текста на токены, элементарные единицы текста

В большинстве случаев токен это слово!

Если пользоваться методом `.split()`, токен — последовательность букв, разделённая пробельным символам

Можно использовать регулярные выражения и модуль `re`

Можно использовать специальные токенизаторы, например из `nltk`:

- RegexpTokenizer
- BlanklineTokenizer
- И ещё около десятка штук

Токенизация с помощью re

```
>>> sentence = """
Контактную информацию вы можете уточнить, перейдя по ссылке
https://minust.ru/structure/00000000000
(выбрать раздел &quot;Территориальные органы и подведомственные
организации&quot;, выбрать регион и открыть вкладку
&quot;Информация и контакты&quot;)
"""
>>> sentence.split()[9:]
['(выбрать', 'раздел', '&quot;Территориальные', 'органы',
'и', 'подведомственные', 'организации&quot;,',
'выбрать', 'регион', 'и', 'открыть', 'вкладку',
'&quot;Информация', 'и', 'контакты&quot;)']
>>> re.split('[^а-яА-ЯёЁ]+', sentence)
['Контактную','информацию','вы','можете', 'уточнить','перейдя',
'по','ссылке', 'выбрать','раздел','Территориальные', 'органы',
'и','подведомственные', 'организации','выбрать','регион','и',
'открыть','вкладку', 'Информация','и', 'контакты','']
```

Принцип работы токенизатора из библиотеки spacy

1. Разбить предложения на как можно большее число частей
2. Соединить части, записанные как исключения

Примеры заложенных исключений:

```
[[{'ORTH': 'Сан'}, {'ORTH': '-'}, {}],
[{'ORTH': 'Сен'}, {'ORTH': '-'}, {}],
[{'ORTH': 'Сент'}, {'ORTH': '-'}, {}],
[{'ORTH': 'Мак'}, {'ORTH': '-'}, {}],
[{'LOWER': 'вверх'}, {'ORTH': '-'}, {'LOWER': 'вниз'}],
[{'LOWER': 'чуть'}, {'ORTH': '-'}, {'LOWER': 'чуть'}],
[{'LOWER': 'чуть'}, {'ORTH': '-'}, {'LOWER': 'чуть'}],
[{'IS_DIGIT': True}, {'ORTH': '-'}, {'LOWER': 'й'}],
[{'IS_DIGIT': True}, {'ORTH': '-'}, {'LOWER': 'х'}],
[{'IS_DIGIT': True}, {'ORTH': '-'}, {'LOWER': 'м'}],
...
```

Разделение на предложения

Простой вариант:

```
>>> sent = 'Hey! Is Mr. Bing waiting for you?'
>>> re.split('[!.?]+', sent)
['Hey', ' Is Mr', ' Bing waiting for you', '']
```

Можно учитывать разные исключения при разделении:

```
>>> sentenceEnders = re.compile(r"""
    (?: # Group for two positive lookbehinds.
    (?<=[.!?]) # Either an end of sentence punct,
    | (?<=[.!?]['"]) # or end of sentence punct and quote.
    ) # End group of two positive lookbehinds.
    (?<! Mr\. ) # Don't end sentence on "Mr."
    (?<! Mrs\. ) # Don't end sentence on "Mrs."
    \s+ # Split on whitespace between sentences.
    """, re.IGNORECASE | re.VERBOSE)
>>> sentenceEnders.split(sent)
['Hey!', 'Is Mr. Bing waiting for you?']
```

Отбор слов

Какие слова могут быть плохие?

- Слишком частые
  русский язык: и, но, я, ты, ...
  английский язык: a, the, I, one, ...
  специфичные для коллекции: «сообщать» в новостях
- Слишком редкие
  (встречаются в ≤ 5 документах)
- Стоп-слова
  (предлоги, междометия, частицы, цифры)

```
>>> from nltk.corpus import stopwords
>>> stopWords = set(stopwords.words('english'))
>>> list(stopWords)[:6]
['doesn', 'of', 'most', 'am', 'and', 'not']
```

Замена сокращений с помощью re.sub

Удаление сокращений в английском языке:

```
>>> def delete_to_be(text_string):
...     return re.sub("('s|'re|n't)\s",u' <stop> ',
...                   text_string.lower())
>>> delete_to_be("Where's your spoon daddy")
'where <stop> your spoon daddy'
>>> delete_to_be("Why don't you like me")
'why do <stop> you like me'
```

Замена специфичных сущностей на теги

```
>>> result_string = re.sub('&quot', ' " ', sentence)
>>> result_string = re.sub('(http|www)\S+', ' <URL> ',
...                        result_string)
>>> result_string = re.sub('([^\w\s<>])', ' \\1 ',
...                        result_string)
>>> " ".join(result_string.split())
['Контактную','информацию', 'вы', 'можете', 'уточнить',
',', 'перейдя', 'по', 'ссылке', '<URL>', '(', 'выбрать',
'раздел', '"', ';', 'Территориальные', 'органы',
'и', 'подведомственные', 'организации',
'"',';',',','выбрать','регион','и','открыть',
'вкладку', '"', ';','Информация','и','контакты',
'"', ';', ')']
```

Нормализация слов — стемминг

Стемминг — отбрасывание окончаний слов

```
>>> from nltk.stem.snowball import SnowballStemmer
>>> stemmer = SnowballStemmer(language='english')
>>> sentence = 'George admitted the talks happened'.split()
>>> " ".join([stemmer.stem(word) for word in sentence])
'georg admit the talk happen'
```

```
>>> sentence = 'write wrote written'.split()
" ".join([stemmer.stem(word) for word in sentence])
'write wrote written'
```

Стемминг для русского языка

Для русского языка стемминг не очень подходит

```
>>> stemmer = SnowballStemmer(language='russian')
>>> sentence = 'опрошенных считают налоги необходимыми'.split()
>>> " ".join([stemmer.stem(word) for word in sentence])
'опрошен счита налог необходим'
```

```
>>> sentence = 'поле пол полёт полка полк'.split()
>>> " ".join([stemmer.stem(word) for word in sentence])
'пол пол полет полк полк'
>>> sentence = 'крутой круче крутейший крутить'.split()
>>> " ".join([stemmer.stem(word) for word in sentence])
'крут круч крут крут'
```

Вспомогательная задача — определение части речи

```
>>> from nltk import wordnet
>>> def get_wordnet_pos(treebank_tag):
...     my_switch = {'J':wordnet.wordnet.ADJ,
...                  'V':wordnet.wordnet.VERB,
...                  'N':wordnet.wordnet.NOUN,
...                  'R':wordnet.wordnet.ADV}
...     for key, item in my_switch.items():
...         if treebank_tag.startswith(key):
...             return item
...     return wordnet.wordnet.NOUN
...
>>> sentence = 'George admitted the talks happened'.split()
>>> pos_taged = nltk.pos_tag(sentence)
>>> pos_taged
[('George', 'NNP'), ('admitted', 'VBD'), ('the', 'DT'),
 ('talks', 'NNS'), ('happened', 'VBD')]
>>> [get_wordnet_pos(tag) for word, tag in pos_taged]
['n', 'v', 'n', 'n', 'v']
```

Нормализация слов — лемматизация

Лемматизация — приведение слова в начальную форму

Лемматизатор WordNet требует для работы метки частей речи!

```
>>> from nltk import WordNetLemmatizer
>>> def simple_lemmatizer(sentence):
...     simple_lemmatizer =
...     tokenized_sent = sentence.split()
...     pos_taged = [(word, get_wordnet_pos(tag))
...         for word, tag in nltk.pos_tag(tokenized_sent)]
...     return " ".join([lemmatizer.lemmatize(word, tag)
...                          for word, tag in pos_taged])
...
>>> simple_lemmatizer('George admitted the talks happened')
'George admit the talk happen'
>>> simple_lemmatizer('write wrote written')
'write write write'
```

Лемматизация для русского языка

Лемматизаторы для русского языка: pymorphy2, mystem3.

```
>>> import pymorphy2
>>> def simple_lemmatizer(sentence):
...     lemmatizer = pymorphy2.MorphAnalyzer()
...     tokenized_sent = sentence.split()
...     return " ".join([lemmatizer.parse(word)[0].normal_form
...                              for word in tokenized_sent])
...
>>> simple_lemmatizer('опрошенных считают налоги необходимы')
'опросить считать налог необходимый'
>>> simple_lemmatizer('поле пол полёт полка полк')
'пол половина полёт полка полк'
```

Pymorphy2 не требует метку части речи, но для улучшения разбора, её можно использовать.

Заключение о нормализации и отборе слов

Отбор слов

- Нужен почти всегда для всех языков
- Можно сократить словарь до 100 000 токенов почти без потери качества

Стемминг

- Плохо работает для русского языка
- Нормально работает для английского, но модели хорошо работают и без него

Лемматизация

- Лучше стемминга для русского языка
- Сильно повышает качество моделей для русского языка
- Хорошо работает и для английского, но модели хорошо работают и без неё
- Гораздо медленнее чем стемминг

Последовательности слов

Рассмотрим разные сущности на примере предложения:
Метод опорных векторов — метод машинного обучения

- **Коллокации** — устойчивые словосочетания
  метод опорных векторов, метод машинного обучения, опорных векторов, машинного обучения
- **n-граммы** — последовательности из n слов
  2-граммы: *метод опорных, опорных векторов, векторов метод, метод машинного, машинного обучения*
- **s-скип-n-граммы** — последовательности из n слов с s пропусками
  1-скип-2-граммы: *метод векторов, опорных метод, векторов машинного, метод обучения*

Выделение n-грамм

В scikit-learn есть встроенное выделение n-грамм

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> s = ['my name is',
...      'your name are',
...      'my father is']
>>> vectorizer = CountVectorizer(ngram_range=(1, 1))
>>> vectorizer.fit_transform(s).toarray()
array([[0, 0, 1, 1, 1, 0],
       [1, 0, 0, 0, 1, 1],
       [0, 1, 1, 1, 0, 0]], dtype=int64)
>>> vectorizer.get_feature_names()
['are', 'father', 'is', 'my', 'name', 'your']
>>> vectorizer = CountVectorizer(ngram_range=(1, 2))
>>> vectorizer.fit_transform(s).toarray()
array([[0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0],
       [1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1],
       [0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]], dtype=int64)
```

Как можно получать коллокации

- Извлечение биграмм на основе частот и морфологических шаблонов.
- Поиск разрывных коллокаций.
- Извлечение биграмм на основе мер ассоциации и статистических критериев. (TopMine)
- Алгоритм TextRank для извлечения словосочетаний.
- Rapid Automatic Keyword Extraction.
- Выделение ключевых слов по tf-idf.

Выделение частотных коллокаций

Статистические алгоритмы выделения коллокаций основаны на том, что коллокацией являются слова, часто встречающиеся рядом друг с другом

Require: D, t, $W' \subset W$
Ensure: $\{(w,u)\}$ — множество коллокаций из 2 слов;
1: $n_{wu} := 0$ {для всех $w, u \in W'$}
2: **for** $i = 1, \ldots, N$ **do**
3: **for** $j = 1, \ldots, n_d - 1$ **do**
4: $n_{wu} = n_{wu} + \mathbb{I}[w^d_{j+1} = w, w^d_j = u]$ {для всех $w, u \in W'$}
5: $s = \{(w,u) \mid n_{wu} > t\}$
6: **return** s

Без выделения стоп-слова подобные алгоритмы будут работать очень плохо!

Стандартные этапы обработки текстов

Можно выделить следующие этапы:

1. Удаление специфичных символов/последовательностей
2. Приведение к нижнему регистру
3. Токенизация
4. Лемматизация
5. Удаление стоп-слов / Сокращение словаря

Почти всегда правильная обработка приводит к улучшению качества и уменьшению времени работы.