Математические методы анализа текстов

Семинар 1

Правила курса. Получение данных. Предобработка текста. Извлечение коллокаций

Мурат Апишев (great-mel@yandex.ru) [1]

МГУ им. М. В. Ломоносова

9 февраля, 2018

[1] Подготовлена с использованием материалов курса «Анализ неструктурированных данных» ФКН ВШЭ

Правила курса

- **Почта курса:** `nlp.msu@gmail.com`
  - Все вопросы адресовать туда.
  - В личку писать не надо.
- **Оценка за курс:** задания (70%) + экзамен (30%)
- **Задания:**
  - 4 лабораторных работы в Jupyter Notebook
  - Разбор научной статьи (выступление и/или реферат)
  - Возможно, будет конкурс на kaggle
  - Окончательные стоимости разбалловки будут позже
- **Куда сдавать:**
  - Присылать на почту с темой «Фамилия Имя - Лабораторная 1»
  - Дедлайн строгий, задание, после него работа не засчитывается

Требования к оформлению заданий

- Любое задание выполняется самостоятельно, плагиат будет жёстко наказываться
- Отчёт — не менее важная часть работы, чем написание кода и организация экспериментов
- Это должен быть формальный и информативный текст, а не школьное сочинение
- Содержание отчёта:
  1. Постановка задачи
  2. Методы решения
  3. Использованные инструменты
  4. Описания экспериментов
  5. Полученные результаты и выводы

О чём будут семинары

- Основная часть теории будет даваться на лекциях
- Семинары будут иметь практическую направленность
- Затронем следующие темы:
  1. Получение и предобработка текстов
  2. Машинный перевод
  3. Классификация, анализ тональности
  4. Синтаксические парсеры
  5. Векторные представления слов и текстов
  6. Тематическое моделирование
  7. Информационный поиск
  8. Генерация текстов
  9. Вопросно-ответные системы

Краулинг и парсинг данных

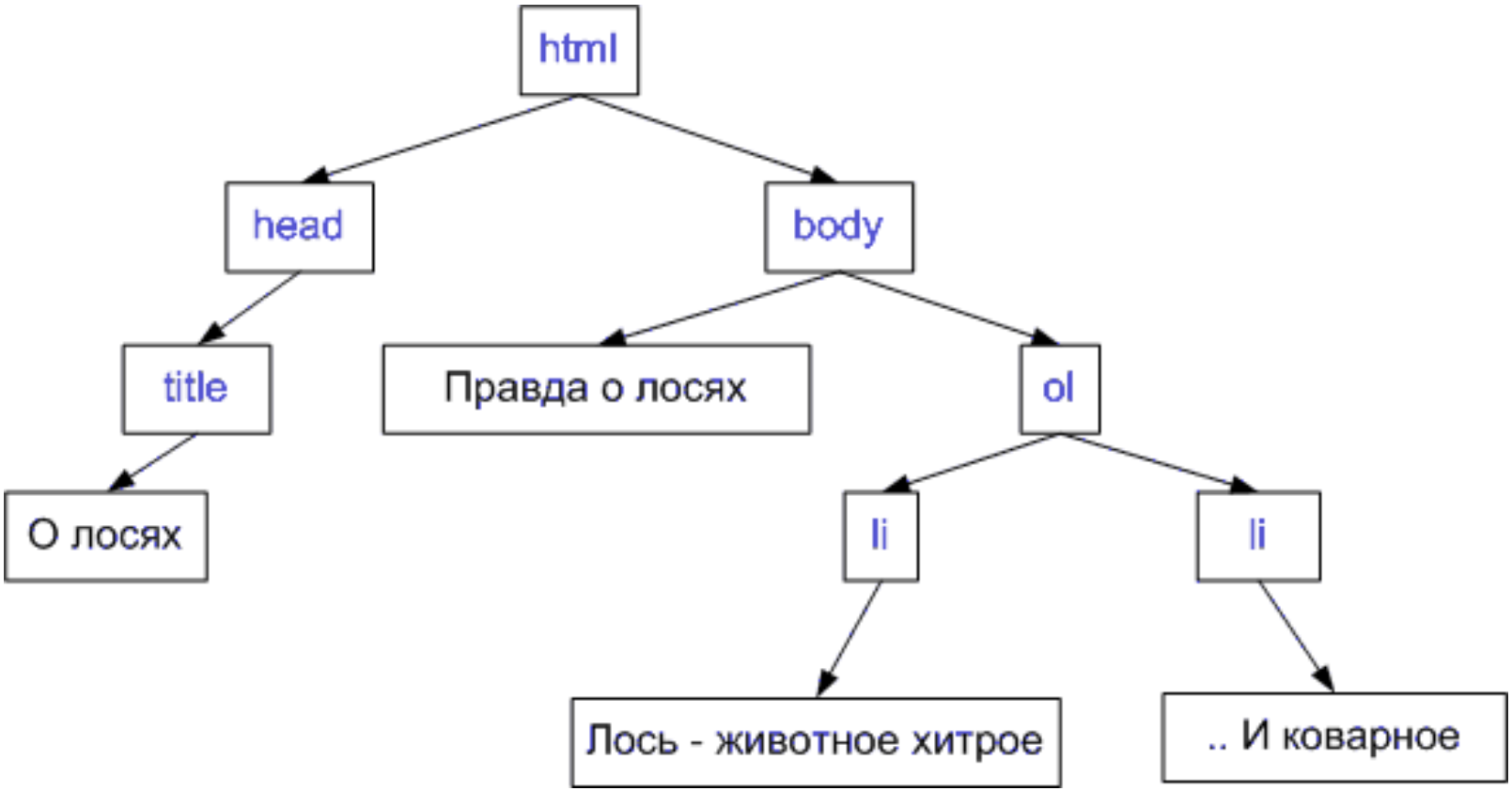
- Одним из основных источников данных является Интернет.
- Нужно уметь выкачивать данные и парсить их в удобный для обработки формат.
- DOM (Document Object Model) — объектная модель, используемая для XML/HTML-документов.
- Согласно DOM-модели, документ является иерархией.
- Каждый HTML-тег образует отдельный элемент-узел, каждый фрагмент текста — текстовый элемент, и т.п.

Источник примера: http://javascript.ru/tutorial/dom/intro

Document Object Model

```
<html>
    <head>
        <title>
            О лосях
        </title>
    </head>
    <body>
        Правда о лосях.
        <ol>
            <li>Лось - животное хитрое</li>
            <li>... И коварное</li>
        </ol>
    </body>
</html>
```

Document Object Model



Самый внешний тег — <html>, поэтому дерево начинает расти от него.

Парсинг HTML/XML: Beautiful Soup

- Всегда можно написать парсер руками ☺
  Но можно сэкономить силы и время
- Beautiful Soup — библиотека для обработки XML/HTML на Python
- Может преобразовать даже неправильную разметку в дерево синтаксического разбора
- Работает достаточно медленно
- Ссылка на документацию:
  http://wiki.python.su/Документации/BeautifulSoup

Парсинг HTML/XML: lxml

- Тоже библиотека для обработки XML/HTML на Python
- Также строит дерево синтаксического разбора, но не является устойчивой к неверной разметке
- Работает быстро
- Ссылка на документацию: http://lxml.de/index.html
- BeautifulSoup стал использовать lxml в качестве внутреннего парсера для ускорения, а в lxml был добавлен модуль soupparser

Коротко о селекторах

CSS — используется как средство описания, оформления внешнего вида веб-страниц, написанных с помощью языков разметки HTML и XHTML.

```
селектор, селектор {
 свойство: значение;
 свойство: значение;
}
```

Селектор определяет элементы, к которым должны быть применены правила.

Пример:

```
.my_table { color: red; }
```

Все элементы класса `my_table` получат значение `red` атрибута `color`.

Коротко о селекторах

Хорошая статья о CSS-селекторах: http://everstudent.ru/blog/htmlcss/30-css-seletors-to-memorize/

Занимательная обучающая игра: http://flukeout.github.io/

XPath-селектор — выбор элемента путём наложения ограничений на его путь в XML-дереве.

Подробнее можно прочесть на Википедии

Scrapy

- Scrapy — асинхронный фреймворк для сбора данных
- Создаёт «веб-паука»
- Использует XPath или CSS селекторы
- Экспор в json, xml, css
- Установка: http://www.w3ii.com/scrapy/scrapy_environment.html

Задача: создать «веб-паука», который обработает страницы Brickset и извлечёт данные о наборах LEGO

Источник примера: https://www.8host.com/blog/web-scraping-s-pomoshhyu-scrapy-i-python-3/

Создаём паука

Передаём ему на вход первую страницу результатов поиска наборов LEGO

```
1 import scrapy
2 class BrickSetSpider(scrapy.Spider):
3 name = "brickset_spider"
4 start_urls = ['http://brickset.com/sets/year-2016']
```

1. Всегда нужно смотреть на скачиваемые данные
2. В любом браузере можно посмотреть исходный код страницы, чтобы понять её структуру

Набор из списка на странице

10252: Volkswagen Beetle

10252-1 | ADVANCED MODELS | VEHICLES | 2016

14 WIDE CAR | CREATOR EXPERT | D2C | LARGE SCALE VEHICLE

VOLKSWAGEN

★★★★★ 1 REVIEW

PIECES 1167

NOTES Labelled Creator - Expert.

RRP $99.99, 89.99€

PPP 8.6c, 7.7c

PACKAGING Box

AVAILABILITY LEGO exclusive

FIRST SOLD

USA: Aug 16, UK/EU: Aug 16

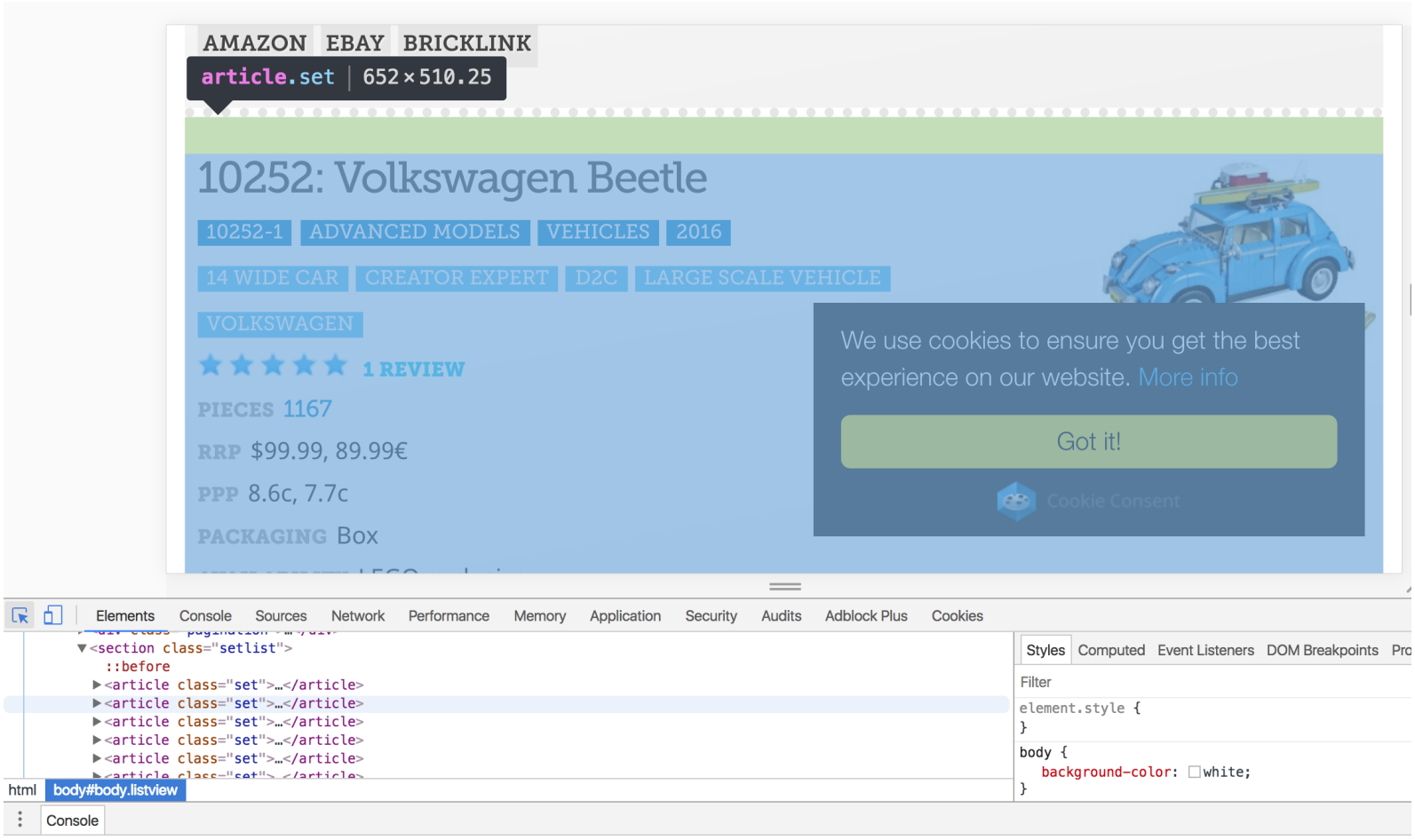
INSTRUCTIONS Yes

ADDITIONAL IMAGES 20

SET TYPE Normal

Выбираем селекторы

Каждый набор на странице имеет класс «set»



Используем CSS-селекторы

Извлечём все наборы со страницы:

```
1 class BrickSetSpider(scrapy.Spider):
2 name = 'brickset_spider'
3 start_urls = ['http://brickset.com/sets/year-2016']
4 def parse(self, response):
5   SET_SELECTOR = '.set'
6   for brickset in response.css(SET_SELECTOR):
7     pass
```

Название наборов хранится в тегах a внутри тега h1:

```
1 brickset.com/sets/year-2016
2 <h1><a href='/sets/10251-1/Brick-Bank'>Brick Bank</a><
    /h1>
```

Ищем и отображаем имя набора

```
1  class BrickSetSpider(scrapy.Spider):
2  name = 'brickset_spider'
3  start_urls = ['http://brickset.com/sets/year-2016']
4  def parse(self, response):
5    SET_SELECTOR = '.set'
6    for brickset in response.css(SET_SELECTOR):
7      NAME_SELECTOR = 'h1 a ::text'
8      yield {
9        'name': brickset.css(NAME_SELECTOR)
10                    .extract_first(),
11     }
```

`::text` — псевдоселектор CSS, который будет извлекать название набора из тега a.

Переход на другие страницы

Каждая страница начинается и заканчивается символом >, который ссылается на следующую страницу результата:

```
<li class="next">
<a href="http://brickset.com/sets/year-2017/page-2">
    &#8250;</a>
</li>
```

```
  NEXT_PAGE_SELECTOR = '.next a ::attr(href)'
  next_page = response.css(NEXT_PAGE_SELECTOR).
    extract_first()
  if next_page:
    yield scrapy.Request(
        response.urljoin(next_page),
        callback=self.parse
    )
```

Предобработка текста

- Итак, данные получены в подходящем формате (например, plain text)
- Следующий шаг: предобработка
- Базовые шаги предобработки:
  1. токенизация
  2. приведение к нижнему регистру
  3. удаление стоп-слов
  4. удаление пунктуации
  5. фильтрация по частоте/длине/соответствию регулярному выражению
  6. лемматизация или стемминг
- Чаще всего применяются все эти шаги, но в разных задачах какие-то могут опускаться, поскольку приводят к потере информации.

Полезные модули

1. `nltk` — один из основных модулей Python для анализа текстов, содержит множество инструментов.
2. `re/regex` — модули для работы с регулярными выражениями
3. `pymorphy2/pymystem3` — лемматизаторы
4. Специализированные модули для обучения моделей (например, CRF)
5. `numpy/pandas/scipy/sklearn` — модули общего назначения
6. `codecs` — полезный модуль для работы с кодировками при использовании Python 2.*

Токенизация, удаление стоп-слов и пунктуации

В `nltk` есть разные токенизаторы:

- RegexpTokenizer
- BlanklineTokenizer
- И ещё около десятка штук

Стоп-слова тоже можно удалять с помощью `nltk` (но лучше дополнительно фильтровать вручную):

```
1 from nltk.corpus import stopwords
2 words = [word for word in word_list
3           if word not in stopwords.words('russian')]
```

Пунктуацию можно удалять с помощью регулярных выражений, а можно просто:

```
1 from string import punctuation
2 s = ''.join(c for c in s if c not in punctuation)
```

Стэмминг и лемматизация

Стэмминг — процесс приведения слова к основе (отрезание окончания и формообразующего суффикса), грубо, но быстро.

- Porter stemmer
- Snowball stemmer
- Lancaster stemmer

Лемматизация — процесс приведения слова к нормальной форме, качественно, но долго.

- pymorphy2 (язык русский, украинский)
- mystem3 (язык русский, английский?)
- Wordnet Lemmatizer (NLTK, язык английский, требует POS метку)
- Metaphraz (язык русский)
- Coda/Cadenza (языки русский и английский)

Пример лемматизации

```
1 import pymorphy2
2
3 text_ru = u'Где твоя ложка, папа?'
4 pymorph = pymorphy2.MorphAnalyzer()
5
6 for word in text_ru.split(u' '):
7     if not re.match(u'([^a-za-яё]+)',word):
8         word = pymorph.parse(word)[0].normal_form
9     print word,
```

Вывод:

```
где твой ложка , папа ?
```

Коллокации

N-граммы — усточивые последовательности из N слов, идущих подряд («машина опорных векторов»)

Коллокация — устойчивое сочетание слов, не обязательно идущих подряд («Он сломал своему противнику руку»)

Примеры коллокаций:

1. *Соединённые Штаты Америки, Европейский Союз*
2. *Машина опорных векторов, испытание Бернулли*
3. *Крепкий чай, крутой кипяток, свободная пресса*

Часто коллокациями бывают именованные сущности (но далеко не всегда).

Как можно получать коллокации

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

Экспериментальные данные

- Датасет представляет собой статьи о 28 резонансных событиях 2017 года.
- Каждое событие представлено 100 сырыми текстами.
- Примеры событий:
  - *Власти Петербурга согласились передать РПЦ Исаакиевский собор*.
  - *Дональд Трамп вступил в должность президента США*.
  - *Умер Дэвид Рокфеллер*.
- Ссылка на данные.
- Начнём с поиска биграмм в теме «Дональд Трамп вступил в должность президента США»
- Ссылка на исходный код (частично будет ниже).

Частотные униграммы без стоп-слов

```
1  import nltk
2  import re
3  import pymorphy2
4  from nltk.corpus import stopwords
5
6  prog = re.compile('[А-Яа-я]+')
7  t1 = prog.findall(s.lower())
8
9  morph = pymorphy2.MorphAnalyzer()
10
11 t2 = [morph.parse(token)[0].normal_form
12         for tok in t1
13             if not tok in stopwords.words('russian')]
14 t3 = nltk.FreqDist(t2)
15 t3.most_common(20)
```

Результат

('трамп', 595)
('президент', 491)
('год', 441)
('который', 428)
('инаугурация', 358)
('сша', 352)
('дональд', 316)
('один', 284)
('россия', 251)
('наш', 223)
('январь', 212)
('это', 198)
('российский', 192)
('время', 184)
('свой', 179)
('быть', 179)
('страна', 173)
('стать', 161)
('человек', 140)
('день', 133)

Частотные биграммы

- Без лемматизации и удаления стоп-слов
- Без POS-тегов, мер ассоциации и т.п.

```
1 bg = list(nltk.bigrams(prog.findall(s.lower())))
2 bgfd = nltk.FreqDist(bg)
3 bgfd.most_common(18)
```

Результат:

(('дональд', 'трамп'), 165)
(('дональда', 'трампа'), 133)
(('президента', 'сша'), 125)
(('в', 'году'), 87)
(('в', 'россии'), 68)
(('избранного', 'президента'), 59)
(('го', 'президента'), 55)
(('инаугурация', 'трампа'), 55)
(('москва', 'января'), 51)
(('по', 'делу'), 50)
(('в', 'должность'), 47)
(('об', 'этом'), 46)
(('в', 'вашингтоне'), 45)
(('из', 'за'), 45)
(('в', 'отношении'), 45)
(('президент', 'сша'), 44)
(('и', 'в'), 40)
(('млрд', 'руб'), 40)

Частотные биграммы

- Без лемматизации и удаления стоп-слов
- С морфологическим шаблоном (Томита) – ещё будет!

```
S -> Adj<gnc-agr[1]> Noun<gnc-agr[1], rt>;
```

```
1 # s - list with colloaction find by Tomita
2 d1 = nltk.FreqDist(s)
3 d1.most_common(16)
```

Результат:

('избранный президент', 87)
('-ый президент', 70)
('белый дом', 69)
('прямая трансляция', 43)
('наша страна', 31)
('этот год', 31)
('соединенный штат', 28)
('новый президент', 27)
('конституционный суд', 25)
('прошлый год', 23)
('весь мир', 21)
('предвыборная кампания', 21)
('ближайшее время', 21)
('новая администрация', 20)
('опасное вождение', 20)
('демократическая партия', 19)

Поиск разрывных коллокаций

- Часто устойчивые словосочетания находятся не рядом.
- **Примеры:**
  - She *knocked* on his *door*.
  - They *knocked* on his heavy *door*.
  - A man *knocked* on the metal front *door*.
- **Что делаем:**
  - Рассмотрим все пары слов в некотором окне.
  - Посчитаем расстояние между словами.
- **Что меряем:**
  - **Матожидание** – показывает, насколько часто слова встречаются вместе.
  - **Дисперсия** – вариабельность позиции.
- **Важно провести лемматизацию.**
- Презентация по теме: http://tpc.at.ispras.ru/wp-content/uploads/2011/10/lecture4-2016.pdf

Пример

s	$\bar{d}$	Count	Word 1	Word 2
0.43	0.97	11657	New	York
0.48	1.83	24	previous	games
0.15	2.98	46	minus	points
0.49	3.87	131	hundreds	dollars
4.03	0.44	36	editorial	Atlanta
4.03	0.00	78	ring	New
3.96	0.19	119	point	hundredth
3.96	0.29	106	subscribers	by
1.07	1.45	80	strong	support
1.13	2.57	7	powerful	organizations
1.01	2.00	112	Richard	Nixon
1.05	0.00	10	Garrison	said

Большое значение дисперсии говорит о том, что словосочетание не слишком интересно.

$$\overline{d} = \frac{\sum_{i=1}^{n} d_i}{n}$$

$$s^2 = \frac{\sum_{i=1}^{n} (d_i - \overline{d})^2}{n-1}$$

- n – число раз, когда два слова встретились.
- d_i – смещение между словами (может быть < 0).
- $\overline{d}$ – выборочное среднее смещений.

Источник примера

Меры ассоциации биграмм – PMI

PMI (Pointwise Mutual Information):

$$\mathrm{PMI}(w_1, w_2) = \log \frac{f(w_1, w_2)}{f(w_1) f(w_2)}$$

где w_i – слово, $f(\cdot)$ – частота слова или биграммы.

- Оценивает независимость совместного появления пары слов.
- Значения величины зависят от размеров корпуса.
- Завышает значимость редких словосочетаний. **Решение:** порог по частоте.
- Выделяет терминологические словосочетания.

Меры ассоциации биграмм – T-Score

$$\text{T-Score}(w_1, w_2) = \frac{f(w_1, w_2) - f(w_1)f(w_2)}{\sqrt{f(w_1, w_2)/N}}$$

где N – общее количество биграмм.

- Является модифицированным ранжированием по частоте.
- Не преувеличивает значимость редких коллокаций ($\Rightarrow$ нет необходимости в пороге).
- Выделяет общеязыковые устойчивые сочетания.
- По-сути – статистический тест Стьюдента, проверяется гипотеза независимой встречаемости двух слов.

Меры ассоциации биграмм – T-Score

t	$C(w^1)$	$C(w^2)$	$C(w^1\ w^2)$	w^1	w^2
4.4721	42	20	20	Ayatollah	Ruhollah
4.4721	41	27	20	Bette	Midler
4.4720	30	117	20	Agatha	Christie
4.4720	77	59	20	videocassette	recorder
4.4720	24	320	20	unsalted	butter
2.3714	14907	9017	20	first	made
2.2446	13484	10570	20	over	many
1.3685	14734	13478	20	into	them
1.2176	14093	14776	20	like	people
0.8036	15019	15629	20	time	last

Источник примера

Меры ассоциации биграмм – χ^2, LLR

χ^2 и *LLR (Log-Likelyhood Ratio)*:

- Так же представляют собой статистические тесты.
- χ^2 сравнивает наблюдаемые частоты в корпусе с ожидаемыми при верной гипотезе о независимости, большое различие приводит к опровержению гипотезы.
- LLR: насколько более правдоподобна одна гипотеза, чем другая:
- χ^2 требует большую выборку наблюдений.
- Здесь можно найти подробные формулы и описания.

Пример использования

Все описанные меры реализованы в `nltk.collocations`:

- `bigram_measures.pmi`
- `bigram_measures.student_t`
- `bigram_measures.chi_sq`
- `bigram_measures.likelihood_ratio`

Код, данные и результаты экспериментов доступны здесь.

Предобработка текстов:

1. Объединим все 2800 текстов в один.
2. Приведём всё к нижнему регистру.
3. Лемматизируем.
4. Удалим стоп-слова.

Пример

```
1  # m - linear list of tokens
2
3  from nltk.collocations import *
4  N_best = 100  # number of bigrams to extract
5
6  # class for association measures
7  bm = nltk.collocations.BigramAssocMeasures()
8
9  # class for bigrams extraction and storing
10 f = BigramCollocationFinder.from_words(m)
11
12 # remove too seldom bigrams
13 f.apply_freq_filter(5)
```

Пример

```
# get top-100 bigrams using simple frequency
raw_freq_ranking = [' '.join(i) for i in
                    f.nbest(bm.raw_freq, N_best)]

# get top-100 bigrams using described measures
tscore_ranking = [' '.join(i) for i in
                  f.nbest(bm.student_t, N_best)]

pmi_ranking =  [' '.join(i) for i in
                f.nbest(bm.pmi, N_best)]

llr_ranking = [' '. join(i) for i in
               f.nbest(bm.likelihood_ratio, N_best)]

chi2_ranking =  [' '.join(i) for i in
                 f.nbest(bm.chi_sq, N_best)]
```

Результаты

	raw_freq	pmi	t-score	chi2	llr
0	владимир путин	азотосодержимый добавка	владимир путин	азотосодержимый добавка	владимир путин
1	прямая линия	аугусто пиночет	прямая линия	алый парус	прямая линия
2	дональд трамп	бактериальный инфекция	дональд трамп	алёсс капут	риа новость
3	курортный сбор	визитный карточка	курортный сбор	аттестат зрелость	дональд трамп
4	риа новость	висок ром	риа новость	аугусто пиночет	курортный сбор
5	премьер министр	воротничок автоматизация	премьер министр	бактериальный инфекция	исаакиевский собор
6	исаакиевский собор	греф предречь	исаакиевский собор	визитный карточка	премьер министр
7	евгений евтушенко	дилер роад	евгений евтушенко	висок ром	санкт петербург
8	алексей навальный	добавление рацион	алексей навальный	воротничок автоматизация	евгений евтушенко
9	президент сша	доза азотосодержимый	президент сша	генри киссинджер	виталий чуркин
10	чемпионат мир	консультант кредитовать	чемпионат мир	греф предречь	денис вороненков

TextRank

TextRank – приложение алгоритма PageRank к задачам NLP:

- Строим граф на основе исходного текста
  - V – вершины (слова)
  - E – рёбра (связи)
- Вычисляем веса вершин по мерам центральности, например, по *PageRank*:

$$\mathrm{PR}(V_i) = (1 - d) + d \sum_{V_j \in In(V_i)} \frac{PR(V_j)}{Out(V_j)}$$

- Извлекаем цепочки с наибольшими весами.

Источник: http://koost.eveel.ru/science/CSEDays2012.pdf

Описание TextRank

- В качестве V можно взять все уникальные леммы текста (допустимо ограничиться прилагательными и существительными: термины в основном являются именными группами).
- Сканируем текст с окном из $N \in [2, 10]$ слов.
- На каждой итерации считаем для пары слов величину связи

$$WC(w_1, w_2) = \begin{cases} 1 - \dfrac{d(w_1, w_2) - 1}{N - 1}, & \text{if } d(w_1, w_2) \in (0, N), \\ 0, & \text{if } d(w_1, w_2) \geqslant N, \end{cases}$$

где w_i – слова, $d(w_1, w_2)$ – расстояние между ними (можно просто взять модуль разности позиций).

- Основание подобной связи – между двумя рядом стоящими словами часто суще- ствует семантическое отношение.

Описание TextRank

- Ранжируем вершины графа на основании значения TextRank, получаемого случайным блужданием для каждой вершины $t \in V$:

$$\mathrm{TR}(t_i) = (1-d) + d \sum_{t_j \in In(t_i)} \frac{w_{ji}}{\sum_{t_k \in Out(t_j)} w_j k} \mathrm{TR}(t_j)$$

  где d – фактор затухания, $In(t)$ – вершины, входящие в t, $Out(t)$ – выходящие из t, w_{ij} – вес соответствующего ребра.

- Упорядочиваем вершины по TR и отбираем T самых лучших (например, $T = 1/3|V|$). Это множество кандидатов C.

- Извлекаем из текста все последовательности слов, состоящие из элементов множества C. **Важно:**

  1. в последовательности должно быть хоть одно существительное;
  2. в случае вложенности надо рассматривать только последовательность с бльшим весом

TextRank: пример

Текст:

Статья посвящена вопросу извлечения терминов из текстов на русском языке при помощи графовых моделей экспериментально исследован алгоритм решения даннои задачи. Сформулированы требования и рекомендации к применению алгоритма в задачах обработки русского языка.

Граф:

TextRank: пример

Множество кандидатов:

- задача – 0,094
- русский – 0,084
- алгоритм – 0,083
- язык – 0,076
- извлечение – 0,064
- обработка – 0,063
- термин – 0,062
- вопрос – 0,060

Выделенные словосочетания:

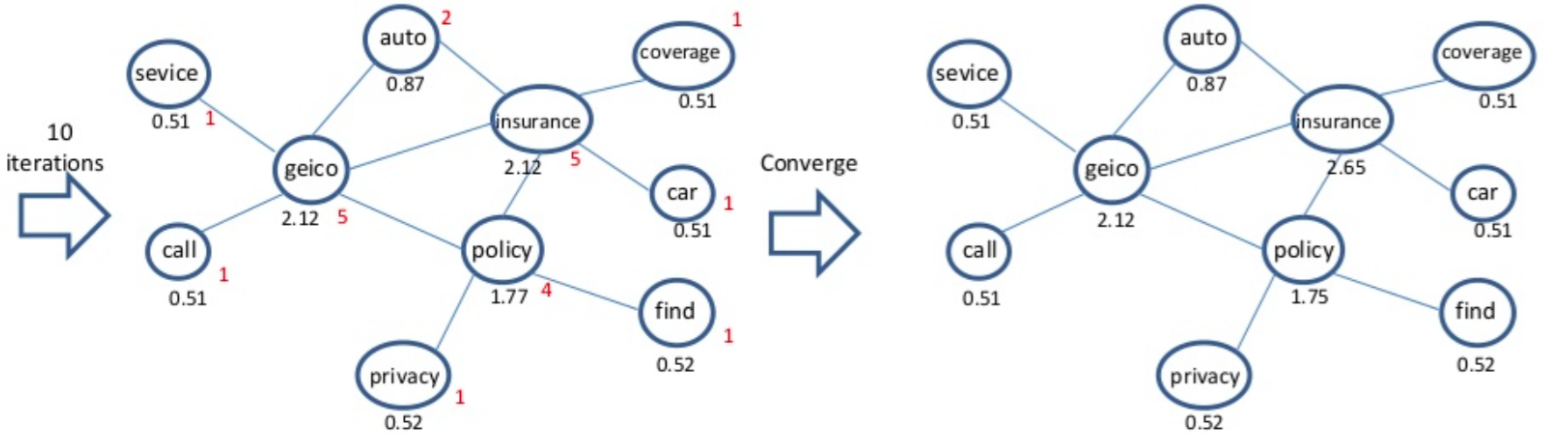
Статья посвящена ***вопросу извлечения терминов*** из текстов на русском языке при помощи графовых моделей экспериментально исследован **алгоритм** решения данной **задачи**. Сформулированы требования и рекомендации к применению ***алгоритма*** в ***задачах обработки русского языка***.

TextRank: ещё пример

Ссылка на источник

$$S(V_i) = (1-d) + d * \sum_{j \in ngbr(V_i)} \frac{1}{|degree(V_j)|} S(V_j)$$

d is the damping factor that usually set to 0.85



$$Score(geico) = 0.15 + 0.85 * \left(\underbrace{\frac{1}{1} * 0.51}_{\text{service}} + \underbrace{\frac{1}{1} * 0.51}_{\text{call}} + \underbrace{\frac{1}{2} * 0.87}_{\text{auto}} + \underbrace{\frac{1}{5} * 2.12}_{\text{insurance}} + \underbrace{\frac{1}{4} * 1.77}_{\text{policy}}\right) = 2.12$$

$$Score(policy) = 0.15 + 0.85 * \left(\underbrace{\frac{1}{1} * 0.52}_{\text{find}} + \underbrace{\frac{1}{1} * 0.52}_{\text{privacy}} + \underbrace{\frac{1}{5} * 2.12}_{\text{insurance}} + \underbrace{\frac{1}{5} * 2.12}_{\text{geico}}\right) = 1.75$$

$$Score(service) = 0.15 + 0.85 * \left(\underbrace{\frac{1}{5} * 2.12}_{\text{geico}}\right) = 0.51$$

Converge Really Quick!
(<= 20 iterations)

TextRank в Gensim

Полный код и данные доступны тут.

```
1 # text is a string with lemmatized tokens
2 from gensim.summarization import keywords
3 kw = keywords(text)
```

Результат (примеры):

- дуров
- telegram пока
- роскомнадзор
- дать компания реестр
- заявление
- канал мессенджер
- возражать против
- блокировка
- россия создатель

RAKE

RAKE (Rapid Automatic Keyword Extraction):

- Фразы-кандидаты – все слова между разделителями.
- Некоторым образом производится оценка фразы.
- Фраза-кандидат ограничивается по частоте и количеству слов.
- Модули: `rake_nltk`, `Rake`, `rake`.
- Ссылка на данные и код.

Пример:

```
1 from rake_nltk import Rake
2 r = Rake(stopwords.words('russian') + ['это', 'вне'])
3 r.extract_keywords_from_text(tokenized_text)
4 r.get_ranked_phrases()
```

Выделение ключевых слов по tf-idf

- **Идея:** хотим выделить слова, которые часто встречаются в данном тексте, и редко – в других текстах.

$$v_{wd} = tf_{wd} \times \log \frac{N}{df_w}$$

где tf_{wd} – число раз, которое слово w встретилось в документе d, df_w – число документов, содержащих w, N – общее число документов.

- Такие слова, как правило, информативны, и значение tf-idf является хорошим признаком.
- Значения tf-idf для слов текста можно получить с помощью `sklearn.feature_extraction.text.TfidfVectorizer`
- Ссылка на данные и код примера.

Выводы (вместо краткого пересказа слайдов)

- Анализ текстов — важная прикладная область машинного обучения.
- Хороший аналитик должен уметь не только применять методы анализа, но и обладать навыками выкачивания данных, их предобработки и выделения признаков[2].
- Теоретические знания можно получить на лекциях, практические — на семинарах и при выполнении заданий.

Успехов!

[2] В идеале надо ещё уметь закодить своё решение на промышленном языке, но это за рамками курса ☺