Анализ текстов

Лекция

Предобработка и выделение признаков. Классификация текстов. Анализ тональности.

Мурат Апишев (great-mel@yandex.ru)

3 октября, 2018

Содержание занятия

- Предобработка текстов
- Регулярные выражения
- Признаки, выделение коллокаций
- Задача классификации текстов
- Метрики качества
- Отбор моделей
- Блендинг и стекинг
- Библиотека Vowpal Wabbit, hashing trick
- Библиотека FastText

Предобработка текста

- Первый шаг любой аналитики – получение данных. Предположим, что данные есть в некотором подходящем для работы формате.
- Следующая задача – 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):

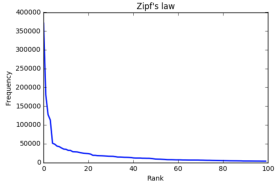
```
1 from nltk.corpus import stopwords
2 words = [word for word in word_list
3           if word not in stopwords.words('russian')]
```

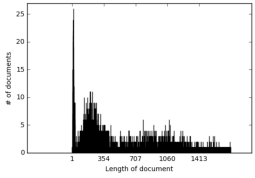
Пунктуацию можно удалять с помощью регулярных выражений, а можно просто:

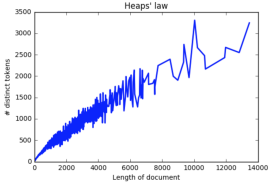
```
1 from string import punctuation
2 s = ''.join(c for c in s if c not in punctuation)
```

Статистические свойства коллекции

- Перед тем, как анализировать коллекцию текстов, необходимо узнать её статистические свойства.
- Каждая задача требует подсчёта специфичных ей величин.
- Есть характеристики, на которые нужно смотреть всегда:
  - Частоты слов в коллекции, закон Ципфа
  - Гистограмма длин документов
  - Закон Хипса для документов







Регулярные выражения

- Регулярные выражения появились от т.н. регулярных автоматов (классификация грамматик по Хомскому).
- По факту это некоторый строковый шаблон, на соответствие которому можно проверить текст.
- Для работы с регулярными выражениями есть множество инструментов и онлайн-сервисов, в Python есть два основных модуля: re и `regex`.
- С синтаксисом можно ознакомиться на странице выбранного инструмента, но основные правила одинаковы, например:
  - `.` – означает наличие одного любого символа
  - `[a-zA-Z0-9]` – означает множество символов из заданного диапазона
  - `+`, `*` – показывают, что следующий перед ними символ или последовательсть символов должны повториться $\geqslant 1$ раз (`r+`) или > 0 раз (`(xa-)*`)

Пример из жизни

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( .*|:*|:.*|;.*|)
```

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 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

```
import nltk
import re
import pymorphy2
from nltk.corpus import stopwords

prog = re.compile('[А-Яа-я]+')
t1 = prog.findall(s.lower())

morph = pymorphy2.MorphAnalyzer()

t2 = [morph.parse(token)[0].normal_form
        for tok in t1
            if not tok in stopwords.words('russian')]
t3 = nltk.FreqDist(t2)
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 (например, Томита)

```
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

```
# m - linear list of tokens

from nltk.collocations import *
N_best = 100  # number of bigrams to extract

# class for association measures
bm = nltk.collocations.BigramAssocMeasures()

# class for bigrams extraction and storing
f = BigramCollocationFinder.from_words(m)

# remove too seldom bigrams
f.apply_freq_filter(5)
```

Пример

```
1  # get top-100 bigrams using simple frequency
2  raw_freq_ranking = [' '.join(i) for i in
3                      f.nbest(bm.raw_freq, N_best)]
4
5  # get top-100 bigrams using described measures
6  tscore_ranking = [' '.join(i) for i in
7                    f.nbest(bm.student_t, N_best)]
8
9  pmi_ranking =  [' '.join(i) for i in
10                  f.nbest(bm.pmi, N_best)]
11
12 llr_ranking = [' '. join(i) for i in
13                 f.nbest(bm.likelihood_ratio, N_best)]
14
15 chi2_ranking =  [' '.join(i) for i in
16                   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:**

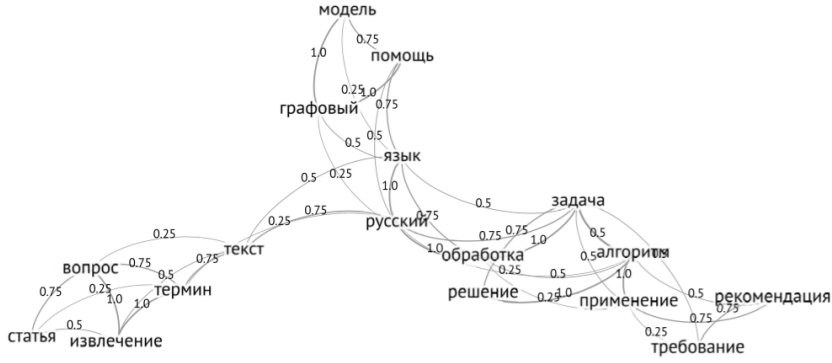
  1. в последовательности должно быть хоть одно существительное;
  2. в случае вложенности надо рассматривать только последовательность с бльшим весом

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:

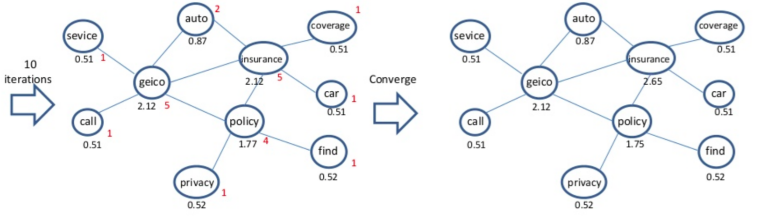
Статья посвящена ***вопросу извлечения терминов*** из текстов на русском языке при помощи графовых моделей экспериментально исследован **алгоритм** решения даннои **задачи**. Сформулированы требования и рекомендации к применению ***алгоритма*** в ***задачах обработки русского языка***.

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



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

Converge Really Quick!
(<= 20 iterations)

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

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

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.

Постановка задачи классификации

Данные:

- $d \in D$ – множество документов (объектов)
- $c \in C$ – множество меток классов

Типы задач:

- Бинарная классификация: $|C| = 2, \forall d \in D \leftrightarrow c \in C$
- Многоклассовая классификация [multiclass]: $|C| = K, K > 2, \forall d \in D \leftrightarrow c \in C$
- Многотемная классификация [multi-label]: $|C| = K, K > 2, \forall d \in D \leftrightarrow \tilde{C} \subseteq C$

Примеры задач

1. Фильтрация спама: $C = \{spam, good\}$
2. Анализ тональности (простой): $C = \{pos, neg, neutral\}$
3. Рубрикация: $C = \{sport, hobby, \dots\}$ – multiclass [+ multi-label]
4. Определение авторства:
   - Этим ли автором написан текст?
   - Каким(-и) из авторов написан текст?
   - Какова возрастная группа автора? Пол автора?

Метрики качества бинарной классификации

Простейшая метрика – *аккуратность* (accuracy): $\mathrm{acc} = \frac{\#(correct)}{\#(total)}$

В жизни почти не используется. **Почему?**

Чаще всего используются *точность* (precision) и *полнота* (recall):

- $precision = Pr = \frac{tp}{tp + fp}$
- $recall = R = \frac{tp}{tp + fn}$
- $F_1 = \frac{2}{\frac{1}{Pr} + \frac{1}{R}} = \frac{2 \cdot Pr \cdot R}{Pr + R}$

		gold standart	
		positive	negative
classification output	positive	tp	fp
	negative	fn	tn

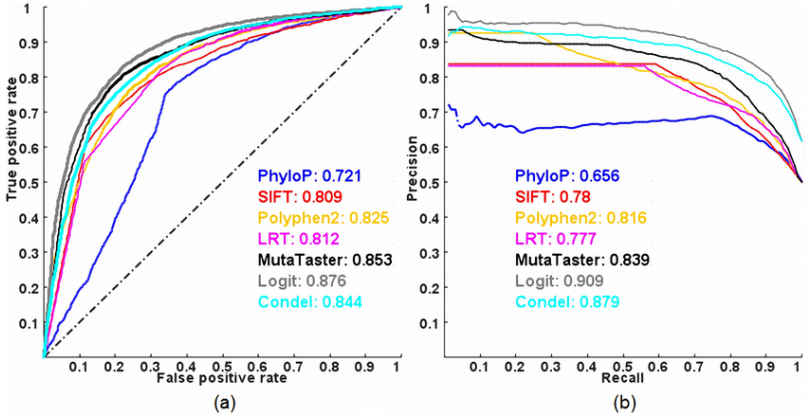
Метрики качества многоклассовой классификации

- Микро-усреднение:
  - $Pr_{micro} = \frac{\sum tp_i}{\sum tp_i + \sum fp_i}$
  - $R_{micro} = \frac{\sum tp_i}{\sum tp_i + \sum fn_i}$
- Макро-усреднение:
  - $Pr_{macro} = \frac{\sum Pr_i}{|C|}$
  - $R_{macro} = \frac{\sum R_i}{|C|}$

		gold standart		
		$class_1$	$class_2$	$class_3$
classification output	$class_1$	tp_1	fp_{12}	fp_{13}
	$class_2$	fn_{21}	tp_2	fp_{23}
	$class_3$	fn_{31}	fn_{32}	tp_3

- Микро-усреднение нивелирует влияние маленьких классов
- Макро-усреднение учитывает все классы в равной степени

Метрики качества: AUC-ROC, AUC-PR



AUC-PR лучше подходит для несбалансированных классов!

Источник картинки

Признаки для текстов

Классификация по правилам:

- если в предложении встречается личное местоимение первого лица и глагол с окончанием женского рода, то пол автора женский
- если доля положительно окрашенных прилагательтельных в отзыве больше доли отрицательно окрашенных прилагательных, то отзыв относится к классу positive

Признаки для применения ML:

- счётчики слов
- TF-IDF слов
- N-граммы (в т.ч. и символьные, + коллокации, термины, именованные сущности и т.п.)

Признаки для текстов

- Текстовые признаки, как правило, очень разреженные
  - в Python существует много типов разреженных матриц с разными свойствами (можно посмотреть здесь)
  - Почти все классификаторы `sklearn` работают с разреженными данными, исключение `GradientBoostingClassifier`
- Обычно используется «Bag-of-words» (но не всегда!)
  - `sklearn.feature_extraction.text.CountVectorizer`
  - `sklearn.feature_extraction.text.TfIdfVectorizer`
- Из разреженных признаков можно получить плотные путём сжатия признакового пространства (SVD, NNMF, PLSA)

Описание данных

- 10 тысяч вопросов со StackOverflow
- Каждый вопрос имеет либо тег **windows**, либо тег **linux**

```
1 import pandas as pd
2 texts = pd.read_csv('windows_vs_linux.10k.tsv',
3                     header=None, sep='\t')
4 texts.columns = ['text', 'is_windows']
5 texts.head(4)
```

	text	is_windows
0	so i find myself porting a game that was orig...	0
1	i ve been using tortoisesvn in a windows envi...	1
2	we are using wmv videos on an internal site a...	1
3	on one linux server running apache and php 5 ...	0

Примеры кода

Собрать «мешок слов» со счётчиками:

```
vectorizer = CountVectorizer(binary=True)
bow = vectorizer.fit_transform(texts.text)
print(type(bow))  # <class 'scipy.sparse.csr.csr_matrix'>
```

Собрать TF-IDF:

```
vectorizer = TfidfVectorizer()
tf_idf = vectorizer.fit_transform(texts.text)
```

Сжать пространство TF-IDF признаков:

```
svd = TruncatedSVD(n_components=200, n_iter=5)
tf_idf_svd = svd.fit_transform(tf_idf)
```

Выбор модели

Кросс-валидация:

- различными стратегиями делим обучающую выборку на N частей
- обучаем модель на $N - 1$ частях, на оставшейся тестируем перебираем все варианты
- параметры выбираем усреднением

```
params = {'C': np.logspace(-5, 5, 11)}

clf = LogisticRegression()
cv = GridSearchCV(clf, params, n_jobs=-1, scoring='roc_auc', cv=5)
cv.fit(bow, texts.is_windows);
```

Выбор модели

```
pd.DataFrame(cv.cv_results_)[['mean_test_score', 'params']]
  .sort_values('mean_test_score', ascending=False)
```

	mean_test_score	params
4	0.965813	{u'C': 0.1}
5	0.962415	{u'C': 1.0}
3	0.961759	{u'C': 0.01}
6	0.955353	{u'C': 10.0}
7	0.948635	{u'C': 100.0}
2	0.945693	{u'C': 0.001}
8	0.945429	{u'C': 1000.0}
9	0.943599	{u'C': 10000.0}
10	0.942903	{u'C': 100000.0}
1	0.790566	{u'C': 0.0001}
0	0.632507	{u'C': 1e-05}

Наиболее значимые слова

```
top = pd.DataFrame([get_top_windows(cv.best_estimator_, 6),
                    get_top_linux(cv.best_estimator_, 6)]).T
top.columns = ['Windows', 'Linux']
top
```

	Windows	**Linux**
0	windows	ubuntu
1	win32	root
2	vista	mono
3	exe	linux
4	dll	kernel
5	batch	bash

Отбор признаков

```
vect = CountVectorizer(binary=True, ngram_range=(1, 4))
bow = vect.fit_transform(texts.text)
print(bow.shape)  # (10000, 2117115)
```

```
k_best = SelectKBest(k=50000)
bow_k_best = k_best.fit_transform(bow, texts.is_windows)
```

```
clf = LogisticRegression()
np.mean(cross_val_score(clf, bow_k_best, texts.is_windows,
                        scoring='roc_auc', cv=5))
```

Получили более высокое качество на кросс-валидации (0.97955)

Есть ли подвох?

Код вычисления метрик (sklearn.metrics)

Аккуратность (доля верных ответов):

```
accuracy_score(predicted, target)
```

Точность, полнота, F_1-score:

```
precision_score(target, prediction)
recall_score(target, prediction)
f1_score(target, prediction)
```

F_1-score для многоклассового случая:

```
f1_score(target, prediction, average = 'micro')
f1_score(target, prediction, average = 'macro')
```

AUC-ROC, AUC-PR:

```
roc_auc_score(target, prediction)
average_precision_score(target, prediction)
```

Multiclass и multilabel классификация

```
1 texts = pd.read_csv('multi_tag.10k.tsv', header=None, sep='\t')
2 texts.columns = ['text', 'tags']
3 print(texts.shape)
4 texts.head(4)
```

	text	tags
0	i want to use a track bar to change a form s ...	c# winforms type-conversion decimal opacity
1	i have an absolutely positioned div containin...	html css css3 internet-explorer-7
2	given a datetime representing a person s birt...	c# .net datetime
3	given a specific datetime value how do i disp...	c# datetime datediff relative-time-span

Многоклассовая классификация

По сути – обобщение бинарной классификации.

Часть классификаторов `sklearn` умеет работать с многоклассовым случаем:

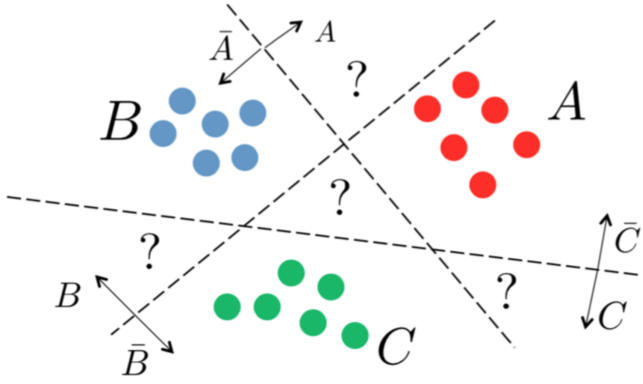
- `KNeighborsClassifier`
- `RandomForestClassifier`
- `SVC`

Ддя остальных есть адапторы:

- `OneVsRestClassifier`
- `OneVsOneClassifier`

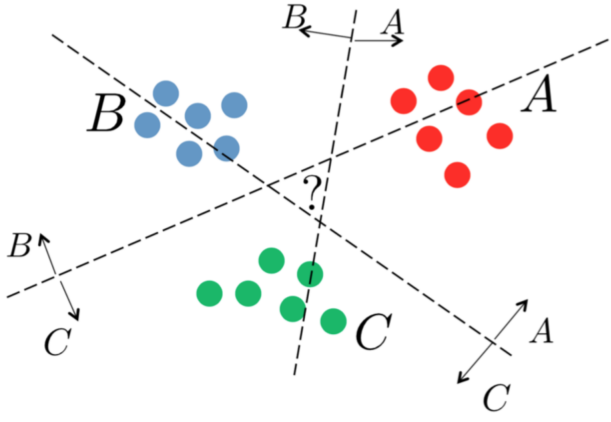
Многоклассовая классификация

One-vs-Rest: обучает $|C|$ моделей, для каждой метки отдельно



Многоклассовая классификация

One-vs-One: обучает $\frac{|C|\cdot(|C|-1)}{2}$ моделей, для каждой метки отдельно



Обработка данных

- Собираем «мешок слов» с помощью `CountVectorizer`
- Отбираем 20 самых популярных тегов
- Тем, кто после фильтрации остался без тегов, ставим новый тег **other**

```
1 tags = texts.tags.apply(lambda x: x.split())
2 all_tags = reduce(lambda s, x: s + x, tags, [])
3 values, count = np.unique(all_tags, return_counts=True)
4
5 top_tags = sorted(zip(count, values), reverse=True)[:20]
```

Выбор модели

Преобразуем списки тегов в матрицу, которая будет содержать индикаторы наличия тега у вопроса:

```
binarizer = MultiLabelBinarizer()
y = binarizer.fit_transform(texts.tags.apply(lambda x: filter_tags(x.split())))
```

Также будет использовать LogisticRegression, но уже вместе с OneVsRestClassifier:

```
params = {'estimator__C': np.logspace(-5, 5, 11)}
clf = OneVsRestClassifier(LogisticRegression())

cv = GridSearchCV(clf, params, n_jobs=-1, scoring=make_scorer(f1_score, average='samples'), cv=5)
cv.fit(bow, y);
```

Выбор модели

```
pd.DataFrame(cv.cv_results_)[['mean_test_score', 'params']]
            .sort_values('mean_test_score', ascending=False)
```

- F_1-score = 0.40473
- Задача стала существенно сложнее
- Попробуем улучшить качество

	mean_test_score	params
10	0.404732	{u'estimator__C': 100000.0}
9	0.404212	{u'estimator__C': 10000.0}
8	0.404140	{u'estimator__C': 1000.0}
7	0.403066	{u'estimator__C': 100.0}
6	0.402630	{u'estimator__C': 10.0}
5	0.390346	{u'estimator__C': 1.0}
4	0.319707	{u'estimator__C': 0.1}
3	0.092920	{u'estimator__C': 0.01}
2	0.000300	{u'estimator__C': 0.001}
0	0.000000	{u'estimator__C': 1e-05}
1	0.000000	{u'estimator__C': 0.0001}

Изменим признаки

Заменим счётчики на TF-IDF:

```
vectorizer = TfidfVectorizer()
tf_idf = vectorizer.fit_transform(texts.text)
```

Попробуем выбрать модель:

```
params = {'estimator__C': np.logspace(-5, 5, 11)}

clf = OneVsRestClassifier(LogisticRegression())
cv = GridSearchCV(clf, params, n_jobs=-1, scoring=make_scorer(f1_score,
    average='samples'), cv=5)
cv.fit(tf_idf, y);
```

По логике должно получиться лучше

Выбор модели

```
1 pd.DataFrame(cv.cv_results_)[['mean_test_score', 'params']]
2             .sort_values('mean_test_score', ascending=False)
```

- F_1-score = 0.38217
- Получилось ощутимо хуже, чем с «мешком слов»
- **Почему?**

	mean_test_score	params
10	0.382173	{u'estimator__C': 100000.0}
9	0.380033	{u'estimator__C': 10000.0}
8	0.376427	{u'estimator__C': 1000.0}
7	0.364680	{u'estimator__C': 100.0}
6	0.334247	{u'estimator__C': 10.0}
5	0.182323	{u'estimator__C': 1.0}
4	0.000700	{u'estimator__C': 0.1}
0	0.000000	{u'estimator__C': 1e-05}
1	0.000000	{u'estimator__C': 0.0001}
2	0.000000	{u'estimator__C': 0.001}
3	0.000000	{u'estimator__C': 0.01}

Выбираем порог

- При вызове predict возвращается 1, если вероятность принадлежности к классу больше 0.5
- Можно выбирать порог самому через кросс-валидацию

```
clf = OneVsRestClassifier(LogisticRegression(C=100000))
y_hat_bow = cross_val_predict(clf, bow, y,
                              method='predict_proba')
y_hat_tf_idf = cross_val_predict(clf, tf_idf, y,
                                 method='predict_proba')
```

Определим функцию, выставляющую тег в зависимости от порога:

```
def get_score(alpha, y, y_hat):
    return f1_score(y, (y_hat > alpha).astype('int'),
                    average='samples')
```

Подбор порога

```
alphas = np.linspace(0.0, 0.01, 100)
scores = [get_score(a, y, y_hat_bow) for a in alphas]

print(np.max(scores))
print(alphas[np.argmax(scores)])
```

BOW:

- Качество с порогом по умолчанию: 0.40473
- Качество с подобранным порогом: 0.45435

TF-IDF:

- Качество с порогом по умолчанию: 0.38217
- Качество с подобранным порогом: **0.49397**

Hashing Trick

- Модели лучше строить с N-граммами
- Но использование N-грамм (даже с грамотным отбором) существенно увеличивает объём словаря
- Выход – вместо самих N-грамм хранить заданное количество хэшей от них:

```
vectorizer = HashingVectorizer(binary=True,
                               ngram_range=(1, 3),
                               n_features=50000)
bow = vectorizer.fit_transform(texts.text)
```

Блендинг

- При наличии несколько моделей можем получать смешенное предсказание
- Если модели не сильно скоррелированы, это может улучшить качество результирующей модели

```
vectorizer = HashingVectorizer(binary=True,
                               ngram_range=(1, 3),
                               n_features=50000)
bow = vectorizer.fit_transform(texts.text)
```

```
vectorizer = TfidfVectorizer()
tf_idf = vectorizer.fit_transform(texts.text)
```

Блендинг

С помощью кросс-валидации предскажем обучающую выборку для каждой модели:

```
clf_lr = OneVsRestClassifier(LogisticRegression(C=100000))
y_hat_lr = cross_val_predict(clf_lr, bow, y, method='predict_proba', cv=
    folds)
```

```
clf_lr = OneVsRestClassifier(LogisticRegression(C=100000))
y_hat_lr_tf_idf = cross_val_predict(clf_lr, tf_idf, y, method='
    predict_proba', cv=folds)
```

Блендинг

Получим качество на каждой модели в отдельности и на их смеси (веса возьмём равными):

```
alphas = np.linspace(0.0, 0.02, 100)

[get_score(a, y, y_hat_lr) for a in alphas]
[get_score(a, y, y_hat_lr_tf_idf) for a in alphas]

[get_score(a, y, 0.5 * y_hat_lr_tf_idf + 0.5 * y_hat_lr)
           for a in alphas]
```

- Качество первой модели: 0.50923
- Качество второй модели: 0.49355
- Качество смеси: **0.52709**

Стекинг

Вместо ручного смешивания результатов можно подавать их на вход другому алгоритму (*мета-алгоритму*)

Подготовим переменную `stacked`, которая будет содержать предсказания предыдущих алгоритмов

```
stacked = np.hstack([y_hat_lr, y_hat_lr_tf_idf])

clf_stacked = OneVsRestClassifier(
                  RandomForestClassifier(n_estimators=100))

y_hat_stacked = cross_val_predict(clf_stacked, stacked,
                                  y, cv=folds,
                                  method='predict_proba')
```

Стекинг

```
alphas = np.linspace(0, 1, 100)
scores = [get_score(a, y, y_hat_stacked) for a in alphas]

plot(alphas, scores);
scatter(alphas[np.argmax(scores)], np.max(scores));

print(np.max(scores))
print(alphas[np.argmax(scores)])
```

0.547874126984

0.232323232323

После подбора порога получили $F_1 = \mathbf{0.54787}$, что больше всех предыдущих результатов.

Стекинг

Стекинг – важная и часто полезная на практике техника, но в её использовании есть подводные камни, связанные с переобучением, поэтому «стекать» надо грамотно.

Полезные ресурсы:

1. Kaggle Ensembling Guide
2. Стекинг и блендинг (Дьяконов)

Библиотека Vowpal Wabbit

- Разработка Yahoo и потом Microsoft
- Библиотека и CLI программа, позволяющая строить линейные модели
- Способна обрабатывать миллиарды объектов с сотнями тысяч признаков

Установка:

- Ubuntu – `apt-get instal vowpal-wabbit`
- Mac OS – `port install vowpal_wabbit`
- Windows – скачать установочник тут
- Варианты установки из официальной вики

Формат ввода

- Использует необычный формат входных данных
- `Label [weight] |Namespace Feature ...|Namespace ...`
  - `Label` – метка класса для задачи классификации или действительное число для задачи регрессии
  - `weight` – вес объекта, по умолчанию у всех одинаковый
  - `Namespace` – все признаки разбиты на области видимости, может использоваться для раздельного использования или создания квадратичных признаков между областями
  - `Feature` – `string[:value]` или `int[:value]` строки будут хешированы, числа будут использоваться как индекс в векторе признаков. `value` по умолчанию равно 1

Параметры

Hashing Trick: Вводится функция h, с помощью которой получается индекс для записи значения в вектор признаков объекта:

$$h : F \to \{0, \ldots, 2^b - 1\}$$

С помощью `--b` можно задавать размер области значений хеш-функции.

Оптимизация: может использоваться SGD или L-BFGS

- SGD по умолчанию, позволяет делать онлайн обучение. Почти всегда необходимо несколько проходов по данным
- L-BFGS включается с помощью `--bfgs`, работает только с данными небольшого размера
- Количество проходов для SGD задаётся с помощью параметра `--passes}`

Параметры оптимизации

Проходим по всем элементам обучающей выборки много раз, на каждом объекте делаем поправку весов:

$$w_{t+1} = w_t + \eta_t \bigtriangledown_w \ell(w_t, x_t)$$

$$\eta_t = \lambda d^k \left(\frac{t_0}{t_0 + w} \right)$$

Здесь t – порядковый номер объекта обучения, k — номер прохода по всей выборке

- λ: `-l` (learning rate)
- d: `--decay_learning_rate`
- t_0: `--initial_t`
- p: `--power_t`
- k_{max}: `--passes`

Оценка качества

average loss – loss by progressive validation

$$\text{Progressive error} = \frac{e_1 + e_2 + \cdots + e_s}{s}$$

e_i – loss на объекте x_i при обучении на объектах $\{x_1, \ldots, x_{i-1}\}$

Функция потерь задаётся параметром `--loss_function`

Vowpal Wabbit поддерживает несколько основных функций потерь для классификации и регрессии (squared, logistic, hinge и т.д.)

Возможность использовать **регуляризацию** с помощью флагов:

- `--l1`
- `--l2`

Данные для бинарной классификации

- Два класса с признаками A и B:

```
-1 | A:1 B:10
 1 | A:-1 B:12
```

- Можно использовать текст без обработки (решим задачу windows vs. linux):

```
 1 | i have a bat file shown below echo off for f d
 1 | i need a way to determine whether the computer
-1 | my c application uses 3rd libraries which do
-1 | currently i m trying to install php 5 3 0 on
 1 | i how to get the windowproc for a form in c fo
```

Обучение и оценивание

Запуск обучения:

```
!vw -d win_vs_lin.train.vw --loss_function logistic -P 10000 -f model.vw --
    passes 100 -c
```

Применение:

```
!vw -i model.vw -t -p output.csv win_vs_lin.test.vw --loss_function
    logistic
```

Результат:

```
y_hat = pd.read_csv('output.csv', header=None)
roc_auc_score(test_texts[1].replace({'-1 ': 0, '1 ': 1}), y_hat[0])
```

0.96415

Многоклассовая классификация

Включается с помощью флага `--multilabel_oaa n`, где n – число классов

Данные будут выглядеть так:

```
3 | i want to use a track bar to change a form s
6 | i have an absolutely positioned div containing
0,3 | given a datetime representing a person s
3 | given a specific datetime value how do i
6,8 | is there any standard way for a web server
```

В остальном всё аналогично бинарному случаю.

Библиотека FastText

- Разработка Facebook
- Библиотека для получения векторных представлений слов и классификации текстов
- Архитектурно схожа с моделью skipgram (word2vec)
- Основана на символьных N-граммах, использует hashing trick
- Работает быстро и качественно
- Ссылки на статьи: 1, 2
- Ссылка на репозиторий
- Документация пакета для Python

Формат данных и обучение

Данные подаются в виде файла, каждый документ – одна строка вида

```
__label__food-safety __label__acidity Dangerous pathogens capable of growing in acidic environments
```

Обучение и сохранение модели:

```
1 classifier = fasttext.supervised('data.train.txt', 'model')
```

Применение обученной модели к тестовым данным:

```
1 result = classifier.test('test.txt')
```

Оценивание результатов

Оценивание качества на тесте:

```
print 'P@1:', result.precision
print 'R@1:', result.recall
print 'Number of examples:', result.nexamples
```

Наиболее вероятные классы (и их вероятности):

```
labels = classifier.predict(texts, k=3)
labels = classifier.predict_proba(texts, k=3)
```