Математические методы анализа текстов

Лекция

Предобработка текстов и выделение признаков. Базовые методы классификации

Мурат Апишев (mel-lain@yandex.ru) [1]

3 сентября, 2019

[1] Подготовлена с использованием материалов курса «Анализ неструктурированных данных» ФКН ВШЭ (ГУ) и кода Богдана Мельника

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):

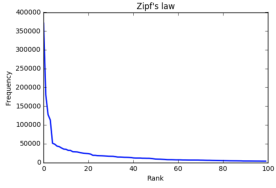
```
1 from nltk.corpus import stopwords
2 words = [word for word in word_list
3           if word not in stopwords.words('russian')]
```

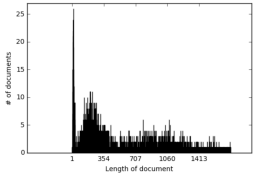
Пунктуацию можно удалять с помощью регулярных выражений, а можно просто:

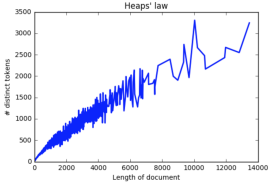
```
1 from string import punctuation
2 s = ''.join(c for c in s if c not in punctuation)
```

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 ?
```

Признаки текста

- Признаки бывают униграмные, N-грамные, документные
- Это могут быть векторы счётчиков, tf-idf, семантические эмбеддиниги
- Или же морфологическая информация о словах документа
- Или же синтасические связи между словами
- Или же ...

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 часто информативны, и значение tf-idf является хорошим признаком.

- Значения tf-idf для слов текста можно получить с помощью `sklearn.feature_extraction.text.TfidfVectorizer`

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).

Методы получения коллокаций

- Извлечение N-грам на основе частот и морфологических шаблонов.
- Поиск разрывных коллокаций.
- Извлечение N-грам на основе мер ассоциации и статистических критериев.
- Более редкие альтернативы (TextRank, RAKE).

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.*
- Начнём с поиска биграмм в теме «Дональд Трамп вступил в должность президента США»

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 или YARGY-парсер)

```
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

Где искать реализацию

Описанные меры реализованы в `nltk.collocations`:

- `bigram_measures.pmi`
- `bigram_measures.student_t`

Есть и другие критерии, например

- `bigram_measures.chi_sq`
- `bigram_measures.likelihood_ratio`

Алгоритм TopMine

- Предложен в El-Kishky, Ahmed, et al. «Scalable topical phrase mining from text corpora.» Proceedings of the VLDB Endowment 8.3 (2014): 305-316.APA
- Ещё один способ извлечения N-грам на основе статистических критериев
- Состоит из двух этапов:
  1. Ищем в документах последовательности слов как можно большей длины, встречающиеся значимое число раз. Для каждого слова и последовательности слов сохраняем частоту
  2. Просматриваем пары слов и последовательностей слов в документе и считаем стат-значимость их совместного появления, пользуясь информацией о частоте встречаемости из первого шага

  Параллельно производится разбивка документов по найденным коллокациям

Постановка задачи классификации

- Классификация – одна из наиболее часто решаемых задач в NLP
- Многие задачи (разметка последовательности, анализ тональности) могут быть сведены к 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

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

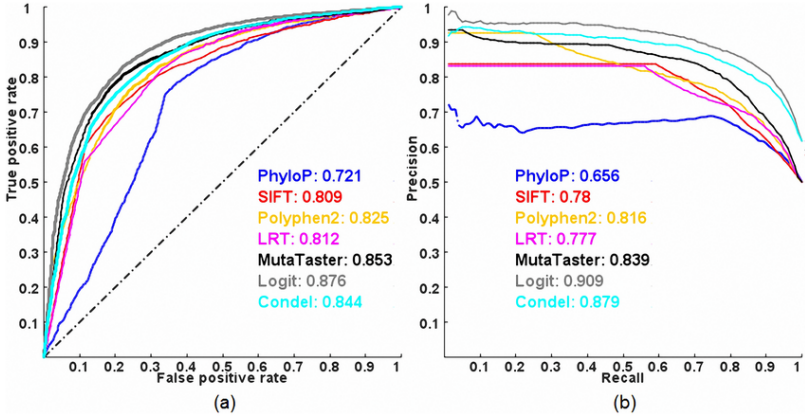
Метрики качества многоклассовой классификации

- 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 – всё, что было выше

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)

Что обычно применяется для классификации текстов

- Наивный байесовский классификатор
- SVM (для тех, кто знает толк в извращениях)
- Логистическая регрессия
- Линейная модель в Vowpal Wabbit
- Рекурентные нейросети
- Свёрточные нейросети
- FastText

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:

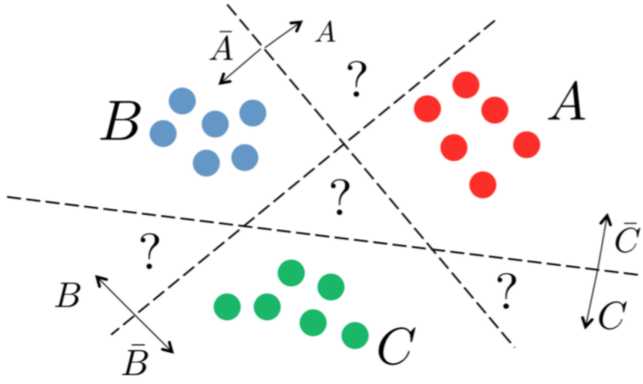
- `KNeighborsClassifier`
- `RandomForestClassifier`
- `SVC`

Ддя остальных есть адапторы:

- `OneVsRestClassifier`
- `OneVsOneClassifier`

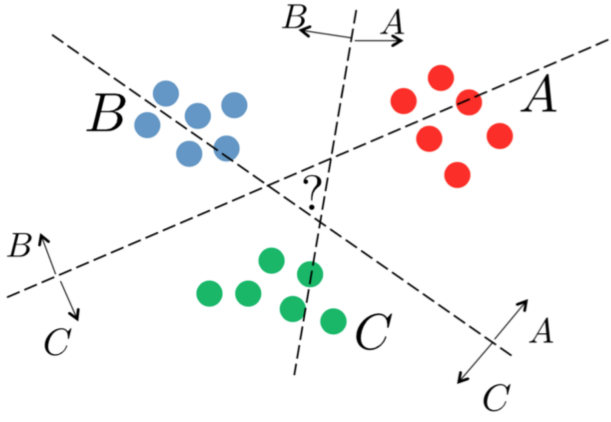
Многоклассовая классификация

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, cv=5,
                  scoring=make_scorer(f1_score,
                                      average='samples'))
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, cv=5,
                  scoring=make_scorer(f1_score,
                                      average='samples'))
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 смашенное 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=10000))
y_hat_lr = cross_val_predict(clf_lr, bow,
                             y, cv=folds,
                             method='predict_proba')
```

```
clf_lr = OneVsRestClassifier(LogisticRegression(C=10000))
y_hat_lr_tf_idf = cross_val_predict(clf_lr, tf_idf,
                                    y, cv=folds,
                                    method='predict_proba')
```

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

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

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

print(np.max(scores))
print(alpha[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)

Задачи сантимент-анализа

Три типа задач (по возрастанию сложности):

1. Полярная тональность (positive / negative / neutral)
2. Ранжированная тональность («звёздочки» от 1 до N)
3. Выявление источника / цели или более сложные типы

http://www.greenbookblog.org/2012/01/02/from-sentiment-analysis-to-enterprise-applications/

Ключевые моменты

- Токенизацию и лемматизацию нужно производить аккуратно, как и фильтрацию словаря
- С опечатками можно бороться с помощью буквенных N-грамм
- `не_` лучше приклеивать к впередистоящему слову, инвертируя его тональность
- Очень важно выделять смайлы и экспрессивную пунктуацию (регулярные выражения)
- Обучение и тестирование нужно производить на схожих данных
- Для учёта редких слов можно логарифмировать частоты

Сложности

Помимо чисто технических проблем, возникают также более сложные семантические:

- Отзывы могут иметь ясный смысл, но при этом не содержать позитивных или негативных слов:

  Это фильм заставляет прочувствовать всю гамму эмоций от «А» до «Я».

- Отзыв может содержать позитивные слова, но на самом деле выражает ожидание:

  Это фильм должен был быть супер крутым. Но не был.

Подходы к решению

- Правила (точно, трудозатратно)

  Я люблю кофе – если сказуемое в группе положительных глаголов и нет отрицаний то positive

- Словари (просто, зависит от предметной области)

слово i из текста	соответствующая тональность $a_i \in$ [1-9] из словаря
хороший	7.47
счастливый	8.21
…	…
скучный	2.95

$$a_{text} = \frac{\sum_{i=1}^{n} a_i}{n}$$

Подходы к решению

- ML: обучение с учителем (классификация) (точно при достаточной обучающей выборке, требуется данные для обучения)
  - Получаем размеченную коллекцию текстов
  - Выделяем признаки (специфичные для тональности)
  - Обучаем классификатор
- ML: обучение без учителя (просто, не требуются данные для обучения, нужен словарь, низкая точность)
  - Выделить ключевые слова (например, по TF-IDF)
  - Определить тональность ключевых слов
  - Сделать вывод о тональности предложения/текста

Сантимент-лексикон

- Существуют наборы слов с оценённой степенью позитивности/негативности, активности.пассивности и т.п.

  Примеры[2]: MPQA subjectivity cues lexicon, SentiWordNet

- Сантимент-лексикон можно размечать самостоятельно, это задача частичного обучения. Необходимо разметить часть примеров и описать правила пополнения.

  Пример: `bad and ugly; cruel but amazing`

  Если два слова связаны and, и тональность одного нам известна, то второе тоже будет иметь такую тональность.

  Если через but — то противоположную.

[2] Ссылка на ресурсы

Пример классификации

```
1  from nltk.classify.util import accuracy
2  from nltk.classify import NaiveBayesClassifier
3  from nltk.corpus import movie_reviews
4
5  def get_feats(tokens):
6      return dict([(token, True) for token in tokens])
7
8  def create_sample(tag, train_ratio):
9      ids = movie_reviews.fileids(tag)
10     feats = [(get_feats(
11                 movie_reviews.words(fileids=[f])),
12                 tag) for f in ids]
13
14     idx = int(len(feats) * train_ratio)
15     train, test = feats[: idx], feats[idx: ]
16     return train, test
```

Пример классификации

```
train_pos, test_pos = create_sample('pos', 0.75)
train_neg, test_neg = create_sample('neg', 0.75)
train = train_pos + train_neg
test = test_pos + test_neg

print('Train on {} docs, test on P{} docs'.format(
          len(train), len(test)))

classifier = NaiveBayesClassifier.train(train)

print('Accuracy: {}'.format(accuracy(classifier, test)))

classifier.show_most_informative_features()
```

Результаты

```
Train on 1500 documents, test on 4000 documents
Accuracy: 0.728

Most Informative Features
     magnificent = True      pos : neg = 15.0 : 1.0
     outstanding = True      pos : neg = 13.6 : 1.0
     insulting = True        neg : pos = 13.0 : 1.0
     vulnerable = True       pos : neg = 12.3 : 1.0
     ludicrous = True        neg : pos = 11.8 : 1.0
     avoids = True           pos : neg = 11.7 : 1.0
     uninvolving = True      neg : pos = 11.7 : 1.0
     astounding = True       pos : neg = 10.3 : 1.0
     fascination = True      pos : neg = 10.3 : 1.0
     idiotic = True          neg : pos = 9.8 : 1.0
```