Data Culture Hack

Извлечение обсуждаемых тематик из корпуса текстов

Мурат Апишев

mel-lain@yandex.ru

mel-lain@ddecisions.ai

30 ноября, 2019

Чему будет посвящён сегодняшний рассказ

- Что есть «тематики» и чем они полезны
- Предобработка текстовых данных
- Выделение из текстов важных слов и словосочетаний
- Методы выделения тематик из текстов
- Тематическое моделирование
- Дополнительный анализ тематик

Что же собираемся извлекать?

Под *тематикой* будем понимать набор слов и словосочетаний, которые

- каким-то образом связаны друг с другом в анализируемых текстах
- характеризуют одну (реже несколько) смысловых сущностей

Например:

автомобиль	прыжки с шестом	регрессия
внедорожник	олимпиада	линейный классификатор
хороший двигатель	длинный бассейн	анализ
высокая подвеска	полные трибуны	данные
бампер	олимпийский факел	svm
удобные кожаные сиденья	допинговая комиссия	сеть
известная марка	мок	нейрон

Предобработка текста

Важно изучить и обработать данные до начала серьёзного анализа!

Базовые шаги предобработки текстов:

- токенизация
- приведение к нижнему регистру
- удаление стоп-слов
- удаление пунктуации
- фильтрация по частоте/длине/соответствию регулярному выражению
- лемматизация или стемминг

Полезные модули языка Python:

`nltk`, `re`, `regex`, `pymorphy2`, `pymystem3`

Пример лемматизации

```
1  import pymorphy2
2
3  text_ru = 'Где твоя большая ложка'
4  pymorph = pymorphy2.MorphAnalyzer()
5
6  result = ''
7  for word in text_ru.split(' '):
8      result += ' {}'.format(pymorph.parse(word)[0].normal_form)
9
10 print(result)
```

Вывод:

```
где твой большой ложка
```

Что делать дальше?

- Почти всегда процесс извлечения тематик разибивается на два основных этапа:
  1. выбор слов или словосочетаний (*токенов*), из которых тематики могут состоять
  2. группировка токенов по выбранной мере близости (например, семантической)
- Сосредоточимся на первом шаге: отборе токенов
- Методы токенизации обычно возвращают список одиночных слов (*униграмм*)
- Из примеров мы видели, что они интерпретируются гораздо хуже словосочетаний (*N-грамм*)
- **Вывод – нужно научиться извлекать из текстов N-граммы**

Как правильно отбирать N-граммы

- Нельзя просто взять всевозможные последовательности слов разной длины
  1. потребуется очень много ресурсов и времени
  2. качество результат сильно пострадает
- Нужно из последовательностей фиксированной длины выбрать наилучшие
- Наилучшие – это те, в которых слова
  1. встречаются вместе неслучайно
  2. являются значимыми (не стоп-слова, не предлоги и т.п.)
- Первое требование можно проверять с помощью статистических критериев (реализации доступны в `nltk`)
- Второе – с помощью анализаторов частей речи (можно воспользоваться `Yargy-parser` или `Tomita-parser`)

T-критерий для оценки неслучайности биграммы

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

где

- где w_i – токен
- $f(\cdot)$ – частота токена или биграммы
- N – общее количество биграмм

Немного деталей:

- Проверяется гипотеза независимой встречаемости двух токенов
- Является модифицированным ранжированием по частоте
- Не преувеличивает значимость редких словосочетаний
- Выделяет общеязыковые устойчивые сочетания

Реализация в NLTK

```
1 import nltk
2 def generate_collocations(tokens):
3     bigram_measures = nltk.collocations.BigramAssocMeasures()
4     finder = nltk.collocations.BigramCollocationFinder.from_words(tokens)
5
6     colls = finder.nbest(bigram_measures.student_t, 10)
7     colls = [{k: finder.ngram_fd[k]} for k in colls]
8
9     return colls
```

- На выходе будет `dict` из сочетания в частоту встречаемости
- `BigramCollocationFinder` можно заменить на `TriramCollocationFinder` или `QuadgramCollocationFinder`
- Есть и другие критерии, например
  - `bigram_measures.pmi`
  - `bigram_measures.chi_sq`
  - `bigram_measures.likelihood_ratio`

Выделение ключевых токенов по TF-IDF

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

$$\text{TfIdf-Score}(w, d) = tf_{wd} \times \log \frac{N}{df_w}$$

где

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

- Значение TF-IDF хорошо коррелирует со смысловой значимостью токена в документе и является полезным признаком

Подсчёт TF-IDF с помощью scikit-learn

```
1  from sklearn.feature_extraction.text import TfidfVectorizer
2  corpus = [
3      'This is the first document.',
4      'This document is the second document.',
5      'And this is the third one.',
6      'Is this the first document?',
7  ]
8
9  vectorizer = TfidfVectorizer()
10 X = vectorizer.fit_transform(corpus)
11
12 print(vectorizer.get_feature_names())
13 print(X.shape)
```

Вывод:

```
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
(4, 9)
```

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Формирование тематик

- Мы выделили множество слов, из которых хочется сформировать тематики
- Следующий шаг – сгруппировать в семантически однородные подмножества
- Эту задачу можно решать разнообразными способами:
  - Построение сжатых векторных представлений токенов с последующей кластеризацией
  - Выделение сообществ в графе, построенном на попарных статистиках токенов (например, на значениях совстречаемости)
  - Вероятностная кластеризация с помощью тематического моделирования
- На самом деле все эти методы имеют общую природу и оперируют одной и той же информацией
- Но на практике проявляются детали, которые могут существенно повлиять на итоговый результат

Но прежде – «мешок слов»

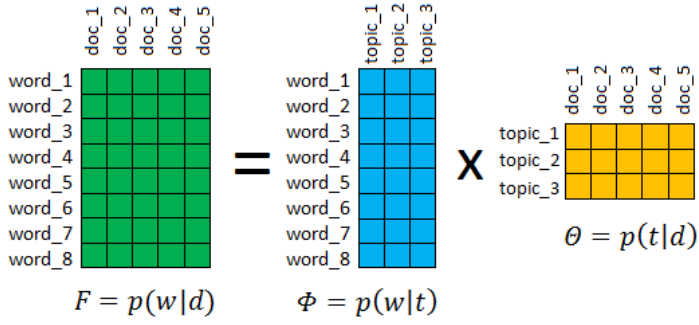
- *Мешок слов (Bag-of-Words)* – способ представления текстового документа с потерей информаций о порядке слов
- Последовательный текст превращается в словарь:
  - ключи – уникальные токены текста
  - значения – частоты встречаемости этих токенов в этом тексте
- По-сути, очень похоже на TF-IDF, но используется только значение TF
- Можно считать в sklearn с помощью кода из примера выше, заменив `TfidfVectorizer` на `CountVectorizer`
- На выходе получается разреженная матрица размера $W \times D$, где W – число уникальных токенов, а D – число документов
- Нормируем её по столбцам и получим *стохастическую* матрицу вероятностей токенов в документе, которую обозначим F

Тематическое моделирование

- *Тематическое моделирование* – набор методов статистического анализа текстов для выявления тематик из корпуса текстов
- Тематическую модель можно рассматривать как
  - разложение матрицы F в произведение двух стохастических матриц с промежуточной размерностью, равной числу искомых тем
  - мягкую бикластеризацию токенов по множеству тем и тем – по множеству документов
  - поиск параметров некоторого генеративного процесса, породившего анализируемые тексты
- Все формулировки эквивалентны (иногда с точностью до незначительных изменений постановки задачи)
- Основные модели: PLSA, LDA, ARTM

Probabilistic Latent Semantic Analysis

$$p(w|d) = \sum_{t \in T} p(w|t)p(t|d)$$

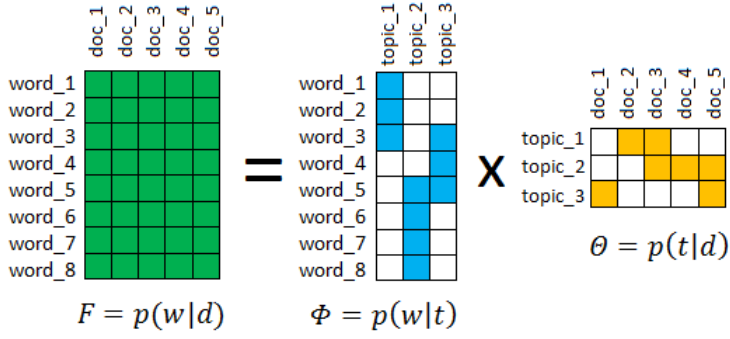


- Каждый синий столбец – распределение на токенах (темы)
- Каждый жёлтый столбец – распределение на темах внутри одного документа
- Φ – это TM, Θ – результат её применения к обучающей выборке
- Выберем наиболее вероятные токены в каждой теме – это и есть тематики

Additive Regularization of Topic Models

Логичные предположения:

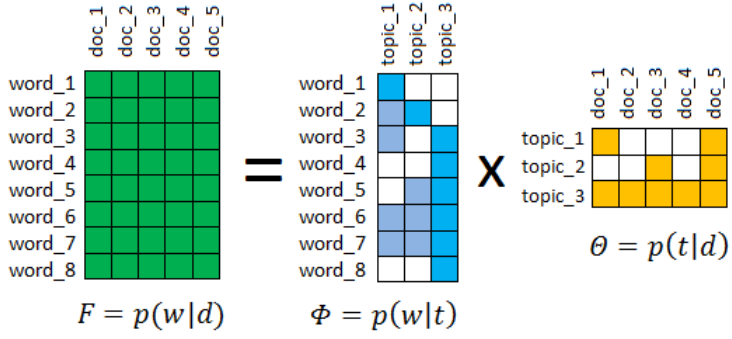
- темы должны состоять из небольшого числа слов, и эти множества слов не должны сильно пересекаться
- каждый документ должен относиться к небольшому числу тем



Additive Regularization of Topic Models

Извлечение специфичной тематики по ключевым словам:

- хотим собрать темы около интересующих слов, а документы — около интересующих тем
- прочие темы хотим сглаживать по неважным словам, чтобы собрать «мусор»



$F = p(w|d)$ $\Phi = p(w|t)$ $\Theta = p(t|d)$

Пакеты для тематического моделирования

Простую модель PLSA или LDA для небольших данных можно построить с помощью gensim:

```
1 from gensim.test.utils import common_texts
2 from gensim.corpora.dictionary import Dictionary
3 from gensim.models import LdaModel
4
5 dictionary = Dictionary(common_texts)
6 corpus = [common_dictionary.doc2bow(text) for text in common_texts]
7
8 lda = LdaModel(corpus, num_topics=5)
9 lda.get_term_topics(dictionary.token2id['survey'], minimum_probability=0.0)
```

Вывод:

```
[(0, 0.002770255), (1, 0.0028463434),
 (2, 0.00055916863), (3, 0.090775795),
 (4, 0.0009080497)]
```

https://radimrehurek.com/gensim/models/ldamodel.html

Пакеты для тематического моделирования

Модели ARTM с гибкой настройкой и возможностью частичного обучения доступны в `bigartm`:

```
import artm

vectorizer = artm.BatchVectorizer(data_path='.', data_format='bow_uci',
                                  collection_name='kos',
                                  target_folder='kos_batches')

model = artm.ARTM(num_topics=10, dictionary=vectorizer.dictionary)
model.scores.add(artm.TopTokensScore(name='tt-score', num_tokens=10))

model.fit_offline(vectorizer, num_collection_passes=5)
model.score_tracker['tt-score'].last_tokens
```

В модель можно добавлять *регуляризаторы* (ограничители), которые позволяют управлять процесс обучения

Данные для примера https://archive.ics.uci.edu/ml/datasets/Bag+of+Words

Простейший пост-анализ тематик

- Автоматический поиск близких и дублирующихся тематик *(с помощью пересечений наборов слов или близостей векторных представлений)*
- Поиск тематических документов *(в TM – из коробки, иначе нужно искать по словам и их векторам)*
- Тональность тематик можно оценивать по тональности относящихся к ним документов *(словарный подход или предобученные модели, пример –* `dostoevsky`*)*
- Оценивание доли тематик в коллекции *(в TM – из коробки, иначе – эвристики на основе доли отнесённых документов)*
- **Но самое важное и надёжное – внимательно посмотреть на результат собственными глазами!**

Теперь поговорим о вашем задании:)