Оглавление

Предисловие 17

Вступление 19

Благодарности 27

Хобсон Лейн 29

Ханнес Макс Хапке 29

Коул Ховард 29

Об этой книге 30

Дорожная карта 30

Что вы найдете в книге 31

О коде 32

Дискуссионный форум liveBook 32

Об авторах 33

Об иллюстрации на обложке 34

От издательства 34

Часть I. Машины для обработки слов

Глава 1. Знакомство с технологией NLP 37

1.1. Естественный язык в сравнении с языком программирования 38

1.2. Волшебство 39

1.2.1. Машины, которые общаются 40

1.2.2. Математика 40

1.3. Практическое применение 42

1.4. Язык глазами компьютера 44

1.4.1. Язык замков 45

1.4.2. Регулярные выражения 46

1.4.3. Простой чат-бот ... 48
1.4.4. Другой вариант ... 52
1.5. Краткая экскурсия по гиперпространству ... 56
1.6. Порядок слов и грамматика ... 58
1.7. Конвейер чат-бота на естественном языке ... 59
1.8. Углубленная обработка ... 62
1.9. IQ естественного языка ... 65
Резюме ... 67

Глава 2. Составление словаря: токенизация слов ... 68
2.1. Непростые задачи: обзор стемминга ... 70
2.2. Построение словаря с помощью токенизатора ... 71
2.2.1. Скалярное произведение ... 80
2.2.2. Измерение пересечений мультимножеств слов ... 81
2.2.3. Улучшение токенов ... 82
2.2.4. Расширяем словарь n-граммами ... 87
2.2.5. Нормализация словаря ... 94
2.3. Тональность ... 103
2.3.1. VADER — анализатор тональности на основе правил ... 104
2.3.2. Наивный байесовский классификатор ... 106
Резюме ... 110

Глава 3. Арифметика слов: векторы TF-IDF ... 111
3.1. Мультимножество слов ... 113
3.2. Векторизация ... 118
3.2.1. Векторные пространства ... 120
3.3. Закон Ципфа ... 126
3.4. Тематическое моделирование ... 129
3.4.1. Возвращаемся к закону Ципфа ... 132
3.4.2. Ранжирование по релевантности ... 133
3.4.3. Инструменты ... 136
3.4.4. Альтернативы ... 137
3.4.5. Okapi BM25 ... 138
3.4.6. Что дальше? ... 139
Резюме ... 139

Глава 4. Поиск смысла слов по их частотностям: семантический анализ ... 140
4.1. От частотностей слов до оценок тем ... 142
4.1.1. Векторы TF-IDF и лемматизация ... 142
4.1.2. Векторы тем ... 143

4.1.3. Мысленный эксперимент ... 144
4.1.4. Алгоритм оценки тем ... 149
4.1.5. LDA-классификатор ... 150
4.2. Латентно-семантический анализ ... 155
4.2.1. Воплощаем мысленный эксперимент на практике ... 158
4.3. Сингулярное разложение ... 160
4.3.1. U — левые сингулярные векторы ... 162
4.3.2. S — сингулярные значения ... 163
4.3.3. V^T — правые сингулярные векторы ... 164
4.3.4. Ориентация SVD-матрицы ... 165
4.3.5. Усечение тем ... 166
4.4. Метод главных компонент ... 168
4.4.1. PCA на трехмерных векторах ... 170
4.4.2. Хватит возиться с лошадьми, возвращаемся к NLP ... 171
4.4.3. Применение PCA для семантического анализа СМС ... 174
4.4.4. Применение усеченного SVD для семантического анализа СМС ... 176
4.4.5. Насколько хорошо LSA классифицирует спам ... 177
4.5. Латентное размещение Дирихле ... 180
4.5.1. Основная идея LDiA ... 181
4.5.2. Тематическая модель LDiA для СМС ... 183
4.5.3. LDiA + LDA = классификатор спама ... 186
4.5.4. Более честное сравнение: 32 темы LDiA ... 188
4.6. Расстояние и подобие ... 190
4.7. Стиринг и обратная связь ... 192
4.7.1. Линейный дискриминантный анализ ... 194
4.8. Мощь векторов тем ... 195
4.8.1. Семантический поиск ... 197
4.8.2. Дальнейшие усовершенствования ... 200
Резюме ... 200

Часть II. Более глубокое обучение: нейронные сети

Глава 5. Первые шаги в нейронных сетях: перцептроны и метод обратного распространения ошибки ... 203
5.1. Нейронные сети, список ингредиентов ... 204
5.1.1. Перцептрон ... 205
5.1.2. Числовой перцептрон ... 205
5.1.3. Коротко про смещение ... 206
5.1.4. Айда кататься на лыжах — поверхность ошибок ... 221

5.1.5. С подъемника — на склон ... 222
5.1.6. Проведем небольшую реорганизацию ... 223
5.1.7. Keras: нейронные сети на Python ... 224
5.1.8. Вперед и вглубь ... 228
5.1.9. Нормализация: «стильный» входной сигнал ... 228
Резюме ... 229

Глава 6. Умозаключения на основе векторов слов (Word2vec) ... 230
6.1. Семантические запросы и аналогии ... 231
6.1.1. Вопросы на аналогию ... 232
6.2. Векторы слов ... 233
6.2.1. Векторные умозаключения ... 237
6.2.2. Вычисление представлений Word2vec ... 240
6.2.3. Использование модуля gensim.word2vec ... 250
6.2.4. Как сгенерировать свои собственные представления векторов слов ... 252
6.2.5. Word2vec по сравнению с GloVe (моделью глобальных векторов) ... 255
6.2.6. FastText ... 256
6.2.7. Word2vec по сравнению с LSA ... 257
6.2.8. Визуализация связей между словами ... 258
6.2.9. Искусственные слова ... 264
6.2.10. Определение сходства документов с помощью Doc2vec ... 266
Резюме ... 268

Глава 7. Сверточные нейронные сети ... 269
7.1. Усвоение смысла ... 271
7.2. Инструментарий ... 272
7.3. Сверточные нейронные сети ... 273
7.3.1. Стандартные блоки ... 274
7.3.2. Размер шага (свертки) ... 275
7.3.3. Формирование фильтров ... 276
7.3.4. Дополнение ... 278
7.3.5. Обучение ... 279
7.4. Окна и правда узкие ... 280
7.4.1. Реализация на Keras: подготовка данных ... 282
7.4.2. Архитектура сверточной нейронной сети ... 288
7.4.3. Субдискретизация ... 288
7.4.4. Дропаут ... 291
7.4.5. Вишенка на торте ... 292

7.4.6. Приступаем к обучению ... 294
7.4.7. Применение модели в конвейере ... 296
7.4.8. Что дальше? ... 297
Резюме ... 299

Глава 8. Нейронные сети с обратной связью: рекуррентные нейронные сети ... 300
8.1. Запоминание в нейронных сетях ... 303
8.1.1. Обратное распространение ошибки во времени ... 308
8.1.2. Когда что обновлять ... 311
8.1.3. Краткое резюме ... 313
8.1.4. Всегда есть какой-нибудь подвох ... 313
8.1.5. Рекуррентные нейронные сети и Keras ... 314
8.2. Собираем все вместе ... 318
8.3. Приступим к изучению прошлого ... 321
8.4. Гиперпараметры ... 321
8.5. Предсказание ... 324
8.5.1. Сохранение состояния ... 325
8.5.2. И в другую сторону ... 326
8.5.3. Что это такое? ... 327
Резюме ... 328

Глава 9. Эффективное сохранение информации с помощью сетей с долгой краткосрочной памятью ... 329
9.1. Долгая краткосрочная память ... 330
9.1.1. Обратное распространение ошибки во времени ... 341
9.1.2. А как же проверка на практике? ... 343
9.1.3. «Грязные» данные ... 345
9.1.4. Возвращаемся к «грязным» данным ... 348
9.1.5. Работать со словами сложно. С отдельными буквами — проще ... 349
9.1.6. Моя очередь говорить ... 354
9.1.7. Моя очередь говорить понятнее ... 357
9.1.8. Мы научились, как говорить, но не что говорить ... 365
9.1.9. Другие виды памяти ... 365
9.1.10. Углубляемся ... 366
Резюме ... 368

Глава 10. Модели sequence-to-sequence и механизм внимания ... 369
10.1. Архитектура типа «кодировщик — декодировщик» ... 370
10.1.1. Декодирование вектора идеи ... 371
10.1.2. Знакомо, правда? ... 374

10.1.3. Диалог с помощью sequence-to-sequence 375
10.1.4. Обзор LSTM..... 376
10.2. Компонуем конвейер sequence-to-sequence 377
10.2.1. Подготавливаем набор данных для обучения модели sequence-to-sequence..... 378
10.2.2. Модель sequence-to-sequence в Keras..... 379
10.2.3. Кодировщик последовательностей..... 380
10.2.4. Декодировщик идеи..... 382
10.2.5. Формируем сеть sequence-to-sequence 383
10.3. Обучение сети sequence-to-sequence..... 384
10.3.1. Генерация выходных последовательностей..... 385
10.4. Создание чат-бота с помощью сетей sequence-to-sequence..... 386
10.4.1. Подготовка корпуса для обучения..... 386
10.4.2. Формирование словаря символов..... 388
10.4.3. Генерируем унитарные тренировочные наборы данных 388
10.4.4. Обучение нашего чат-бота sequence-to-sequence..... 389
10.4.5. Формируем модель для генерации последовательностей 390
10.4.6. Предсказание последовательности 391
10.4.7. Генерация ответа..... 391
10.4.8. Общаемся с нашим чат-ботом 392
10.5. Усовершенствования 393
10.5.1. Упрощаем обучение с помощью группирования данных..... 393
10.5.2. Механизм внимания 394
10.6. На практике 396
Резюме 398

Часть III. Поговорим серьезно. Реальные задачи NLP

Глава 11. Выделение информации: выделение поименованных сущностей и формирование ответов на вопросы..... 401
11.1. Поименованные сущности и отношения..... 401
11.1.1. База знаний 402
11.1.2. Выделение информации..... 405
11.2. Регулярные паттерны..... 405
11.2.1. Регулярные выражения..... 407
11.2.2. Выделение информации и признаков при машинном обучении..... 407
11.3. Заслуживающая выделения информация 409
11.3.1. Выделение GPS-координат..... 409
11.3.2. Выделение дат..... 410

11.4. Выделение взаимосвязей (отношений) ... 415
11.4.1. Частеречная (POS) разметка ... 416
11.4.2. Нормализация имен сущностей ... 420
11.4.3. Нормализация и выделение отношений ... 421
11.4.4. Паттерны слов ... 421
11.4.5. Сегментация ... 422
11.4.6. Почему не получится разбить по ('.!?') ... 424
11.4.7. Сегментация предложений с помощью регулярных выражений ... 425
11.5. На практике ... 427
Резюме ... 428

Глава 12. Начинаем общаться: диалоговые системы ... 429
12.1. Языковые навыки ... 430
12.1.1. Современные подходы ... 432
12.1.2. Гибридный подход ... 438
12.2. Подход сопоставления с паттернами ... 439
12.2.1. Сопоставляющий с паттернами чат-бот на основе AIML ... 440
12.2.2. Сетевое представление сопоставления с паттерном ... 447
12.3. Заземление ... 448
12.4. Информационный поиск ... 450
12.4.1. Проблема контекста ... 451
12.4.2. Пример чат-бота на основе информационного поиска ... 453
12.4.3. Чат-бот на основе поиска ... 456
12.5. Порождающие модели ... 459
12.5.1. Разговор в чате про nlpia ... 459
12.5.2. Достоинства и недостатки каждого из подходов ... 462
12.6. Подключаем привод на четыре колеса ... 462
12.6.1. Will — залог нашего успеха ... 463
12.7. Процесс проектирования ... 464
12.8. Маленькие хитрости ... 467
12.8.1. Задавайте вопросы с предсказуемыми ответами ... 467
12.8.2. Развлекайте пользователей ... 467
12.8.3. Если все остальное не дает результата — ищите! ... 468
12.8.4. Стремитесь к популярности ... 468
12.8.5. Объединяйте людей ... 468
12.8.6. Проявляйте эмоции ... 469
12.9. На практике ... 469
Резюме ... 470

Глава 13. Масштабирование: оптимизация, распараллеливание и обработка по батчам ... 471
13.1. Слишком много хорошего (данных) ... 472
13.2. Оптимизация алгоритмов NLP ... 472
13.2.1. Индексация ... 473
13.2.2. Продвинутая индексация ... 475
13.2.3. Продвинутая индексация с помощью пакета Annoy ... 477
13.2.4. Зачем вообще использовать приближенные индексы ... 481
13.2.5. Решение проблемы индексации: дискретизация ... 482
13.3. Алгоритмы с постоянным расходом RAM ... 483
13.3.1. gensim ... 484
13.3.2. Вычисления на графах ... 485
13.4. Распараллеливание вычислений NLP ... 486
13.4.1. Обучение моделей NLP на GPU ... 486
13.4.2. Арендовать или покупать? ... 487
13.4.3. Варианты аренды GPU ... 488
13.4.4. Тензорные процессоры ... 489
13.5. Сокращение объема потребляемой памяти при обучении модели ... 490
13.6. Как почерпнуть полезную информацию о модели с помощью TensorBoard ... 493
13.6.1. Визуализация вложений слов ... 493
Резюме ... 496

Приложения

Приложение A. Инструменты для работы с NLP ... 498
A.1. Anaconda3 ... 498
A.2. Установка пакета nlpia ... 499
A.3. IDE ... 500
A.4. Система управления пакетами Ubuntu ... 501
A.5. Mac ... 502
A.5.1. Система управления пакетами для Macintosh ... 502
A.5.2. Дополнительные пакеты ... 502
A.5.3. Настройки ... 502
A.6. Windows ... 504
A.6.1. Переходите в виртуальность ... 505
A.7. Автоматизация в пакете nlpia ... 505

Приложение Б. Эксперименты с Python и регулярные выражения ... 506
Б.1. Работа со строковыми значениями ... 507
Б.1.1. Типы строк: str и bytes ... 507
Б.1.2. Шаблоны в Python: .format() ... 508
Б.2. Ассоциативные массивы в Python: dict и OrderedDict ... 508
Б.3. Регулярные выражения ... 508
Б.3.1. | — OR ... 509
Б.3.2. () — группы ... 510
Б.3.3. [] — классы символов ... 511
Б.4. Стиль ... 511
Б.5. Овладейте в совершенстве ... 512

Приложение В. Векторы и матрицы: базовые элементы линейной алгебры ... 513
В.1. Векторы ... 513
В.2. Расстояния ... 515

Приложение Г. Инструменты и методы машинного обучения ... 520
Г.1. Выбор данных и устранение предвзятости ... 520
Г.2. Насколько хорошо подогнана модель ... 521
Г.3. Знание — половина победы ... 523
Г.4. Перекрестное обучение ... 524
Г.5. Притормаживаем модель ... 525
Г.5.1. Регуляризация ... 525
Г.5.2. Дропаут ... 526
Г.5.3. Нормализация по мини-батчам ... 527
Г.6. Несбалансированные тренировочные наборы данных ... 527
Г.6.1. Супердискретизация ... 528
Г.6.2. Субдискретизация ... 528
Г.6.3. Дополнение данных ... 529
Г.7. Метрики эффективности ... 530
Г.7.1. Оценка эффективности классификатора ... 530
Г.7.2. Оценка эффективности регрессора ... 532
Г.8. Советы от профессионалов ... 533

Приложение Д. Настройка GPU на AWS ... 535
Д.1. Шаги создания экземпляра с GPU на AWS ... 536
Д.1.1. Контроль затрат ... 547

Приложение Е. Хеширование с учетом локальности 549
Е.1. Векторы высокой размерности принципиально различны 550
Е.1.1. Индексы и хеши векторного пространства 550
Е.1.2. Мыслим многомерно 551
Е.2. Многомерная индексация 554
Е.2.1. Хеширование с учетом локальности 555
Е.2.2. Приближенный метод поиска ближайших соседей 555
Е.3. Предсказание лайков 556

Источники информации 558
Приложения и идеи проектов 558
Курсы и учебные руководства 560
Утилиты и пакеты 560
Научные статьи и обсуждения 561
Конкурсы и премии 564
Наборы данных 565
Поисковые системы 565

Глоссарий 569
Акронимы 570
Терминология 574

3 Арифметика слов: векторы TF-IDF

В этой главе

- Подсчет слов и частотностей термов для анализа смысла.
- Предсказание вероятностей вхождений слов с помощью закона Ципфа.
- Векторные представления слов и способы их использования.
- Поиск релевантных документов из корпуса на основе обратных частотностей документов.
- Оценка сходства пар документов с помощью коэффициентов Отиаи и метрики Okapi BM25.

После того как мы собрали и посчитали слова (токены), а также объединили их по основам или леммам, настало время использовать полученную информацию для чего-нибудь интересного. Обнаружение слов хорошо подходит для простых задач вроде получения сводных показателей использования слов или поиска по ключевым словам. Но вам бы хотелось узнать, какие слова играют более важную роль для конкретного документа или даже корпуса в целом, чтобы затем искать в корпусе релевантные документы на основе этого показателя важности.

Такой подход снижает вероятность ошибочного срабатывания детектора спама на отдельном бранном слове или нескольких слегка напоминающих спам-слова

в сообщении электронной почты. При наличии широкого диапазона слов разной степени позитивности показателей (меток) можно измерить позитивность и дружелюбность какого-нибудь твита. Если знать, с какой частотой конкретные слова появляются в документе *относительно* остальных документов, можно еще больше определить уровень «позитивности» . В этой главе мы расскажем о более гибких способах измерения частоты слов и их применения в документах. Этот подход на протяжении десятилетий был основным при создании признаков на основе текстов на естественном языке в коммерческих поисковых системах, а также фильтрах спама.

Следующим шагом нашего путешествия в мир обработки естественного языка является превращение слов в непрерывные числовые величины, а не просто в целые числа, отражающие количества слов, или бинарные векторы, указывающие наличие/отсутствие конкретных слов. Над векторными представлениями слов в непрерывном пространстве можно производить более интересные математические операции. Наша цель — найти числовое представление, которое отражало бы важность или информационное содержание представляемых слов. Подождите до главы 4, и вы узнаете, как превратить это информационное наполнение в числа, отражающие *смысл* слов.

В данной главе мы рассмотрим три набирающих популярность способа представления слов и их значения.

- *Мультимножества слов* — векторы количеств, или частотностей, слов.
- *Мультимножества n-грамм* — векторы количеств пар слов (биграмм), троек слов (триграмм) и т. д.
- *Векторы TF-IDF* — показатели слов, наилучшим образом отражающие степень их важности.

ВАЖНО

TF-IDF расшифровывается как *«частотность терма умножить на обратную частотность документа»* (term frequency inverse document frequency). Частотность слова — количество вхождений слова в документ, о котором мы говорили в предыдущих главах. Обратная частотность документа означает, что это количество для конкретного слова делится на число документов, в которых оно встречается.

Каждый из этих методов может применяться как отдельно, так и в качестве части конвейера NLP. Все описанные выше модели являются статистическими в том смысле, что они основаны на *частотностях* слов. Позже мы рассмотрим различные способы заглянуть глубже в связи между словами, закономерности их использования и нелинейности.

Все описанные выше «поверхностные» алгоритмы NLP обладают большими возможностями и применяются на практике для выполнения многих задач, таких как фильтрация спама и анализ тональностей.

3.1. Мультимножество слов

В предыдущей главе мы создали вашу первую модель векторного пространства текста. Для этого мы создали унитарные представления всех слов, а затем объединили все эти векторы с помощью бинарного OR (или усеченной версии `sum`) в векторное представление всего текста. Получившийся в результате бинарный вектор мультимножества слов, будучи загруженным в структуру данных вроде DataFrame библиотеки Pandas, оказывается отличным индексом для поиска документов.

Далее мы рассмотрели еще более удобное векторное представление с подсчетом количества вхождений (частотности) слова в тексте. На первый взгляд, чем чаще встречается слово в тексте, тем больший вклад вносит в его смысл. Документ, в котором часто упоминаются «крылья» и «руль», с большей вероятностью связан с самолетами или воздушными путешествиями, чем документ со словами «кошки» и «гравитация». Если некоторые слова были классифицированы как отражающие положительные эмоции — вроде *good*, *best*, *joy* и *fantastic*, то тональность содержащего их документа, скорее всего, положительная. Впрочем, понятно, что основанный на таких простых правилах алгоритм может легко ошибиться.

Рассмотрим пример, в котором подсчет частотностей слов оказывается полезным:

```
>>> from nltk.tokenize import TreebankWordTokenizer
>>> sentence = """The faster Harry got to the store, the faster Harry,
...     the faster, would get home."""
>>> tokenizer = TreebankWordTokenizer()
>>> tokens = tokenizer.tokenize(sentence.lower())
>>> tokens
['the',
 'faster',
 'harry',
 'got',
 'to',
 'the',
 'store',
 ',',
 'the',
 'faster',
 'harry',
 ',',
 'the',
 'faster',
 ',',
 'would',
 'get',
 'home',
'.']
```

Этот простой список нужен, чтобы выделить из документа уникальные слова и найти их количества. Словарь Python отлично подходит для этой цели. Поскольку

необходимо хранить еще и количества вхождений слов, можно воспользоваться типом Counter, как мы делали в предыдущих главах:

```
>>> from collections import Counter
>>> bag_of_words = Counter(tokens)
>>> bag_of_words
Counter({'the': 4,
         'faster': 3,
         'harry': 2,
         'got': 1,
         'to': 1,
         'store': 1,
         ',': 3,
         'would': 1,
         'get': 1,
         'home': 1,
         '.': 1})
```

Как и в любом хорошем словаре Python, порядок ключей перемешивается. Новый порядок оптимизирован для хранения, обновления и поиска, а не для согласованного отображения. Информация, заключенная в порядке слов исходного высказывания, отбрасывается.

ПРИМЕЧАНИЕ

Объект collections.Counter — неупорядоченная коллекция, также называемая мультимножеством. В зависимости от платформы и версии Python счетчик может отображаться в, казалось бы, логичном порядке, например в лексикографическом или в порядке токенов в высказывании. Но рассчитывать на какой-либо определенный порядок токенов (ключей) как в объекте Counter, так и в обычном классе dict языка Python нельзя.

Для коротких документов, как этот, даже в неупорядоченном мультимножестве слов содержится немало информации об исходном подтексте предложения. Причем информации в мультимножестве достаточно для весьма серьезных задач, таких как обнаружение спама, анализ тональности (позитивность, счастье и т. д.), и даже для выявления таких тонких эмоций, как сарказм. Да, это просто набор слов, но наполненный смыслом и информацией. Итак, проранжируем эти слова — отсортируем их в более удобном для понимания порядке. У объекта Counter есть весьма удобный метод most_common, предназначенный именно для этой цели:

```
>>> bag_of_words.most_common(4)
[('the', 4), (',', 3), ('faster', 3), ('harry', 2)]
```

По умолчанию функция most_common() сортирует все токены в порядке убывания частотности, но мы ограничились только первыми четырьмя из них

Количество вхождений слова в заданном документе называется *частотностью терма* (term frequency, TF). Иногда можно встретить нормализованные путем деления на общее число термов в документе количества слов[1].

Итак, наши четыре чаще всего встречающихся терма (токена): *the*, *«,»*, *harry* и *faster*. Впрочем, *the* и запятая не несут много информации об основной мысли этого документа. Скорее всего, эти неинформативные токены будут встречаться вам еще много раз. Поэтому в данном случае их лучше игнорировать, как и целый список стандартных английских стоп-слов и знаков препинания. Конечно, это не всегда имеет смысл делать, но такое решение позволит пока упростить наш пример. Это сокращает список самых употребляемых токенов в векторе TF (мультимножестве слов) до *harry* и *faster*.

Подсчитаем частотность терма `harry` из описанного выше объекта `Counter` (`bag_of_words`):

```
>>> times_harry_appears = bag_of_words['harry']
>>> num_unique_words = len(bag_of_words)
>>> tf = times_harry_appears / num_unique_words
>>> round(tf, 4)
0.1818
```

Количество уникальных токенов в исходном документе

Притормозим и обратим внимание на нормализованную частоту термов — фразу (и сопутствующие ей вычисления), которая встречается на протяжении всей книги. Под этим термином понимается количество слов относительно длины документа. Зачем вообще делить на длину документа? Представим, что *dog* встречается три раза в документе A и 100 раз в документе B. Это слово явно играет куда более важную роль во втором случае. Но если документ A — письмо ветеринару из 30 слов, а B — «Война и мир» Л. Н. Толстого (приблизительно 580 тыс. слов!)? Тогда наши изначальные выводы меняются на диаметрально противоположные. Следующие уравнения учитывают длину документа:

$$TF(\text{dog}, document_A) = 3 / 30 = 0{,}1.$$

$$TF(\text{dog}, document_B) = 100 / 580\,000 = 0{,}00017.$$

Теперь у вас есть что-то, что позволяет понять разницу между двумя документами, их связь с *dog* и друг с другом. Таким образом, вместо чистых количеств слов можно использовать для описания документов из корпуса нормализованные частотности термов. Аналогичным образом можно вычислить этот показатель для каждого из слов и узнать относительную важность данного терма для каждого документа. Наш главный герой, Гарри, и его жажда скорости, несомненно, являются центром описанной истории. Мы добились значительных успехов в превращении текста в числа, намного выходящих за пределы фиксации наличия/отсутствия заданного

[1] Впрочем, нормализованная частотность, по существу, представляет собой вероятность, поэтому, возможно, не стоит называть ее частотностью.

слова в тексте. Очевидно, что это довольно «притянутый за уши» пример, но скоро мы увидим, насколько значимые результаты могут быть получены при таком подходе. Рассмотрим больший фрагмент текста. Возьмем несколько первых абзацев из статьи «Википедии» о воздушных змеях *(kites)*:

> *A kite is traditionally a tethered heavier-than-air craft with wing surfaces that react against the air to create lift and drag. A kite consists of wings, tethers, and anchors. Kites often have a bridle to guide the face of the kite at the correct angle so the wind can lift it. A kite's wing also may be so designed so a bridle is not needed; when kiting a sailplane for launch, the tether meets the wing at a single point. A kite may have fixed or moving anchors. Untraditionally in technical kiting, a kite consists of tether-set-coupled wing sets; even in technical kiting, though, a wing in the system is still often called the kite.*
>
> *The lift that sustains the kite in flight is generated when air flows around the kite's surface, producing low pressure above and high pressure below the wings. The interaction with the wind also generates horizontal drag along the direction of the wind. The resultant force vector from the lift and drag force components is opposed by the tension of one or more of the lines or tethers to which the kite is attached. The anchor point of the kite line may be static or moving (such as the towing of a kite by a running person, boat, free-falling anchors as in paragliders and fugitive parakites or vehicle).*
>
> *The same principles of fluid flow apply in liquids and kites are also used under water.*
>
> *A hybrid tethered craft comprising both a lighter-than-air balloon as well as a kite lifting surface is called a kytoon.*
>
> *Kites have a long and varied history and many different types are flown individually and at festivals worldwide. Kites may be flown for recreation, art or other practical uses. Sport kites can be flown in aerial ballet, sometimes as part of a competition. Power kites are multi-line steerable kites designed to generate large forces which can be used to power activities such as kite surfing, kite landboarding, kite fishing, kite buggying and a new trend snow kiting. Even Man-lifting kites have been made.*
>
> *«Википедия»*

Присвоим этот текст переменной:

```
>>> from collections import Counter
>>> from nltk.tokenize import TreebankWordTokenizer
>>> tokenizer = TreebankWordTokenizer()
>>> from nlpia.data.loaders import kite_text
>>> tokens = tokenizer.tokenize(kite_text.lower())
>>> token_counts = Counter(tokens)
>>> token_counts
Counter({'the': 26, 'a': 20, 'kite': 16, ',': 15, ...})
```

Kite_text = A kite is traditionally …, все как в примере выше

ПРИМЕЧАНИЕ

> TreebankWordTokenizer возвращает kite. (с точкой) в качестве токена. Treebank Tokenizer предполагает, что документ уже был сегментирован на отдельные предложения, поэтому он игнорирует пунктуацию только в конце строки. Сегментация предложений — очень сложный вопрос, и мы будем обсуждать ее только в главе 11. Тем не менее синтаксический анализатор spaCy работает быстрее и точнее, чем Treebank, ввиду того, что он выполняет сегментацию и токенизацию предложений (наряду со многими другими действиями)[1] за один проход. Так что лучше применять для реальных приложений spaCy, а не использовавшиеся для простых примеров выше компоненты NLTK.

Ладно, вернемся к примеру. Вы уже заметили огромное количество стоп-слов? Наверняка эта статья из «Википедии» не об артиклях *the*, *a*, союзе *and* и т. д. Их пока отбросим:

```
>>> import nltk
>>> nltk.download('stopwords', quiet=True)
True
>>> stopwords = nltk.corpus.stopwords.words('english')
>>> tokens = [x for x in tokens if x not in stopwords]
>>> kite_counts = Counter(tokens)
>>> kite_counts
Counter({'kite': 16,
         'traditionally': 1,
         'tethered': 2,
         'heavier-than-air': 1,
         'craft': 2,
         'wing': 5,
         'surfaces': 1,
         'react': 1,
         'air': 2,
         ...,
         'made': 1})}
```

Определенные выводы о содержании документа можно сделать, исходя даже из одной информации о частотностях слов в нем. Слова *kite(s)*, *wing* и *lift* обладают большой важностью. Даже если вы не имеете понятия о содержании этого документа и просто наткнулись на него в своей обширной базе данных (масштабов Google), то смогли бы «программным образом» сделать вывод, что он как-то связан с полетами или подъемами в воздух или же с воздушными змеями.

Если рассматривать несколько документов в корпусе, то все становится немного интереснее. Набор документов может, например, быть *целиком* посвящен воздушным змеям. Логично предположить, что во всех документах из него упоминается веревка *(string)* и ветер *(wind)*, а частотность термов `TF("string")` и `TF("wind")` будет высокой во всех документах. Теперь рассмотрим способ более изящного представления этих чисел в математических целях.

---

[1] См. веб-страницу spaCy 101: Everything you need to know по адресу spacy.io/usage/spacy-101#annotations-token.

3.2. Векторизация

Мы уже сталкивались с простейшими преобразованиями текста в числа. Но мы просто сохранили числа в словаре, сделав первый шаг из мира текста в мир математики. Теперь мы отправимся дальше по этому пути. Вместо описания документа в терминах частотного словаря мы сформируем из этих количеств слов вектор. В Python он будет представлять собой список, но в общем случае он может быть упорядоченной коллекцией или массивом. Это можно проделать быстро с помощью следующего кода:

```
>>> document_vector = []
>>> doc_length = len(tokens)
>>> for key, value in kite_counts.most_common():
...     document_vector.append(value / doc_length)
>>> document_vector
[0.07207207207207207,
 0.06756756756756757,
 0.036036036036036036,
 ...,
 0.0045045045045045045]
```

Над этим списком (вектором) можно уже непосредственно производить математические операции.

ПРИМЕЧАНИЕ

Существует много способов ускорить обработку подобных структур данных[1].

Применять математику лишь к одному элементу не очень интересно. Одного вектора для одного документа недостаточно. Лучше взять еще парочку документов и создать для них векторы. Но содержащиеся во всех векторах значения должны относиться к общей точке отсчета. Для проведения с ними вычислений они должны отражать точку в пространстве относительно единого начала координат. Вектору нужны общая точка отсчета и одинаковые масштабы («единицы измерения») в каждом из их измерений. Первый этап данного процесса — нормализация количеств слов путем подсчета нормализованных частотностей термов вместо простых количеств вхождений слов в документе (как мы делали в предыдущем разделе). Второй — приведение всех векторов к единой длине (размеру).

Кроме того, значение каждого элемента вектора должно отражать одно и то же слово в векторах для всех документов. Но наше письмо ветеринару вряд ли будет включать столько же слов, как «Война и мир» (а может, и будет, кто знает?). Не переживайте, если некоторые из векторов будут содержать нулевые значения на некоторых позициях. Находим все уникальные слова в каждом из наших двух документов, а затем — все уникальные слова в объединении этих двух множеств. Такие наборы слов часто называются *лексиконом*, что отражает уже встречавшееся нам в предыдущих главах понятие, только в терминах конкретного корпуса. Посмотрим, как это работает с документами немного меньшего размера, нежели *«Война*

[1] См. веб-страницу NumPy по адресу www.numpy.org.

и мир». Вернемся к нашему Гарри. У вас уже есть один «документ» о нем, увеличим наш корпус еще на парочку:

```
>>> docs = ["The faster Harry got to the store, the faster and faster Harry
⇒ would get home."]
>>> docs.append("Harry is hairy and faster than Jill.")
>>> docs.append("Jill is not as hairy as Harry.")
```

ПРИМЕЧАНИЕ

Для удобства, чтобы не набирать эти тексты вручную, вы можете импортировать их из пакета nlpia: from nlpia.data.loaders import harry_docs as docs.

Взглянем на наш лексикон для корпуса из трех документов:

```
>>> doc_tokens = []
>>> for doc in docs:
...     doc_tokens += [sorted(tokenizer.tokenize(doc.lower()))]
>>> len(doc_tokens[0])
17
>>> all_doc_tokens = sum(doc_tokens, [])
>>> len(all_doc_tokens)
33
>>> lexicon = sorted(set(all_doc_tokens))
>>> len(lexicon)
18
>>> lexicon
[',',
 '.',
 'and',
 'as',
 'faster',
 'get',
 'got',
 'hairy',
 'harry',
 'home',
 'is',
 'jill',
 'not',
 'store',
 'than',
 'the',
 'to',
 'would']
```

Каждый из трех векторов документов должен содержать 18 значений, даже если в соответствующем документе содержатся не все 18 слов из лексикона. Каждому токену выделяется место в векторах в соответствии с его позицией в лексиконе. Некоторые из этих количеств токенов будут равны нулю, что нам, собственно, и нужно:

```
>>> from collections import OrderedDict
>>> zero_vector = OrderedDict((token, 0) for token in lexicon)
>>> zero_vector
```

```
OrderedDict([(',', 0),
             ('.', 0),
             ('and', 0),
             ('as', 0),
             ('faster', 0),
             ('get', 0),
             ('got', 0),
             ('hairy', 0),
             ('harry', 0),
             ('home', 0),
             ('is', 0),
             ('jill', 0),
             ('not', 0),
             ('store', 0),
             ('than', 0),
             ('the', 0),
             ('to', 0),
             ('would', 0)])
```

Далее мы копируем базовый вектор, обновляем его значения для каждого документа и сохраняем в массиве:

Функция copy.copy() создает независимую копию, отдельный экземпляр нашего нулевого вектора, а не переиспользует ссылку (указатель) на местоположение в памяти исходного объекта. В противном случае мы бы перезаписывали один и тот же объект zero_vector новыми значениями на каждой итерации цикла и не начинали бы с «чистой доски» на каждом проходе

```
>>> import copy
>>> doc_vectors = []
>>> for doc in docs:
...     vec = copy.copy(zero_vector)
...     tokens = tokenizer.tokenize(doc.lower())
...     token_counts = Counter(tokens)
...     for key, value in token_counts.items():
...         vec[key] = value / len(lexicon)
...     doc_vectors.append(vec)
```

Итак, у нас есть три вектора. По одному на каждый документ. «И что дальше? Что мы можем с ними сделать?» — спросите вы. С векторами, содержащими количества слов, можно делать множество интересных вещей, как и с любыми другими, так что сначала узнаем больше о векторах и векторных пространствах[1].

3.2.1. Векторные пространства

Векторы — краеугольный камень линейной алгебры (векторной алгебры). По своей сути это упорядоченные последовательности чисел (координат) в векторном пространстве. Они описывают место (положение) в данном пространстве. Могут использоваться и для задания конкретного направления и модуля (расстояния) в этом пространстве. *Пространство* — это совокупность всех возможных векторов,

[1] Если вы хотите узнать больше о линейной алгебре и векторах, см. приложение В.

которые могут в нем встречаться. Таким образом, вектор с двумя значениями будет располагаться в двумерном векторном пространстве, вектор с тремя значениями в трехмерном векторном пространстве и т. д.

Кусок миллиметровки или сетка пикселов на изображении — это допустимые двумерные векторные пространства. Нетрудно заметить, как важен порядок этих координат. Если поменять местами координаты x и y для местоположений на миллиметровке без соответствующей корректировки операций над векторами, то все решения задач линейной алгебры окажутся зеркально отраженными. Миллиметровка и сетка пикселов изображения — примеры евклидовых пространств, потому что оси координат x и y перпендикулярны друг другу. Обсуждаемые в этой главе векторные пространства являются евклидовыми пространствами[1].

А как насчет широты и долготы на карте или глобусе? Карта и глобус — определенно двумерные векторные пространства, ввиду того что представляют собой упорядоченный список двух чисел: широты и долготы. Каждая из пар широта — долгота описывает точку на приближенно сферической бугристой поверхности Земли. Вдобавок угол между осями координат широты и долготы не равен в точности 90°, поэтому векторное пространство широты и долготы не является линейным. Это означает, что вы должны быть осторожны, когда вычисляете такие вещи, как расстояние/близость (сходство) между двумя точками, представленными парой 2D-векторов широты и долготы или векторов в любом неевклидовом пространстве. Представьте, например, вычисление расстояния между координатами широты и долготы Портленда, штат Орегон, и Нью-Йорка, штат Нью-Йорк[2].

На рис. 3.1 показан один из способов изображения 2D-векторов с координатами (5, 5), (3, 2) и (–1, 1). «Вершина» вектора (острый конец стрелки) указывает на местоположение в векторном пространстве. Поэтому вершины векторов на этом графике находятся в точках, соответствующих указанным трем парам координат. Хвост вектора («задняя» часть стрелки) всегда в начале координат (0, 0).

А как насчет трехмерных векторных пространств? Положения и скорости в трехмерном физическом мире, в котором мы с вами живем, можно отразить с помощью координат x, y и z трехмерного вектора. Или рассмотреть криволинейное пространство из троек «широта — долгота — высота», описывающих местоположения объектов вблизи поверхности Земли.

К счастью, мы не ограничены обычным 3D-пространством. Наше пространство может быть 5-, 10- или 5000-мерным! Линейная алгебра во всех случаях работает одинаково. Но по мере роста размерности пространства могут потребоваться

---

[1] Авторы используют термин rectilinear space в качестве синонима евклидова пространства. В русскоязычной литературе понятия прямолинейного пространства не выделяют, а rectilinear в серьезных математических словарях переводят как «линейный». Но линейное (векторное) пространство — более общее понятие, чем евклидово, и подобный перевод был бы просто некорректен. — *Примеч. пер.*

[2] Для обеспечения правильности вычислений понадобится что-то вроде пакета GeoPy (geopy.readthedocs.io).

большие вычислительные мощности. Возможны также определенные проблемы проклятия размерности, но этот вопрос мы отложим до последней главы[1].

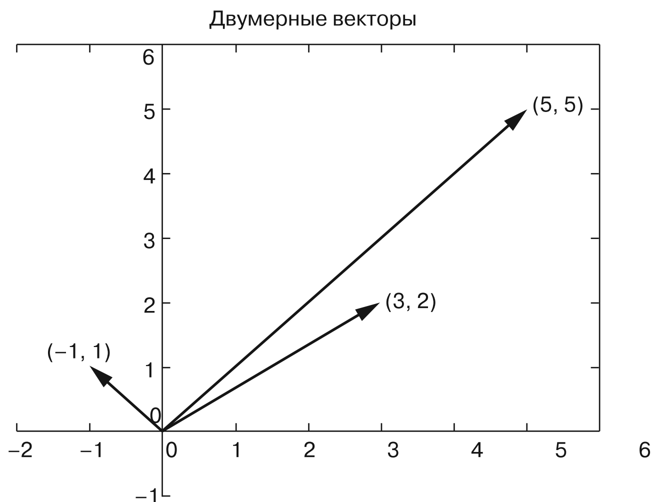

Рис. 3.1. Двумерные векторы

Для векторного пространства документов на естественном языке размерность векторного пространства равна числу различных слов во всем корпусе. Для TF (и далее и TF-IDF) мы иногда будем использовать для обозначения этой размерности прописную букву K. Это число отдельных слов также является размером словаря корпуса, поэтому в теоретических статьях его обычно обозначают $|V|$. Любой документ в K-мерном векторном пространстве можно описать с помощью K-мерного вектора. В случае нашего корпуса из трех документов о Гарри и Джилл K = 18. Поскольку наглядно представить себе пространство размерности больше трех непросто, проигнорируем большую их часть и рассмотрим пока что только два, чтобы изобразить его векторы на лежащей перед вами плоской странице. На рис. 3.2 K уменьшено до двух для 2D-представления 18-мерного векторного пространства Гарри и Джилл.

K-мерные векторы аналогичны данным, просто это сложно представить наглядно. Теперь у нас есть векторные представления всех документов в одном пространстве,

[1] Проклятие размерности состоит в экспоненциальном удалении векторов друг от друга в смысле евклидова расстояния по мере увеличения размерности. Множество простых операций становятся нереализуемыми на практике при векторах размерностью выше 10 или 20. Примерами таких операций может быть сортировка большого списка векторов по их удаленности от вектора запроса («эталонного» вектора) (поиск приблизительного ближайшего соседа). Чтобы глубже вникнуть, ознакомьтесь со статьей по адресу ru.wikipedia.org/wiki/Проклятие_размерности, поэкспериментируйте с пакетом annoy языка Python (github.com/spotify/annoy) или поищите в Google Scholar информацию по запросу high dimensional approximate nearest neighbors (scholar.google.com/scholar?q=high+dimensional+approximate+nearest+neighbor).

а значит, появилась возможность их сравнить. Измерить евклидово расстояние между векторами можно путем их вычитания и вычисления длины расстояния между ними, называемого расстоянием в смысле L^2-нормы (метрики L^2). Это расстояние по прямой от места, задаваемого острием (вершиной) одного вектора, до местоположения, соответствующего острию другого вектора. В приложении С, посвященном линейной алгебре, рассказывается, почему этот метод не подходит для векторов количества слов (частотностей термов).

Два вектора считаются подобными, если они имеют одинаковое направление. Их модули (длины) также могут быть равны, означая, что векторы количеств слов (частотностей термов) относятся к документам примерно одинаковой длины. Но важна ли длина документов при оценке подобия векторных представлений слов в документах? Скорее всего, нет.

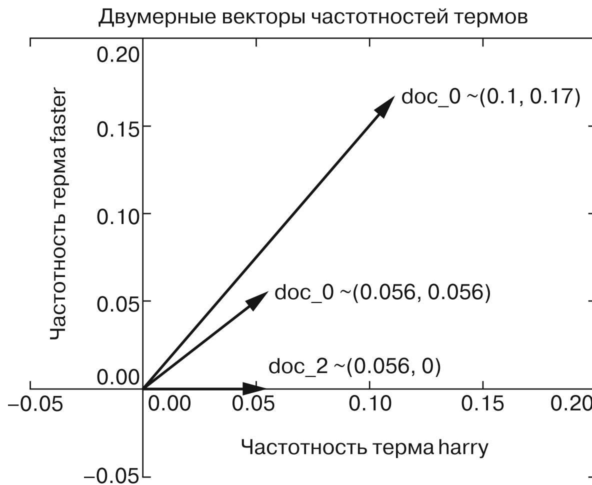

Рис. 3.2. Двумерные векторы частотностей термов

Желательно, чтобы наша оценка уровня подобия документов учитывала одни и те же слова аналогичное число раз в одинаковых пропорциях. Благодаря столь точным оценкам можно быть уверенными, что отражаемые ими документы, вероятно, посвящены близким вещам.

Коэффициент Отиаи (коэффициент косинусного подобия) представляет собой просто косинус угла между двумя векторами (theta), показанными на рис. 3.3. Его можно вычислить с помощью евклидова скалярного произведения по формуле:

$$A \cdot B = |A|\,|B| \cdot \cos \Theta.$$

Коэффициент Отиаи удобно вычислять, поскольку скалярное произведение не требует вычисления каких-либо тригонометрических функций. Кроме того, диапазон принимаемых коэффициентом Отиаи значений удобен для большинства задач машинного обучения: от –1 до +1.

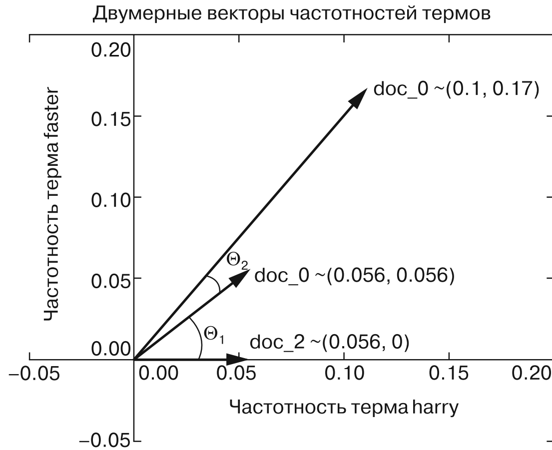

Рис. 3.3. Двумерные theta

На языке Python эта формула выглядит следующим образом:

```
a.dot(b) == np.linalg.norm(a) * np.linalg.norm(b) / np.cos(theta)
```

Решая это уравнение относительно `cos(theta)`, получаем коэффициент Отиаи:

$$\cos\Theta = \frac{A \cdot B}{|A||B|}.$$

Можно сделать это на чистом Python, не используя пакет `numpy`, как в листинге 3.1.

Листинг 3.1. Вычисление косинусного сходства на языке Python

```
>>> import math
>>> def cosine_sim(vec1, vec2):
...     """ Let's convert our dictionaries to lists for easier matching."""
...     vec1 = [val for val in vec1.values()]
...     vec2 = [val for val in vec2.values()]
...
...     dot_prod = 0
...     for i, v in enumerate(vec1):
...         dot_prod += v * vec2[i]
...
...     mag_1 = math.sqrt(sum([x**2 for x in vec1]))
...     mag_2 = math.sqrt(sum([x**2 for x in vec2]))
...
...     return dot_prod / (mag_1 * mag_2)
```

Итак, нужно вычислить скалярное произведение двух векторов — умножить элементы векторов попарно, а затем просуммировать полученные результаты. После этого поделить результат на норму (модуль, то есть длину) каждого вектора. Норма вектора равна евклидову расстоянию от его головы до хвоста — квадратному корню суммы квадратов его элементов. Это *нормализованное скалярное произведение*, как и значение косинуса, располагается между –1 и 1. Оно также является косинусом угла между этими двумя векторами. Данное значение равно доле более длинного

вектора, охваченной перпендикулярной проекцией на него более короткого вектора. Из этого ясно, насколько наши два вектора направлены в одну сторону.

Равный 1 косинусный коэффициент отражает одинаковые нормализованные векторы, указывающие в аналогичном направлении по всем измерениям. Длины (модули) этих векторов могут быть различными, но они указывают в одном и том же направлении. Помните, что мы поделили скалярное произведение на норму каждого вектора, причем это можно делать до или после вычисления скалярного произведения. Таким образом, при его вычислении векторы нормализованы, длины обоих равны 1. Мы можем сделать вывод: чем ближе значение косинусного сходства к 1, тем меньше угол между векторами. В NLP документы, косинусный коэффициент векторов которых близок к 1, содержат похожие слова в одинаковой пропорции. Таким образом, документы, векторы которых близки друг к другу, скорее всего, посвящены одному и тому же.

При косинусном коэффициенте, равном 0, векторы не имеют общих компонентов. Они находятся под углом 90° относительно друг друга по всем измерениям. Для векторов TF NLP такая ситуация возникает только в случае, если в двух документах нет общих слов. Поскольку в них используются совершенно разные слова, они обсуждают абсолютно разные вещи. Это не обязательно означает различный их смысл или тематику, просто в них применяются разные слова.

При значении косинусного коэффициента, равном –1, векторы полностью противоположные. Они указывают в противоположных направлениях. Такого не может произойти ни с простыми векторами количеств слов (частотностей термов), ни даже с нормализованными TF-векторами (которые мы обсудим далее). Векторы количеств слов (частотностей термов) не могут быть отрицательными. Таким образом, последние всегда будут находиться в одном квадранте векторного пространства. Ни один из векторов частотностей термов не может «пробраться» в какой-либо из квадрантов, расположенных «за спиной» других векторов. Ни один компонент какого-либо из TF-векторов не может быть противоположным какому-либо компоненту другого вектора ввиду того, что частотность слова просто не может быть отрицательной.

В этой главе вы не увидите никаких отрицательных значений косинусного сходства. Однако в следующей мы разработаем концепцию слов и тем, противоположных друг другу. В таком случае вы увидите документы, слова и темы, значение косинусного сходства которых меньше 0 или даже равно –1.

ПРОТИВОПОЛОЖНОСТИ ПРИТЯГИВАЮТСЯ

Из нашего способа вычисления косинусного коэффициента следует интересное свойство. Если косинусные коэффициенты двух векторов или документов равны –1 (то есть они являются противоположностями) относительно третьего вектора, они должны быть подобны друг другу — в точности идентичны. Однако документы, которые представляют эти векторы, могут не совпадать. В них не только может быть разный порядок слов, но и один может быть намного длиннее другого, если в нем используются те же слова в той же пропорции.

Позже вы узнаете о векторах, которые намного точнее моделируют документ, а пока вы получили хорошее представление о необходимых инструментах.