Об авторах

Саваш Йылдырым (Savaş Yıldırım) окончил факультет информационных технологий Стамбульского технического университета и имеет степень доктора философии в области обработки естественного языка (NLP). В настоящее время является адъюнкт-профессором Стамбульского университета Билги, Турция, и приглашенным исследователем в Университете Райерсона, Канада. Активный лектор и исследователь с более чем 20-летним опытом преподавания курсов по машинному обучению, глубокому обучению и NLP. Внес значительный вклад в турецкое сообщество NLP, разработав большое количество приложений и ресурсов с открытым исходным кодом. Он также предоставляет комплексные консультации компаниям, занимающимся ИИ, по их проектам, связанным с исследованиями. В свободное время пишет книги и снимает короткометражные фильмы, а также занимается йогой.

Прежде всего я хотел бы поблагодарить мою дорогую подругу Айлин Октай за ее постоянную поддержку и терпение на протяжении длительного процесса написания этой книги. Я также хотел бы поблагодарить своих коллег из факультета информационных технологий Стамбульского университета Билги за их поддержку.

Мейсам Асгари-Ченаглу (Meysam Asgari-Chenaghlu) – менеджер по ИИ в компании Carbon Consulting, а также кандидат наук в Тебризском университете. Он был консультантом ведущих телекоммуникационных и банковских компаний Турции. Работал над различными проектами, включая понимание естественного языка и семантический поиск.

Прежде всего я хотел бы поблагодарить мою любящую и терпеливую жену Наржес Никзад-Хасмахи за ее поддержку и понимание. Я также хотел бы поблагодарить моего отца за его поддержку; пусть его душа упокоится с миром. Большое спасибо Carbon Consulting и моим коллегам.

О рецензенте

Александр Афанасьев – инженер-программист с 14-летним опытом работы в различных отраслях и на разных должностях. В настоящее время Александр является независимым подрядчиком, который реализует идеи в области компьютерного зрения, NLP и создания передовых систем сбора данных в области анализа киберугроз. Раньше Александр уже выступал рецензентом книги *Selenium Testing Cookbook* от издательства Packt. Помимо повседневной работы, он активно участвует в деятельности Stack Overflow и GitHub.

Я хочу поблагодарить авторов этой книги за их усердную работу и представленные в этой книге новаторские идеи; благодарю замечательную команду редакторов и координаторов с их отличными коммуникативными навыками и мою семью, которая всегда поддерживала мои идеи и мою работу.

Оглавление

Предисловие

Мы стали свидетелями больших изменений в *обработке естественного языка* (natural language processing, NLP), которые случились за последние 20 лет. За это время мы испробовали в деле разные подходы и, наконец, вступили в новую эру доминирования удивительной нейросетевой архитектуры трансформеров. Эта архитектура глубокого обучения унаследовала многие методы своих предшественников. В число данных методов входит *контекстное векторное представление слов* (contextual word embedding), *многопоточное самовнимание* (multi-head self-attention), *позиционное кодирование* (positional encodings), *параллелизируемые архитектуры* (parallelizable architectures), *сжатие моделей* (model compression), *перенос обучения* (transfer learning) и *кросс-языковые модели* (cross-lingual model). Архитектура трансформеров начала свое развитие с различных нейронных подходов к NLP, постепенно превратилась в архитектуру кодировщика-декодера на основе механизма внимания и продолжает развиваться в этом направлении по сей день. Сейчас в литературе появляются описания новых успешных вариантов этой архитектуры. Появились отличные модели, которые используют только кодировщик, такие как BERT, или только декодер, такие как GPT.

На протяжении всей книги мы будем возвращаться к упомянутым методам, а работа с трансформерами не составит для нас труда благодаря библиотеке Transformers от сообщества Hugging Face. Мы предложим пошаговые решения широкого спектра задач в области NLP, начиная от обобщения и заканчивая ответами на вопросы. Мы покажем, что с помощью нейросетевых трансформеров можно достичь самых современных результатов.

Для кого эта книга

Эта книга предназначена для исследователей, работающих в области глубокого обучения, практических специалистов по NLP, преподавателей машинного обучения / NLP и студентов, которые хотят начать свой путь в машинное обучение с обработки естественного языка. Знание машинного обучения хотя бы на начальном уровне и хорошие навыки программирования на Python помогут вам извлечь максимальную пользу из этой книги.

Какие темы охватывает эта книга

В *главе 1* дается краткое введение в историю NLP и проводится сравнение традиционных методов и моделей глубокого обучения, таких как CNN, RNN и LSTM, и моделей нейросетевых трансформеров.

В *главе 2* мы более подробно расскажем об использовании трансформеров. Для токенизаторов и таких моделей, как BERT, мы приведем практические примеры.

В *главе 3* вы узнаете о том, как обучать языковые модели с автоэнкодером на любом заданном языке с нуля, включая предварительное обучение и обучение моделей конкретным задачам.

В *главе 4* представлены теоретические основы авторегрессионных языковых моделей и рассказано о том, как предварительно обучить их на собственном корпусе. Вы узнаете, как выполнить предварительное обучение любой языковой модели, такой как GPT-2, на собственном тексте и использовать модель в различных задачах, таких как генерация текста на естественном языке.

В *главе 5* вы узнаете, как настроить предварительно обученную модель для классификации текста и для любой последующей задачи классификации, такой как анализ тональности или многоклассовая классификация.

В *главе 6* рассказывается о тонкой настройке языковых моделей для задач классификации токенов, таких как NER, POS-теги и ответы на вопросы.

В *главе 7* вы узнаете о методах представления текста и о том, как эффективно использовать трансформерную архитектуру, особенно для задач без обучающего набора, таких как кластеризация, семантический поиск и выделение тем текстов.

В *главе 8* показано, как создавать эффективные модели с помощью дистилляции, усечения и дискретизации. Вы узнаете об эффективных разреженных трансформерах, таких как Linformer и BigBird, и о том, как с ними работать.

В *главе 9* вы узнаете о предварительном обучении многоязыковой и межъязыковой модели и различиях между многоязычным и межъязычным предварительным обучением. Кроме того, в этой главе мы рассматриваем модели причинно-следственной связи и языка перевода.

В *главе 10* подробно описано, как выполнять приложения NLP на основе трансформеров в средах, где доступны два вида процессоров – центральный и графический. Здесь также будет описано использование платформы TensorFlow Extended (TFX) для развертывания машинного обучения.

В *главе 11* представлены две различные технические концепции: визуализация механизма внимания и отслеживание эксперимента. Мы испытаем их в действии на примере сложных инструментов, таких как exBERT и BertViz.

Как получить максимальную отдачу от этой книги

Чтобы получить максимальную отдачу от этой книги, читателю необходимо владеть навыками программирования на языке Python, знать основы NLP и глубокого обучения и понимать, как работают глубокие нейронные сети.

> **Важное примечание**
>
> Весь код в этой книге соответствует версии Python 3.6, поскольку некоторые библиотеки для версии Python 3.9 находятся в стадии разработки.

Программное и аппаратное обеспечение	Необходимая операционная система
Transformers	Windows, macOS, Linux
TensorFlow и PyTorch	Windows, macOS, Linux
Python 3.6x	Windows, macOS, Linux
Jupyter Notebook	Windows, macOS, Linux
Google Colaboratory	Windows, macOS, Linux
Docker	Windows, macOS, Linux
Locust.io	Windows, macOS, Linux
Git	Windows, macOS, Linux

Если вы используете цифровую версию этой книги, мы рекомендуем скачать код из репозитория книги на GitHub по приведенной ниже ссылке. Это поможет вам избежать любых потенциальных ошибок, связанных с копированием и вставкой кода.

Скачивание исходного кода примеров

Вы можете скачать файлы примеров кода для этой книги с GitHub по адресу https://github.com/PacktPublishing/Mastering-Transformers. Если есть обновление кода, оно появится в репозитории GitHub.

Видеоролики Code in Action

Видеоролики *Code in Action* для этой книги (на английском языке) можно посмотреть на YouTube по адресу https://bit.ly/3i4vFzJ.

Условные обозначения и соглашения, принятые в книге

В книге используются следующие типографские соглашения.

Курсив – используется для смыслового выделения важных положений, новых терминов, имен команд и утилит, а также слов и предложений на естественном языке.

`Моноширинный шрифт` – применяется для листингов программ, а также в обычном тексте для обозначения имен переменных, функций, типов, объектов, баз данных, переменных среды, операторов, ключевых слов и других программных конструкций и элементов исходного кода.

`Моноширинный полужирный шрифт` – используется для обозначения команд или фрагментов текста, которые пользователь должен ввести дословно без изменений, а также в листингах программ, если необходимо обратить особое внимание на фрагмент кода.

`Моноширинный курсив` – применяется для обозначения в исходном коде или в командах шаблонных меток-заполнителей, которые должны быть заменены соответствующими контексту реальными значениями.

Советы или важные примечания

Представляют собой текст, помещенный в рамку.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Часть I

Последние разработки в области NLP, подготовка рабочей среды и приложение Hello World

В этой части книги вы познакомитесь с архитектурой трансформеров на вводном уровне. Вы создадите свою первую программу `hello-world`, загрузив предоставленные сообществом предварительно обученные языковые модели и запустив соответствующий код с графическим процессором или без него. Но перед этим мы подробно расскажем об установке и использовании библиотек `tensorflow`, `pytorch`, `conda`, `transformers` и `sentenceTransformers`.

Этот раздел состоит из следующих глав:

- главы 1 «От последовательности слов к трансформерам»;
- главы 2 «Знакомство с трансформерами на практике».

1

От последовательности слов к трансформерам

В этой главе мы расскажем о том, что изменилось в *обработке естественного языка* за два десятилетия. За это время мы испробовали в деле разные подходы и, наконец, вступили в новую эру доминирования удивительной нейросетевой архитектуры трансформеров. Все подходы по-своему помогают нам представить слова и документы для решения задач NLP. *Дистрибутивная семантика* (distributional semantics) описывает значение слова или документа при помощи векторного представления, рассматривая дистрибутивные вхождения в коллекции статей. Векторные представления используются для решения многих задач в процессах обработки естественного языка, как связанных, так и не связанных с машинным обучением. В течение многих лет для задач генерации текстов на естественном языке широко использовались языковые модели на основе *n*-грамм. Однако у этих традиционных подходов есть много недостатков, которые мы будем обсуждать на протяжении всей главы.

Далее мы обсудим классические архитектуры *глубокого обучения* (deep learning, DL), такие как *рекуррентные нейронные сети* (recurrent neural network, RNN), *нейронные сети с прямым распространением* (feed-forward neural network, FFNN) и *сверточные нейронные сети* (convolutional neural network, CNN). Благодаря использованию этих архитектур удалось повысить быстродействие приложений в области NLP и преодолеть ограничения традиционных подходов. Однако и у этих моделей есть свои проблемы и недостатки. В последнее время большой интерес вызывают модели-трансформеры, демонстрирующие удивительную эффективность во всех задачах NLP, от классификации до генерации текста. Однако главный успех трансформеров заключается в том, что они значительно повысили быстродействие многоязычных и многопотоковых процессов NLP, а также одноязычных и однопотоковых задач. Благодаря архитектуре трансформеров стало намного проще применять в NLP *перенос*

обучения, назначение которого – сделать модели повторно применяемыми для разных задач или разных языков.

Мы начнем с механизма внимания, а затем кратко обсудим архитектуру трансформеров и различия между предыдущими моделями NLP. Параллельно с теоретическими рассуждениями мы будем демонстрировать практические примеры на основе популярной среды разработки NLP. Для простоты будем использовать как можно более короткие ознакомительные примеры кода.

В этой главе мы рассмотрим следующие темы:

- эволюция подходов NLP в направлении трансформеров;
- понятие дистрибутивной семантики;
- использование глубокого обучения;
- обзорное знакомство с архитектурой трансформеров;
- использование переноса обучения совместно с трансформерами.

Технические требования

Для упражнений по программированию мы будем использовать Jupyter Notebook. Нам потребуется интепретатор Python версии 3.6.0 или новее, а также следующие пакеты, которые необходимо установить с помощью команды `pip install`:

- `sklearn`;
- `nltk` (версия 3.5.0);
- `gensim` (версия 3.8.3);
- `fasttext`;
- `keras` (версия 2.3.0 или новее);
- `Transformers` (версия 4.00 или новее).

Все блокноты Jupyter с упражнениями доступны на GitHub по адресу: https://github.com/PacktPublishing/Advanced-Natural-Language-Processing-with-Transformers/tree/main/CH01.

Для просмотра видеоролика Code in Action перейдите по ссылке: https://bit.ly/2UFPuVd.

Эволюция подходов NLP в направлении трансформеров

Мы стали свидетелями больших изменений в обработке естественного языка, которые случились за последние 20 лет. За это время мы испробовали в деле разные подходы и, наконец, вступили в новую эру доминирования удивительной нейросетевой архитектуры *трансформеров (Transformer)*. Эта архитектура глубокого обучения унаследовала многие методы своих предшественников. Ее эволюция началась с различных методов нейронного моделирования, постепенно перешла к архитектуре энкодера-декодера с механизмом внимания и продолжает развиваться. Архитектура трансформеров и ее вариации достигли успеха благодаря следующим разработкам минувшего десятилетия:

- контекстно-векторное представление слов;
- улучшенные алгоритмы токенизации для обработки невидимых или редких слов;

- использование дополнительных токенов запоминания в предложениях, таких как `Paragraph ID` в `Doc2vec` или токен классификации CLS в языковой модели BERT;
- механизмы внимания, которые устраняют необходимость кодировать всю информацию предложения в один вектор контекста;
- механизм многопоточного самовнимания;
- позиционное кодирование порядка слов;
- параллелизируемые архитектуры, ускоряющие обучение и точную настройку;
- сжатие моделей (дистилляция, дискретизация и т. д.);
- перенос обучения (многоязычное и многозадачное обучение).

В течение многих лет мы использовали традиционные подходы NLP, такие как *языковые модели n-грамм (n-gram language models)*, *модели извлечения информации на основе TF-IDF* (TF-IDF-based information retrieval models) и *матрицы терминов документа с позиционным кодированием (one-hot encoded document-term matrices)*. Все эти подходы внесли большой вклад в решение различных задач NLP, таких как классификация последовательностей, генерация и понимание текстов на естественном языке и т. д.

С другой стороны, у этих традиционных методов NLP есть свои слабые стороны, например неспособность решить проблемы разреженности (sparsity), представления невидимых слов, отслеживания протяженных зависимостей (long-term dependency) и др. Для устранения этих недостатков были разработаны подходы на основе DL, такие как:

- RNN;
- CNN;
- FFNN;
- несколько вариантов, объединяющих RNN, CNN и FFNN.

В 2013 году Word2vec – модель двухслойного кодировщика слов FFNN – решила проблему размерности, создавая короткие и плотные представления слов, которые называются *векторным представлением* (word embedding). Эта модель-предшественник позволяла генерировать быстрые и эффективные статические векторные представления слов. Она преобразовывала исходные текстовые данные в данные машинного обучения (точнее, *самообучения*) путем либо предсказания целевого слова с использованием контекста, либо предсказания соседних слов на основе скользящего окна. Создатели еще одной широко используемой и популярной модели – GloVe – утверждали, что модели, основанные на подсчете, могут работать лучше нейронных моделей. Она использует как глобальную, так и локальную статистику текстового корпуса, чтобы изучать векторы на основе статистики совпадения слов. Эта модель хорошо справляется с некоторыми синтаксическими и семантическими задачами, как показано на рис. 1.1. На рисунке хорошо видно, что смещения векторов определений помогают сформировать векторно-ориентированное восприятие. Мы можем сформировать обобщение гендерных отношений, которое является семантическим отношением смещения между мужчиной и женщиной (*мужчина → женщина*). Затем можем арифметически вычислить вектор термина *актриса*, сложив вектор термина *актер* и вычисленное ранее смещение. Точно

так же мы можем исследовать синтаксические отношения, такие как формы множественного числа слов. Например, если известны векторы слов *актер*, *актеры* и *актриса*, мы можем вычислить вектор множественного числа женского рода (*актрисы*).

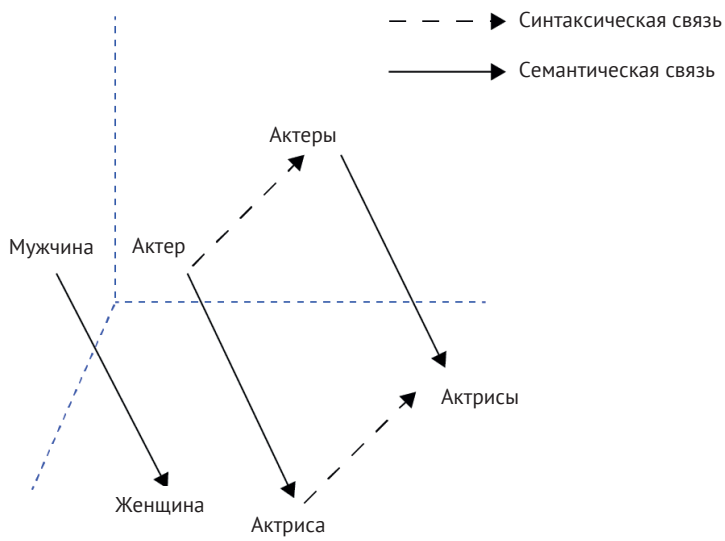

Рис. 1.1. Смещение векторов слов для извлечения отношений

В задачах преобразования последовательности в последовательность (sequence to sequence, seq2seq) в качестве кодировщиков и декодеров стали использовать рекуррентные и сверточные архитектуры, такие как RNN, CNN и с долгой краткосрочной памятью (long short-term memory, LSTM). Основная проблема, с которой столкнулись эти ранние модели, заключалась в многозначности слов. Смысл слов приходилось игнорировать, поскольку каждому слову назначалось одно фиксированное представление, что является особенно серьезной проблемой для многозначных слов и семантики предложений.

Следующим новаторским моделям нейронных сетей, таким как *универсальная языковая модель с тонкой настройкой* (universal language model finetuning, ULMFit) и *векторное представление на основе языковых моделей* (embeddings from language models, ELMo), удалось закодировать информацию на уровне предложений и наконец разрешить (хотя и не полностью) проблему многозначности, с которой не справлялись модели со статичными представлениями слов. Эти два важных подхода основаны на сетях LSTM и реализуют концепции предварительной тренировки и тонкой настройки. Они позволяют нам применять перенос обучения, используя модели, предварительно обученные общей задаче на огромных текстовых наборах данных.

Затем мы можем легко выполнить тонкую настройку, выполнив дообучение предварительно обученной сети на размеченных данных целевой задачи. В данном случае представления слов отличаются от традиционных векторов, поскольку каждое представление слова является функцией всего входного предложения. Современная архитектура трансформеров основана именно на этой идее.

Параллельно с эволюцией представлений слов развивалась идея механизма внимания, которая произвела сильное впечатление в области NLP и привела к значительным успехам, особенно в задачах типа seq2seq. Более ранние методы передавали последнее состояние (известное как *вектор контекста (context vector)*, или *вектор смысла (thought vector*), полученное из всей входной последовательности, в выходную последовательность без связывания или исключения. Механизм внимания способен построить более сложную модель, связав токены, определенные во входной последовательности, с конкретными токенами в выходной последовательности. Например, предположим, что у вас есть ключевая фраза *Government of Canada* (Правительство Канады) в исходном предложении, которое нужно перевести с английского на турецкий. В выходном предложении токен *Kanada Hükümeti* образует сильные связи с исходной фразой и более слабую связь с оставшимися словами в исходном предложении, как показано на рис. 1.2.

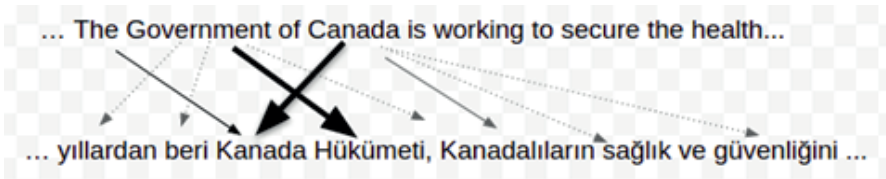

Рис. 1.2. Схематическое представление механизма внимания

Модели, использующие механизм внимания, более успешно решают задачи seq2seq, такие как перевод, ответы на вопросы и резюмирование текста.

В 2017 году была предложена успешная модель кодировщика-декодера на основе нейросети, относящейся к типу трансформеров. Ее архитектура основана на FFNN, но без использования рекуррентности RNN и применяет только механизмы внимания (Vaswani et al., *All you need is attention*, 2017). Модели, основанные на трансформерах, к настоящему времени преодолели многие трудности, с которыми сталкивались другие подходы, и стали новой ключевой парадигмой. На протяжении всей этой книги вы будете изучать, как работают модели на основе трансформеров.

Что такое дистрибутивная семантика?

Дистрибутивная семантика (Distributional semantics) описывает значение слова в виде векторного представления, в первую очередь исследуя характеристики встречаемости, а не его словарные определения. Теория предполагает, что слова, встречающиеся вместе в одной и той же среде, имеют схожие значения. Впервые ее сформулировал ученый Харрис (*Distributional Structure Word*, 1954). Например, слова *собака* и *кошка* чаще всего встречаются в одном и том же контексте. Одним из преимуществ дистрибутивного подхода является возможность исследовать и отслеживать так называемые *лексико-семантические изменения* – семантическую эволюцию слов с течением времени и в разных областях.

Традиционные подходы на протяжении многих лет опирались на языковые модели *неупорядоченных наборов слов* (Bag of Words, BoW) и *n*-граммы для построения представления слов и предложений. В подходе BoW слова и доку-

менты представляются с помощью *прямого унитарного кодирования* (one-hot encoding), которое является разреженным способом представления, также известным как *модель векторного пространства* (Vector Space Model, VSM).

Классификация текста, выявление сходства слов, извлечение семантических отношений, устранение неоднозначности смысла слов – эти и многие другие задачи NLP решали с помощью методов унитарного кодирования в течение многих лет. В свою очередь, модели языка на основе *n*-грамм присваивают вероятности последовательностям слов, чтобы мы могли либо вычислить вероятность того, что последовательность принадлежит корпусу, либо сгенерировать случайную последовательность на основе данного корпуса.

Реализация BoW

BoW – это метод представления документов путем подсчета слов в них. Основная структура данной методики – это матрица документ–термин. Давайте рассмотрим простую реализацию BoW на языке Python. В следующем фрагменте кода показано, как построить матрицу терминов документа с помощью библиотеки Python `sklearn` для демонстрационного корпуса `toy_corpus`, состоящего из трех предложений:

```
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
import pandas as pd
toy_corpus= ["the fat cat sat on the mat",
             "the big cat slept",
             "the dog chased a cat"]
vectorizer=TfidfVectorizer()
corpus_tfidf=vectorizer.fit_transform(toy_corpus)
print(f"The vocabulary size is \
{len(vectorizer.vocabulary_.keys())} ")
print(f"The document-term matrix shape is\
{corpus_tfidf.shape}")
df=pd.DataFrame(np.round(corpus_tfidf.toarray(),2))
df.columns=vectorizer.get_feature_names()
```

Результатом работы кода является матрица документ–термин, представленная на рис. 1.3. Размерность матрицы очень мала (3×10), но в реалистичном сценарии размеры матрицы могут быть довольно большими, например 10 тыс. × 10 млн.

```
The vocabulary size is 10
The document-term matrix shape is (3, 10)
```

	big	cat	chased	dog	fat	mat	on	sat	slept	the
0	0.00	0.25	0.00	0.00	0.42	0.42	0.42	0.42	0.00	0.49
1	0.61	0.36	0.00	0.00	0.00	0.00	0.00	0.00	0.61	0.36
2	0.00	0.36	0.61	0.61	0.00	0.00	0.00	0.00	0.00	0.36

Рис. 1.3. Матрица терминов документа

Данная матрица основана на подсчете, где значения ячеек сформированы в соответствии с весовой схемой «частота терминат – обратная частота документа» (term frequency-inverse document frequency, TF-IDF). Этот подход не учитывает положение слов. Поскольку порядок слов во многом определяет значение фразы, его игнорирование приводит к потере смысла. Это обычная проблема метода BoW, которая наконец решена с помощью механизма рекурсии в RNN и позиционного кодирования в трансформерах.

Каждый столбец в матрице обозначает вектор слова в словаре, а каждая строка обозначает вектор документа. Для вычисления сходства или несходства слов, а также документов могут применяться метрики семантического сходства. В большинстве случаев для улучшения представления документа мы используем биграммы, такие как `cat_sat` и `the_street`. Например, когда параметр `ngram_range=(1,2)` передается в `TfidfVectorizer`, он строит векторное пространство, содержащее как униграммы (`big`, `cat`, `dog`), так и биграммы (`big_cat`, `big_dog`). Такие модели также называют BoW-*n*-граммами (bag-of-grams), потому что они являются естественным продолжением BoW.

Если слово обычно используется в каждом документе, как, например, английский предлог *and*, его называют часто встречающимся, или высокочастотным. И наоборот, некоторые слова почти не встречаются в документах, поэтому их называют низкочастотными (или редкими) словами. Поскольку наличие в тексте высокочастотных и низкочастотных слов может помешать правильной работе модели, в качестве решения применяют TF-IDF, который является одним из наиболее важных и хорошо известных механизмов взвешивания.

Обратная частота документа (inverse document frequency, IDF) – это статистический вес для измерения важности слова в документе. Например, хотя слово *the* (определенный артикль) довольно часто встречается в английском языке, у него очень маленькая различающая способность, зато слово *chased* (гналась) может быть очень информативным и давать подсказки о теме текста. Это связано с тем, что часто используемые слова (стоп-слова, функциональные слова) имеют малую различающую способность при понимании документов.

Различимость терминов также зависит от предметной области – например, список статей про глубокое обучение, скорее всего, будет содержать слово «сеть» почти в каждом документе. IDF может уменьшить веса всех терминов, используя их *частоту документов* (document frequency, DF), которая вычисляется по количеству документов, включающих термин. *Частота термина* (term frequency, TF) – это исходное количество вхождений термина (слова) в документе.

Некоторые преимущества и недостатки модели BoW на основе TF-IDF перечислены в табл. 1.

Таблица 1. Преимущества и недостатки модели TF-IDF BoW

Преимущества	Недостатки
• простота реализации • результаты поддаются толкованию • адаптация к предметной области	• стремительный рост размерности • нет решения для невидимых слов • сложность выявления семантических отношений и синонимов • игнорируется порядок слов • медленно работает с большими словарями

Решение проблемы размерности

Для устранения размерности модели BoW широко используется *латентный семантический анализ* (Latent Semantic Analysis, LSA), позволяющий выявлять семантику в низкоразмерном пространстве. Это линейный метод, который фиксирует попарные корреляции между терминами. Вероятностные методы на основе LSA можно по-прежнему рассматривать как единый слой скрытых тематических переменных. Однако современные модели DL включают в себя несколько скрытых слоев с миллиардами параметров. Кроме того, модели на основе трансформеров показали, что они могут обнаруживать скрытые представления намного лучше, чем такие традиционные модели.

Когда мы решаем задачи *понимания естественного языка* (natural language understanding, NLU), традиционный процесс начинается с подготовительных шагов, таких как *токенизация*, *выделение морфологических основ* (stemming), *обнаружение именных словосочетаний* (noun phrase detection), *разбиение на части* (chunking), *удаление стоп-слов* (stop-word elimination) и многого другого. После этого создается матрица документ–термин на основе какого-либо алгоритма взвешивания (самым популярным остается TF-IDF). Далее эта матрица служит входными табличными данными для процедуры *машинного обучения* (machine learning, ML), анализа эмоциональной окраски, сходства документов, кластеризации документов или оценки степени релевантности между запросом и документом. Аналогичным образом термины, представленные в виде матрицы, можно использовать для задачи классификации токенов, включая распознавание именованных объектов, извлечение семантических отношений и т. д.

Этап классификации включает в себя прямую реализацию таких алгоритмов машинного обучения на размеченных данных, как *машина опорных векторов* (support vector machine, SVM)[1], *случайный лес (random forest)*, *логистика*, *наивный байесовский алгоритм* и *множественное обучение* (ускорение, или бэггинг). На практике реализация этой последовательности выглядит приблизительно как в следующем фрагменте кода:

```
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
labels= [0,1,0]
clf = SVC()
clf.fit(df.to_numpy(), labels)
```

Как видно на примере этого кода, мы можем с легкостью использовать алгоритм подгонки модели (операцию `fit`) благодаря API `sklearn`. Чтобы применить обученную модель к обучающим данным, достаточно выполнить следующий код:

```
clf.predict(df.to_numpy())
Output: array([0, 1, 0])
```

Итак, переходим к следующему разделу!

[1] Категория нейросетей прямого распространения. – *Прим. перев.*