Введение в представление текста

В НЛП мы обычно преобразовываем входной текст, слово или символ в числовой формат, чтобы применять к ним различные математические операции (сравнивать числа, находить закономерности в числовом вводе и т.д.).

Самый удобный способ — **предоставить вектор** для каждой текстовой единицы. Под вектором мы подразумеваем упорядоченную последовательность чисел. В НЛП мы можем использовать их для описания любого элемента, будь то одно слово или целый текст. То, что кодирует вектор, зависит исключительно от нашей цели и подхода. Этот процесс называется **встраиванием (или текстовым представлением)**: это может быть встраивание документа или слова.

Простые модели для встраивания документов

Начнем с представления **«мешок слов» BOW**, которое проще всего реализовать. При таком представлении мы рассматриваем **каждое слово независимо**, без учета окружающего контекста. Мы описываем текст как последовательность всех содержащихся в нем слов, но не сохраняем первоначальный порядок и место. Полученный вектор кодирует текст и хранит информацию о встречаемости в нем слов.

Модель часто используется при классификации документов и поиске информации, но также находит применение во многих других задачах НЛП.

Давайте посмотрим на пример. У нас есть три отзыва, каждый из которых состоит из одного предложения:

```
Обзор I: easily the best album of the year.
Обзор II: the album is amazing.
Обзор III: loved the clean production!
```

Во-первых, нам нужно разработать словарь и перечислить все известные слова в данных. Если пренебречь знаками препинания, это может выглядеть следующим образом:

`"easily", "the", "best", "album", "of", "year", "is", "amazing", "loved", "clean", "production"` .

Длина вектора должна равняться длине словаря, поэтому здесь будет **11**.

Следующим шагом является подсчет всех вхождений этих слов в каждом отзыве. Во-первых, **«the»** появляется дважды в первом обзоре, поэтому мы вставляем **2** в соответствующую ячейку напротив названия документа. Мы получаем следующие представления, в которых число в векторе представляет количество связанного слова:

```
review_1 = [1, 2, 1, 1, 1, 1, 0, 0, 0, 0, 0]
review_2 = [0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0]
```

```
review_3 = [0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1]
```

Могут быть разные способы подсчета очков. Вы можете просто указать, встречается ли слово в документе или нет. Это приводит к двоичным векторам с 0 для каждого отсутствующего слова и 1 для каждого существующего слова. Теперь наше представление немного изменится:

```
review_1 = [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
review_2 = [0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0]
review_3 = [0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1]
```

Мы создаем бинарные векторы, когда нас больше заботит наличие слов, а не их необработанное количество. Простейший анализ настроений является примером задачи, в которой мы можем применить это представление.

Иногда, когда мы оцениваем вхождения в модели мешка слов, мы получаем часто встречающиеся слова, которые не важны для рассматриваемого нами документа. Тем не менее, могут быть слова с меньшими баллами, которые имеют отношение к теме нашего текста, но пакет слов их не учитывает. Для этого у нас есть `TF-IDF`, чтобы помочь нам увеличить пропорциональность и выделить те слова, которые содержат полезную информацию.

TF-IDF (**TF** — term frequency, **IDF** — inverse document frequency, Подробнее посмотрите по ссылке) учитывается не только в конкретном документе, но и во всей текстовой коллекции. Некоторые слова встречаются в каждом документе (например, "do", "say", "have"), и их оценки могут быть высокими даже для одного текста. **Идея TF-IDF** состоит в том, чтобы изменить масштаб подсчета, чтобы устранить такое смещение. Это приводит к взвешенному представлению в виде набора слов.

Для получения **TF-IDF** необходимо рассчитать две метрики:

- **Частота термина** фиксирует частоту слова в одном выбранном документе.
- **Обратная частота документа** измеряет редкость слова во всем наборе документов. Чем чаще встречается слово, тем ближе его вес к 0.

Основание логарифма может быть разным, но чаще всего используется натуральный логарифм. **Оценка TF-IDF** является произведением этих двух показателей.

Простые модели встраивания слов

Мы используем `BOW` и `TF-IDF` для представления большого текста (документа, романа, статьи). Для встраивания токенов предпочтительно использовать `Word2Vec` или `GloVe`.

В чем основное различие между этими двумя типами моделей? **Первый**, встраивание документов, создает один вектор, где каждое слово или **n-грамма** представляет собой только одну цифру (в векторе много цифр). **Второе**, встраивание слов, генерирует вектор, где слово представлено целым вектором, а не только одной цифрой в векторе.

Вложения слов позволяют нам фиксировать семантические и синтаксические отношения. Например, у **«lamp» (лампа)** и **«lantern» (фонарь)** будут очень похожие векторы, а у **«lantern» (фонарь)** и **«can» (банка)** будут разные представления.

Мы не можем просто подсчитать значения вручную, как мы это сделали с **мешком слов** и **TF-IDF**. Вместо этого веса должны быть получены во время обучения на колоссальных наборах текстов. В качестве альтернативы мы можем использовать предварительно обученные встраивания, предоставляемые проектами GloVe и fastText. Длина вектора может быть задана как параметр модели.

Основная идея **Word2Vec** заключается в том, что значение слова сильно зависит от контекста, поэтому **Word2Vec** аппроксимирует смысл слова через векторы его окружения, так что мы можем отметить синонимы и антонимы, а также другие типы языковых явлений и отношений (для во-первых, известным примером является то, что «man» относится к «king», как «woman» относится к «queen»).

Word2Vec имеет две модели создания векторов:

- Непрерывный bag-of-words
- Непрерывный skip-gram

Word2Vec вычисляет вероятности слов в корпусе и использует их для создания вложений. В результате все слова из словаря получают свои векторные представления. Подобные слова (то есть слова, встречающиеся в сходных контекстах) будут иметь схожие векторы.

Другая модель встраивания слов, **GloVe**, во многом похожа на Word2Vec. Центральная концепция GloVe заключается в простом наблюдении, что соотношения вероятностей совместного появления слов имеют потенциал для кодирования той или иной формы значения.

Возьмем, к примеру, вероятности совпадения целевых слов «ice» и «steam » с различными контрольными словами из словаря. Вот некоторые фактические вероятности из корпуса из 6 миллиардов слов (взято с официальной страницы GloVe):

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

ice чаще встречается с **solid**, чем с **gas**, а **steam** чаще встречается с **gas**, чем с **solid**. Оба слова часто встречаются вместе с их общим свойством **water**, и оба нечасто встречаются вместе с несвязанным словом **fashion**. Основной задачей GloVe является изучение векторов слов, скалярное произведение которых равно логарифму вероятности совпадения слов.

Для получения дополнительной информации обратитесь к официальной странице GloVe.

Другой подход — **быстрое кодирование (one-hot encoding)**. Все начинается с индексации всех слов в предложении. Например, у нас есть следующий текст:

> Tomorrow I have to go to school.

Индексация слов будет выглядеть так: | Tomorrow | I | have | to | go | school | | ---------|---|------|----|----|--------|| 0 | 1 | 2 | 3 | 4 | 5 |

Вы можете заметить, что мы не индексируем одно и то же слово дважды. Итак, слово `tomorrow` находится в позиции 0; его однократный вектор будет выглядеть так: `[1, 0, 0, 0, 0, 0]`

Длина вектора равна количеству слов в словаре. Матрица этого предложения будет следующей:

	0	1	2	3	4	5
Tomorrow	1	0	0	0	0	0
I	0	1	0	0	0	0
have	0	0	1	0	0	0
to	0	0	0	1	0	0
go	0	0	0	0	1	0
to	0	0	0	1	0	0
school	0	0	0	0	0	1

Вы могли заметить, что индекс 3 встречается дважды. Это потому, что соответствующее слово (to) тоже встречается дважды.

Расширенные встраивания документов

Более сложными способами встраивания большого текста являются модель `N-gram`, `doc2vec`, `Universal Sentence Encoder` и многие другие.

Модель N-грамм — полезный инструмент для предсказания следующего слова: эта модель стоит за **T9**, проверкой опечаток и многими другими технологиями. Центральная концепция состоит в том, чтобы присвоить вероятность появления **N-граммы** или вероятность того, что **слово появится следующим** в последовательности слов.

Мы можем сравнивать документы, назначая вероятности появления определенной **n-граммы**. Например, если в одном документе **3-грамма** встречается один раз, а в другом 52 раза, то эти два документа будут иметь разные вероятности того, что эта n-грамма встретится в тексте. Следовательно, можно предположить, что эти два документа совершенно разные. После вычисления вероятностей для множества n-грамм мы получим вектор, где каждая цифра — это число вероятности.

Модель Doc2Vec очень похожа на **Word2Vec**. Она использует Word2Vec в качестве основы. В doc2vec мы используем word2vec для вычисления вектора для каждого

слова. Получаем **вектор n** (количество слов в документе), затем добавляем еще один вектор, отвечающий за идентификацию документа. Вы можете прочитать, как использовать **Doc2Vec в Gensim** в его официальной документации.

Universal Sentence Encoder встраивает каждое предложение, а затем семантически сравнивает предложения. Простая модель встраивает предложения с помощью технологии Word2Vec, и мы получаем векторы предложений. Затем STS Benchmark оценивает, насколько показатели сходства, вычисленные с использованием вложений предложений, согласуются с человеческими суждениями.

Эталонный тест требует, чтобы системы возвращали оценки сходства для разнообразного набора пар предложений. Затем используется корреляция Пирсона для оценки качества машинных оценок сходства с человеческими суждениями. Вы можете узнать больше об использовании USE в Python в статье Universal Sentence Encoder на TensorFlow.

Контекстуализированные встраивания слов

Word2Vec игнорирует ближайший контекст. То же самое относится и к GloVe. Что это значит? Некоторые слова имеют два или более значений, которые могут различаться. У нас всегда будет один вектор, если мы будем предварительно обучать нашу модель, не обращая внимания на контекст. Это вызывает много проблем, особенно если мы анализируем слово с большим количеством значений, например **stick**. То же самое и в GloVe: слово всегда будет иметь вектор `[0,1, 200, 0,3, -4,1, -0,0007, 100, 101, 3]`, будь то глагол или существительное.

Вот почему у нас есть контекстуализированные встраивания слов. Отличным примером контекстуализированной модели являются вложения **ELMo**. Они основаны на двунаправленной модели **LSTM**. ELMo обрабатывает все предложение целиком, прежде чем векторизовать каждое слово внутри него. Модели, которые не обращают внимания на контекст, называются неконтекстуализированными вложениями слов.

Другими контекстуализированными вложениями слов являются **Open AI Transformer**, **Google BERT** и **ULM-Fit**. OpenAI был первой языковой моделью на основе трансформера с тонкой настройкой. Точнее, он использует только декодер преобразователя. Это однонаправленное языковое моделирование.

Представления двунаправленного кодировщика от преобразователя (BERT) — это двунаправленное языковое моделирование на основе маскированного преобразователя. Есть две версии этой модели по умолчанию: большая и базовая. Скачать первую можно на Hugging Face.

Universal Language Model. Тонкая настройка для классификации текста (ULM-Fit) — еще одна языковая модель, предложенная в 2018 году. Сначала ULM-Fit тренирует длинную память (LM) на огромном общем корпусе доменов с двунаправленной длинной памятью. Затем он настраивает LM на данные целевой задачи. В итоге он настраивается как классификатор на целевую задачу.

Подведем итоги

Мы обсудили три типа встраивания: простое и расширенное встраивание документов и простое встраивание слов. В первой группе у нас есть мешок слов и TF-IDF. Узнали про Doc2Vec и Universal Sentence Encoder, а также про концепцию модели N-грамм со сложными встраиваниями документов. И также обсудили Word2Vec и GloVe.