Word2vec

Шапкин Антон
Национальный Исследовательский Университет
Высшая Школа Экономики, Факультет Компьютерных Наук
Москва, Россия
aashapkin_1@edu.hse.ru

***Аннотация* — Данная статья содержит информацию о реализации утилиты word2vec через Skip-gram Model и NegativeSampling, а также рассказывает о преимуществах, недостатках и способах применения данного инструмента.**

Ключевые слова — word2vec; Skip-gram; NegativeSampling;

Введение

В XXI веке, при возрастающем потоке информации, появилась необходимость обработки естественного языка. Это направление математической лингвистики и искусственного интеллекта, изучающее проблемы компьютерного анализа естественного (человеческого) языка. В жизни, мы каждый день сталкиваемся с алгоритмами этой области, скрытыми под красивой оберткой, например в почтовых сервисах, где алгоритмы используют для классификации писем (на классы: спам, социальные сети и т.д.), во время поиска информации (Yandex, Google) и многих других аспектах жизнедеятельности. Данное направление не стоит на месте, так в 2013 году был разработан инструмент word2vec, в результате применения которого мы получаем векторные представления слов. Алгоритмы такого типа лежат в основе многих систем обработки естественного языка, без которых мы уже не можем представить наш мир (Siri, GoogleTranslate) и каждый день разрабатываются все новые и новые программы, использующие их, а утилита word2vec один из самых популярных и эффективных методов представления слов в виде векторов.

Рассмотрим реализацию данного инструмента, а также изучим его преимущества, недостатки и способы применения.

Основная часть

Формализация задачи

Формализуем задачу: по корпусу документов $T = \{D_1, D_2, \ldots, D_n\}$ необходимо построить вектора для всех слов из словаря V. Алгоритмы word2vec опираются на дистрибутивную семантику (т.е. мы знаем слово по контексту). Так, мы устанавливаем окно из q слов, которое скользит по всем документам корпуса и генерирует образцы для обучения модели. Для каждого слова w мы будем иметь множество контекстов (контекстных слов) $C(w) = \{c_1, c_2, \ldots, c_m\}$.

Стоит сразу отметить, что для каждого слова есть 2 представления в виде вектора: embedding и context. Embedding вектор – это представление слова в виде вектора как центрального слова, в то время как context – его представление как контекстного слова.

Согласно [1], в word2vec существует две основных модели обучения: Continuous Bag of Words Model и Skip-gram Model.

Continuous Bag of Words Model

Как сказано в [2], основная идея метода Continuous Bag of Words Model заключается в том, что мы рассматриваем вероятности $p(w|c)$ (слова при условии контекста). Здесь мы рассматриваем как то, что было до слова, так и то, что идет после него. Таким образом, по ~2q меткам (q слов до центрального слова, q слов после него) мы предсказываем центральное слово. Например, в предложении “Петя сказал,

что этот автобус красный”. Если брать q = 2 и рассмотреть случай, когда окно установлено так, что центральным словом является “сказал”, мы получаем в качестве меток ∅, “Петя”, “что”, “этот”, а целевое слово - “сказал”.

Skip-gram Model

Модель Skip-gram Model похожа на CBOW, однако здесь мы предсказываем контекст по слову. Точнее, мы используем каждое текущее слово в качестве входных данных и прогнозируем слова в определенном диапазоне до и после текущего слова. В данном методе перед нами стоит задача найти такие параметры θ для вероятностей $p(c|w;\theta)$, которые максимизируют вероятность корпуса:

$$\arg\max_{\theta} \prod_{w\in V} \prod_{c\in C(w)} p(c|w;\theta).$$

Это верно, поскольку мы заинтересованы в прогнозировании контекста по заданному центральному слову, т.е. мы хотим максимизировать $p(c|w)$ для каждой пары (c, w) которая встречалась в тексте. Вспомним, что сумма вероятностей по всем исходам равна 1. Получается, что мы неявно устремляем $p(c|w) \to 0$ для всех несуществующих пар (c, w).Таким образом, в исходной задаче максимизации мы делаем произведение близким к 1, если наша модель хорошая, и близким к 0, иначе.

Для подсчета данного выражения, необходимо модифицировать его. Прологарифмировав выражение, сможем перейти к максимизации суммы:

$$\arg\max_{\theta} \sum_{(w,c)\in WC} \log p(c|w;\theta) \qquad (*),$$

где WC – множество всех возможных пар (c, w), которые мы извлекли из текста.

Теперь мы перешли к суммам, что значительно приятнее произведений. Однако, как мы определяем $p(c|w;\theta)$? Как сказано в [3], один из подходов для параметризации данной модели – Softmax (функция, преобразующая вектор z размерности k в вектор z_i' той же размерности, где каждая координата полученного вектора представлена вещественным числом от 0 до 1 и $\sum_i z_i' = 1$).

В нашем случае:

$$p(c|w;\theta) = \frac{e^{v_c * v_w}}{\sum_{c\prime\in V} e^{v_{c\prime} * v_w}},$$

где v_c, v_w –векторное представление для слов c и w соответственно, параметры θ – это v_{c_i} и v_{w_i} для $w \in V, c \in V, i = 0, \dots, d$, где d - embedding size (размерность вложения, т.е. размерность векторов, которые мы получим в результате применения инструмента word2vec).

Подставив это в $(*)$, а также воспользовавшись свойством логарифма, получаем:

$$\arg\max_{\theta} \sum_{(w,c)\in WC} \log p(c|w) =$$

$$= \sum_{(w,c)\in WC} \log e^{v_c * v_w} - \log \sum_{c\prime\in V} e^{v_{c\prime} * v_w}$$

Недостатки Skip-gram Model

Несмотря на то, что данная формула может быть вычислена, мы получаем очень дорогую с точки зрения вычислений формулу, потому что вероятности $p(c|w;\theta)$ очень долго считать из-за суммы $\sum_{c'\in V} e^{v_{c'} * v_w}$. Данная проблема имеет несколько решений, одно из которых – hierarchical softmax. Другое решение – NegativeSampling (негативное сэмплирование или отрицательный выбор). Оба метода позволяют вычислять распределение вероятностей быстрее, чем за линейное от размера словаря время. Расскажем про один из них подробнее.

NegativeSampling

Данный метод основан на модели Skip-gram, однако из-за того, что фактически он оптимизирует другую цель, мы получаем численно более выгодный алгоритм.

Рассмотрим пару (w, c). Раньше мы прогнозировали контекст по слову, теперь переключимся на модель, которая берет входное и выходное слово и вычисляет вероятность их соседства.

Как и в [3], обозначим за $p(D = 1|w, c)$ вероятность того, что пара (w, c) встречалась в

корпусе. Тогда, $p(D = 0|w, c) = 1 - p(D = 1|w, c)$ – вероятность того, что пара не встречалась.

Как и в модели Skip-gramm, получаем:

$$\arg\max_{\theta} \prod_{(w,c)\in WC} p(D = 1|w, c;\ \theta) =$$

$$= \arg\max_{\theta} \log \prod_{(w,c)\in WC} p(D = 1|w, c;\ \theta) =$$

$$= \arg\max_{\theta} \sum_{(w,c)\in WC} \log p(D = 1|w, c;\ \theta)$$

Величина $p(D = 1|w, c; \theta)$ может быть определена с помощью Softmax:

$$p(D = 1|w, c; \theta) = \frac{1}{1 + e^{-v_c * v_w}}$$

Однако, данная задача имеет тривиальное решение, а именно: $p(D = 1|w, c) = 1$ для всех пар (w, c).

Для решения данной проблемы, введем в набор данных отрицательные образцы. Более формально, сгенерируем множество WC', состоящее из случайных пар (w, c), для которых известно, что они не встречались в корпусе.

Тогда, получим:

$$\arg\max_{\theta} \prod_{(w,c)\in WC} p(D = 1|w, c;\ \theta) \prod_{(w,c)\in WC'} p(D = 0|w, c;\ \theta)$$

Прологарифмируем и воспользуемся тем, что:

$$p(D = 1|w, c; \theta) = \frac{1}{1 + e^{-v_c * v_w}} \quad \text{и}$$

$$p(D = 0|w, c) = 1 - p(D = 1|w, c).$$

Тогда получим:

$$\arg\max_{\theta} \sum_{(w,c)\in WC} \log \frac{1}{1 + e^{-v_c * v_w}} \sum_{(w,c)\in WC'} \log\left(1 - \frac{1}{1 + e^{-v_c * v_w}}\right)$$

Что в свою очередь равно:

$$\arg\max_{\theta} \sum_{(w,c)\in WC} \log \frac{1}{1 + e^{-v_c * v_w}} \sum_{(w,c)\in WC'} \log \frac{1}{1 + e^{v_c * v_w}}$$

Такой подход обеспечивает отличный компромисс между производительностью и статистической эффективностью.

Word2vec

Теперь, когда мы понимаем основные идеи skip-gram и NegativeSampling, есть возможность перейти к рассмотрению процесса обучения word2vec.

Определим размер словаря V_{size}, а также размерность вложения E_{size}.

Согласно [4], в начале обучения создаём две матрицы: Embeddings (для представления слов как центральных) и Contexts (для представления слов как контекстных), инициализируя их случайными значениями. В этих матрицах хранятся вложения для каждого слова в нашем словаре, поэтому (V_{size}, E_{size}) – их размер.

Далее инициируем обучение модели. На каждом шаге мы рассматриваем какой-либо положительный и связанные с ним отрицательные примеры. После этого, мы улучшаем векторные представления данных слов. На первом этапе, данное действие производится последовательно по всем документам. Затем, повторяем данные этапы некоторое количество раз. Наконец, после завершения процесса, мы откладываем матрицу Contexts и используем обученную матрицу Embeddings для последующих задач.

В процессе обучения word2vec участвуют два ключевых гиперпараметра: размер окна и количество отрицательных образцов.

Особенности и преимущества

Согласно [5], модели, натренированные на очень большом количестве текстов, показывают удивительные результаты: алгебраические операции на векторах отображают семантические операции. Самый известный пример — формула «король − мужчина + женщина», в результате применения которой мы получим вектор, наиболее схожий с вектором слова королева. Аналогично, вектору Чехия + валюта сопоставляется вектор слова крона.

Еще одним преимуществом является то, что данная утилита не требует большой вычислительной мощности для процесса обучения. Кроме того, инструмент не нуждается в предобработки текстов.

Недостатки word2vec

Однако, несмотря на все преимущества, имеются недостатки. Как сказано в [6], данный инструмент хорошо работает на словах и словосочетаниях, но показывает плохие результаты на длинных текстах.

Еще один минус word2vec – с его помощью не могут быть представлены слова, которые не встречались в выборке.

Способы применения

Несмотря на недостатки, Word2vec активно участвует в нашей повседневной жизни. Он используется для вычисления семантической близости (поиск синонимов), машинного перевода (находится линейное отображение одного языка в другой, например, бегать отображается в run, и затем применяется word2vec), а также кластеризации и классификации текстов.

Полезная информация

Кроме того, компания Google предоставила в открытый доступ модель [7], обученную на огромном количестве данных (примерно 100 миллиардов слов в сумме во всех документах). Она содержит 300-мерные представления 3 миллионов слов, что сокращает время обучения модели до 0. Недостаток данной модели для пользователей большинства стран – в ней только англоязычные слова.

Заключение

В заключении хотелось бы сказать, что метод word2vec, разработанный в 2013 году, стал значительным прорывом в своей области. Конечно, и до этого существовали методы вложений, однако их результаты не были так хороши. Я знаком с некоторыми базовыми средствами обработки естественного языка (TF-IDF, Bag Of Words), однако на практике данные методы показывают не столь удовлетворительный результат, а также требуют предварительной обработки текста. Word2vec в свою очередь обходится без этого, а также не требует больших вычислительных мощностей. По моему мнению, данный алгоритм имеет большой потенциал и может быть использован (и уже применен) для создания многих полезных ресурсов.

Список источников

[1] Страница на neerc *Векторное представление слов*

[2] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean – *Efficient Estimation of Word Representation in Vector Space*

[3] Yoav Goldberg and Omer Levy *word2vec Explained: Deriving Mikolov's et al.'s Negative-Sampling Word-Embedding Method*

[4] Статья на habr - *Word2vec в картинках*

[5] *Видео от академии Яндекса*

[6] Статья на habr – *Word2vec классификация текстовых документов*

[7] Сервис - *Google code*