Математические методы анализа текстов

Векторные представления слов. word2vec

Мурат Апишев

10 апреля, 2017

Векторные представления слов

car	**new york**
cars	new york city
muscle car	brooklyn
sports car	long island
compact car	syracuse
autocar	manhattan
automobile	washington
pickup truck	bronx
racing car	yonkers
passenger car	poughkeepsie
dealership	new york state

Векторное представление слова *(word embedding)* — вещественный вектор в пространстве с фиксированной невысокой размерностью.

Вход — коллекция текстов.

Выход — векторные представления слов из словаря коллекции.

Векторные представления слов (2D t-SNE)

Deep learning, Yann LeCun, Yoshua Bengio & Geoffrey Hinton, Nature 521, 436–444 (28 May 2015) | doi:10.1038/nature14539

Зачем нужны word embeddings

Сжатые векторные представления слов

1. полезны сами по себе, например, для поиска синонимов или опечаток в поисковых запросах.
2. используются в качестве признаков для решения самых различных задач:
   - выявление именованных сущностей
   - тэгирование частей речи
   - машинный перевод
   - кластеризация документов
   - ранжирование документов
   - анализ тональности текста

Сравнение с One-hot encoding

Оба подхода представляют собой способ превращения слова в вектор признаков. **Но:**

Размерность:

- ✗ Признаковое пространство в one-hot векторах имеет размерность, равную мощности словаря коллекции, т.е. тысячи и десятки тысяч. Эта размерность растёт вместе с ростом словаря.
- ✓ Сжатые векторные представления строятся в пространствах фиксированной размерности порядка десятков и сотен.

Сравнение с One-hot encoding

Оба подхода представляют собой способ превращения слова в вектор признаков. **Но:**

Семантическая близость:

- ✗ Никак не учитывает семантическую близость слов, все векторы одинаково далеки друг от друга в признаковом пространстве.
- ✓ Сжатые векторные представления для семантически близких слов близки как векторы (например, по косинусной мере). Это позволяет работать со словами, которых раньше не было в корпусе.

Как делали до word2vec

1. По корпусу текстов D со словарём T строим матрицу *со-встречаемостей* $X_{|T|\times|T|}$.

   Возможны различные варианты учёта со-встречаемости слов:

   - сумма по всей коллекции числа попаданий пары слов в окно фиксированного размера;
   - количество документов, хоть раз содержащих пару слов;
   - количество документов, хоть раз содержащих пару слов в окне.

   Можно понижать размерность:

2. SVD-разложение: $X = USV^T$.

3. Из столбцов матрицы U выбираются первые K компонент.

Почему это плохой метод

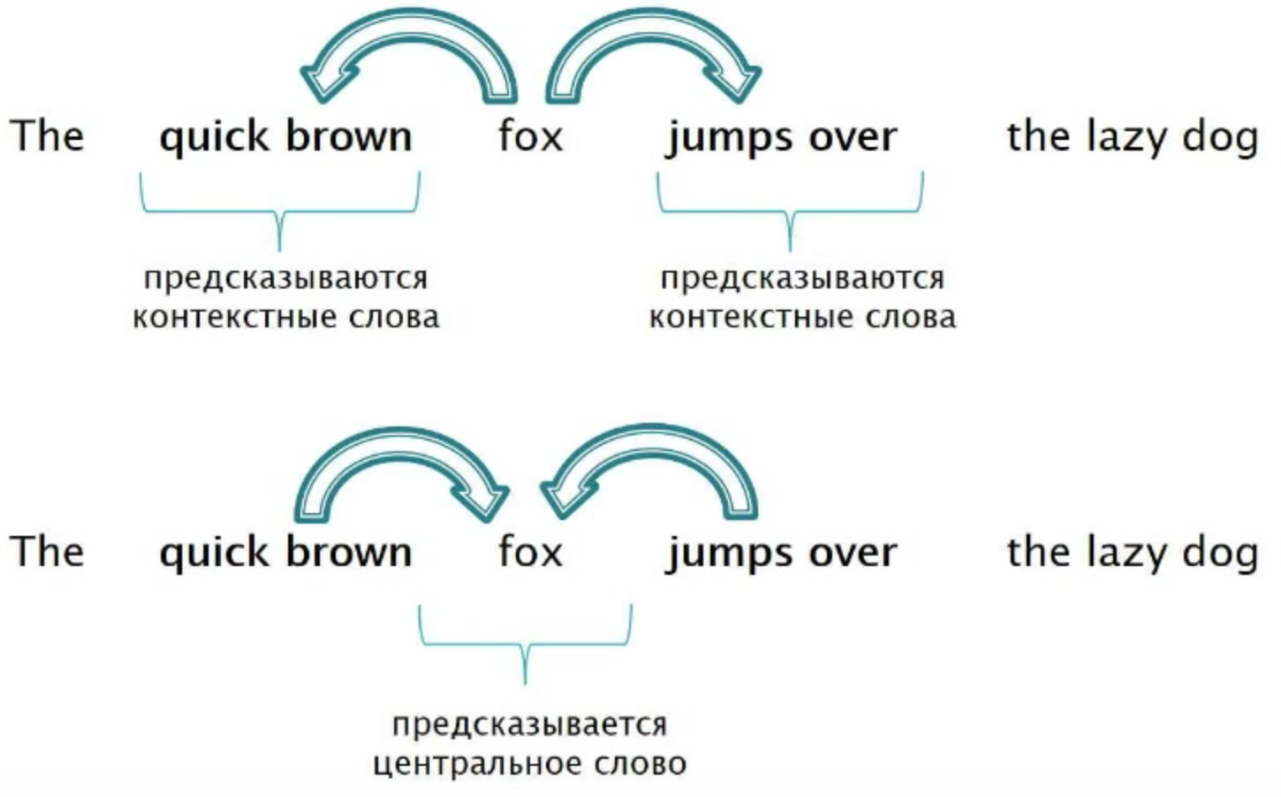
- Относительно низкое качество получаемых представлений.
- Сложность работы с очень большой и разреженной матрицей.
- Сложность добавления новых слов/документов.

Кстати: вместо матрицы со-встречаемостей можно использовать и обычный «мешок слов».

word2vec: Don't count, predict!

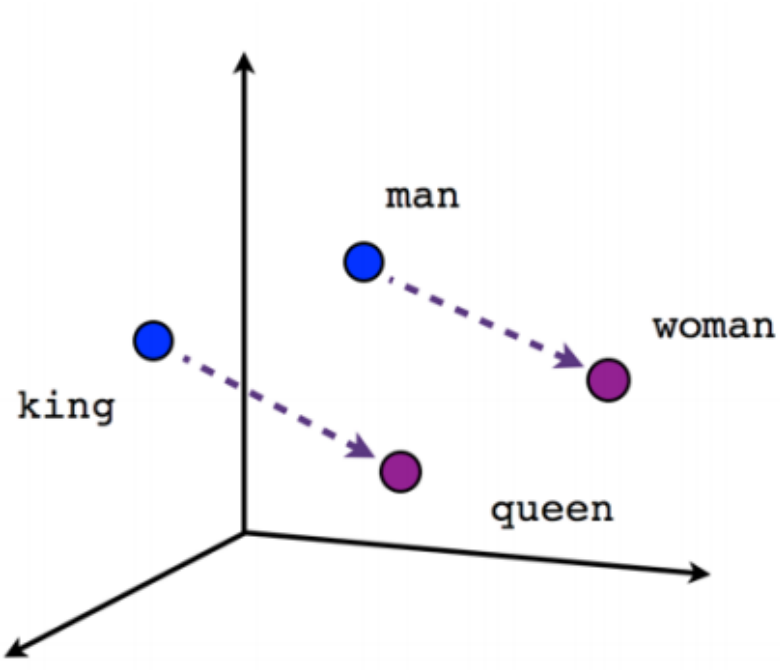
word2vec — группа алгоритмов для получения векторных представлений слов.

Две модели: Continuous BOW и Skip-gram.

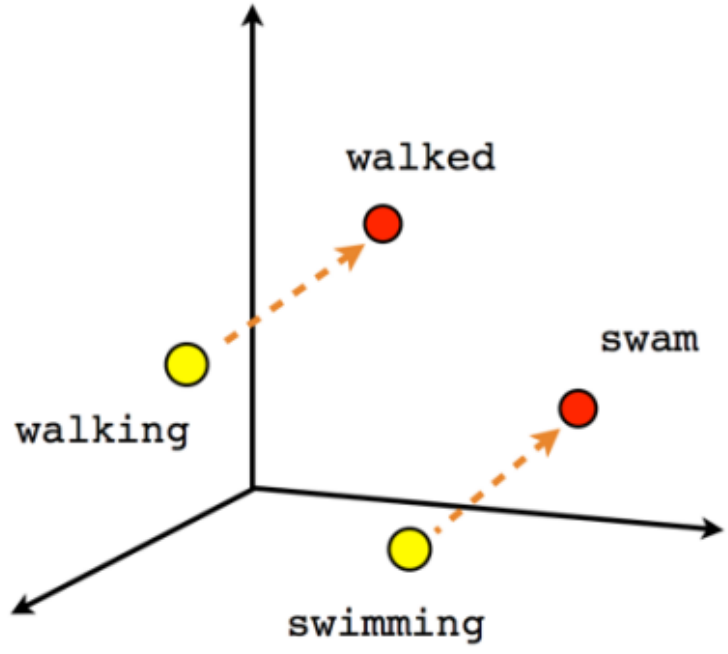


https://www.youtube.com/watch?v=jvGbFS5AYOY

Полезные свойства



Male-Female



Verb tense

Модель CBOW

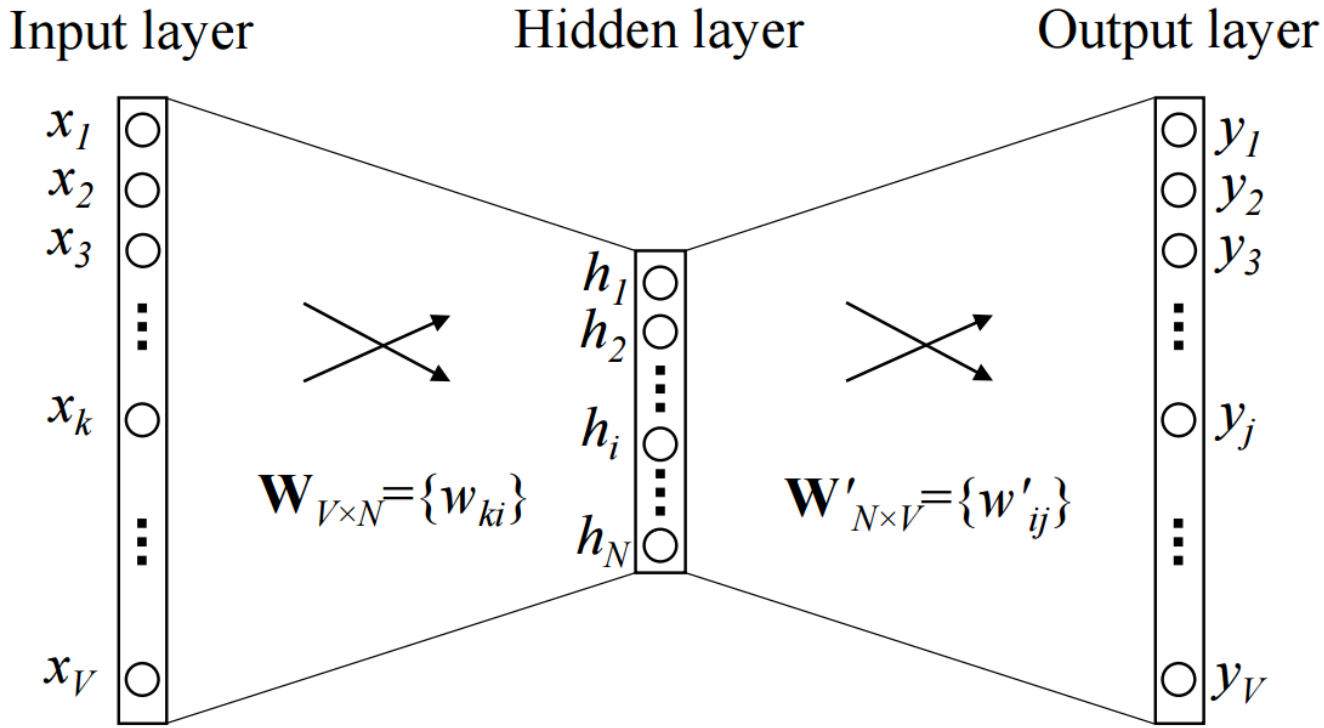
Логарифм правдоподобия: $\mathcal{L} = \sum_{w \in D} \log p(w|c, \theta)$

- θ — параметры модели;
- w — текущее слово;
- c — контекст текущего слова.

Обучаем с помощью простой нейросети:

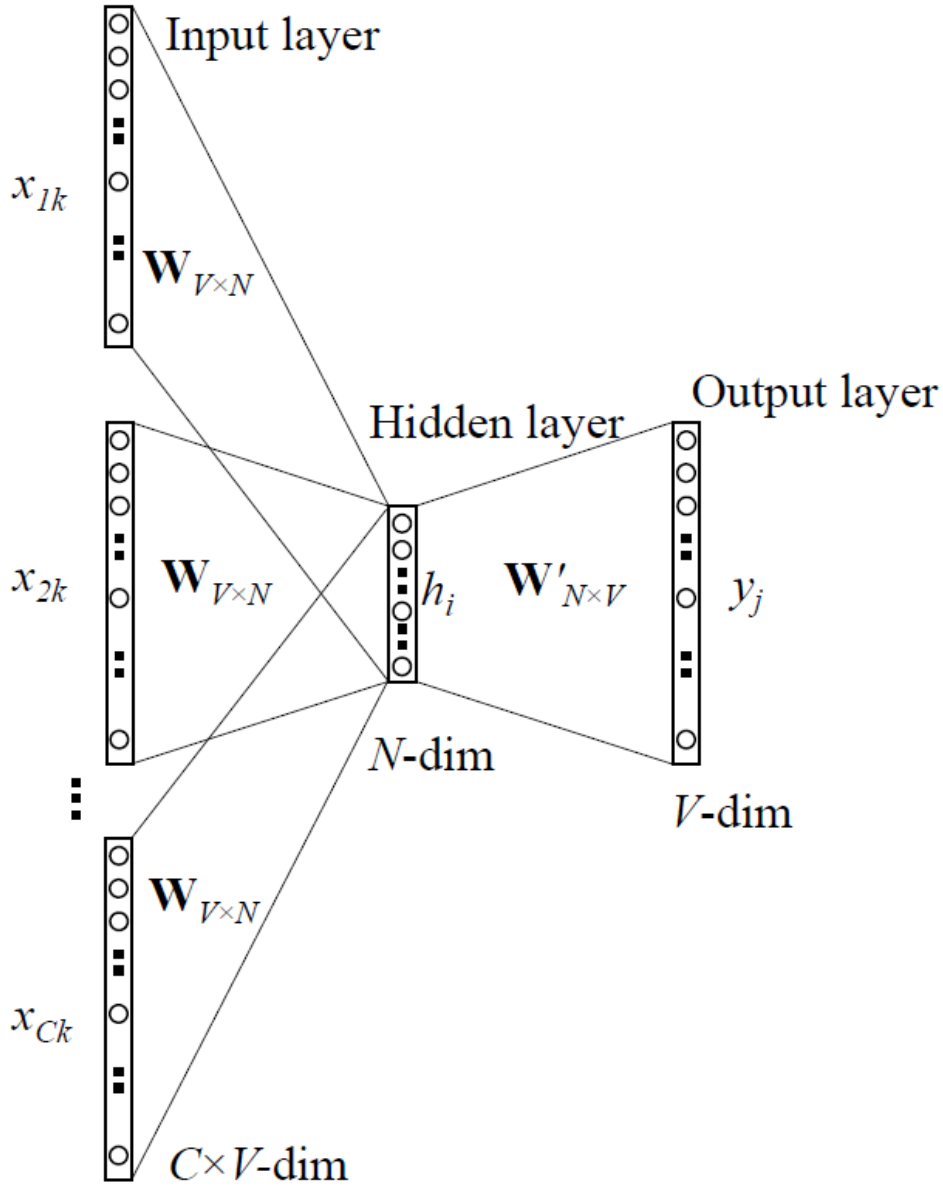
- Скользящим окном проходим по всей коллекции.
- **Вход:** one-hot представление слова (вектор длины $|T|$).
- **Выход:** распределение на словах коллекции (вектор длины $|T|$).
- Вероятность $p(w|c, \theta)$ моделируется softmax-функцией.

Модель CBOW (единичный контекст)

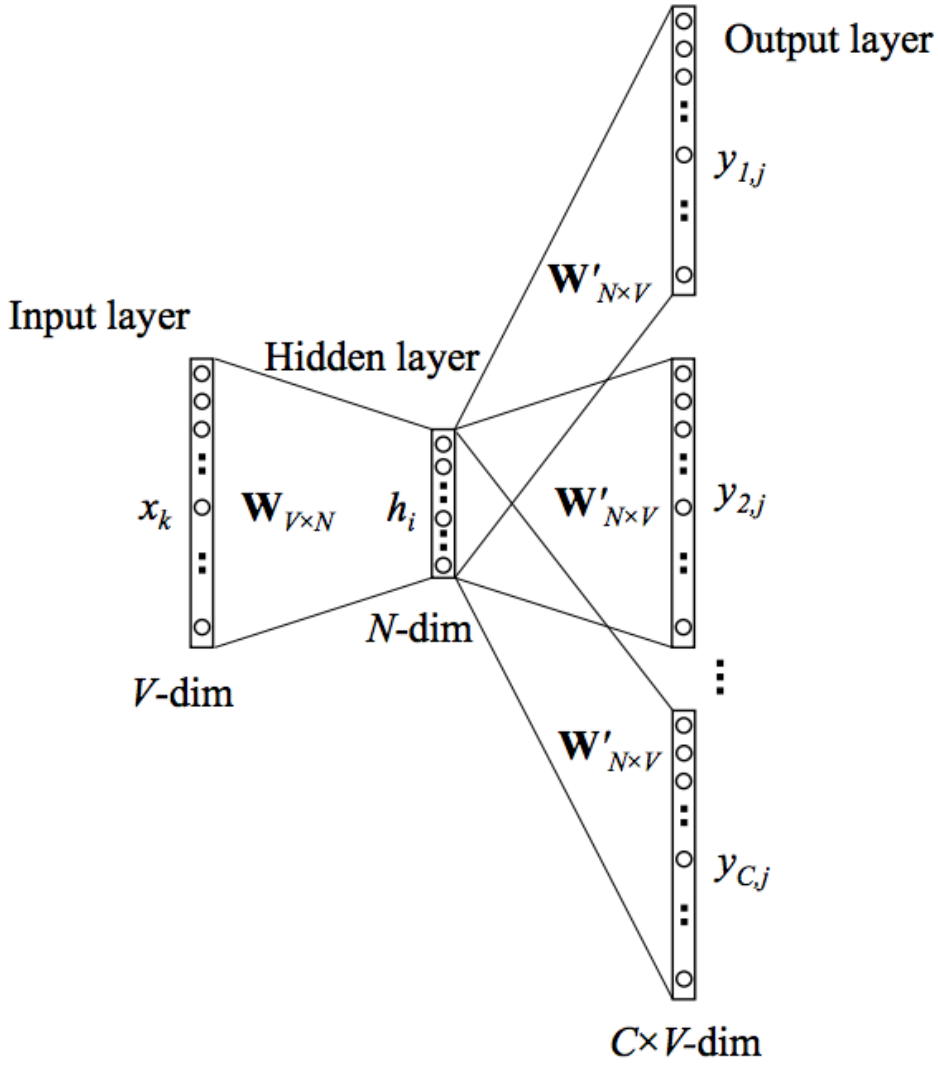


$$p(w|c, v_w, v_c) = \frac{\exp(v_w^T v_c)}{\sum_{w\in D} \exp(v_w^T v_c)}, \qquad \theta = \{v_w, v_c\}$$

Модель CBOW (общий случай)



Модель Skip-gram



Negative Sampling

- Подсчёт нормировочной константы в softmax — дорогая операция.
- Можно изменить постановку задачи и функционал качества.
- Решаем задачу бинарной классификации:

  $z = 1$ — пара $(w, s) \in D$, $z = 0$ — нет $(s \in c(w))$.

$$p(z = 1|(w, s)) = \frac{1}{1 + \exp(-v_w^T v_s)} = \sigma(v_w^T v_s)$$

- Запишем новый функционал праводоподобия:

$$\mathcal{L} = \sum_{(w,s)\in D_1} \log \sigma(v_w^T v_s) + \sum_{(w,s)\in D_2} \log \sigma(-v_w^T v_s),$$

$$D_1 = \{(w, s) : s \in c(w)\}, D_2 = \{(w, s) : s \notin c(w)\}$$

Negative Sampling

$$\mathcal{L} = \sum_{(w,s)\in D_1} \log \sigma(v_w^T v_s) + \sum_{(w,s)\in D_2} \log \sigma(-v_w^T v_s),$$

- Но множество всех отрицательных примеров отсутствует.
- Выход — для каждого рассматриваемого слова w генерировать для отрицательных примеров случайные слова из T.
- Функционал оптимизируется с помощью SGD.
- Вместо негативного сэмплирования можно использовать т. н. *иерархический softmax*.

Пример: исправление опечаток

Word: преключение

— приключение 0.748698
преключения 0.726111
приключения 0.692828
приключеия 0.670168
прключение 0.666706
приключеня 0.663286
прключения 0.660438
приключени 0.659609

Word: avito

— awito 0.693721
авито 0.675299
fvito 0.661414
авита 0.659454
irr 0.642429
овито 0.606189
avito 0.598056

Источник: https://habrahabr.ru/post/249215/

Реализации

- Оригинальный word2vec
- Medallia/Word2VecJava
- FastText
- Spark MLLib Word2Vec
- Gensim word2vec
- и другие

gensim — пакет для тематического моделирования, включает ряд полезных инструментов (часто в качестве удобной обёртки над готовыми реалищациями).

Предоставляет интерфейс для работы с оригинальным word2vec.

Пример: word2vec в gensim

Обучем модель на подсете данных английской википедии.

Ссылка на данные:
`https://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2`

Импортируем основные модули:

```
from gensim.corpora import WikiCorpus
from gensim.models import Word2Vec
from gensim.models.word2vec import LineSentence
```

Пример: word2vec в gensim

Подготовим данные — 100K статей:

```
f = 'enwiki-latest-pages-articles.xml.bz2'
with open('wiki.en.text', 'w') as fout:
    w = WikiCorpus(f, lemmatize=False, dictionary={})
    for i, text in enumerate(wiki.get_texts()):
        fout.write(' '.join(text) + '\n')
        if i == 99999:
            sys.exit()
```

Пример: word2vec в gensim

Обучим модель:

```
model = Word2Vec(LineSentence('wiki.en.text'),
                 size=200,
                 window=5,
                 min_count=3,
                 workers=8)

# trim unneeded model memory = use (much) less RAM
model.init_sims(replace=True)

model.save('wiki.en.word2vec.model')
```

Пример: word2vec в gensim

Используем модель:

- `model.most_similar('queen', topn=3)`

```
[(u'king', 0.6691948175430298),
 (u'princess', 0.648743808269507),
 (u'empress', 0.6162152886390686)]
```

- `model.most_similar(positive=['woman', 'king'], negative=['man'], topn=2)`

```
[(u'queen', 0.6960216164588928),
 (u'empress', 0.597904801368713 4)]
```

Спасибо за внимание!