Математические методы анализа текстов

Семинар 5

Векторные представления слов

Мурат Апишев (great-mel@yandex.ru)

МГУ им. М. В. Ломоносова

30 марта, 2018

Векторные представления слов

- *Векторное представление слова* (word embedding) — сопоставление текстовому слову некоторого числового вектора фиксированной размерности
- Естественно, векторное представление может строиться не только для слов, но и для произвольных объектов
- Векторы могут обладать разнообразными полезными свойствами, например, отражать семантическую близость между словами
- Способы получения:
  - One-hot encoding
  - SVD
  - Topic modeling
  - word2vec/GloVe/FastText/StarSpace

Зачем нужны векторные представления слов

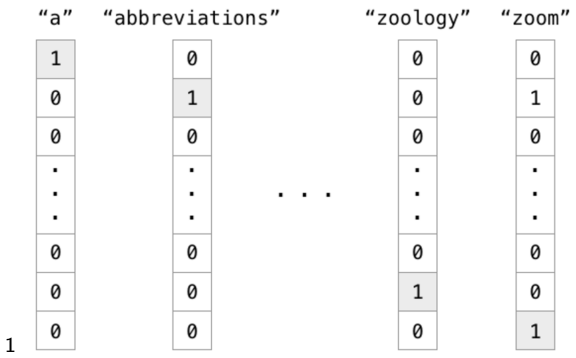
1. ВПС полезны сами по себе, например, для поиска синонимов или опечаток в поисковых запросах

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

One-hot encoding

Самый простой способ кодирования категориальных признаков:



Полученные векторы огромные и ортогональные

[1] https://ayearofai.com/lenny-2-autoencoders-and-word-embeddings-oh-my-576403b0113a

SVD

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

   Понижаем размерность:

2. SVD-разложение: $X = USV^T$

3. Из столбцов матрицы U выбираются первые K компонент

Недостатки SVD

1. Относительно низкое качество получаемых представлений
2. Сложность работы с очень большой и разреженной матрицей
3. Сложность добавления новых слов/документов (решается инкрементальными методами построения)

Что такое word2vec

word2vec — группа алгоритмов, предназначенных для получения вещественных векторных представлений слов в пространстве с невысокой фиксированной размерностью (word embedding)

Вход — коллекция текстов

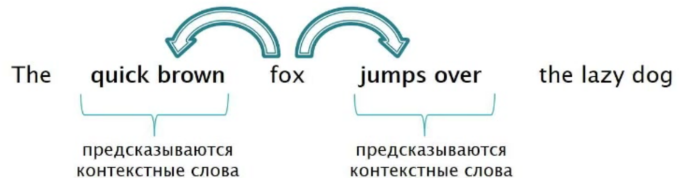
Выход — векторные представления слов

Как правило, в векторном представлении семантически близкие слова оказываются рядом

Идея: «Слова со схожими значениями разделяют схожий контекст»

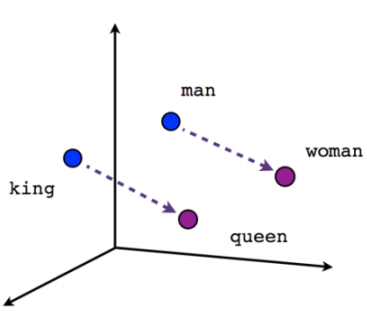
Don't count, predict!

Две модели: **Skip-gram** и **Continuous BOW**

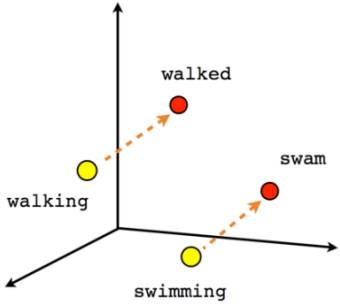


Источник: https://www.youtube.com/watch?v=jvGbFS5AYOY

Полезные свойства

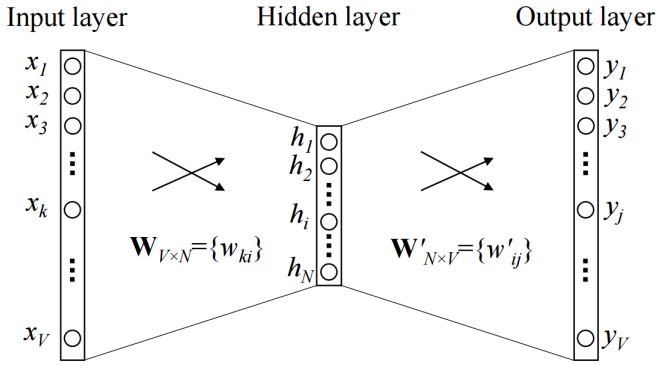


Male-Female

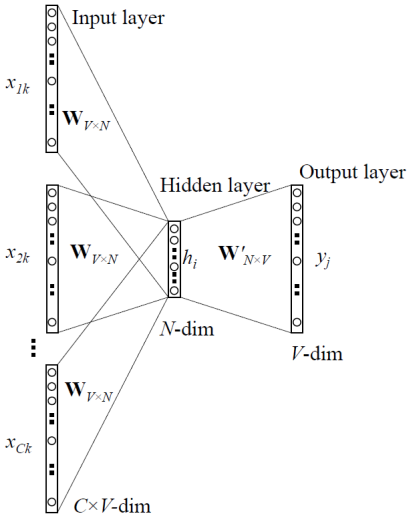


Verb tense

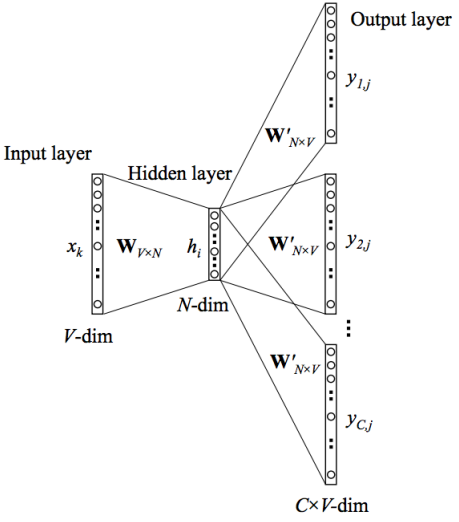
Модель CBOW (единичный контекст)



Модель CBOW (общий случай)



Модель Skip-gram



Проблема

Подсчёт softmax — вычислительно дорогая операция

Применяются различные методы аппроксимации:

1. Softmax-based
   - **Иерархический softmax**
   - Дифференциальный softmax
   - CNN-softmax
2. Sampling-based
   - ...
   - **Negative Sampling**
   - ...

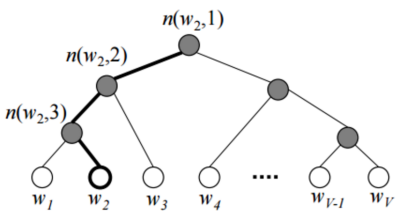
Ссылка на статью с примерами

Иерархический softmax

Вычисление softmax — дорогая операция, $O(V)$
Иерархический softmax — $O(log(V))$

Построим бинарное дерево, листьями которого будут уникальные слова коллекции (например дерево Хаффмана)

Выходы скрытого слоя связываются с внутренними узлами дерева (V-1 штук)



Ссылка на источник картинки

Иерархический softmax

Идея: в процессе обучения при фиксированном контексте нас интересует только предсказываемое слово

Вероятность того, что w будет выходным словом:

$$p(w = w_{out}) = \prod_{j=1}^{L(w)-1} \sigma(\{n(w, j+1) = lch(n(w, j))\} v_{n(w,j)}^T u)$$

- $L(w)$ — длина пути от корня до слова w
- $n(w, j)$ — j-я вершина на этом пути
- $\sigma(x)$ — сигмоида
- $\{\text{true}\} = +1$, $\{\text{false}\} = -1$
- $lch(n)$ — левый потомок вершины n
- $u = v_{w_{inp}}$ в случае skip-gram, $u = 1/h \sum_{k=1}^{h} v_{inp,k}$ (усреднённый вектор контекста) в случае CBOW

Иерархический softmax

Вероятность того, что w будет выходным словом:

$$p(w = w_{out}) = \prod_{j=1}^{L(w)-1} \sigma(\{n(w, j+1) = lch(n(w, j))\} v_{n(w,j)}^T u)$$

Считаем множество бинарных вероятностей, на каждом шаге можно пойти налево или направо с вероятностями

$$p(n, left) = \sigma(v_n^T u)$$
$$p(n, right) = 1 - p(n, left) = \sigma(-v_n^T u)$$

Затем на каждом шаге вероятности перемножаются и получается искомая формула

Negative Sampling

- 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 правдоподобия:

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

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

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

word2vec и PMI

Pointwise Mutual Information:

$$PMI(w, c) = \log \frac{\#(w, c)|D|}{\#(w)\,\#(c)}$$

$$PPMI(w, c) = \max\{PMI(w, c), 0\}$$

$$SPPMI(w, c) = PPMI(w, c) - \log k$$

k — число отрицательных примеров (negative samples)

Можно показать, что модель SGNS является разложением матрицы SPPMI (строки — слова, столбцы — контексты)

Важно: под контекстом в данном случае понимается *центральное слово*, на основании которого предсказываются стоящие рядом слова

GloVe

- SVD + word2vec (условно)
- [2] Строим матрицу $X \in \mathbb{R}^{V \times V}$, x_{ij} — количество раз, когда слово i встречается в контексте слова j (в окне $\leqslant 9$ слов)
- $P_{ij} = \dfrac{x_{ij}}{X_i}$ – вер-ть j в контексте i (X_i – сумма $i-$й строки).
- Строим функцию $F(w_i, w_j, \hat{w}_k)$, показывающую, какое из слов i и j вероятнее увидеть в контексте k (> 1 если i чаще)

$$F(w_i, w_j, \hat{w}_k) = \frac{P_{ik}}{P_{jk}}$$

- Все w_x — векторные представления соответствующих слов

[2] Ссылка на источник описания

GloVe

- Предлагается следующая функция

$$F((w_i - w_j)^T \hat{w}_k) = \frac{F(w_i^T \hat{w}_k)}{F(w_j^T \hat{w}_k)}, \quad F(w_i^T \hat{w}_k) = P_{ik}$$

- Тогда можно взять $F(x) = \exp(x)$, а w_i таким, что

$$w_i^T \hat{w}_k = \log(P_{ik}) = \log(x_{ik}) - \log(X_i)$$

- С учётом фиксированности данных о коллекции, перепишем задачу

$$w_i^T \hat{w}_k + b_i + \hat{b}_k = \log(x_{ik}), \quad b_i + \hat{b}_k = \log(X_i), \quad b_x - \text{bias}$$

- Оптимизация с помощью AdaGrad

GloVe

- Оптимизируемый функционал:

$$J = \sum_{i,j=1}^{V} f(x_{ij})(w_i^T \hat{w}_j + b_i + \hat{b}_j - \log(x_{ij}))^2$$

- $f(x)$ должна
  - $f(0) = 0$
  - $f(x)$ не убывает
  - $f(x)$ относительно мала для больших x
-

$$f(x) = \begin{cases} (x/x_{max})^\alpha, & x < x_{max} \\ 1, & \text{else} \end{cases}$$

- $\alpha = 0.75$, $x_{max} = 100$

Исправление опечаток с помощью СВП

Word: преключение	Word: avito
— приключение 0.748698	— awito 0.693721
преключения 0.726111	авито 0.675299
приключения 0.692828	fvito 0.661414
приключеия 0.670168	авита 0.659454
прключение 0.666706	irr 0.642429
приключеня 0.663286	овито 0.606189
прключения 0.660438	avito 0.598056
приключени 0.659609	

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

Инструменты для получения СВП

- Medallia/Word2VecJava
- **FastText**
- StarSpace
- Spark MLLib Word2Vec
- **Gensim word2vec**
- **GloVe**
- **AdaGram**
- и другие

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

Gensim word2vec

Попробуем построить модели на двух датасетах:

1. Коллекция текстов русской Википедии (50K)
2. Коллекция текстов Луркоморье ($\approx$ 17K)

Данные представлены в виде plain text. Фильтрация слов не производилась. Корпус Луркоморья лемматизирован, Википедии – нет

Подключим необходимые модули:

```
1 import os
2 import sys
3 import codecs
4
5 from gensim.models import Word2Vec
6 from gensim.models.word2vec import LineSentence
7 from gensim.corpora import WikiCorpus
```

Подготовка данных Википедии

```
f = 'ruwiki-latest-pages-articles.xml.bz2'
with codecs.open('wiki.ru.text', 'w',
                 'utf-8', errors='ignore') as fout:

    wiki = WikiCorpus(f, lemmatize=False,
                      dictionary={}, processes=2)

    for i, text in enumerate(wiki.get_texts()):
        fout.write(u' '.join([t.decode('utf-8',
                                       'ignore')
                              for t in text]) + u'\n')

        if i == 49999:
            sys.exit()
```

Лемматизация Википедии

```
import pymorphy2
morph = pymorphy2.MorphAnalyzer()
with codecs.open('wiki.ru.text', 'r', 'utf-8') as fin:
    with codecs.open('wiki_data.txt',
                     'w', 'utf-8') as fout:

        for i, line in enumerate(fin):
            for w in line.strip().split(' '):
                fout.write(u'{} '.format(
                    morph.parse(w)[0].normal_form))
            fout.write('\n')
```

Процесс долгий, занимает ≈ 3.5 часа.

Обучение моделей

```
model = Word2Vec(LineSentence('wiki_data.txt'),
                                 size=400,
                                 window=5,
                                 min_count=5,
                                 workers=2)
model.init_sims(replace=True)  # clear memory
model.save('wiki_w2v.model')


model = Word2Vec(LineSentence('lurk_data.txt'),
                                 size=400,
                                 window=5,
                                 min_count=5,
                                 workers=2)
model.init_sims(replace=True)  # clear memory
model.save('lurk_w2v.model')
```

Результаты

Загрузить модель обратно можно так:

```
m = Word2Vec.load('wiki_w2v.model')
```

Получаем вектор для слова:

```
vec = m[u'россия']
print type(vec), vec.size
# <type 'numpy.ndarray'> 400
```

Находим похожие слова (по косинусной мере):

```
for t in m.most_similar(
                positive=[u'россия'],
                topn=5):
    print t[0], t[1]
```

Результаты

Результат обучения векторных представлений сильно зависит от коллекции. Сравним топ-5 самых близких слов к заданным:

Википедия

`most_similar(россия)`

российский 0.5653642416
рф 0.523694574833
украина 0.492026507854
ссср 0.473026573658
финляндия 0.464367419481

`most_similar(тролль)`

муметь 0.717674195766
гоблин 0.559770524502
великан 0.557757973671
злобный 0.55741250515
гном 0.554968833923

Луркоморье

`most_similar(россия)`

беларусь 0.645048737526
европа 0.622894406319
украина 0.622316598892
рашка 0.619276404381
германия 0.609378278255

`most_similar(тролль)`

троллинг 0.725703835487
троль 0.660580933094
лжец 0.582996308804
провокатор 0.57004237175
толстый 0.568691492081

Результаты

А можно делать ещё вот так:

```
for t in m.most_similar(
                positive=[u'король', u'женщина'],
                negative=[u'мужчина'],
                topn=5):
    print t[0], t[1]
```

Википедия

королева 0.622572660446
изабелла 0.548147201538
монарх 0.520876228809
королевство 0.510757803917
алиенора 0.496694564819

Луркоморье

королева 0.646822690964
лорд 0.634696245193
цезарь 0.631121397018
император 0.630542576313
шут 0.618379890919

Чем больше данных, тем выше качество!

GloVe

```
pip install glove_python
```

```
from glove import Corpus, Glove

sentences = []
with codecs.open('lurk_data.txt',
                 'r', 'utf-8') as fin:

    for line in fin:
        sents.append(line.strip().split(' '))

corpus = Corpus()
corpus.fit(sents, window=5)

model = Glove(no_components=400, learning_rate=0.05)
```

Обучение и результаты

```
model.fit(corpus.matrix,
          epochs=10,
          no_threads=2,
          verbose=True)
model.add_dictionary(corpus.dictionary)
```

Извлекаем векторы:

```
model.word_vectors[model.dictionary[u'тролль']]
```

Находим наиболее похожие слова:

```
for t in glove.most_similar(u'тролль', number=6):
    print t[0], t[1]
```

Результаты

`most_similar(политика)`

идеология 0.727189130492
сабеский 0.714538286076
политический 0.713631574774
пропаганда 0.7007617694
францыя 0.699854020775

`most_similar(сбербанк)`

конгресс 0.882172501104
мид 0.864097843135
ввс 0.862849876553
югославия 0.861940880969
бельгия 0.855773979309

`most_similar(вконтакте)`

вконтактика 0.921803002496
страничка 0.850177460389
контактика 0.838132152427
паблик 0.814504293447
фесбук 0.812322068049

`most_similar(сахалин)`

восьмо 0.81161460995
седьмо 0.803145980402
седьмой 0.791646824613
девятый 0.769437015377
двенадцатый 0.766602958844

Результат сильно зависит от семантики текстов корпуса!

FastText и AdaGram

```
pip install fasttext
```

```
import fasttext

# second arg is a filename for model
model = fasttext.skipgram('lurk_data.txt', 'model',
                          dim=400)
print model[u'европа']
```

```
pip install Cython numpy pip install
git+https://github.com/lopuhin/python-adagram.git

adagram-train lurk_data.txt lurk.pkl
```

```
import adagram
vm = adagram.VectorModel.load('lurk.pkl')
print vm.word_sense_vector(u'европа', 1)
print vm.sense_neighbors(u'европа', 0)
```

Предобученные модели

- Процесс обучения СВП **очень небыстрый**.
- При работе с общеупотребительной лексикой, можно пользоваться предобученными моделями
- Ссылка на модели для разных библиотек

```
1 from gensim.models import Word2Vec
2
3 # Load Google's pre-trained Word2Vec model.
4 f = './GoogleNews-vectors-negative300.bin'
5 model = Word2Vec.load_word2vec_format(f, binary=True)
```

Аналогично модели можно скачать и загрузить в GloVe и FastText

Векторные представления документов

- Тематическое моделирование (столбцы матрицы Θ)
- Взвешенная сумма векторов слов документа (в качестве весов можно брать, например, tf-idf)
- doc2vec (paragraph2vec)
- context2vec

doc2vec доступен в библиотеке gensim. Рекомендуется пользоваться моделью `DBOW` с параметром `dbow_words=0`

Итоги занятия

- Сжатые векторные представления слов можно строить с помощью word2vec, GloVe, FastText, StarSpace
- Векторные представления слов и документов активно используются в качестве признаков в различных задачах текстовой аналитики
- Реализации многимх инструментов по обработке текстов (или обёртки над иными реализациями) можно найти в библиотеке gensim

Теперь послушаем ваши доклады!