Реализация алгоритма векторизации слов Word2vec

Выполнена студентом группы ФН12-11М
Мамаевым Алексеем

Москва, 2021

Word embedding

Отображение слов в n-мерные действительные векторы

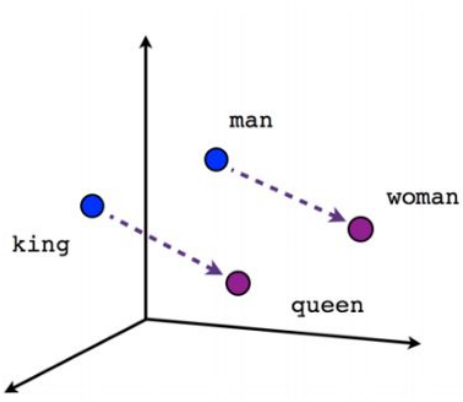
$$vec : W \rightarrow V \subseteq \Re^n$$

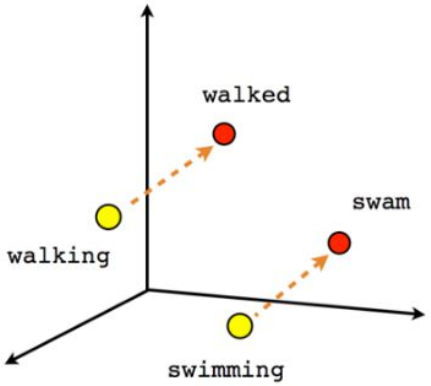
Не является сюръекцией, но можно ввести отображение вектора в ближайшее слово

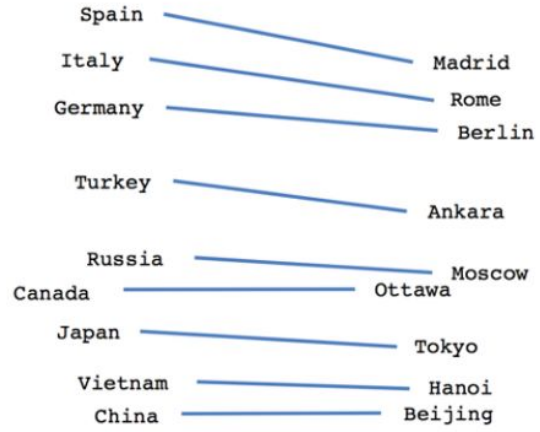
$$vec^{-1} : \Re^n \rightarrow W$$

Свойства

- Близкие по значению слова
- Аналогии слов





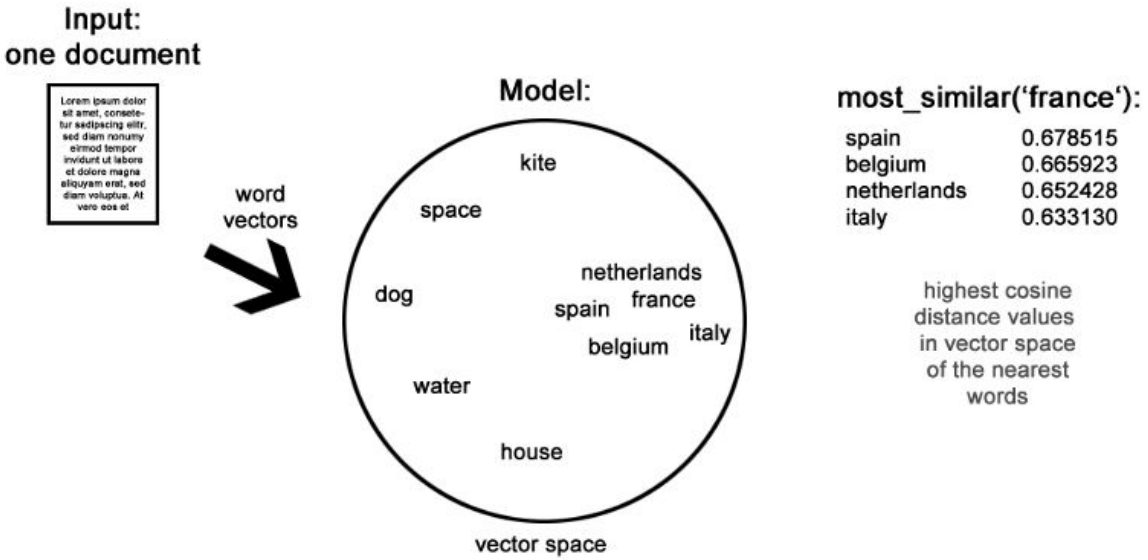


Применение

Семантический анализ, анализ тональности

Рекомендательные системы

Поисковые фразы

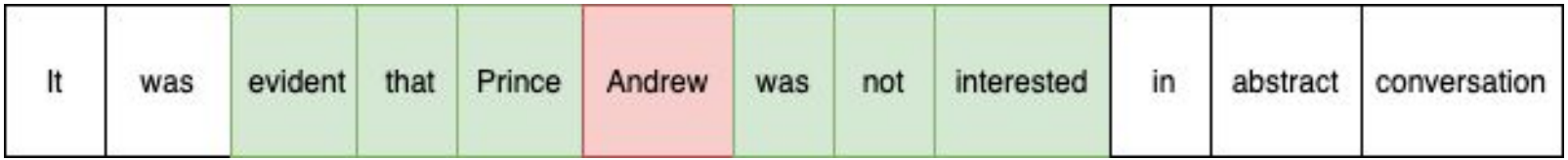


Word2vec

Разработан в Google в 2013

Алгоритмы обучения:

- Skip-gram
- CBOW
- Negative Samplings



Классическая задача

Максимизация softmax косинусных расстояний

$$\forall w \; P(w|c) = \frac{exp(vec(w)^T vec(c))}{\sum_{v \in W} exp(vec(v)^T \; vec(c))} \rightarrow max$$

Упрощенная задача

Вычисления и оптимизация только в контексте слова

$$\sum_{c \in C} exp(vec(w)^T \, vec(c)) + \sum_{c \notin C} exp(-vec(w)^T \, vec(c))$$

$$\sum_{c \in C} \frac{1}{1 + exp(-vec(w)^T vec(c))} - \sum_{c \notin C} \frac{1}{1 + exp(-vec(w)^T vec(c))}$$

Реализация

Python 3

Torch: Embeddings, NN, Adam

Numpy

```
negative_words = torch.randint(1, self.vocab_size,
 size=(batch_size, self.negative_samples_n))
negative_embs = self.embeddings(negative_words).permute(0, 2, 1)
negative_probs = torch.sigmoid(torch.bmm(batch_embs, negative_embs))
negative_loss = torch.nn.functional.
                binary_cross_entropy(negative_probs,
                      negative_probs.new_zeros(negative_probs.shape))
```

Тестирование

Подбор близких слов

```
word2vec.most_similar("dragoons", topk=40)
```

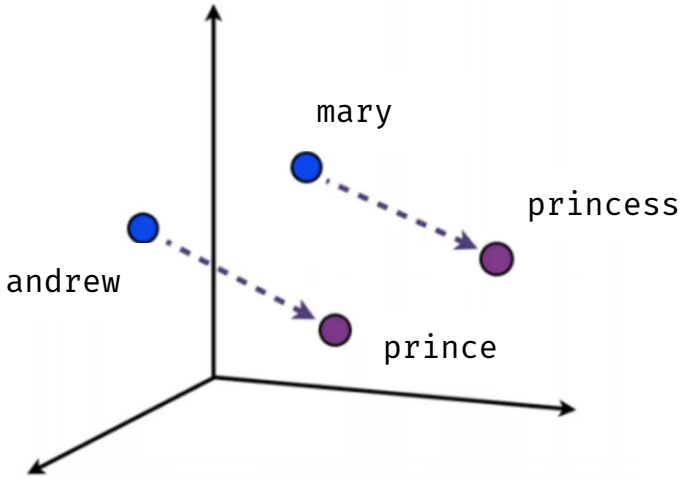
```
[('defend', 0.43215737),
 ('uhlans', 0.40895194),
 ('ferdinand', 0.3891443),
 ('noncommissioned', 0.35259026),
 ('seated', 0.34550554),
 ('stationed', 0.34464905),
```

```
word2vec.most_similar("hussars")
```

```
[('line', 0.4702038),
 ('crowd', 0.430907),
 ('carts', 0.4300836),
 ('prisoners', 0.41685408),
 ('village', 0.41162157),
 ('between', 0.41049403),
 ('orderly', 0.41002244),
 ('horses', 0.40474528),
 ('road', 0.3989981)]
```

Тестирование

Сохранение аналогий




```
word2vec.most_similar_vector(
    word2vec.get_vector("andrew")
    - word2vec.get_vector("prince")
    + word2vec.get_vector("princess")
)
```

```
[('mary', 0.8978914),
 ('she', 0.8663386),
 ('andrew', 0.83834875),
 ('after', 0.822762),
 ('her', 0.81443346),
 ('him', 0.8091848),
 ('was', 0.79340523),
 ('sha', 0.7916329),
 ('nat', 0.78282255),
```

```
word2vec.most_similar_vector(
    word2vec.get_vector("prince")
    - word2vec.get_vector("andrew")
    + word2vec.get_vector("mary")
)
```

```
[('princess', 0.87626827),
 ('she', 0.8108855),
 ('that', 0.79813194),
```

Тестирование

Сравнение с Gensim.models.Word2Vec

Word2Vec: ~150 секунд

Gensim: ~25 секунд

Достоинства и недостатки

+ Простая реализация (~200 строк)
+ Приемлемый результат работы
+ Допустимое время обучения
- Медленнее Gensim
- Недостаточная точность поиска аналогов на малом объеме

Заключение

- Реализован алгоритм Word2Vec
- Изучены классические и используемые принципы работы алгоритма
- Рассмотрено обучение и использование модели на большом объеме данных
- Произведено сравнение с аналогами

Спасибо за внимание