

Задачи и алгоритмы обработки текстов

NLP (Natural Language Processing)

Что такое NLP?

- Набор техник\подходов для решения задач, в которых присутствует текст, как (один из) видов данных

Может быть вещью в себе:

- Машинный перевод

Или частью большей системы:

- Поиск страницы релевантной запросу

Примеры задач обработки текста

- Анализ тональности высказываний
 - например, отзывы на товары
- Классификация сообщений пользователя
 - “Хочу записаться на стрижку”, “спасибо, не надо”
- Поиск релевантных документов
- Синтез текстов
- Машинный перевод
 - Например, с английского на французский
- Определение авторства произведения
- Экстракция сущностей из текста
 - Найти\опознать все имена собственные и адреса в тексте
- И многое многое другое ...

Отличия задач обработки текста

- Работа с последовательностями переменной длины
 - В отличие от матриц (компьютерное зрение) или фиксированных векторов (табличные данные)
- Данные существенно дискретны
 - К изображению кошки (матрице) можно добавить 0.01 и получить изображение кошки, с текстом так не сделать
- Неочевидно как перевести данные в числовой вид
- Разнородность данных:
 - Омонимы, синонимы
 - Опечатки
 - Словообразование (“превед”, “квантификация” и т.д.)
 - Разное написание:
 - “Привет”, “ПРИВЕТ!”, “ЗДАРОВА!!!!”, “ЗДАРОВА!!!!!!!!!!!!!!!!!!!!1111)))”
 - и многое другое ...

Как представить текст в виде набора признаков

Варианты:

- Модель умеет работать с последовательностями
 - Рассмотрим в следующей лекции
- Модель не умеет работать с последовательностями
 - Рассмотрим сейчас

Как представить текст в виде набора признаков

Классический подход

- “Добро пожаловать в Нью-Йорк!”
- Токенизация
 - [“Добро”, “пожаловать”, “в”, “Нью-Йорк!”]
- Нормализация
 - [“добро”, “пожаловать”, “в”, “нью-йорк”]
- Перевод в числовой вид
 - [5, 1, 3, 7]
- Сведение к вектору
 - (0, 1, 0, 1, 1, 0, 1, 0, 0,0)

Токенизация

Разделение текста на слова (токены)

Сложности:

- [“Нью-Йорк”] вместо [“Нью”, “Йорк”]
- [“+7 (555) 512 2565”] или [TEL_NUMBER] вместо [“7”, “555”, “5122565”]
- [“П Р И В Е Т!”] вместо [“П”, “Р”, “И”, “В”, “Е”, “Т!”]
- [“20.11.2012”] вместо [“20”, “11”, “2012”]
- [“感谢”, ...] вместо [“谢谢你翻译这句话”]
 - В некоторых языках слова не разделяют пробелами
 - [“привет”, “как”, “дела”] вместо [“приветкадела”] - хотя это касается и языков где так не делают

Токенизация

Обычно решают с помощью набора регулярных выражений и других эвристик (происходящих из лингвистики)

- Проще простого
 - Делим по пробелам\знакам препинания
- Просто
 - Делим с помощью нескольких регулярных выражений
- Посложнее
 - Делим с помощью специально описанных грамматик
 - Правила для городов
 - Телефонных номеров
 - Дат и других сущностей
 - Обработка знаков препинания
 - ... и многое другое :-)

Токенизация

Обычно решают с помощью набора регулярных выражений\грамматик и других эвристик

- Проще простого
 - “Добро пожаловать в Нью-Йорк!”.split()
- Просто
 - nltk.tokenize.regexp
- Посложнее
 - используем NLTK/Spacy/другие специфичные токенизаторы

Важно: нет токенизатора, который обрабатывает все случаи

Сведение к вектору

Классический подход - “мешок слов” aka Bag of Words

- Нумеруем каждое слово

1. Абак
 2. Абакан
 3. Бурда
 4. Волынка
 5. ...
9999999. Яхта

Мешок слов

Переводим список токенов в список чисел

- ["привет", "как", "дела"] -> [1678, 4390, 9054]
- `indices =`
`[dictionary.get(word, UNK) for word in tokens]`

Создаем вектор - мешок слов:

```
vector = [0] * len(dictionary)
```

```
for k in indices:
```

```
    vector[k] += 1
```

(0, 0, 0,, 1, 0, 0,, 1, 0, 0, ..., 1, 0, 0, 0 ...)

Мешок слов

Пример: анализ тональности

- Применим логистическую регрессию к мешку слов
- `sigmoid(weights * vector).sum()`

Пример:

- “Это плохой фильм” -> (0, 0, ..., 1, ..., 1, ..., 1, ..., 0)
- $0.0001 * \text{“это”} - 1.5 * \text{“плохой”} + 0.0001 * \text{“фильм”} = -1.5 + 0.0002$
- “Это хороший фильм”
- $0.0001 * \text{“это”} + 1.2 * \text{“хороший”} + 0.0001 * \text{“фильм”} = 1.2002$
- Но что если “Это не плохой фильм”?

Мешок слов

Характеристики мешка слов:

- Не учитывает порядок слов
- Разреженный (много нулей) - можно использовать sparse представления и большие словари
- Каждое слово - новое измерение
- Сильно зависит от препроцессинга (“Привет” и “привет” это два разных слова или одно?)
- Чувствителен к опечаткам
- Не учитывает синонимов/омонимов

Стоп слова

Полезны ли слова, которые встречаются почти везде:

- “в”, “на”, “это”, “что”, “то”, ...

Полезны ли слова, которые встречаются один раз среди всех примеров?

При использовании классического подхода (мешок слов) такие слова обычно удаляют

tf-idf

$TF(w)$ = term frequency

- Сколько раз слово w встретилось в документе

$DF(w)$

- Сколько документов содержит слово w

N - количество документов

$IDF(w)$

- $\log (N / DF(w) + 1)$

$tf-idf = TF (w) \quad * \quad IDF (w)$

tf-idf. Пример: поиск документа

Документы:

1. In information retrieval, **tf-idf** or **TFIDF**, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a **word** is to a document in a collection or corpus.[1] It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The **tf-idf** value increases proportionally to the number of times a **word** appears in the document and is offset by the number of documents in the corpus that contain the **word**, which helps to adjust for the fact that some **words** appear more frequently in general. **tf-idf** is one of the most popular term-weighting schemes today; 83% of text-based recommender systems in digital libraries use **tf-idf**. [2]
2. The bag-of-words model is a simplifying representation used in natural language processing and information retrieval (IR). In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its **words**, disregarding grammar and even **word** order but keeping multiplicity. The bag-of-words model has also been used for computer vision.[1] The bag-of-words model is commonly used in methods of document classification where the (frequency of) occurrence of each **word** is used as a feature for training a classifier[2]. An early reference to "bag of **words**" in a linguistic context can be found in Zellig Harris's 1954 article on Distributional Structure.[3]

$$\text{tf}(\text{tf-idf}, 1) = 5 \quad \text{tf}(\text{word}, 1) = 4 \quad \text{df}(\text{tf-idf}) = 1 \quad \text{idf}(\text{tf-idf}) = 1.1$$

$$\text{tf}(\text{tf-idf}, 2) = 0 \quad \text{tf}(\text{word}, 2) = 4 \quad \text{df}(\text{word}) = 2 \quad \text{idf}(\text{word}) = 0.69$$

$$\text{tf-idf}(\text{tf-idf}, 1) = 5 * 1.1 = 5.5 \quad \text{tf-idf}(\text{word}, 1) = 4 * 0.69 = 2.76$$

$$\text{tf-idf}(\text{tf-idf}, 2) = 0 * 1.1 = 0 \quad \text{tf-idf}(\text{word}, 2) = 4 * 0.69 = 2.76$$

Зачем нужен tf-idf

Документы:

1. $\text{tf_idf}(\dots, 5, 4, \dots) = (\dots, 5.5, 2.76, \dots)$
2. $\text{tf_idf}(\dots, 0, 4, \dots) = (\dots, 0, 2.76, \dots)$

Идея tf-idf:

- Увеличить веса специфичных слов (встречающихся в малом числе документов)
- Уменьшить вес слов встречающихся во многих документах

Применение tf-idf к мешку слов помогает в разных задачах

Стемминг

Нахождение “корня” слова с помощью эвристических правил и алгоритмов.

Пример: суффиксный стеммер

(.*)ed -> \1

(.*)ely -> \1

(.*)ing -> \1

supervised, supervisely, supervising -> supervis

Лемматизация

Приведение слова к словарной (нормальной) форме.

Используются морфологические анализаторы (определяется часть речи, делается морфологический разбор)

Работают медленнее чем алгоритмы стемминга, дают более корректные результаты с точки зрения лингвистики

Стемминг и лемматизация - способы “нормализации” слов

Как учитывать связи и порядок слов?

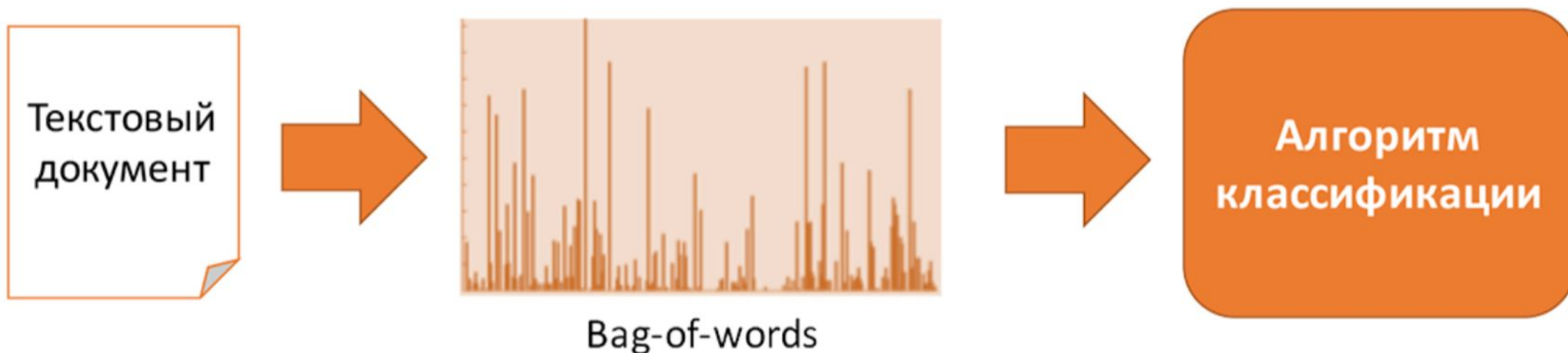
- N-gram - последовательность из n идущих подряд слов в тексте
 - униграммы ($n=1$), биграммы ($n=2$), триграммы ($n=3$)
- K-skip-N-gram - последовательность из n идущих подряд слов в тексте, причём расстояние между соседними должно составлять не более k токенов
- **Пример: "Набор подряд идущих токенов"**
- 2-gram: набор подряд, подряд идущих, идущих токенов
- 1-skip-2-gram: набор подряд, подряд идущих, идущих токенов, набор идущих, подряд токенов

1-skip-2-gram

Source Text	Training Samples			
<table><tr><td>The</td><td>quick</td><td>brown</td></tr></table> fox jumps over the lazy dog. ➡	The	quick	brown	(the, quick) (the, brown)
The	quick	brown		
The <table><tr><td>quick</td><td>brown</td><td>fox</td></tr></table> jumps over the lazy dog. ➡	quick	brown	fox	(quick, the) (quick, brown) (quick, fox)
quick	brown	fox		
The quick <table><tr><td>brown</td><td>fox</td><td>jumps</td></tr></table> over the lazy dog. ➡	brown	fox	jumps	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
brown	fox	jumps		
The quick brown <table><tr><td>fox</td><td>jumps</td><td>over</td></tr></table> the lazy dog. ➡	fox	jumps	over	(fox, quick) (fox, brown) (fox, jumps) (fox, over)
fox	jumps	over		

Bag of words pipeline

- Препроцессинг текста
- Bag-of-words на словах и N-граммах как векторное представление текста
- Linear models, SVD, Random forest и т.д. для классификации



Word2Vec

- Другой способ векторного представления
- То, что называют "pretrained word embeddings"
- Статья издана 7 сентября 2013 (Google research)

Word2Vec

Основная идея: дистрибутивная гипотеза

- Слова встречающихся в схожих контекстах имеют близкие значения

Задача:

- предсказать слово по его контексту (соседним словам)

или

- предсказать контекст по слову

Для обучения обычно используют большой корпус текста (википедия, common crawl ...)

Word2Vec skip-gram negative sampling

Задача: по слову предсказать его контекст

Обучающая выборка:

“Предсказать по слову его контекст”

(“по”, “слову”, 1)

(“его”, “слову”, 1)

(“предсказать”, “слову”, 0)

(“контекст”, “слову”, 0)

(“выборка”, “слову”, 0)

Положительные пример: выбираем из текста

Отрицательный пример: ставим в пару случайное слово из текста

Word2Vec skip-gram negative sampling

Задача: по слову предсказать его контекст

Модель:

```
vocab_size = 100_000
emb_size = 200
context_embedding = Linear(vocab_size, emb_size)
word_embedding = Linear(vocab_size, emb_size)
output = Linear(emb_size, 1)

def forward(word, context):
    embd_ctx = context_embedding(context)
    embd_word = word_embedding(word)
    return sigmoid(output(embd_ctx * embd_word))
```

Word2Vec skip-gram negative sampling

Задача: по слову предсказать его контекст

Обучение:

```
for word, context, label in dataset:
    pred = model(word, context)
    loss = binary_cross_entropy(pred, label)
    loss.backward()
    optimizer.step()
    model.zero_grad()
```

Получение векторов:

```
vectors = [word_embedding(word) for word in vocab]
```

Word2Vec

Skip-gram

- Сэмплирование плохих примеров в negative sampling можно базировать на встречаемости слов
- Вместо negative sampling, можно честно считать softmax и минимизировать негативное лог. правдоподобие (неэффективно для больших словарей)
- Можно использовать иерархический softmax

CBOW

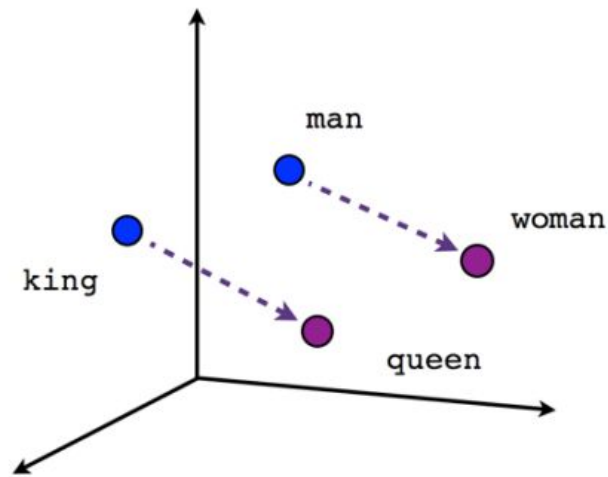
- Вместо предсказания контекста по слову, можно предсказывать слово по контексту

Word2Vec

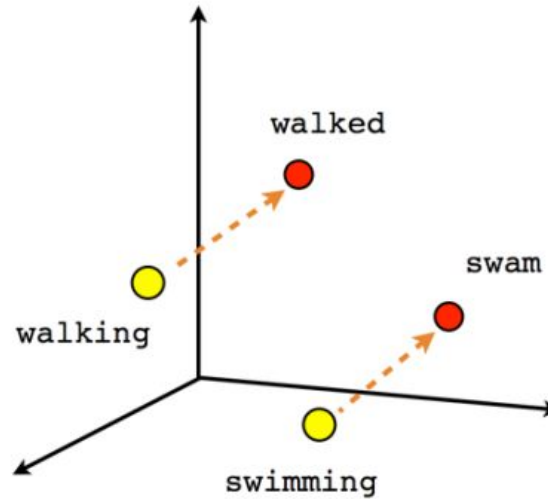
- Получаем “dense” представление слов как векторов небольшой размерности (300-1000)
- Вектора обучают на больших неразмеченных наборах текстов
 - После чего их можно использовать в своих задачах
- Для получения вектора примера берут среднее векторов слов
- Или используют WMD (word mover distance)

Интересное свойство word2vec

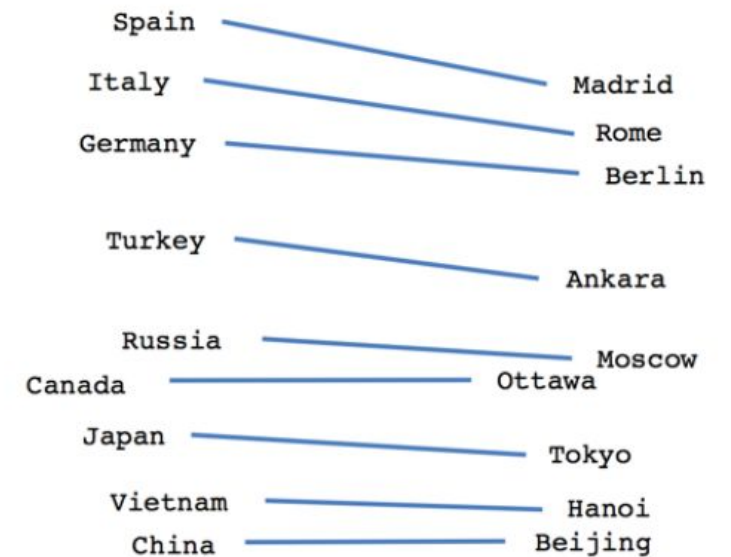
- King - Man + Woman = Queen
- Paris - France + Italy = Rome



Male-Female



Verb tense

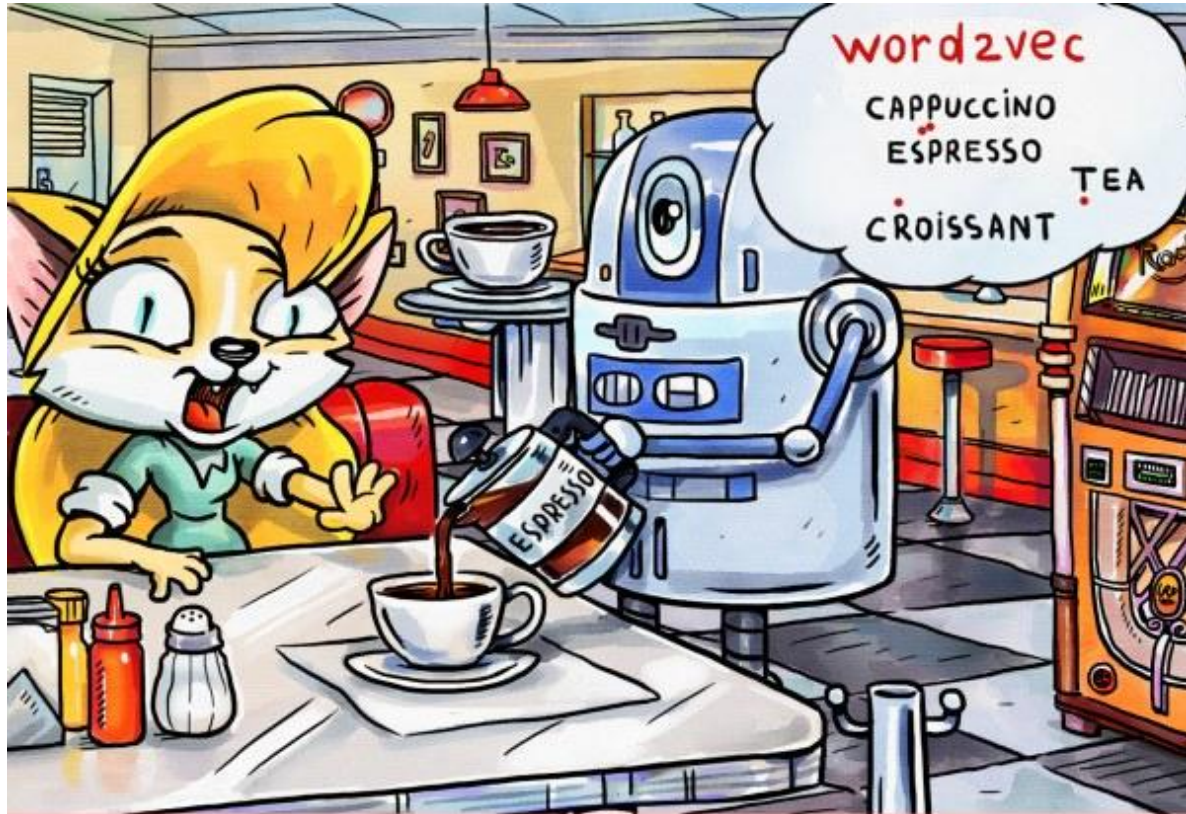


Country-Capital

TSNE на word2vec



Интересное свойство word2vec



- Espresso? But I ordered a cappuccino!
- Don't worry, the cosine distance between them is so small that they are almost the same thing.

Близость векторов слов не означает что слова - синонимы, и тем более что они синонимы в вашей конкретной задаче

Word2Vec как распределенное представление

Пример т.н. “распределенного представления”

- Объект (слово) преобразуется в вектор в некотором пространстве
- Отношения между объектами - похожесть векторов

Другой пример:

- поиск похожих изображений
 - Евклидово\косинусное расстояние между эмбедингами CNN
 - Методы:
 - Сиамские сети
 - arcface

Расширения word2vec

- Алгоритм word2vec модифицируют, чтобы :
 - Решать проблему омонимов
 - Давать контекстно-зависимые вектора
 - Давать устойчивость к опечаткам и новым словоформам
 - и т.д.
- Советую посмотреть: ELMO
- Сейчас рассмотрим FastText

FastText

Модификация алгоритма word2vec, устойчивая к опечаткам и новым словоформам

- Обучаются вектора слов и вектора sub-words (3-грамм)
- В случае, если слово есть в словаре используется его вектор
- Иначе вектор слова есть среднее векторов его триграмм

Позволяет обойтись без специального токена UNK для неизвестных слов

В некоторых задачах позволяет избежать добавления модели исправления опечаток

Идея word2vec

Идея word2vec оказалась полезной не только для слов:

- Для документов
- Для графов
- В рекомендательных системах (последовательности просмотров\покупок)
- ...