

# Семинар. Работа с текстами.

---

Даулбаев Талгат

14 марта 2017 г.

# Слайд про то, что тексты надо превращать в вектора

Пусть есть множество  $D$ , состоящее из документов  $d_1, \dots, d_N$ .

Каждый текст будем считать объектом.

Цель — решить задачу машинного обучения. Например, классифицировать спам.

Что можно считать признаками текста?

# Term Frequency

Признак — «частота» встречаемости каждого слова  $t$  в тексте  $d$ .

Называется «term frequency» и обозначается  $tf_{t,d}$ .

Разные способы вычисления  $tf_{t,d}$ :

- 1, если слово  $t$  входит в документ  $d$ , иначе — 0;
- $\nu_{t,d}$  — количество вхождений слова  $t$  в документ  $d$ ;
- $\nu_{t,d}$ , делённое на количество слов в  $d$ ;
- $1 + \log(\nu_{t,d})$ ;
- и так далее...

В чём проблемы такого представления?

Если слово  $t$  часто встречается в  $d$ , но редко — в других документах, то это слово важное. Но  $tf_{t,d}$  не учитывает этого.

Введём характеристику «document frequency»  $df_{t,D}$  — количество документов, в которых есть слово  $t$ .

А с помощью неё — «inverted document frequency»  $idf_{t,D}$ , которая тем больше, чем меньше  $df_{t,D}$ .

Пусть  $N$  — общее количество документов (объектов).

Самые распространённые варианты вычисления  $\text{idf}_{t,D}$ :

- $\log \frac{N}{\text{df}_{t,D}}$  (формула из учебников)
- $\log \frac{N}{\text{df}_{t,D}} + 1$  (формула из scikit-learn)
- $\max \left\{ 0, \log \frac{N - \text{df}_{t,D}}{\text{df}_{t,D}} \right\}$

Произведение  $tf$  и  $idf$  так и называется «term frequency — inverse document frequency»:

$$tf-idf_{t,d,D} = tf_{t,d} \times idf_{t,D}$$

Значение  $tf-idf_{t,d,D}$

- высокое, когда  $t$  встречается в малом числе документов;
- низкое, когда:
  - слово  $t$  встречается в большом числе документов;
  - слово  $t$  нечасто встречается в документе  $d$ ;

## TF-IDF: пример

- $d_1 = \text{«Я на выборы никогда не ходил»}$
- $d_2 = \text{«Не жалею, не зову»}$
- $d_3 = \text{«Ходор Ходор Ходор»}$

Пусть  $\text{idf}_{t,D} = \log \frac{N}{\text{df}_{t,D}}$ , а  $\text{tf}_{t,d}$  — доля  $t$  в  $d$ .

Для слова  $t_1 = \text{«не»}$  в  $d_1$ :

$$\text{tf-idf}_{t_1,d_1,D} = \frac{1}{6} \times \log \frac{3}{2} \approx 0.07$$

Для слова  $t_2 = \text{«выборы»}$  в  $d_1$ :

$$\text{tf-idf}_{t_2,d_1,D} = \frac{1}{6} \times \log \frac{3}{1} \approx 0.18$$

Два варианта:

- `sklearn.feature_extraction.text.TfidfVectorizer` — преобразует набор «сырых» текстов в матрицу с tf-idf признаками.
- `sklearn.feature_extraction.text.TfidfTransformer` — преобразует матрицу со счётчиками вхождений слов в tf-idf матрицу.



Два варианта:

1. `sklearn.feature_extraction.text.TfidfVectorizer` — преобразует набор «сырых» текстов в матрицу с tf-idf признаками.
2. `sklearn.feature_extraction.text.TfidfTransformer` — преобразует матрицу со счётчиками вхождений слов в tf-idf матрицу.

Пойдём по второму пути. Набор «сырых» текстов можно преобразовать в матрицу со счётчиками вхождений слов с помощью `sklearn.feature_extraction.text.CountVectorizer`.

# CountVectorizer: пример

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import pandas as pd
```

```
texts = ["I am a lumberjack",
        "And I am okay",
        "I sleep all night and I work all day"]
```

```
# token_pattern изменён, чтобы учитывать и однобуквенные слова
count = CountVectorizer(token_pattern=r"\b\w+\b")
# .toarray() для преобразования в плотный массив:
result = count.fit_transform(texts).toarray()
# .get_feature_names() - названия признаков
df = pd.DataFrame(data=result, columns=count.get_feature_names())
```

# CountVectorizer: пример

```
texts = ["I am a lumberjack",  
         "And I am okay",  
         "I sleep all night and I work all day"]
```

	a	all	am	and	day	i	lumberjack	night	okay	sleep	work
0	1	0	1	0	0	1	1	0	0	0	0
1	0	0	1	1	0	1	0	0	1	0	0
2	0	2	0	1	1	2	0	1	0	1	1

# TfidfTransformer: пример

```
from sklearn.feature_extraction.text import TfidfTransformer
```

```
texts = ["I am a lumberjack",  
         "And I am okay",  
         "I sleep all night and I work all day"]
```

```
tfidf = TfidfTransformer()  
result = tfidf.fit_transform(df).toarray()  
new_df = pd.DataFrame(data=result, columns=count.get_feature_names())
```

	a	all	am	and	day	i	lumberjack	night	okay	sleep	work
0	0.69	0.0	0.23	0.00	0.00	0.08	0.69	0.00	0.0	0.00	0.00
1	0.00	0.0	0.30	0.30	0.00	0.11	0.00	0.00	0.9	0.00	0.00
2	0.00	0.7	0.00	0.12	0.35	0.09	0.00	0.35	0.0	0.35	0.35

**Проблема:** «dog» и «dogs», «человек» и «человеку» считаются разными словами.

Лемматизация — приведение слова к его нормальной форме:

- для существительных — именительный падеж, единственное число (кошками → кошка);
- для прилагательных — именительный падеж, единственное число, мужской род (красивая → красивый);
- для глаголов, причастий, деепричастий — глагол в инфинитиве (страдающий → страдать).

# Лемматизация на русском языке

```
import pymorphy2
import string
morph = pymorphy2.MorphAnalyzer()
text = """Нам не нужно других миров. Нам нужно
зеркало... Мы бьёмся над контактом и никогда не найдём его. Мы в
глупом положении человека, рвущегося к цели, которой он боится,
которая ему не нужна. Человеку нужен человек!"""
text = text.translate(str.maketrans("", "", string.punctuation))
normal_forms_list = []
for word in text.split():
    norm_form = morph.parse(word)[0].normal_form
    normal_forms_list.append(norm_form)
' '.join(normal_forms_list)
```

## Результат:

мы не нужно другой мир мы нужно зеркало мы биться над контакт и  
никогда не найти он мы в глупый положение человек рваться к цель  
который он бояться который он не нужный человек нужный человек

Лемматизация — сложная штука. Куда проще стемминг — выделение из слова «основы» по эвристическим правилам.

Примеры:

- «кошки» → «кош»
- «кошачий», «кошачье» → «кошач»

Есть много разных стеммеров.

```
from nltk.stem.snowball import SnowballStemmer  
ss = SnowballStemmer('english')  
ss.stem('dogs')
```

Нужна библиотека nltk (pip install nltk).

Придётся загрузить дополнительные пакеты. Для этого запустить nltk.download() и открыть всплывающее окно.

Стемминг текста тоже делается пословно.



# The Beatles or Snoop Dogg?<sup>1</sup>



Обучите логистическую регрессию различать тексты Beatles и Snoop Dogg'а.

Архив с данными — на странице курса на гитхабе.

---

<sup>1</sup>спасибо за идею Максиму Новикову (ИАД-16, 2015/2016)