[Машинное обучение для текстов 2](#_Toc133298829)

[Структура курса 2](#_Toc133298830)

[Ваши цели 2](#_Toc133298831)

[Векторизация текстов 3](#_Toc133298832)

[Введение 3](#_Toc133298833)

[Чему вы научитесь 3](#_Toc133298834)

[Постановка задачи 3](#_Toc133298835)

[Лемматизация 3](#_Toc133298836)

[Задача 4](#_Toc133298837)

[Регулярные выражения 5](#_Toc133298838)

[Задача 6](#_Toc133298839)

[Мешок слов и N-граммы 7](#_Toc133298840)

[Создание мешка слов 10](#_Toc133298841)

[Задача 11](#_Toc133298842)

# Машинное обучение для текстов

Вы научитесь решать задачи классификации и регрессии для текстов.

Узнаете, как делать из текстов табличные данные.

#### Структура курса

Анализ текстов помогает компаниям оценить отношение к бренду и узнать, какой контент может понравиться потребителям. Быстрее человека в разы машина определит, какой тематики и тональности текст, есть ли в нём токсичные слова.

Чтобы научить машину работать с текстами, их нужно преобразовать в понятный для неё формат. Поэтому изучение курса вы начнёте с алгоритмов извлечения признаков из текстов. Узнаете, что такое мешок слов и N-граммы. Обучите логистическую регрессию определять тональность текста.

Затем познакомитесь с новым способом создания признаков из текстов — языковыми представлениями. Освоите word2vec и BERT. Узнаете, как работать с уже обученными языковыми моделями BERT и применять их в задаче классификации.

В конце курса вас ждёт проект по классификации текстов.

💡 Английский язык пригодится не только для того, чтобы разбираться в документации. Общение на английском с коллегами и работодателями — неотъемлемая часть работы в международных компаниях. Если хотите подтянуть знание языка, обратите внимание на курс «Английский для аналитиков» от Яндекс Практикума.

Обучение на курсе построено вокруг типичных ситуаций в работе аналитика: обсуждения данных и их визуализации, стендапов, планирования спринтов, сбора требований и защиты проекта перед заказчиком, а материал закрепляется на встречах с иностранцами из IT. В результате после курса вам будет легче общаться и работать в англоязычной среде. Для студентов Практикума действует скидка 10% по промокоду STUDENTSPRACTICUM.

[**Узнать больше ➞**](https://practicum.yandex.ru/english/english_for_career/#!/tab/491633859-2?utm_source=practicum&utm_medium=students_course&utm_campaign=prct_st_crs_data_scientist&utm_content=it_english)

## Ваши цели

* Научиться вычислять величины TF-IDF для текстов.
* Понять, как находить языковые представления моделью BERT.
* Построить модель классификации для текстов.

Это сложный курс: нужно освоить много инструментов работы с текстами.

Разобраться, что такое модель BERT, вам помогут наставники на вебинаре.

## Векторизация текстов

### Введение

Научимся извлекать из текстов признаки.

#### Чему вы научитесь

* Узнаете, что такое мешок слов и N-граммы.
* Освоите метод вычисления признаков TF-IDF.
* Построите модель для классификации текстов.

### Постановка задачи

Построим модель, которая классифицирует твиты положительной и отрицательной тональности.

### Лемматизация

Прежде чем извлечь признаки из текста, упростим его.

Рассмотрим этапы предобработки текста:

1. **Токенизация** (англ. *tokenization*) — разбиение текста на токены: отдельные фразы, слова, символы.
2. **Лемматизация** (англ. *lemmatization*) — приведение слова к начальной форме — лемме.

Функция лемматизации русского текста есть в библиотеках:

* pymorphy2 (англ. *python morphology,* «морфология для Python»), \*\*
* UDPipe (англ. *universal dependencies pipeline*, «конвейер для построения общих зависимостей»*),*
* pymystem3.

Рассмотрим, как работать с библиотекой pymystem3.

Создадим класс для лемматизации:

Скопировать кодPYTHON

from pymystem3 import Mystem

m = Mystem()

Функции lemmatize() передадим «лемматизируй это»:

Скопировать кодPYTHON

m.lemmatize("лемматизируй это")

Функция вернула лемму каждого слова. Для функции конец строки — это тоже знак, поэтому находится и его лемма (\n) (от англ. *new line*, «новая строка»):

Скопировать код

['лемматизировать', ' ', 'это', '\n']

Функцией join() объединим элементы списка в строку, разделив их пробелом (но можно и без него):

Скопировать кодPYTHON

" ".join(['лемматизировать', ' ', 'это', '\n'])

Получаем:

Скопировать код

'лемматизировать это \n'

В этой теме мы будем работать с русскоязычным корпусом коротких текстов RuTweetCorp.

Перед вами уменьшенный датасет — 5000 записей. Каждая запись содержит текст поста и оценку его тональности. Если пост позитивный, то метка «1», если негативный — «0». Осторожно: твиты могут содержать обсценную (от лат. *obscenus*, «непристойный») лексику.



Импортируем данные:

Скопировать кодPYTHON

import pandas as pd

data = pd.read\_csv('tweets.csv')

Чтобы алгоритмы умели определять тематику и тональность текста, их нужно обучить на корпусе (англ. *corpus*). Это набор текстов, в котором эмоции и ключевые слова уже размечены.

Создадим корпус постов. Преобразуем столбец 'text' в список текстов. Переведём тексты в стандартный для Python формат: кодировку Unicode U.

Изменим кодировку методом astype():

Скопировать кодPYTHON

corpus = data['text'].values.astype('U')

#### Задача

Напишите функцию lemmatize(text). На вход она принимает текст из корпуса и возвращает лемматизированную строку. Возьмите первый текст из датасета tweets.csv. Код напечатает на экране исходный и лематизированный тексты.

import pandas as pd

from pymystem3 import Mystem

data = pd.read\_csv('/datasets/tweets.csv')

corpus = data['text'].values.astype('U')

def lemmatize(text):

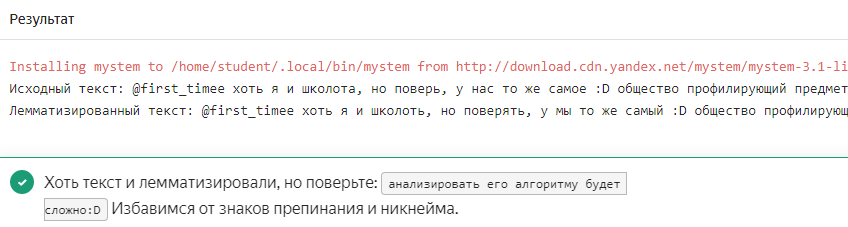
m = Mystem()

m.lemmatize(text)

return "".join(m.lemmatize(text))

print("Исходный текст:", corpus[0])

print("Лемматизированный текст:", lemmatize(corpus[0]))



### Регулярные выражения

От лишних символов текст очистят **регулярные выражения** (англ. regular expressions).

Это инструмент для поиска слова или числа по **шаблону** (англ. pattern). Он определяет, из каких частей состоит строка и какие в них символы. Например, нужно найти все даты в таком формате записи: 02.02.2020. Их шаблон — это два числа, точка, два числа, точка, четыре числа.

Шаблон для email-адреса такой: строка на латинице, символ @, строка на латинице, точка, строка на латинице.

Для работы с регулярными выражениями в Python есть встроенный модуль **re** (сокр. от regular expressions):

Скопировать кодPYTHON

import re

Познакомимся с функцией **re.sub()** (от англ. substitute, «заменять»). Она находит в тексте все совпадения по шаблону и заменяет их заданной строкой.

Скопировать кодPYTHON

*# pattern — шаблон*

*# replacement — на что заменять*

*# text — текст, в котором искать совпадения*

re.sub(pattern, replacement, text)

Вернёмся к нашей задаче. В лемматизированном тексте твита нужно оставить только кириллические символы и пробелы. Чтобы их найти, напишем регулярное выражение.

Оно начинается с символа r, заключается в кавычки и квадратные скобки:

Скопировать кодPYTHON

r'[]'

В квадратных скобках перечисляют все символы, подходящие под шаблон (в любом порядке, без пробелов). Запишем, что ищем буквы от «а» до «я». Они могут быть как в нижнем, так и верхнем регистрах, поэтому получаем:

Скопировать кодPYTHON

*# диапазон букв обозначается дефисом:*

*# а-я — это то же самое, что абвгдежзийклмнопрстуфхцчшщъыьэюя*

r'[а-яА-Я]'

Поскольку Python не знает, что буква Ё должна входить в диапазон, добавим её в нижнем и верхнем регистрах:

Скопировать кодPYTHON

r'[а-яА-ЯёЁ]'

Возьмём исходный текст твита. Под наш шаблон подходят кириллические символы и пробелы, которые как раз нужно оставить. Но если мы вызовем функцию re.sub(), их заменят пробелы. Чтобы указать, что символы под шаблон не подходят, перед набором символов поставим знак «домика» (^):

Скопировать кодPYTHON

*# уже лемматизированный текст*

text = "@first\_timee хоть я и школота, но поверь, у мы то же самый :d общество профилировать предмет типа)"

re.sub(r'[^а-яА-ЯёЁ ]', ' ', text)

Скопировать код

' хоть я и школота но поверь у мы то же самый общество профилировать предмет типа '

Так в тексте остались только кириллические символы и пробелы. После этой операции в тексте можно обнаружить лишние пробелы, для анализа они — помеха. Пробелы устраняются комбинацией функций join() и split().

Возьмём пример текста с лишними пробелами: в середине, начале и конце строки. Методом split() преобразуем его в список. Если не указывать аргументы у split(), он делает разбиение по пробелам или группам пробелов:

Скопировать кодPYTHON

text = " хоть я и школота но поверь "

text.split()

Получили список без пробелов:

Скопировать код

['хоть', 'я', 'и', 'школота', 'но', 'поверь']

Методом join() объединим элементы в строку через пробел:

Скопировать кодPYTHON

" ".join(['хоть', 'я', 'и', 'школота', 'но', 'поверь'])

Пробелов ровно столько, сколько нужно:

Скопировать кодPYTHON

'хоть я и школота но поверь'

#### Задача

Напишите функцию clear\_text(text), которая оставит в тексте только кириллические символы и пробелы. На вход она принимает текст, а возвращает очищенный текст. Дополнительно уберите лишние пробелы.

Напечатайте на экране исходный текст, а затем очищенный и лемматизированный тексты (уже в прекоде).

import pandas as pd

from pymystem3 import Mystem

import re

data = pd.read\_csv('/datasets/tweets.csv')

corpus = list(data['text'])

def lemmatize(text):

m = Mystem()

lemm\_list = m.lemmatize(text)

lemm\_text = "".join(lemm\_list)

return lemm\_text

def clear\_text(text):

template = r'[^а-яА-ЯёЁ ]'

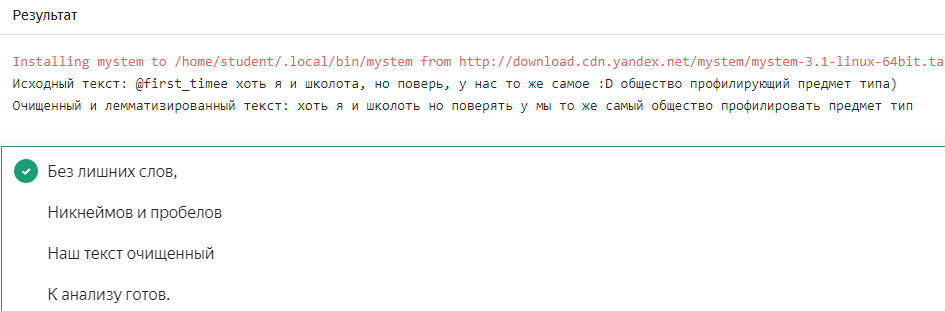
text = re.sub( r'[^а-яА-ЯёЁ ]',' ', text)

text = " ".join(text.split())

return text

print("Исходный текст:", corpus[0])

print("Очищенный и лемматизированный текст:", lemmatize(clear\_text(corpus[0])))



### Мешок слов и N-граммы

Переведём тексты в понятный для машины формат — векторный.

Преобразовать слова в векторы поможет модель **«мешок слов»** (англ. *bag* *of* *words*). Она преобразует текст в вектор, не учитывая порядок слов. Отсюда и название — «мешок».

Возьмём начало стихотворения Игоря Северянина:

Скопировать код

Ананасы в шампанском! Ананасы\*\* в шампанском!

Удивительно вкусно, искристо и остро!

Лемматизируем его:

Скопировать код

ананас в шампанский

ананас в шампанский

удивительно вкусно искристый и остро

Посчитаем количество вхождений каждого слова:

* «ананас», «в», «шампанский» — по 2 раза;
* «удивительно», «вкусно», «искристый», «и», «остро» — по 1 разу.

Получили такой результат:



Вектор этого текста:

Скопировать кодPYTHON

[2, 2, 2, 1, 1, 1, 1, 1]

Когда текстов несколько, мешок слов преобразует их в матрицу. Её строки — это тексты, а столбцы — уникальные слова из всех текстов корпуса. Числа на пересечении строк и столбцов показывают, сколько раз в тексте встречается уникальное слово.

Начало формы

Перед вами корпус. Какой вектор описывает лемматизированный текст «видеть Гpека в pека pак»:

Скопировать кодPYTHON

[

'ехать Гpека чеpез pека',

'видеть Гpека в pека pак',

'сунуть Гpека pука в pека',

'pак за pука Гpека цап'

]

Последовательность слов для вектора такая:

Скопировать кодPYTHON

['ехать', 'Гpека', 'чеpез', 'pека', 'видеть', 'в', 'pак', 'сунуть', 'pука', 'за', 'цап']



[1, 1, 1, 1, 1]



[0, 1, 0, 1, 1]



Тоже правильный ответ

[0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0]

Верно! А если составить векторы для других текстов из этой скороговорки, можно узнать, как ещё перекликаются реки, раки и руки.



Неправильный ответ

[1, 4, 1, 2, 1, 2, 2, 1, 2, 1, 1]

Каждый вектор в мешке слов соответствует одному тексту. Этот вектор описывает весь корпус.

Конец формы

В мешке слов учитывается каждое уникальное слово. Но порядок слов и связи между ними не учитываются. Например, есть такой лемматизированный текст:

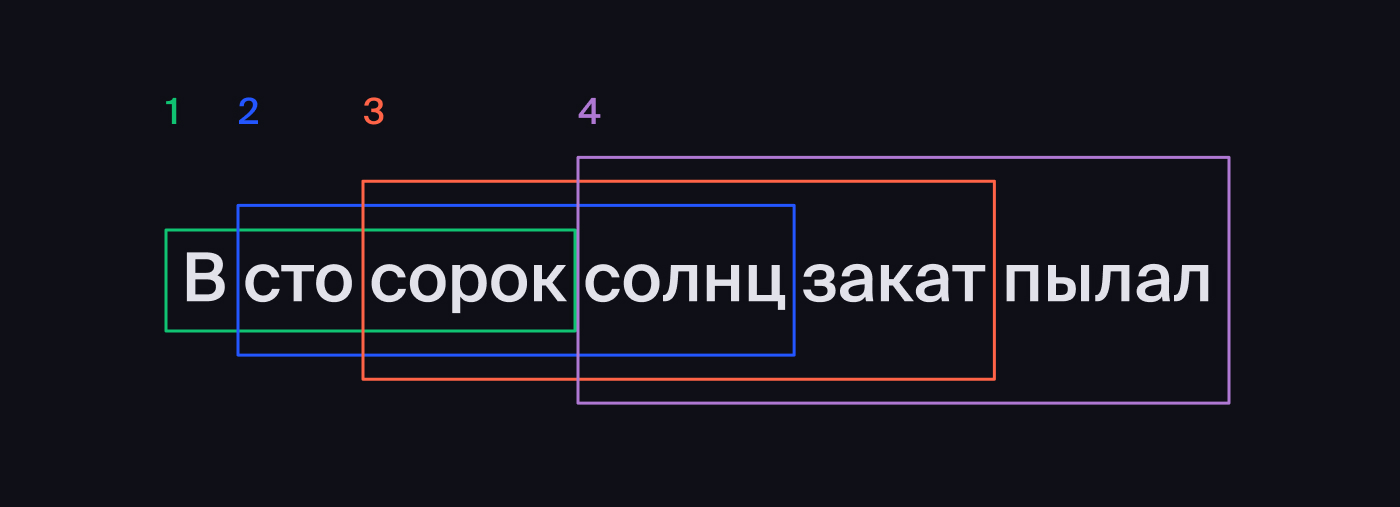
Скопировать код

Фёдор ехать из Москва в Петербург

Его набор слов: «Фёдор», «ехать», «Москва», «Петербург», «из», «в». Так куда едет Фёдор? Чтобы ответить на вопрос, посмотрим на словосочетания, или **N-граммы** (англ. *N-grams*).

*N*-грамма — это последовательность из нескольких слов. *N* указывает на количество элементов и может быть любым. Например, если *N* равно 1, получаются слова, или **униграммы** (лат. *unus*, «один»). При *N*=2 выходят словосочетания из двух слов — **биграммы** (лат. *bis*, «дважды»). Если *N*=3, то это уже **триграммы** (лат. *tres*, *\*\**«три»), т. е. из трёх слов.

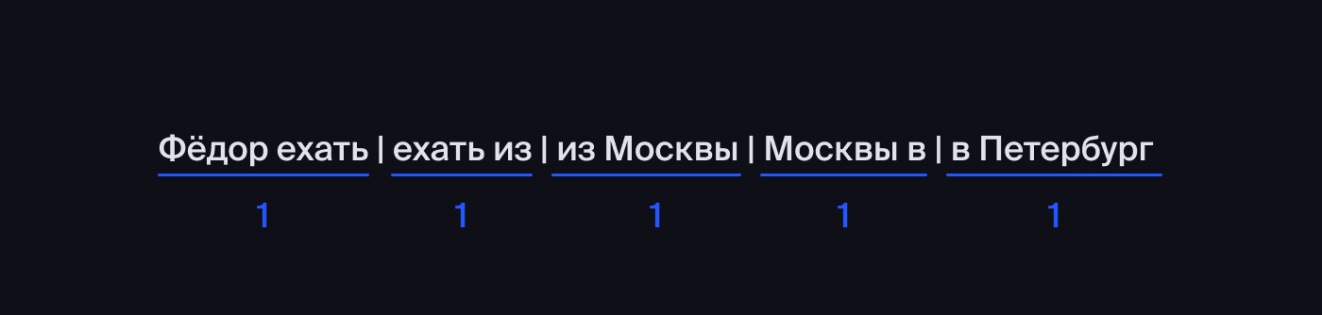
Как выглядят триграммы предложения «В сто сорок солнц закат пылал»?



Мы получили четыре триграммы: «в сто сорок», «сто сорок солнц», «сорок солнц закат», «солнц закат пылал». Слово «закат» не может быть началом следующей триграммы: после него остаётся только одно слово, а нужно два.

Вернёмся к нашему Фёдору и найдём в тексте биграмму. Получится такой набор: «Фёдор ехать», «ехать из», «из Москвы», «Москвы в», «в Петербург». Всё теперь ясно: точки А и Б найдены. Фёдор отправился из Москвы в Петербург.

Аналогично модели «мешок слов» *N*-граммы текста можно преобразовывать в векторы. Например, вектор для текста о Фёдоре выглядит так:



Начало формы

Сколько биграмм получится из цитаты Даниила Хармса. *N*-грамма не должна разбиваться точкой:

Скопировать кодPYTHON

'Предмет обезоружен. Он стручок. Вооружена только куча.'



2



3



Тоже правильный ответ

4

Слово обезоружено. Вооружены N-граммы.



Неправильный ответ

6

Насчитали больше.

### Создание мешка слов

Научимся создавать мешок слов и находить **стоп-слова** (англ. stopwords).

Чтобы преобразовать корпус текстов в мешок слов, обратимся к классу **CountVectorizer()** (англ. count vectorizer, «счётчик слов для создания векторов»). Он находится в модуле **sklearn.feature\_extraction.text** (англ. «построение признаков текста»).

Импортируем его:

Скопировать кодPYTHON

from sklearn.feature\_extraction.text import CountVectorizer

Создадим счётчик:

Скопировать кодPYTHON

count\_vect = CountVectorizer()

Передадим счётчику корпус текстов. Для этого вызовем знакомую вам функцию fit\_transform(). Счётчик выделит из корпуса уникальные слова и посчитает количество их вхождений в каждом тексте корпуса. Отдельные буквы счётчик как слова не учитывает.

Скопировать кодPYTHON

*# bow, от англ. bag of words*

bow = count\_vect.fit\_transform(corpus)

Метод вернёт матрицу, в которой одна строка — это текст, а столбец — уникальное слово из всего корпуса. Число на их пересечении покажет, сколько раз в тексте встречалось нужное слово.

Из прошлого урока возьмём корпус про Греку:

Скопировать кодPYTHON

corpus = [

'ехать Гpека чеpез pека',

'видеть Гpека в pека pак',

'сунуть Гpека pука в pека',

'pак за pука Гpека цап'

]

Создадим для него мешок слов. Чтобы получить размер матрицы, посмотрим атрибут shape:

Скопировать кодPYTHON

bow.shape

Скопировать код

(4, 10)

Всё верно, у нас 4 текста и 10 уникальных слов (предлог «в» не учитывается).

Представим мешок слов в виде матрицы:

Скопировать кодPYTHON

print(bow.toarray())

Скопировать код

array([[0, 1, 0, 0, 1, 1, 0, 0, 0, 1],

[1, 1, 0, 1, 1, 0, 0, 0, 0, 0],

[0, 1, 1, 0, 1, 0, 0, 1, 0, 0],

[1, 0, 1, 0, 1, 0, 1, 0, 1, 0]], dtype=int64)

Список уникальных слов в мешке образует **словарь**. Он хранится в счётчике и вызывается методом **get\_feature\_names()** (англ. «получить имена признаков»):

Скопировать кодPYTHON

count\_vect.get\_feature\_names()

В корпусе про Греку и рака словарь такой:

Скопировать кодPYTHON

['pак',

'pека',

'pука',

'видеть',

'гpека',

'ехать',

'за',

'сунуть',

'цап',

'чеpез']

CountVectorizer() также нужен для расчёта N-грамм. Чтобы он считал словосочетания, укажем размер N-граммы через аргумент **ngram\_range** (англ. «диапазон N-грамм»). Например, если мы ищем словосочетания по два слова в фразе, то диапазон зададим такой:

Скопировать кодPYTHON

count\_vect = CountVectorizer(ngram\_range=(2, 2))

Со словосочетаниями счётчик работает так же, как и со словами.

У больших корпусов и мешки слов выходят большие, но часть слов в них может быть бессмысленной. Например, что можно сказать о тексте по местоимениям, союзам и предлогам? Чаще всего от них можно избавиться, причём тема текста и смысл предложения не изменятся. Когда мешок слов меньше и чище, проще найти слова, важные для классификации текстов.

Чтобы почистить мешок слов, найдём **стоп-слова**, то есть слова без смысловой нагрузки. Их много, и для каждого языка — свои. Разберём пакет stopwords, который находится в модуле nltk.corpus библиотеки **nltk** (англ. Natural Language Toolkit, «инструментарий естественного языка»):

Скопировать кодPYTHON

from nltk.corpus import stopwords

Чтобы пакет заработал, загрузим список стоп-слов:

Скопировать кодPYTHON

import nltk

nltk.download('stopwords')

Вызовем функцию **stopwords.words()**, передадим ей аргумент **'russian'**, то есть русскоязычные стоп-слова:

Скопировать кодPYTHON

stop\_words = set(stopwords.words('russian'))

При создании счётчика передадим список стоп-слов в счётчик векторов CountVectorizer():

Скопировать кодPYTHON

count\_vect = CountVectorizer(stop\_words=stop\_words)

Теперь счётчик знает, какие слова нужно исключить из мешка слов.

Конец формы

#### Задача

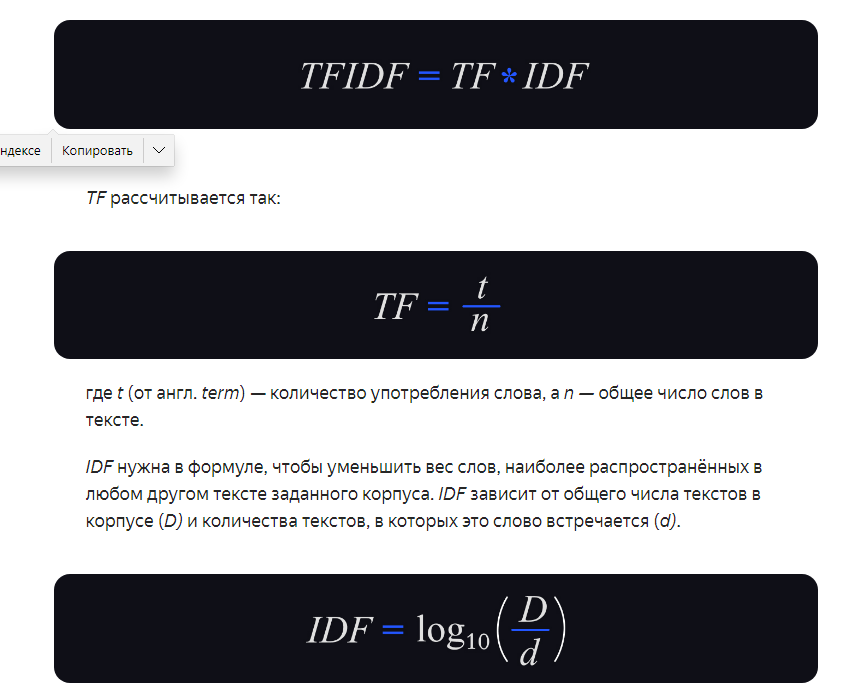
|  |  |
| --- | --- |
| 1.  В файле tweets\_lemm.csv находится датасет, в который мы добавили столбец *lemm\_text* с очищенными и лемматизированными текстами.  Создайте два мешка слов для корпуса твитов: очищенный от стоп-слов и с ними. Напечатайте их размеры на экране (уже в прекоде). | import pandas as pd  from sklearn.feature\_extraction.text import CountVectorizer  from nltk.corpus import stopwords  # < напишите код здесь >  data = pd.read\_csv("/datasets/tweets\_lemm.csv")  corpus = data['lemm\_text'].values.astype('U')  # создайте мешок слов без учёта стоп-слов  count\_vect = CountVectorizer()  bow = count\_vect.fit\_transform(corpus)  # < напишите код здесь >  print("Размер мешка без учёта стоп-слов:", bow.shape)  # создайте новый мешок слов с учётом стоп-слов  stop\_words = set(stopwords.words('russian'))  count\_vect = CountVectorizer(stop\_words=stop\_words)  bow = count\_vect.fit\_transform(corpus)  # < напишите код здесь >  print("Размер мешка с учётом стоп-слов:", bow.shape) |
| 2.  Создайте счётчик *N*-грамм для корпуса твитов. В каждой фразе по два слова. Напечатайте на экране размер *N*-грамм (уже в прекоде). | import pandas as pd  from sklearn.feature\_extraction.text import CountVectorizer  # импортируйте CountVectorizer  data = pd.read\_csv("/datasets/tweets\_lemm.csv")  corpus = list(data['lemm\_text'])  count\_vect = CountVectorizer(ngram\_range=(2, 2))  n\_gramm = count\_vect.fit\_transform(corpus)  # создайте n-грамму n\_gramm, для которой n=2  print("Размер:", n\_gramm.shape) |

### TF-IDF

Мешок слов учитывает частоту употребления слов. Посмотрим, как часто уникальное слово встречается во всём корпусе и в отдельном его тексте.

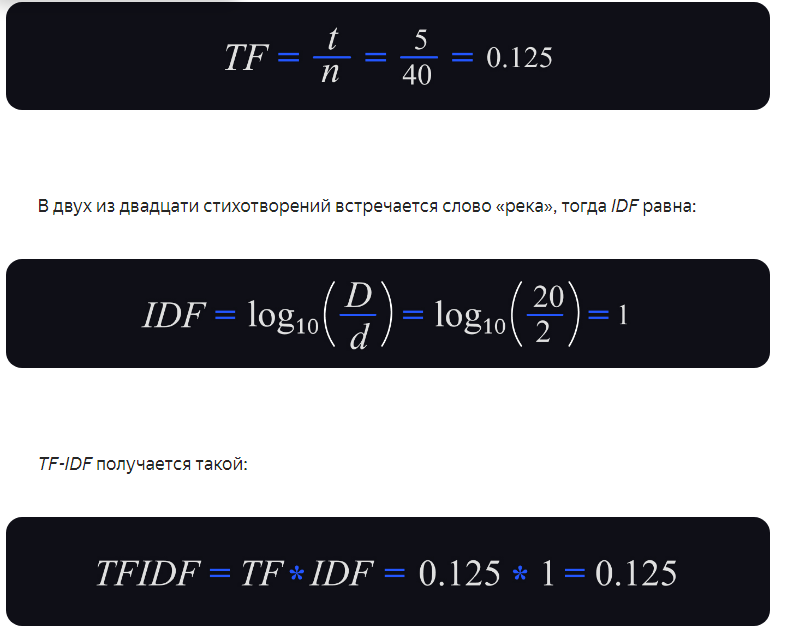
Оценка важности слова определяется величиной **TF-IDF** (от англ. term frequency, «частота терма, или слова»; inverse document frequency, «обратная частота документа, или текста»). То есть TF отвечает за количество упоминаний слова в отдельном тексте, а IDF отражает частоту его употребления во всём корпусе.

Формула TF-IDF такая:

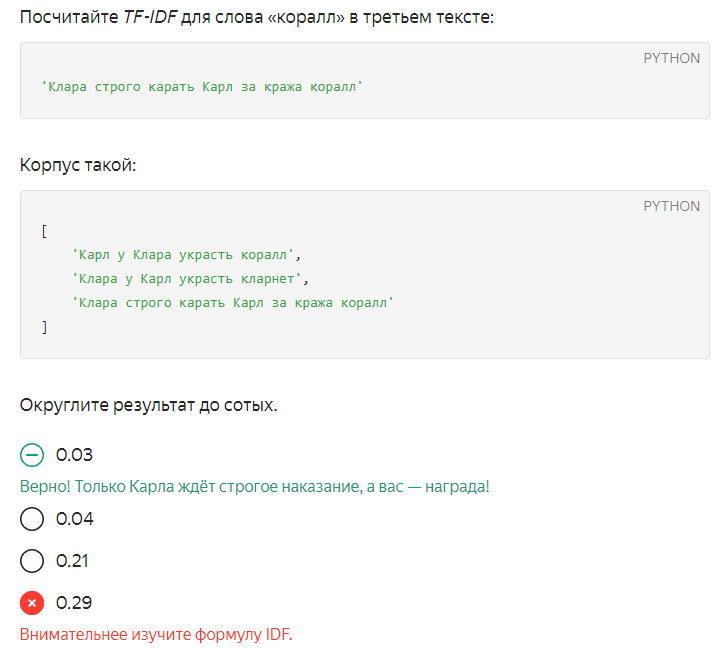


Например, рассмотрим корпус из 20-ти стихотворений. Возьмём первое: в нём 40 слов, нам интересно слово «река». В стихе оно встречается пять раз. Всего в корпусе два стихотворения с «рекой». Рассчитаем *TF-IDF* для слова «река» в первом стихотворении корпуса.

*TF* равна:



Большая величина TF-IDF говорит об уникальности слова в тексте по отношению к корпусу. Чем чаще оно встречается в конкретном тексте и реже в остальных, тем выше значение TF-IDF.



### TF-IDF в sklearn

Вычислим TF-IDF для корпуса текстов.

Рассчитать TF-IDF можно и в библиотеке sklearn. Класс **TfidfVectorizer()** (от англ. «счётчик величин TF-IDF») находится в модуле sklearn.feature\_extraction.text. Импортируем его:

from sklearn.feature\_extraction.text import TfidfVectorizer

По аналогии с CountVectorizer() создадим счётчик, указав в нём стоп-слова:

count\_tf\_idf = TfidfVectorizer(stop\_words=stopwords)

Чтобы посчитать TF-IDF для корпуса текстов, вызовем функцию fit\_transform():

tf\_idf = count\_tf\_idf.fit\_transform(corpus)

Передав TfidfVectorizer() аргумент ngram\_range, можно рассчитать N-граммы.

Если данные разделены на обучающую и тестовую выборки, функцию fit() запускайте только на обучающей. Иначе тестирование будет нечестным: в модели будут учтены частоты слов из тестовой выборки.

#### Задача

Создайте матрицу cо значениями TF-IDF по корпусу твитов. Сохраните её в переменной tf\_idf. Укажите стоп-слова. Напечатайте на экране размер матрицы (уже в прекоде).

import pandas as pd

import nltk

from nltk.corpus import stopwords as nltk\_stopwords

from sklearn.feature\_extraction.text import TfidfVectorizer

data = pd.read\_csv("/datasets/tweets\_lemm.csv")

corpus = data['lemm\_text'].values.astype('U')

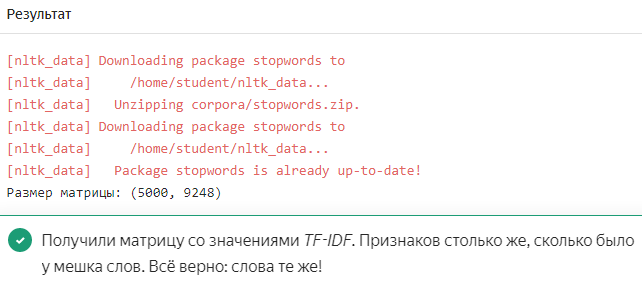
nltk.download('stopwords')

stopwords = set(nltk\_stopwords.words('russian'))

count\_tf\_idf = TfidfVectorizer(stop\_words=stopwords)

tf\_idf = count\_tf\_idf.fit\_transform(corpus)

print("Размер матрицы:", tf\_idf.shape)



### Классификация тональности текста

Для определения тональности применим величины TF-IDF как признаки.

Анализ тональности текста, или **сентимент-анализ** (от англ. sentiment, «настроение»), выявляет эмоционально окрашенные слова. Этот инструмент помогает компаниям оценивать, например, реакцию на запуск нового продукта в интернете. На разбор тысячи отзывов человек потратит несколько часов, а компьютер — пару минут.

Оценить тональность — значит отметить текст как позитивный или негативный. То есть мы решаем задачу классификации, где целевой признак равен «1» для положительного текста и «0» для отрицательного. Признаки — это слова из корпуса и их величины TF-IDF для каждого текста.

#### Задача

Обучите логистическую регрессию так, чтобы она определяла тональность текста.

Подсчитайте величину TF-IDF для текстов. Лемматизированные тексты твитов для обучения находятся в файле tweets\_lemm\_train.csv. Целевой признак вы найдёте в столбце positive.

Обученной моделью классификации определите результаты предсказания для тестовой выборки твитов, которая лежит в файле tweets\_lemm\_test.csv. В этой выборке целевого признака нет. Сохраните предсказания в столбце positive. Таблицу с результатом сохраните как csv-файл, но чтобы тренажёр принял файл, не указывайте расширение (например, назовите файл 'predictions')

Значение accuracy вашей модели должно быть не меньше 0.62.

Файлы: [tweets\_lemm\_train](https://code.s3.yandex.net/datasets/tweets_lemm_train.csv) , [tweets\_lemm\_test](https://code.s3.yandex.net/datasets/tweets_lemm_test.csv).

**Подсказка**

Для предсказания ответов повторно вычислите величину *TF-IDF* для вектора с тестовой выборкой. Примените метод *transform()* к объекту *TfidfVectorizer.*