Практическая работа **2_3_4**

Практическое задание **Классификация текстовых данных**

Для задачи классификация текстовых данных будем использовать набор твитов, относящихся к теме стихийных бедствий, несчастных случаев, происшествий и т.п.

1. Для анализа используем файл ***disasters_social_media.csv***, который загрузим в ***googleColab*** с помощью программного кода:

In [1]:

```
# Загрузить любой файл с компьютера в google.colab
from google.colab import files
uploaded = files.upload()
for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
    name=fn, length=len(uploaded[fn])))
```

Choose File **No file selected**

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving disasters_social_media.csv to disasters_social_media.csv
User uploaded file "disasters_social_media.csv" with length 2208398 bytes
```

1. Экспортируем библиотеки и необходимые модули для обработки текста:

In [2]:

```
import pandas as pd
import numpy as np
import nltk
import re
from nltk.corpus import stopwords
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[2]:

```
True
```

1. Считаем данные

In [3]:

```
df_raw = pd.read_csv('disasters_social_media.csv', encoding='latin-1') # Чтение данных
```

1. Структура данных

1. Посмотрим на структуру данных с помощью функции **.head():** выведем первые пять строк из набора данных:

In [4]:

```
df_raw.head(5)
```

Out[4]:

	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	choose_one	choose_one:confidence	choose_one_g
0	**778243823**	**True**	**golden**	**156**	**NaN**	**Relevant**	**1.0000**	**Relev**
1	**778243824**	**True**	**golden**	**152**	**NaN**	**Relevant**	**1.0000**	**Relev**
2	**778243825**	**True**	**golden**	**137**	**NaN**	**Relevant**	**1.0000**	**Relev**
3	**778243826**	**True**	**golden**	**136**	**NaN**	**Relevant**	**0.9603**	**Relev**
4	**778243827**	**True**	**golden**	**138**	**NaN**	**Relevant**	**1.0000**	**Relev**

1. Выведем несколько последних строк данных:

In [5]:

```
df_raw.tail()
```

Out[5]:

	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	choose_one	choose_one:confidence	choose_g
10871	**778261105**	**True**	**golden**	**100**	**NaN**	**Relevant**	**0.7629**	
10872	**778261106**	**True**	**golden**	**90**	**NaN**	**Relevant**	**0.9203**	
10873	**778261107**	**True**	**golden**	**102**	**NaN**	**Relevant**	**1.0000**	
10874	**778261108**	**True**	**golden**	**96**	**NaN**	**Relevant**	**0.8419**	
10875	**778261109**	**True**	**golden**	**97**	**NaN**	**Relevant**	**0.8812**	

Получили прямоугольную таблицу данных. Ее строки соответствуют *объектам* – *твитам*, а столбцы – *признакам** этих объектов. Объекты также называются наблюдениями* или *примерами* (***samples***), а *признаки* – *атрибутами* (***features***).

атрибутами (***features***).

1. Определим размер массива:

In [6]:

```
df_raw
```

Out[6]:

	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	choose_one	choose_one:confidence	choose_
0	778243823	True	golden	156	NaN	Relevant	1.0000	
1	778243824	True	golden	152	NaN	Relevant	1.0000	
2	778243825	True	golden	137	NaN	Relevant	1.0000	
3	778243826	True	golden	136	NaN	Relevant	0.9603	
4	778243827	True	golden	138	NaN	Relevant	1.0000	
...	...	...	...	...	...	...	...	
10871	778261105	True	golden	100	NaN	Relevant	0.7629	
10872	778261106	True	golden	90	NaN	Relevant	0.9203	
10873	778261107	True	golden	102	NaN	Relevant	1.0000	
10874	778261108	True	golden	96	NaN	Relevant	0.8419	
10875	778261109	True	golden	97	NaN	Relevant	0.8812	

10876 rows × 13 columns

[]:10876 rows × 13 columns

1. Выведем названия столбцов:

In [7]:

```
pd.read_csv('disasters_social_media.csv', nrows=1).columns.tolist()
```

Out[7]:

```
['_unit_id',
 '_golden',
 '_unit_state',
 '_trusted_judgments',
 '_last_judgment_at',
 'choose_one',
 'choose_one:confidence',
 'choose_one_gold',
 'keyword',
 'location',
 'text',
 'tweetid',
 'userid']
```

Нас будут интересовать два столбца: ***'choose_one'*** и ***'text'***. Мерой релевантности конкретного твита в теме, относящейся к стихийным бедствиям, являются значения столбца ***'choose_one'***.

Релевантный – важный, существенный; уместный, актуальный в определенных обстоятельствах; способный служить для точного определения чего-либо.

1. Выведем значения выходного столбца

In [8]:

```
df_raw.choose_one.values # Значения выходного столбца
```

Out[8]:

```
array(['Relevant', 'Relevant', 'Relevant', ..., 'Relevant', 'Relevant',
       'Relevant'], dtype=object)
```

1. Узнаем количество уникальных значений в столбце ***'choose_one'*** с помощью функции ***set()***:

In [9]:

```
set(df_raw.choose_one.values)
```

Out[9]:

```
{"Can't Decide", 'Not Relevant', 'Relevant'}
```

'choose_one' является категориальным признаком. Признак **'choose_one'** называется ответом; признак ***'text'*** – входным признаком. Задача: Требуется по имеющейся таблице научиться по новому объекту, которого нет в таблице, но для которого известны значения входных признаков, по возможности с небольшой ошибкой предсказывать значение выделенного признака (ответа).

Если ответ количественный, то задача называется задачей *восстановления регрессии*. Если ответ категориальный, то задача называется задачей *классификации*.

2. Сокращение размера данных

1. Нас интересует только предсказание «релевантный» или «не релевантный» твит, поэтому удалим строки ***'Can't decide'*** (*'Не могу решить'*):

In [10]:

```
df = df_raw[df_raw.choose_one != "Can't Decide"] # берет только строки, в которых нет "Can't Decide" в столбце choose_one
```

1. Посмотрим, насколько уменьшился размер массива:

In [11]:

```
df
#df.shape
```

Out[11]:

	_unit_id	_golden	_unit_state	_trusted_judgments	_last_judgment_at	choose_one	choose_one:confidence	choose_
0	778243823	True	golden	156	NaN	Relevant	1.0000	
1	778243824	True	golden	152	NaN	Relevant	1.0000	
2	778243825	True	golden	137	NaN	Relevant	1.0000	
3	778243826	True	golden	136	NaN	Relevant	0.9603	
4	778243827	True	golden	138	NaN	Relevant	1.0000	
...	...	...	...	...	...	...	...	
10871	778261105	True	golden	100	NaN	Relevant	0.7629	
10872	778261106	True	golden	90	NaN	Relevant	0.9203	
10873	778261107	True	golden	102	NaN	Relevant	1.0000	
10874	778261108	True	golden	96	NaN	Relevant	0.8419	
10875	778261109	True	golden	97	NaN	Relevant	0.8812	

10860 rows × 13 columns

[]: 10860 rows × 13 columns

Всего **16** строк содержали признак ***'Can't decide'.***

1. Теперь сосредоточимся только на столбцах **'text'** и **'choose_one',** сократив тем самым количество столбцов с **13** до **2:**

In [12]:

```
df = df[['text','choose_one']] # берем только столбцы 'text' и 'choose_one'
df
```

Out[12]:

	text	choose_one
0	Just happened a terrible car crash	Relevant
1	Our Deeds are the Reason of this #earthquake M...	Relevant
2	Heard about #earthquake is different cities, s...	Relevant
3	there is a forest fire at spot pond, geese are...	Relevant
4	Forest fire near La Ronge Sask. Canada	Relevant
...	...	...
10871	M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...	Relevant
10872	Police investigating after an e-bike collided ...	Relevant
10873	The Latest: More Homes Razed by Northern Calif...	Relevant
10874	MEG issues Hazardous Weather Outlook (HWO) htt...	Relevant
10875	#CityofCalgary has activated its Municipal Eme...	Relevant

10860 rows × 2 columns

Перевод некоторых сообщений**:**

Just happened a terrible car crash – Только что произошла ужасная автокатастрофа

Our Deeds are the Reason of this #earthquake – Наши поступки являются причиной этого # землетрясения

Heard about #earthquake is different cities – Слышали о #землетрясении в разных городах**.**

Police investigating after an e-bike collided – Полиция проводит расследование после столкновения электронного велосипеда

1. Преобразуем категориальные признаки в количественные**.** Перейдем от текстовых данных**:** ***'Relevant'*** и ***'Not Relevant'*** к числовым бинарным данным**.**

Закодируем числом **1** релевантные твиты и **0 –** нерелевантные**.**

In [13]:

```
relevance = {'Relevant':1,'Not Relevant':0}
df['relevance'] = df.choose_one.map(relevance) # ставит в соответствие релевантным значениям 1 и нерелевантным - 0
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

1. Просмотрим результаты кодировки**:**

In [14]:

```
df
```

Out[14]:

	text	choose_one	relevance
0	Just happened a terrible car crash	Relevant	1
1	Our Deeds are the Reason of this #earthquake M...	Relevant	1
2	Heard about #earthquake is different cities, s...	Relevant	1

	text	choose_one	relevance
3	there is a forest fire at spot pond, geese are...	Relevant	1
4	Forest fire near La Ronge Sask. Canada	Relevant	1
...	...	...	...
10871	M1.94 [01:04 UTC]?5km S of Volcano Hawaii. htt...	Relevant	1
10872	Police investigating after an e-bike collided ...	Relevant	1
10873	The Latest: More Homes Razed by Northern Calif...	Relevant	1
10874	MEG issues Hazardous Weather Outlook (HWO) htt...	Relevant	1
10875	#CityofCalgary has activated its Municipal Eme...	Relevant	1

10860 rows × 3 columns

3. Обработка текста

Приступим к обработке текста, созданию мешка слов и расчету статистической меры, используемой для оценки важности слова в контексте документа – ***tf-idf***. ***TF-IDF*** (от англ. ***TF – term frequency***, ***IDF – inverse document frequency***) – статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

1. Применим функции для нормализации и токенизации твитов. Пример программного кода извлечения слов (фунция ***extract_words***), удаления стопслов и специальных символов приведен ниже, но вы можете улучшить его, используя навыки, которые получили ранее. Например, указать слова, которые следует игнорировать, или провести операции лемматизации или стемминга. Чем больше вы предварительно обработайте данные, тем лучше может быть ваша модель!

In [15]:

```
def extract_words(sentence):
  ignore_words = stopwords.words()
  words = re.sub("[^\w]", " ", sentence).split() # заменяет все специальные символы на '
'
  words = [word.lower() for word in words]
  words_cleaned = [w.lower() for w in words if w not in ignore_words]
  return words_cleaned
```

4. Создание мешка слов

Нам необходимо знать, как часто каждое отдельное слово появляется в нашем наборе данных. Мы можем представить это в виде словаря, который имеет формат ***{'word':frequency}***, где каждый ключ – это слово, а частоты (***frequency***) – это количество раз появления слова в нашем наборе данных.

Этот словарь известен как ***hash map***, и он может быть построен итеративно путем циклического перебора каждого токена в документе. Если токен отсутствует в ***hash map***, добавим его в ***hash map*** и установим значение его частоты = 1. Если токен уже присутствует в ***hash map***, увеличим его частоту на 1. Это можно сделать с помощью двух функций:

1. создадим функцию **map_book**, которая принимает словарь под названием ***hash_map***, а также токены из твита и обновляет ***hash_map*** каждым словом из токенов;
2. затем создадим функцию (ее можно назвать ***make_hash_map***), которая будет перебирать все твиты и вызывать функцию (***map_book***) для обновления ***hash_map***.
3. 1) создадим функцию ***map_book***

In [16]:

```
#
```

```
# вычисляет частоту появления слов
def map_book(hash_map, tokens):
  if tokens is not None:
    for word in tokens:
      # слово присутствует
      if word in hash_map:
        hash_map[word] = hash_map[word] + 1
      else:
        hash_map[word] = 1
    return hash_map
  else:
    return None
```

1. **2)** Создадим функцию **make_hash_map**

In [17]:

```
def make_hash_map(df):
  hash_map = {}
  for index, row in df.iterrows():
    hash_map = map_book(hash_map, extract_words(row['text']))
  return hash_map
```

18). Предлагаем взять всего **500** самых популярных токенов**:**

In [18]:

```
# определяет функцию frequent_vocab следующими входными данными: word_freq и max_features
def frequent_vocab(word_freq, max_features):
  counter = 0 # инициализирует счетчик значением ноль
  vocab = [] # создает пустой список, который называется vocab
  # перечисляет слова в словаре в порядке убывания частоты
  for key, value in sorted(word_freq.items(), key=lambda item: (item[1], item[0]), rever
se=True):
    # функция цикла для получения топ (max_features) количества слов
    if counter < max_features:
      vocab.append(key)
      counter+=1
    else: break
  return vocab
```

19) Создадим **hash map (**слово **–** частота**)** из токенизированного набора данных**:**

In [19]:

```
hash_map = make_hash_map(df) # создает hash map (слово-частота) из токенизированного набо
ра данных
vocab=frequent_vocab(hash_map, 500)
vocab
```

Out[19]:

```
['co',
 'http',
 'https',
 'û_',
 'amp',
 'like',
 'fire',
 'get',
 'new',
 'via',
 '2',
 'news',
 'people',
 'emergency',
 'video',
 'disaster',
 'would',
 '3',
```

'police',
'still',
'body',
'suicide',
'us',
'1',
'storm',
'first',
'time',
'burning',
'crash',
'rt',
'attack',
'got',
'back',
'california',
'day',
'know',
'fires',
'two',
'buildings',
'going',
'today',
'see',
'world',
'nuclear',
'bomb',
'love',
'hiroshima',
'year',
'full',
'go',
'5',
'û',
'dead',
'youtube',
'life',
'watch',
'old',
'car',
'killed',
'train',
'think',
'4',
'last',
'ûªs',
'accident',
'let',
'good',
'make',
'say',
'gt',
'2015',
'w',
'could',
'way',
'may',
'many',
'families',
'need',
'even',
'years',
'best',
'mass',
'bombing',
'collapse',
'right',
'home',
'water',
'forest',
'death',
'army',

'work',
'black',
'another',
'really',
'please',
'lol',
'wildfire',
'never',
'hot',
'god',
'read',
'mh370',
'look',
'help',
'bomber',
'fatal',
'live',
'northern',
'obama',
'8',
'pm',
'japan',
'11',
'school',
'much',
'feel',
'city',
'wild',
'flood',
'reddit',
'great',
'9',
'stop',
'near',
'night',
'shit',
'said',
'latest',
'injured',
'6',
'typhoon',
'top',
'homes',
'services',
'floods',
'hope',
'fear',
'ever',
'legionnaires',
'atomic',
'house',
'flames',
'10',
'truck',
'state',
'post',
'getting',
'15',
'wreck',
'ass',
'everyone',
'every',
'damage',
'content',
'change',
'red',
'earthquake',
'cross',
'summer',
'since',
'severe',
'plan',

'oil',
'coming',
'7',
'weather',
'military',
'found',
'food',
'ûò',
'thunderstorm',
'heat',
'family',
'debris',
'without',
'next',
'little',
'wounded',
'natural',
'lightning',
'hit',
'gonna',
'evacuation',
'devastated',
'cause',
'always',
'well',
'smoke',
'set',
'times',
'survive',
'national',
'looks',
'free',
'destroyed',
'boy',
'terrorist',
'story',
'photo',
'murder',
'malaysia',
'injuries',
'check',
'bloody',
'bad',
'trapped',
'spill',
'someone',
'service',
'says',
'run',
'refugees',
'liked',
'fucking',
'flooding',
'failure',
'70',
'women',
'tonight',
'show',
'oh',
'loud',
'fall',
'china',
'area',
'30',
'08',
'ûª',
'thunder',
'saudi',
'rain',
'movie',
'landslide',
'game',

'boat',
'around',
'week',
'weapon',
'terrorism',
'rescue',
'p',
'girl',
'confirmed',
'call',
'blood',
'bag',
'air',
'wind',
'survived',
'put',
'migrants',
'injury',
'head',
'hail',
'drought',
'weapons',
'survivors',
'screaming',
'report',
'released',
'panic',
'kills',
'hurricane',
'hostage',
'explosion',
'destroy',
'burned',
'big',
'whole',
'warning',
'save',
'rescued',
'missing',
'hazard',
'breaking',
'attacked',
'thing',
'past',
'mosque',
'made',
'heard',
'fuck',
'destruction',
'bridge',
'apocalypse',
'40',
'violent',
'trauma',
'sinking',
'sinkhole',
'responders',
'rescuers',
'ok',
'hundreds',
'high',
'drowning',
'bags',
'wave',
'self',
'saw',
'real',
'massacre',
'keep',
'island',
'dust',
'demolished',

'deaths',
'crush',
'crashed',
'collapsed',
'catastrophic',
'0',
'white',
'use',
'stock',
'river',
'lives',
'lava',
'hazardous',
'explode',
'evacuate',
'due',
'county',
'blown',
'wreckage',
'update',
'trouble',
'traumatised',
'tragedy',
'structural',
'screamed',
'quarantine',
'outbreak',
'must',
'market',
'long',
'light',
'exploded',
'drown',
'displaced',
'derailment',
'collision',
'collided',
'charged',
'catastrophe',
'bang',
'away',
'august',
'ambulance',
'screams',
'ruin',
'quarantined',
'meltdown',
'least',
'group',
'electrocuted',
'crushed',
'calgary',
'bombed',
'blew',
'bleeding',
'battle',
'05',
'00',
'wounds',
'three',
'suspect',
'riot',
'part',
'panicking',
'obliteration',
'mudslide',
'investigators',
'inundated',
'harm',
'flattened',
'engulfed',
'deluged',

'curfew',
'caused',
'blast',
'better',
'bagging',
'b',
'arson',
'anniversary',
'airplane',
'wrecked',
'windstorm',
'twister',
'thought',
'something',
'sirens',
'seismic',
'sandstorm',
'rioting',
'obliterated',
'obliterate',
'horrible',
'hijacking',
'half',
'devastation',
'detonation',
'desolation',
'danger',
'cliff',
'casualties',
'bus',
'baby',
'woman',
'went',
'twitter',
'sure',
'sunk',
'snowstorm',
'send',
'security',
'road',
'power',
'minute',
'iran',
'hostages',
'derailed',
'derail',
'demolish',
'cyclone',
'collide',
'chemical',
'bioterrorism',
'whirlwind',
'things',
'pkk',
'phone',
'murderer',
'line',
'hijacker',
'heart',
'fedex',
'famine',
'died',
'zone',
'volcano',
'ur',
'tomorrow',
'song',
'rubble',
'pandemonium',
'longer',
'eyewitness',
'detonated',

```
 'detonate',
 'demolition',
 'came',
 'blazing',
 'airport',
 'ûó',
 'tsunami',
 'trying',
 'stretcher',
 'south',
 'sound',
 'razed',
 'kids',
 'hijack',
 'gets',
 'fatalities',
 'fan',
 'evacuated',
 'electrocute',
 'ebay',
 'avalanche',
 'armageddon',
 'annihilated',
 'affected',
 'ûªt',
 'yet',
 'turkey',
 'soon',
 'remember',
 'rainstorm',
 'kill',
 'government',
 'goes',
 'fatality',
 'drowned',
 'deluge',
 'casualty',
 'building',
 'america',
 'west',
 'used',
 'tornado',
 'shoulder',
 'shooting',
 'land',
 'isis',
 'fight',
 'far',
 'blight',
 'bioterror']
```

20.

In [20]:

```
# Определяем функцию bagofwords со следующими входными данными: sentence и words
def bagofwords(sentence, words):
  sentence_words = extract_words(sentence) #токенизирует предложения/твиты и присваивает их значение переменной sentence_words
  # подсчитывает частоту появления слова
  bag = np.zeros(len(words)) #создает массив NumPy, состоящий из нулей с размером len(words)
  # Циклически перебираем данные и добавляем значение 1, когда токен присутствует в твите
  for sw in sentence_words:
    for i,word in enumerate(words):
      if word == sw:
        bag[i] += 1

  return np.array(bag) # возвращает мешок слов для одного твита
```

1. Теперь мы используем эту функцию в цикле для всего набора данных:

In [21]:

```
# настройте массив NumPy с заданным размером для хранения мешка слов
n_words = len(vocab)
n_docs = len(df)
bag_o = np.zeros([n_docs,n_words])
# используйте цикл, чтобы добавить новую строку для каждого твита
for ii in range(n_docs):
  # вызывает предыдущую функцию 'bagofwords'. Обратите внимание навходные данные: sentenc
e и words
  bag_o[ii,:] = bagofwords(df['text'].iloc[ii], vocab)
bag_o.shape
```

Out[21]:

```
(10860, 500)
```

(10860, 500)

5. Использование статистической меры для оценки важности слова в контексте документа – tf-idf

Определим общую частоту и обратную частоту документа

Нам хотелось бы работать со словами, которые несут наибольший смысл в предложениях / твитах. Но возникает вопрос: а слова, которые встречаются чаще всего, они действительно несут наибольший смысл и важны для классификации и построения предсказательной модели?

1. Посмотрим на слова внутри нашего мешка слов. Выведем **20** самых популярных слов и сколько раз они встречаются:

In [22]:

```
sorted(hash_map.items(), key=lambda item: item[1], reverse=True)[:20]
```

Out[22]:

```
[('co', 6800),
 ('http', 6154),
 ('https', 618),
 ('û_', 514),
 ('amp', 510),
 ('like', 492),
 ('fire', 365),
 ('get', 336),
 ('new', 329),
 ('via', 325),
 ('2', 310),
 ('news', 288),
 ('people', 283),
 ('emergency', 229),
 ('video', 227),
 ('disaster', 220),
 ('would', 214),
 ('3', 202),
 ('police', 199),
 ('still', 180)]
```

Как следует из вывода, **20** самых распространенных слов преимущественно не дают никакой информации о твитах. Это остатки **URL**-адресов твитов, а также некоторые общие слова, часто встречающиеся во всем тексте. Их едва ли можно рассматривать в качестве важных характеристик. Для улучшения нашей модели необходимо сделать нечто большее, чем просто выявить наиболее часто встречающиеся слова. Возможно, следует искать слова, которые часто встречаются в некоторых документах, но не во всех.

Простой и удобный способ оценить важность термина для какого-либо документа относительно всех остальных

документов – если слово встречается в каком-либо документе часто, при этом встречаясь редко во всех остальных документах, – это слово имеет большую значимость для того самого документа. Этот способ известен как принцип "общая частота слов **(tf)** – обратная частота документов **(idf)**", или **tfidf.**

Формула **tfidf** имеет вид:

$$w_{i,j} = tf_{i,j} \cdot \log\left(\frac{N}{df_i}\right),$$

i – это индекс слова, **j** – индекс документа, **N** – количество документов (токенов), в которых появляется слово с индексом **i.**

В мешке слов каждая строка – это документ, а каждый столбец – частота слова в этом документе. Это уже часть ***'term frequency'*** (частоты слова – ***tfi j ,***) в ***tfidf***.

Обратная частота документа

1. Вычислим обратную частоту документа по следующему правилу: для каждого слова подсчитаем количество документов, в которых оно появляется, а затем возьмем логарифм от инверсии этого числа.

Построим вектор ***idf***, реализуя следующие два шага:

1. Определим частоту появления для каждого слова.
2. Разделим количество документов **(N)** на количество документов, в которых встречается слово с индексом **i**, и возьмем логарифм от результата.

In [23]:

```
#инициализируйте 2 переменные, представляющие количество твитов (numdocs) и количество то
кенов/слов (numwords)
numdocs, numwords = np.shape(bag_o)

#Переход к формуле tfidf, как указано выше
N = numdocs
word_frequency = np.empty(numwords)

#Подсчет количества документов, в которых появляется слово
for word in range(numwords):
  word_frequency[word]=np.sum((bag_o[:,word]>0))

idf = np.log(N/word_frequency)
idf.shape
idf
```

Out[23]:

```
array([0.64339282, 0.74842242, 2.87774463, 3.07225142, 3.18581871,
       3.15295704, 3.47871106, 3.50288142, 3.5245206 , 3.51829005,
       3.65448692, 3.67244073, 3.6760705 , 3.88566982, 3.91294424,
       4.02498343, 3.9601228 , 4.05109458, 4.08883491, 4.1223576 ,
       4.1223576 , 4.18085381, 4.17484778, 4.14534712, 4.18085381,
       4.18689612, 4.19297517, 4.19909139, 4.25588899, 4.20524526,
       4.2822063 , 4.27556176, 4.26240367, 4.24298559, 4.28889529,
       4.24941648, 4.26240367, 4.3440817 , 4.28889529, 4.30240901,
       4.31610785, 4.32302829, 4.32302829, 4.33701454, 4.31610785,
       4.38756682, 4.37286067, 4.44081133, 4.32999696, 4.35119917,
       4.41003967, 4.44865451, 4.40249247, 4.38756682, 4.41764427,
       4.41764427, 4.50534985, 4.45655969, 4.46452786, 4.52215697,
       4.47256003, 4.46452786, 4.47256003, 4.5137181 , 4.46452786,
       4.50534985, 4.53066766, 4.50534985, 4.53066766, 5.07333389,
       4.5392514 , 4.56545377, 4.54790947, 4.59236123, 4.61071037,
       4.56545377, 4.91081496, 4.62001276, 4.60149371, 4.62001276,
       4.61071037, 4.6294025 , 4.6294025 , 4.6294025 , 4.64845069,
       4.63888124, 4.66786878, 4.67772108, 4.64845069, 4.68767141,
       4.68767141, 4.74954681, 4.71813061, 4.7284934 , 4.7602421 ,
       4.70787411, 4.70787411, 4.71813061, 4.7284934 , 4.74954681,
       4.7389647 , 4.7602421 , 4.74954681, 4.74954681, 4.78198209,
       4.77105302, 4.82693347, 4.80420522, 5.16570721, 4.82693347,
       4.88612235, 4.89839244, 5.05873509, 4.88612235, 4.87400099,
       4.86202479, 4.91081496, 4.85019034, 4.86202479, 5.00238215,
```

4.91081496, 4.88612235, 4.88612235, 4.88612235, 4.91081496,
4.93613277, 4.89839244, 4.92339374, 4.89839244, 4.94903617,
4.94903617, 4.92339374, 4.93613277, 4.94903617, 4.94903617,
4.9887765 , 5.04434635, 4.93613277, 5.42164058, 4.96210825,
4.96210825, 4.9887765 , 4.97535348, 5.00238215, 4.9887765 ,
5.03016172, 4.9887765 , 5.01617547, 5.21530415, 5.01617547,
5.00238215, 5.03016172, 5.01617547, 5.00238215, 5.03016172,
5.07333389, 5.13395851, 5.01617547, 5.05873509, 5.03016172,
5.07333389, 5.03016172, 5.08814897, 5.04434635, 5.03016172,
5.07333389, 5.04434635, 5.05873509, 5.04434635, 5.16570721,
5.07333389, 5.11845432, 5.07333389, 5.07333389, 5.10318685,
5.07333389, 5.10318685, 5.23239858, 5.10318685, 5.11845432,
5.10318685, 5.11845432, 5.10318685, 5.08814897, 5.10318685,
5.10318685, 5.11845432, 5.11845432, 5.10318685, 5.13395851,
5.11845432, 5.4642002 , 5.14970687, 5.19849703, 5.13395851,
5.34159787, 5.16570721, 5.16570721, 5.13395851, 5.16570721,
5.14970687, 5.16570721, 5.21530415, 5.16570721, 5.16570721,
5.24979033, 5.19849703, 5.18196773, 5.19849703, 5.14970687,
5.16570721, 5.18196773, 5.14970687, 5.2674899 , 5.16570721,
5.14970687, 5.21530415, 5.19849703, 5.18196773, 5.18196773,
5.19849703, 5.16570721, 5.16570721, 5.2674899 , 5.21530415,
5.18196773, 5.57926953, 5.24979033, 5.21530415, 5.2674899 ,
5.23239858, 5.32254968, 5.18196773, 5.19849703, 5.18196773,
5.18196773, 5.19849703, 5.19849703, 5.21530415, 5.21530415,
5.23239858, 5.23239858, 5.19849703, 5.2674899 , 5.21530415,
5.24979033, 5.19849703, 5.24979033, 5.21530415, 5.21530415,
5.30385755, 5.21530415, 5.2674899 , 5.23239858, 5.2674899 ,
5.23239858, 5.23239858, 5.23239858, 5.24979033, 5.23239858,
5.23239858, 5.24979033, 5.28550841, 5.24979033, 5.24979033,
5.23239858, 5.23239858, 5.2674899 , 5.2674899 , 5.24979033,
5.30385755, 5.2674899 , 5.24979033, 5.2674899 , 5.24979033,
5.24979033, 5.28550841, 5.28550841, 5.38081859, 5.28550841,
5.28550841, 5.30385755, 5.2674899 , 5.28550841, 5.2674899 ,
5.28550841, 5.28550841, 5.4010213 , 5.30385755, 5.44269399,
5.30385755, 5.30385755, 5.38081859, 5.36101596, 5.30385755,
5.36101596, 5.36101596, 5.30385755, 5.38081859, 5.32254968,
5.38081859, 5.30385755, 5.30385755, 5.34159787, 5.32254968,
5.34159787, 5.32254968, 5.32254968, 5.32254968, 5.30385755,
5.30385755, 5.4642002 , 5.32254968, 5.34159787, 5.44269399,
5.38081859, 5.42164058, 5.36101596, 5.36101596, 5.32254968,
5.32254968, 5.32254968, 5.34159787, 5.32254968, 5.34159787,
5.36101596, 5.36101596, 5.34159787, 5.34159787, 5.34159787,
5.34159787, 5.34159787, 5.36101596, 5.38081859, 5.50865196,
5.34159787, 5.4010213 , 5.34159787, 5.34159787, 5.34159787,
5.36101596, 5.34159787, 5.34159787, 5.62927995, 5.34159787,
5.36101596, 5.34159787, 5.4642002 , 5.34159787, 5.36101596,
5.36101596, 5.36101596, 5.36101596, 5.38081859, 5.38081859,
5.38081859, 5.38081859, 5.4861791 , 5.36101596, 5.36101596,
5.36101596, 5.36101596, 5.55517198, 5.50865196, 5.4010213 ,
5.53164148, 5.53164148, 5.4010213 , 5.38081859, 5.38081859,
5.4010213 , 5.38081859, 5.38081859, 5.38081859, 5.38081859,
5.38081859, 5.38081859, 5.38081859, 5.38081859, 5.38081859,
5.38081859, 5.4010213 , 5.38081859, 5.4642002 , 5.38081859,
5.38081859, 5.38081859, 5.4010213 , 5.4010213 , 5.42164058,
5.4010213 , 5.42164058, 5.4010213 , 5.44269399, 5.4010213 ,
5.4010213 , 5.4010213 , 5.4010213 , 5.4010213 , 5.4010213 ,
5.44269399, 5.4010213 , 5.4010213 , 5.4010213 , 5.42164058,
5.4642002 , 5.4010213 , 5.53164148, 5.4642002 , 5.44269399,
5.44269399, 5.4861791 , 5.44269399, 5.42164058, 5.44269399,
5.42164058, 5.44269399, 5.53164148, 5.44269399, 5.42164058,
5.44269399, 5.44269399, 5.44269399, 5.42164058, 5.42164058,
5.42164058, 5.42164058, 5.42164058, 5.42164058, 5.44269399,
5.4642002 , 5.44269399, 5.50865196, 5.44269399, 5.4642002 ,
5.44269399, 5.50865196, 5.53164148, 5.4642002 , 5.4642002 ,
5.4642002 , 5.50865196, 5.85885439, 5.4861791 , 5.53164148,
5.4642002 , 5.4642002 , 5.4861791 , 5.50865196, 5.4642002 ,
5.4642002 , 5.4642002 , 5.4642002 , 5.4642002 , 5.4861791 ,
5.50865196, 5.50865196, 5.4861791 , 5.50865196, 5.50865196,
5.4861791 , 5.4861791 , 5.53164148, 5.4861791 , 5.53164148,
5.50865196, 5.4861791 , 5.4861791 , 5.4861791 , 5.4861791 ,
5.50865196, 5.4861791 , 5.4861791 , 5.4861791 , 5.57926953,
5.55517198, 5.53164148, 5.53164148, 5.55517198, 5.53164148,

```
       5.55517198, 5.57926953, 5.50865196, 5.50865196, 5.55517198,
       5.50865196, 5.50865196, 5.53164148, 5.50865196, 5.53164148,
       5.53164148, 5.55517198, 5.55517198, 5.55517198, 5.70932266,
       5.60396214, 5.53164148, 5.53164148, 5.53164148, 5.53164148])
```

Завершим расчет ***tfidf*,** умножив наш мешок слов **(**частоту слов**)** на ***idf***

In [24]:

```
#инициализирует массив tfidf
tfidf = np.empty([numdocs, numwords])

#циклически перебирает твиты, перемножая частоту появления слов (представленную мешком сл
ов) с idf
for doc in range(numdocs):
  tfidf[doc, :]=bag_o[doc, :]*idf

tfidf.shape
```

Out[24]:

```
(10860, 500)
```

Массив **tfidf** состоит из значений каждого из **500** токенов в **10860** твитах**. TF-IDF –**универсальная метрика для измерения важности**.**

Пример–пояснение

Пусть в некотором документе X**,** содержащем **100** слов**,** есть слово «интеллект»**,** которое встречается **5** раз**.** Таким образом**, TF** слова «интеллект» равняется **5/100** или **0.05.** А теперь представим**,** что всего у нас есть **1000** документов **(**включая документ X**),** и слово «интеллект» встречается в **10** из них**.** Таким образом**, IDF** слова «интеллект» равняется **lg(1000/10)** или **2.** Таким образом**, TF-IDF** слова «интеллект» равняется **2 * 0.05** или **0.1.**

Вместо документов у нас могут быть какие-нибудь абстрактные категории или конкретные ящики**,** а вместо слов **–** абстрактные объекты или конкретные фрукты**:**

1. Теперь у нас есть массив **tfidf.** Начнем обучать нашу модель и делать прогнозы**.** Обучаться**,** или**,** как говорят**,** строить модель**,** мы будем на обучающей выборке**,** а проверять качество построенной модели **–** на тестовой**.** Подключим библиотеку **scikit-learn,** которая содержит ряд моделей машинного обучения**.**

Подключим библиотеку **scikit-learn,** которая содержит ряд моделей машинного обучения

In [25]:

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split #для разделения данных на обучающий
и тестовый наборы
from sklearn.model_selection import GridSearchCV #чтобы узнать лучший параметр для нашей
модели
```

Пояснения:

– Метод опорных векторов **(Support Vector Machines – SVM) –** это набор методов обучения**,** используемых для классификации**,** регрессии и обнаружения выбросов**.**

– GridSearchCV – состоит из двух частей**: GridSearch** и **CV –** поиск по сетке и перекрестная проверка**.**

В моделе машинного обучения параметры**,** которые необходимо найти**,** называются гиперпараметрами**.** Для выполнения этого поиска можно использовать **Scikit-Learn GridSearchCV.**

– Модуль **train_test_split** разбивает датасет на данные для обучения и тестирования**.** Разбиение на тестовую и обучающую выборку должно быть случайным**.** Обычно используют разбиения в пропорции **50% : 50%, 60% : 40%, 75% : 25%** и т.д**.** Мы воспользуемся функцией **train_test_split** и разобьем данные на обучающую**/**тестовую выборки в отношении **75% : 25%.**

X_train, y_train – это обучающая выборка**; X_test, y_test –** тестовая выборка**.**

In [26]:

```
# разбиваем X_all и y_all на обучающие и тестовые наборы
X_train, X_test, y_train, y_test = train_test_split(tfidf,df['relevance'].values,shuffle
=True)
# выведем количественное соотношение обучающей и тестовой выборок
N_train, _ = X_train.shape
N_test, _ = X_test.shape
print(N_train, N_test)
```

```
8145 2715
```

Оцените, в какой пропорции разбиты данные на тестовую и обучающую выборки. Ответ вставьте в виде комментария

Ответ

Мы получили **8145** данных для обучающей выборки и **2715** для тестовой выборки. Следовательно, так как всего **10860** данных, то соотношение будет как **75% (обучающая): 25% (тестовая).**

In [27]:

```
print (tfidf)
print (df['relevance'])
```

```
[[0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 ...
 [0.64339282 0.74842242 0.         ... 0.         0.         0.        ]
 [0.64339282 0.74842242 0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
0        1
1        1
2        1
3        1
4        1
        ..
10871    1
10872    1
10873    1
10874    1
10875    1
Name: relevance, Length: 10860, dtype: int64
```

на выходе получили сперва массив **tfidf,** который состоит из значений каждого из **500** токенов и релевантный столбец **(**ранее релевантный столбец был преобразован из столбца **choose_one)**

In [28]:

```
print(X_train.shape, X_test.shape)
```

```
(8145, 500) (2715, 500)
```

X обучающие данные с массивом **tfidf,** который состоит из значений каждого из **500** токенов **(8145, 500)** и **X** тестовые данные с массивом **tfidf,** который состоит из значений каждого из **500** токенов **(2715, 500)**

In [29]:

```
print(y_train.shape, y_test.shape)
```

```
(8145,) (2715,)
```

y обучающие данные **y (8145,)** и **y** тестовые данные **(2715,)**

In [30]:

```
# Создаем экземпляр модели
model = SVC()
# обучаем модель на тренировочном наборе - обучение с учителем
model.fit(X_train,y_train)
```

Out[30]:

```
SVC()
```

SVC() - Метод опорных векторов. Этот метод выбран для обучения, то есть метод обучения, используемый для классификации, регрессии и обнаружения выбросов.

In [31]:

```
y_pred = model.predict(X_test) # предсказать значение целевой переменной
print(y_pred)
```

```
[0 1 1 ... 0 1 0]
```

Вывод предсказательных значений **[0 1 1 ... 0 1 0]**

In [32]:

```
# Используем метод score для получения точности модели
score = model.score(X_test, y_test)
print(score)
```

```
0.7863720073664825
```

Определили насколько наша модель хорошо обучилась. Показатель **0.7863720073664825 -** это покатель не совсем идеальный, но достаточный.

In [33]:

```
print('Точность классификатора логистической регрессии на тестовом наборе: {:.3f}'.format
(score))
```

```
Точность классификатора логистической регрессии на тестовом наборе: 0.786
```

In [34]:

```
score = model.score(X_train,y_train)
print(score)
```

```
0.8812768569674647
```

Определили насколько наша модель хорошо предсказала. Показатель **0.8812768569674647 -** это пхороший показатель для предсказания, при условии, что у обучающей была данная оценка ниже. То есть предсказание очень хорошее.

In [35]:

```
score = model.score(X_train,y_train)
print(score)
```

```
0.8812768569674647
```

In [36]:

```
print('Точность классификатора логистической регрессии на обучающем наборе: {:.3f}'.forma
t(score))
```

```
Точность классификатора логистической регрессии на обучающем наборе: 0.881
```

Работа выполнена.