Практическое занятие 2.3.4. 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:

```
# 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])))
```

2. Экспортируем библиотеки и необходимые модули для обработки текста:

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

3. Считаем данные

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

1. Структура данных

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

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

5. Выведем несколько последних строк данных:

```
df_raw.tail()
```

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).

6. Определим размер массива:

```
df_raw
```

```
[]: 10876 rows × 13 columns
```

7. Выведем названия столбцов:

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

```
['_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.

8. Выведем значения выходного столбца

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

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

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

```
[]: {"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

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

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

11. Посмотрим, насколько уменьшился размер массива:

```
df
#df.shape
```

```
[]: 10860 rows × 13 columns
```

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

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

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

df
[]:
```

	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

13. 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.

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

14. Просмотрим результаты кодировки:

```
df
[]:
```

	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
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.

15. Применим функции для нормализации и токенизации твитов. Пример программного кода извлечения слов (фунция `extract_words`), удаления стоп-слов 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!

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

16). 1) создадим функцию map_book

```
# 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
```

17). 2) Создадим функцию make_hash_map

```
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:

```
# определяет функцию frequent_vocab следующими входными данными: wor
d_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: (it
em[1], item[0]), reverse=True):
        # функция цикла для получения топ (max_features) количества с
лов
        if counter < max_features:
            vocab.append(key)
            counter+=1
        else: break
    return vocab
```

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

```
hash_map = make_hash_map(df)    # создает hash map (слово-частота) из
токенизированного набора данных
```

```
vocab=frequent_vocab(hash_map, 500)
vocab
[]:
'volcano',
 'ur',
 'tomorrow',
 'song',
 'rubble',
 'pandemonium',
 'longer',
 'eyewitness',
 'detonated',
 'detonate',
 'demolition',
 'came',
 'blazing',
 'airport',
```

20)

```
# Определяем функцию bagofwords со следующими входными данными: sent
ence и words
def bagofwords(sentence, words):
    sentence_words = extract_words(sentence) #токенизирует предложен
ия/твиты и присваивает их значение переменной sentence_words
    # подсчитывает частоту появления слова
    bag = np.zeros(len(words)) #создает массив NumPy, состоящий из н
улей с размером len(words)
    # Циклически перебираем данные и добавляем значение 1, когда ток
ен 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
```

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

```
# 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 на
 входные данные: sentence и words
    bag_o[ii,:] = bagofwords(df['text'].iloc[ii], vocab)
bag_o.shape
```

```
[]: (10860, 500)
```