Введение в обработку естественного языка

Урок 2. Создание признакового пространства

ДЗ 2

Задание 1.

Задание: обучите три классификатора:

1) на токенах с высокой частотой

2) на токенах со средней частотой

3) на токенах с низкой частотой

Сравните полученные результаты, оцените какие токены наиболее важные для классификации.

Задание 2.

найти фичи с наибольшей значимостью, и вывести их

Задание 3.

1) сравнить count/tf-idf/hashing векторайзеры/полносвязанную сетку (построить classification_report)

2) подобрать оптимальный размер для hashing векторайзера

3) убедиться что для сетки нет переобучения

Выполнил ***Соковнин ИЛ***

Выводы по дз

Задание 1:

Самый лучший результат получался по полному набору токенов. Хороший результат получился при использовании наиболее популярных токенов, как для CountVectorizer, так и для TfidfVectorizer.

Задание 2:

Фичи с наибольшей значимостью несколько отличаются для разных частот.

Задание 3:

1. Лучше всего отработал TfIdfVectorizer.
2. HashingVectorizer приближается к лучшему результату на больших размерах > 10000

В [1]:
```
import pandas as pd
import numpy as np
import re
```

В [2]:
```
# Сброс ограничений на количество символов в записи
pd.set_option('display.max_colwidth', None)
```

В [3]:
```python
with open('./data/corpus', 'r') as f:
    i = 0
    for l in f.readlines():
        if i < 2:
            t = l.split(' ', 1)
            t0 = t[0][-1]
            t1 = t[1][:100]
            print(t[0], '\n', t0, '\n', t1, '... \n')

            i += 1
        else:
            break
```

```
__label__2
 2
 Stuning even for the non-gamer: This sound track was beautiful! It paints the senery in your mind so ...

__label__2
 2
 The best soundtrack ever to anything.: I'm reading a lot of reviews saying that this is the best 'ga ...
```

В [4]:
```python
# Создаём dataframe из файла

with open('./data/corpus', 'r') as f:
    df = pd.DataFrame({'category': t[0][-1], 'text': t[1]} for t in (l.split(' ', 1) for l in f.readlines()))

df.head()
```

Out[4]:

	category	text
0	2	Stuning even for the non-gamer: This sound track was beautiful! It paints the senery in your mind so well I would recomend it even to people who hate vid. game music! I have played the game Chrono Cross but out of all of the games I have ever played it has the best music! It backs away from crude keyboarding and takes a fresher step with grate guitars and soulful orchestras. It would impress anyone who cares to listen! ^_^\n
1	2	The best soundtrack ever to anything.: I'm reading a lot of reviews saying that this is the best 'game soundtrack' and I figured that I'd write a review to disagree a bit. This in my opinino is Yasunori Mitsuda's ultimate masterpiece. The music is timeless and I'm been listening to it for years now and its beauty simply refuses to fade.The price tag on this is pretty staggering I must say, but if you are going to buy any cd for this much money, this is the only one that I feel would be worth every penny.\n
2	2	Amazing!: This soundtrack is my favorite music of all time, hands down. The intense sadness of "Prisoners of Fate" (which means all the more if you've played the game) and the hope in "A Distant Promise" and "Girl who Stole the Star" have been an important inspiration to me personally throughout my teen years. The higher energy tracks like "Chrono Cross ~ Time's Scar~", "Time of the Dreamwatch", and "Chronomantique" (indefinably remeniscent of Chrono Trigger) are all absolutely superb as well.This soundtrack is amazing music, probably the best of this composer's work (I haven't heard the Xenogears soundtrack, so I can't say for sure), and even if you've never played the game, it would be worth twice the price to buy it.I wish I could give it 6 stars.\n
3	2	Excellent Soundtrack: I truly like this soundtrack and I enjoy video game music. I have played this game and most of the music on here I enjoy and it's truly relaxing and peaceful.On disk one. my favorites are Scars Of Time, Between Life and Death, Forest Of Illusion, Fortress of Ancient Dragons, Lost Fragment, and Drowned Valley.Disk Two: The Draggons, Galdorb - Home, Chronomantique, Prisoners of Fate, Gale, and my girlfriend likes ZelbessDisk Three: The best of the three. Garden Of God, Chronopolis, Fates, Jellyfish sea, Burning Orphange, Dragon's Prayer, Tower Of Stars, Dragon God, and Radical Dreamers - Unstealable Jewel.Overall, this is a excellent soundtrack and should be brought by those that like video game music.Xander Cross\n
4	2	Remember, Pull Your Jaw Off The Floor After Hearing it: If you've played the game, you know how divine the music is! Every single song tells a story of the game, it's that good! The greatest songs are without a doubt, Chrono Cross: Time's Scar, Magical Dreamers: The Wind, The Stars, and the Sea and Radical Dreamers: Unstolen Jewel. (Translation varies) This music is perfect if you ask me, the best it can be. Yasunori Mitsuda just poured his heart on and wrote it down on paper.\n

preprocessing

В [5]:
```python
import nltk
from nltk import tokenize as tknz
from string import punctuation
```

В [6]:
```python
stop_words = nltk.corpus.stopwords.words('english')
punctuation_marks = list(punctuation)
noise = set(stop_words + punctuation_marks)
```

В [7]:
```python
def remove_noise(token, noise):
    """
    Удаляем шум из токенов.
    """

    remove_sw = [word for word in token if not word in noise]

    return  remove_sw
```

В [8]:
```python
# лематизация

from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

def get_lemmatizer(words, lemmatizer, pos):
    """
    lemmitization
    """

    lemmas = []
    for word in words:
        lemmas.append(lemmatizer.lemmatize(word, pos = nltk.corpus.wordnet.VERB) )

    return lemmas
```

В [9]:
```python
%%time

# токенизация
# удалим стоп слова и знаки препинания
# лематизация
df['text_tokens'] = df['text'].apply(tknz.word_tokenize)
df['text_tokens'] = df['text_tokens'].apply(remove_noise, noise=noise)
df['text_tokens'] = \
        df['text_tokens'].apply(get_lemmatizer, lemmatizer = WordNetLemmatizer(), pos = wordnet.VERB)
df.head(3)
```

```
Wall time: 10.1 s
```

Out[9]:

	category	text	text_tokens
0	2	Stuning even for the non-gamer: This sound track was beautiful! It paints the senery in your mind so well I would recomend it even to people who hate vid. game music! I have played the game Chrono Cross but out of all of the games I have ever played it has the best music! It backs away from crude keyboarding and takes a fresher step with grate guitars and soulful orchestras. It would impress anyone who cares to listen! ^_^\n	[Stuning, even, non-gamer, This, sound, track, beautiful, It, paint, senery, mind, well, I, would, recomend, even, people, hate, vid, game, music, I, play, game, Chrono, Cross, game, I, ever, play, best, music, It, back, away, crude, keyboarding, take, fresher, step, grate, guitars, soulful, orchestras, It, would, impress, anyone, care, listen, ^_^]
1	2	The best soundtrack ever to anything.: I'm reading a lot of reviews saying that this is the best 'game soundtrack' and I figured that I'd write a review to disagree a bit. This in my opinino is Yasunori Mitsuda's ultimate masterpiece. The music is timeless and I'm been listening to it for years now and its beauty simply refuses to fade.The price tag on this is pretty staggering I must say, but if you are going to buy any cd for this much money, this is the only one that I feel would be worth every penny.\n	[The, best, soundtrack, ever, anything, I, 'm, read, lot, review, say, best, 'game, soundtrack, I, figure, I, 'd, write, review, disagree, bite, This, opinino, Yasunori, Mitsuda, 's, ultimate, masterpiece, The, music, timeless, I, 'm, listen, years, beauty, simply, refuse, fade.The, price, tag, pretty, stagger, I, must, say, go, buy, cd, much, money, one, I, feel, would, worth, every, penny]
2	2	Amazing!: This soundtrack is my favorite music of all time, hands down. The intense sadness of "Prisoners of Fate" (which means all the more if you've played the game) and the hope in "A Distant Promise" and "Girl who Stole the Star" have been an important inspiration to me personally throughout my teen years. The higher energy tracks like "Chrono Cross ~ Time's Scar~", "Time of the Dreamwatch", and "Chronomantique" (indefinably reminescent of Chrono Trigger) are all absolutely superb as well.This soundtrack is amazing music, probably the best of this composer's work (I haven't heard the Xenogears soundtrack, so I can't say for sure), and even if you've never played the game, it would be worth twice the price to buy it.I wish I could give it 6 stars.\n	[Amazing, This, soundtrack, favorite, music, time, hand, The, intense, sadness, ``, Prisoners, Fate, '', mean, 've, play, game, hope, ``, A, Distant, Promise, '', ``, Girl, Stole, Star, '', important, inspiration, personally, throughout, teen, years, The, higher, energy, track, like, ``, Chrono, Cross, Time, 's, Scar~, '', ``, Time, Dreamwatch, '', ``, Chronomantique, '', indefinably, reminescent, Chrono, Trigger, absolutely, superb, well.This, soundtrack, amaze, music, probably, best, composer, 's, work, I, n't, hear, Xenogears, soundtrack, I, ca, n't, say, sure, even, 've, never, play, game, would, worth, twice, price, buy, it.I, wish, I, could, give, 6, star]

ДЗ 2

В [10]:
```python
from collections import Counter
```

В [11]:
```python
# Создадим словарь наших текстов
dictionary = []
for ts in df.text_tokens:
    for t in ts:
        dictionary.append(t)

dictionary[:5]
```

Out[11]:
```
['Stuning', 'even', 'non-gamer', 'This', 'sound']
```

В [12]:
```python
#
# Одной строкой
#
dictionary = [ t for ts in df.text_tokens for t in ts ]
# dictionary[:5]
```

В [13]:
```python
# Подсчитать частому слов в списке и отсортировать по частоте
counts = Counter(dictionary)
# counts.items()
print(dict(list(counts.items())[:5]))
```

```
{'Stuning': 1, 'even': 1249, 'non-gamer': 1, 'This': 3556, 'sound': 645}
```

В [14]:
```python
# Сортировка по частоте
sorted_counts = sorted(counts.items(), key=lambda item: (-item[1]))
sorted_counts[:5]
```

Out[14]:
```
[('I', 21131), ('book', 7347), ("'s", 5758), ('The', 5349), ("n't", 5297)]
```

В [15]:
```python
# Частотный словарь
freq_dictionary = list(tp[0] for tp in sorted_counts)
freq_dictionary[:10]
```

Out[15]:
```
['I', 'book', "'s", 'The', "n't", 'This', "''", 'read', 'It', 'one']
```

В [16]:
```python
#
# То же самое одной строкой
#
freq_dicts = list(tp[0] for tp in sorted(Counter(dictionary).items(), key=lambda x: -x[1]))
# freq_dictionary[:10]
```

В [17]:
```python
# Создадим четыре набора
freq_dicts = {
    'all': set(freq_dicts),
    'high frequency': set(freq_dicts[:len(freq_dicts)//20]),  # < 5%
    'medium frequency': set(freq_dicts[len(freq_dicts)//20 : len(freq_dicts)//5]),  # от 5 до 20%
    'low frequency': set(freq_dicts[len(freq_dicts)//5:]),  # > 20%
}
```

В [18]:
```python
# freq_dicts['high frequency']
```

Создаём и обучаем модель

В [19]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# Извлечение фичей из текстовых данных - векторизаторы
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, HashingVectorizer
```

В [20]:
```python
from sklearn.model_selection import train_test_split

# Создаём тренировочный и тестовый наборы данных
df_train, df_test = train_test_split(df, test_size=0.2, random_state=42)
```

В [21]:
```python
def frequency_filtered_text(df_, freq_type):
    '''
    Фильтрация текстов по частотному словарю
    '''

    filter_text_tokens = []
    for tt in df_['text_tokens']:
        filter_tokens = []
        for t in tt:
            if t in freq_dicts[freq_type]:
                filter_tokens.append(t)
        filter_text_tokens.append(' '.join(filter_tokens))

    return filter_text_tokens
```

В [22]:
```python
# def filtered_text(df_, freq_type):
#     return [ ' '.join(t for t in tt if t in freq_dicts[freq_type]) for tt in df_['text_tokens'] ]
```

В [48]:
```python
freq_type = 'high frequency'
freq_dicts['high frequency']
frequency_filtered_text(df, freq_type)[0]
frequency_filtered_text(df_train, freq_type)[0]
```

Out[48]:
```
"Though one reviewer felt format book poor choice I find perfect I leave copy one bag I pick new two store I find let f
ind I like book little think much cook advance combine heat time come This do many recipes call `` cook rice '' `` cook
'' I find also allow use quickly new note though book mean Many recipes call products lovely lovely book must try live
earth"
```

В [24]:
```
from sklearn.metrics import classification_report

def get_fit_and_test(freq_type):

    # Тренировочный набор данных
    x_train = frequency_filtered_text(df_train, freq_type)
    y_train = df_train['category']

    # Тестовый набор данных
    x_test = frequency_filtered_text(df_test, freq_type)
    y_test = df_test['category']

    return x_train, y_train, x_test, y_test
```

В [25]:
```
# Классификаторы
# from sklearn.linear_model import LogisticRegression
# from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.naive_bayes import GaussianNB
# from sklearn.tree import DecisionTreeClassifier
# from sklearn.svm import SVC
```

В [26]:
```
def print_important_features(vec, model):
    '''
    Находим и выводим фичи с наибольшей значимостью

    '''

    # hasattr(obj, name) - возвращаем флаг, указывающий на то, содержит ли объект указанный атрибут.
    if vec is not None and hasattr(vec, 'get_feature_names'):
        feature_names = vec.get_feature_names()
        # print(feature_names), print(model.coef_[0])

        # Создаём отсортированный  zip-объект ( список кортежей - [(), (), ...] )
        # [(-2.3982348391433557, 'poor'), (-1.9768447288749496, 'worst'), (-1.9529763977726964, 'bore'), ...]
        coefs_with_importances = sorted(zip(model.coef_[0], feature_names))  #  zip-объект (список кортежей) [(), (), ..

        # Выводим n_important фичей
        n_important = 10;

        print("\nФичи с наибольшей положительной значимостью: ")
        for feature in reversed(coefs_with_importances[-n_important:]):
            print(f"{feature[1]} : {feature[0]:.3f}")

        print("\nФичи с наибольшей отрицательной значимостью: ")
        for feature in coefs_with_importances[:n_important]:
            print(f"{feature[1]} : {feature[0]:.3f}")

    print()
```

В [27]:
```
def fit_and_test(freq_type, vec=None, model=None):
    '''
    Обучаем и тестируем модель

    vec - векторайзер
    model - классификатор
    '''

    print('Частоты слов: ' + freq_type)
    x_train, y_train, x_test, y_test = get_fit_and_test(freq_type)

    # Задаём классификатор по умолчанию
    if model == None:
        model = LogisticRegression(random_state=42)  # Логистическая регрессия

    # bow - bag of words (мешок слов)
    bow = vec.fit_transform(x_train)

    # Обучение модели
    model.fit(bow, y_train)

    # Выводим наиболее важные фичи
    print_important_features(vec, model)

    # Генерируем прогнозы
    pred = model.predict(vec.transform(x_test))

    # Строим текстовый отчет по основным показателям классификации.
    # В отчете отображается точность, частота отзыва, значение F1 и другая информация по каждой категории.
    print(classification_report(pred, y_test))
```

В [28]:
```
for freq_type in freq_dicts:
    print(freq_type)
```

```
all
high frequency
medium frequency
low frequency
```

Задание 1.

Задание: обучите три классификатора:

1. на токенах с высокой частотой
2. на токенах со средней частотой
3. на токенах с низкой частотой

Сравните полученные результаты, оцените какие токены наиболее важные для классификации.# CountVectorizer(ngram_range=(1, 1))

Задание 2.

найти фичи с наибольшей значимостью, и вывести их

Задание 3.

1. сравнить count/tf-idf/hashing векторайзеры/полносвязанную сетку (построить classification_report)
2. подобрать оптимальный размер для hashing векторайзера
3. убедиться что для сетки нет переобучения

CountVectorizer

В [29]:
```python
print('CountVectorizer:')
for freq_type in freq_dicts:
    fit_and_test(freq_type, CountVectorizer(ngram_range=(1, 1)))
```

```
CountVectorizer:
Частоты слов: all

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to c
onverge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.h
tml)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modu
les/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names i
s deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out inste
ad.
  warnings.warn(msg, category=FutureWarning)


Фичи с наибольшей положительной значимостью:
excellent : 2.366
perfect : 2.014
government : 1.428
awesome : 1.416
amaze : 1.332
wonderful : 1.328
today : 1.273
love : 1.270
great : 1.266
works : 1.264

Фичи с наибольшей отрицательной значимостью:
poor : -2.398
worst : -1.977
bore : -1.953
waste : -1.934
boring : -1.871
disappoint : -1.641
not : -1.546
disappointment : -1.518
awful : -1.456
useless : -1.443

              precision    recall  f1-score   support

           1       0.86      0.85      0.86      1054
           2       0.83      0.85      0.84       946

    accuracy                           0.85      2000
   macro avg       0.85      0.85      0.85      2000
weighted avg       0.85      0.85      0.85      2000

Частоты слов: high frequency

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to c
onverge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.h
tml)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modu
les/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names i
s deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out inste
ad.
  warnings.warn(msg, category=FutureWarning)


Фичи с наибольшей положительной значимостью:
excellent : 2.262
perfect : 2.046
intense : 1.787
fantastic : 1.650
government : 1.580
brown : 1.572
finest : 1.561
works : 1.496
debut : 1.484
heart : 1.475
```

```
Фичи с наибольшей отрицательной значимостью:
boring : -2.804
worst : -2.477
poor : -2.413
disappointment : -2.364
bore : -2.112
mislead : -2.035
beware : -1.968
errors : -1.848
too : -1.842
waste : -1.814

              precision    recall  f1-score   support

           1       0.83      0.82      0.83      1049
           2       0.81      0.82      0.81       951

    accuracy                           0.82      2000
   macro avg       0.82      0.82      0.82      2000
weighted avg       0.82      0.82      0.82      2000

Частоты слов: medium frequency
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names i
s deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out inste
ad.
  warnings.warn(msg, category=FutureWarning)
```

```
Фичи с наибольшей положительной значимостью:
awsome : 1.708
epic : 1.600
solid : 1.482
medical : 1.454
dog : 1.427
clan : 1.426
attend : 1.405
outstanding : 1.392
investment : 1.389
mature : 1.387

Фичи с наибольшей отрицательной значимостью:
awful : -1.957
wrong : -1.894
junk : -1.830
horribly : -1.743
stain : -1.622
sadly : -1.602
false : -1.600
essentially : -1.583
drivel : -1.574
contrive : -1.566

              precision    recall  f1-score   support

           1       0.65      0.69      0.67       972
           2       0.69      0.64      0.66      1028

    accuracy                           0.66      2000
   macro avg       0.67      0.67      0.66      2000
weighted avg       0.67      0.66      0.66      2000

Частоты слов: low frequency

Фичи с наибольшей положительной значимостью:
heart : 1.703
excelent : 1.161
must : 1.157
highly : 1.119
future : 1.092
own : 1.065
rock : 1.026
eerily : 1.014
wife : 0.982
war : 0.977

Фичи с наибольшей отрицательной значимостью:
money : -1.239
pseudo : -1.151
harlequin : -1.137
crappy : -1.078
bootleg : -1.040
started : -1.021
slow : -1.001
half : -1.000
disapointing : -0.989
boo : -0.989

              precision    recall  f1-score   support
```

```
           1       0.73      0.59      0.65      1271
           2       0.46      0.61      0.53       729

    accuracy                           0.60      2000
   macro avg       0.60      0.60      0.59      2000
weighted avg       0.63      0.60      0.61      2000
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names i
s deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out inste
ad.
  warnings.warn(msg, category=FutureWarning)
```

TfidfVectorizer

```
В [30]: print('TfidfVectorizer:')
for freq_type in freq_dicts:
    fit_and_test(freq_type, TfidfVectorizer(ngram_range=(1, 1)))
```

```
TfidfVectorizer:
Частоты слов: all

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names i
s deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out inste
ad.
  warnings.warn(msg, category=FutureWarning)


Фичи с наибольшей положительной значимостью:
great : 7.558
love : 5.946
excellent : 5.098
best : 4.451
good : 3.941
perfect : 3.495
well : 3.472
easy : 3.065
wonderful : 2.944
must : 2.743

Фичи с наибольшей отрицательной значимостью:
not : -5.304
waste : -5.019
bore : -4.500
poor : -4.355
worst : -4.309
disappoint : -4.233
bad : -4.231
money : -3.696
return : -2.944
nothing : -2.828

              precision    recall  f1-score   support

           1       0.88      0.85      0.86      1071
           2       0.83      0.86      0.85       929

    accuracy                           0.86      2000
   macro avg       0.86      0.86      0.86      2000
weighted avg       0.86      0.86      0.86      2000

Частоты слов: high frequency

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names i
s deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out inste
ad.
  warnings.warn(msg, category=FutureWarning)


Фичи с наибольшей положительной значимостью:
great : 6.629
love : 5.375
excellent : 5.158
best : 4.191
perfect : 3.598
good : 3.567
well : 2.979
wonderful : 2.972
easy : 2.949
amaze : 2.722

Фичи с наибольшей отрицательной значимостью:
not : -5.035
waste : -4.888
bore : -4.641
poor : -4.523
worst : -4.447
disappoint : -4.232
bad : -3.823
money : -3.410
terrible : -2.913
boring : -2.832

              precision    recall  f1-score   support

           1       0.87      0.85      0.86      1063
           2       0.83      0.85      0.84       937

    accuracy                           0.85      2000
   macro avg       0.85      0.85      0.85      2000
weighted avg       0.85      0.85      0.85      2000

Частоты слов: medium frequency
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names i
s deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out inste
ad.
  warnings.warn(msg, category=FutureWarning)


Фичи с наибольшей положительной значимостью:
epic : 1.814
medical : 1.813
awsome : 1.698
attend : 1.637
fast : 1.510
ya : 1.486
ease : 1.473
turner : 1.465
outstanding : 1.422
amazing : 1.400

Фичи с наибольшей отрицательной значимостью:
awful : -2.156
wrong : -2.058
junk : -1.921
false : -1.897
sadly : -1.741
none : -1.673
drivel : -1.642
missing : -1.631
horribly : -1.563
dissapointing : -1.526

              precision    recall  f1-score   support

           1       0.73      0.69      0.71      1100
           2       0.65      0.69      0.67       900

    accuracy                           0.69      2000
   macro avg       0.69      0.69      0.69      2000
weighted avg       0.70      0.69      0.69      2000

Частоты слов: low frequency

Фичи с наибольшей положительной значимостью:
heart : 1.609
well : 1.252
must : 1.195
highly : 1.123
rock : 1.093
excelent : 1.053
great : 0.995
provoking : 0.962
own : 0.960
handy : 0.942

Фичи с наибольшей отрицательной значимостью:
money : -1.368
re : -1.361
what : -1.122
pseudo : -1.090
00 : -1.065
harlequin : -1.059
there : -1.048
either : -1.043
self : -1.041
half : -1.039

              precision    recall  f1-score   support

           1       0.74      0.60      0.66      1289
           2       0.46      0.62      0.53       711

    accuracy                           0.60      2000
   macro avg       0.60      0.61      0.59      2000
weighted avg       0.64      0.60      0.61      2000


C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function get_feature_names i
s deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out inste
ad.
  warnings.warn(msg, category=FutureWarning)
```

HashingVectorizer

В [31]:
```python
for n_features in [100, 200, 500, 1000, 10000, 100000, 1000000]:
    print(f'HashingVectorizer with {n_features} features:')
    fit_and_test('all', HashingVectorizer(analyzer='word', n_features=n_features))
```

```
HashingVectorizer with 100 features:
Частоты слов: all

              precision    recall  f1-score   support

           1       0.66      0.66      0.66      1024
           2       0.64      0.63      0.64       976

    accuracy                           0.65      2000
   macro avg       0.65      0.65      0.65      2000
weighted avg       0.65      0.65      0.65      2000

HashingVectorizer with 200 features:
Частоты слов: all

              precision    recall  f1-score   support

           1       0.69      0.70      0.70      1022
           2       0.68      0.67      0.68       978

    accuracy                           0.69      2000
   macro avg       0.69      0.69      0.69      2000
weighted avg       0.69      0.69      0.69      2000

HashingVectorizer with 500 features:
Частоты слов: all

              precision    recall  f1-score   support

           1       0.78      0.78      0.78      1032
           2       0.77      0.76      0.76       968

    accuracy                           0.77      2000
   macro avg       0.77      0.77      0.77      2000
weighted avg       0.77      0.77      0.77      2000

HashingVectorizer with 1000 features:
Частоты слов: all

              precision    recall  f1-score   support

           1       0.81      0.78      0.79      1078
           2       0.75      0.79      0.77       922

    accuracy                           0.78      2000
   macro avg       0.78      0.78      0.78      2000
weighted avg       0.78      0.78      0.78      2000

HashingVectorizer with 10000 features:
Частоты слов: all

              precision    recall  f1-score   support

           1       0.86      0.85      0.85      1050
           2       0.83      0.84      0.84       950

    accuracy                           0.85      2000
   macro avg       0.85      0.85      0.85      2000
weighted avg       0.85      0.85      0.85      2000

HashingVectorizer with 100000 features:
Частоты слов: all

              precision    recall  f1-score   support

           1       0.86      0.85      0.85      1051
           2       0.83      0.84      0.84       949

    accuracy                           0.84      2000
   macro avg       0.84      0.84      0.84      2000
weighted avg       0.85      0.84      0.85      2000

HashingVectorizer with 1000000 features:
Частоты слов: all

              precision    recall  f1-score   support

           1       0.86      0.85      0.85      1052
           2       0.83      0.84      0.84       948

    accuracy                           0.85      2000
   macro avg       0.85      0.85      0.85      2000
weighted avg       0.85      0.85      0.85      2000
```

Полносвязнная сетка

В [63]:
```
?TextVectorization
```

В [65]:
```
?Sequential
```

В [66]:
```
?Embedding
```

В [115]:
```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Embedding, Flatten
from tensorflow.keras.layers import GlobalAveragePooling1D, Conv1D, GRU, LSTM, Dropout
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization
```

В [160]:
```
def custom_standardization(input_data):

    return input_data


# Create the layer.
vectorize_layer_0 = TextVectorization(  # Текстовый векторизационный слой
        standardize=custom_standardization,
        max_tokens=n_features,
        output_mode='int',
        output_sequence_length=10
    )

# Make a text-only dataset (no labels) and call adapt to build the vocabulary.
text_data = frequency_filtered_text(df, freq_type)
vectorize_layer_0.adapt(text_data)  # создаем "словарь".

print(vectorize_layer_0.get_vocabulary()[:10])
print()

# Create the model that uses the vectorize text layer
model = tf.keras.models.Sequential()

# Start by creating an explicit input layer. It needs to have a shape of
# (1,) (because we need to guarantee that there is exactly one string
# input per batch), and the dtype needs to be 'string'.
model.add(tf.keras.Input(shape=(1,), dtype=tf.string))

# The first layer in our model is the vectorization layer. After this
# layer, we have a tensor of shape (batch_size, max_len) containing vocab
# indices.
model.add(vectorize_layer_0)

# print(vectorize_layer.get_vocabulary()[1], vectorize_layer.get_vocabulary()[921])
input_data = ["The foo go to bar", "I read The book"]
model.predict(input_data)
```

```
['', '[UNK]', 'I', 'book', "'s", 'The', "n't", 'This', "''", 'read']
```

Out[160]:
```
array([[  5,   1,  26,   1, 921,   0,   0,   0,   0,   0],
       [  2,   9,   5,   3,   0,   0,   0,   0,   0,   0]], dtype=int64)
```

В [174]:
```python
# tf.keras.layers.experimental.preprocessing. TextVectorization
# https://spec-zone.ru/tensorflow~2.4/keras/layers/experimental/preprocessing/textvectorization
def custom_standardization(input_data):

    return input_data

max_len = 200  # Sequence length to pad the outputs to.
embedding_dim = 200

vectorize_layer = TextVectorization(  # Текстовый векторизационный слой
    standardize=custom_standardization,
    max_tokens=n_features,  # Maximum vocab size
    output_mode='int',
    output_sequence_length=max_len
)

# Make a text-only dataset (no labels) and call adapt to build the vocabulary.
text_data = frequency_filtered_text(df, freq_type)
vectorize_layer.adapt(text_data)  # создаем "словарь".
print(vectorize_layer.get_vocabulary()[:10])


# ЗАДАЧА КЛАССИФИКАЦИИ ТЕКСТОВЫХ ДАННЫХ С WORD EMBEDDINGS В TENSORFLOW
# https://python-school.ru/blog/nlp-classification-with-emdeddings/
def build_nn_vectorizer(n_features, freq_type='all'):

    # Create the model that uses the vectorize text layer
    model = tf.keras.models.Sequential()

    # Start by creating an explicit input layer. It needs to have a shape of
    # (1,) (because we need to guarantee that there is exactly one string
    # input per batch), and the dtype needs to be 'string'.
    model.add(tf.keras.Input(shape=(1,), dtype=tf.string))

    # The first layer in our model is the vectorization layer. After this
    # layer, we have a tensor of shape (batch_size, max_len) containing vocab
    # indices.
    model.add(vectorize_layer)

    model.add(Embedding(
                input_dim=n_features,  #  размер словаря = n_features
                output_dim=embedding_dim,  # размерность выходной матрицы Embedding.
                input_length=max_len  #  размерность входного слоя.
    ))

    model.add(Flatten())  # слой Flatten, который выпрямляет слой Embdedding;
    model.add(Dense(1, activation='sigmoid'))  # один выходной нейрон с функцией активацией sigmoid,
                                               # который выводит вероятность принадлежности к классу 1 (позитивный отзыв)
                                               # или 0 (негативный отзыв).

#     model.compile(optimizer='adam',  # оптимизатор
#                   loss=tf.keras.losses.BinaryCrossentropy()  # функцию потерь
#                  )

    return model
```

```
['', '[UNK]', 'I', 'book', "'s", 'The', "n't", 'This', "''", 'read']
```

В [175]:
```python
# from tensorflow.keras.preprocessing.sequence import pad_sequences

x_train = frequency_filtered_text(df_train, freq_type)
y_train = df_train['category'].tolist()

x_test = frequency_filtered_text(df_test, freq_type)
y_test = df_test['category'].tolist()


print(np.array(x_train).shape, np.array(y_test).shape)
x_train[0], y_train[0]
```

```
(8000,) (2000,)
```

Out[175]:
```
("Though one reviewer felt format book poor choice I find perfect I leave copy one bag I pick new two store I find let
find I like book little think much cook advance combine heat time come This do many recipes call `` cook rice '' `` coo
k '' I find also allow use quickly new note though book mean Many recipes call products lovely lovely book must try liv
e earth",
 '2')
```

В [176]:
```
model = build_nn_vectorizer(10000)
model.summary()
```

```
Model: "sequential_58"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 text_vectorization_62 (Text  (None, 200)              0
 Vectorization)

 embedding_21 (Embedding)    (None, 200, 200)          2000000

 flatten_12 (Flatten)        (None, 40000)             0

 dense_21 (Dense)            (None, 1)                 40001

=================================================================
Total params: 2,040,001
Trainable params: 2,040,001
Non-trainable params: 0
_________________________________________________________________
```

В [182]:
```
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy'])

history = model.fit(
    x_train,
    y_train,
    epochs=15,
    validation_data=(x_test, y_test),
    batch_size=128)
```

```
      return _run_code(code, main_globals, None,
    File "C:\ProgramData\Anaconda3\lib\runpy.py", line 87, in _run_code
      exec(code, run_globals)
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py", line 16, in <module>
      app.launch_new_instance()
    File "C:\ProgramData\Anaconda3\lib\site-packages\traitlets\config\application.py", line 845, in launch_instance
      app.start()
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelapp.py", line 612, in start
      self.io_loop.start()
    File "C:\ProgramData\Anaconda3\lib\site-packages\tornado\platform\asyncio.py", line 149, in start
      self.asyncio_loop.run_forever()
    File "C:\ProgramData\Anaconda3\lib\asyncio\base_events.py", line 570, in run_forever
      self._run_once()
    File "C:\ProgramData\Anaconda3\lib\asyncio\base_events.py", line 1859, in _run_once
      handle._run()
    File "C:\ProgramData\Anaconda3\lib\asyncio\events.py", line 81, in _run
      self._context.run(self._callback, *self._args)
    File "C:\ProgramData\Anaconda3\lib\site-packages\tornado\ioloop.py", line 690, in <lambda>
      lambda f: self._run_callback(functools.partial(callback, future))
    File "C:\ProgramData\Anaconda3\lib\site-packages\tornado\ioloop.py", line 743, in _run_callback
```

В [ ]:

В [ ]:

В [ ]:

В [ ]:

Выводы

Задание 1:

Самый лучший результат получался по полному набору токенов. Хороший результат получился при использовании наиболее популярных токенов, как для CountVectorizer, так и для TfidfVectorizer.

Задание 2:

Фичи с наибольшей значимостью несколько отличаются для разных частот.

Задание 3:

1. Лучше всего отработал TfIdfVectorizer.
2. HashingVectorizer приближается к лучшему результату на больших размерах > 10000

```
В [ ]:
```