In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
%matplotlib inline
```

In [2]:
```
# Начальная форма ДатаФрейма
df = pd.DataFrame(columns = ['ids', 'text', 'score'])
for bar in ['train/', 'test/']:
    for ton in ['pos', 'neg']:
        tmp_df = pd.DataFrame()

        os.chdir(bar+ton)
        files = os.listdir()

        ids = []
        feedbacks = []
        scores = []

        for file in files:
            feedback=''
            tmp_id = file.split('_')[0]
            tmp_score = file.split('_')[1][:-4]
            with open(file, encoding='utf-8') as f:
                for line in f:
                    feedback += line
            ids.append(tmp_id)
            feedbacks.append(feedback)
            scores.append(tmp_score)
        os.chdir('../..')

        tmp_df['ids'] = ids
        tmp_df['text'] = feedbacks
        tmp_df['score'] = scores
        df = df.append(tmp_df)
```

In [3]:
```
print('df_shape: {}'.format(df.shape))
df.head()
```

```
df_shape: (50000, 3)
```

Out[3]:

	ids	text	score
0	0	Bromwell High is a cartoon comedy. It ran at t...	9
1	10000	Homelessness (or Houselessness as George Carli...	8
2	10001	Brilliant over-acting by Lesley Ann Warren. Be...	10
3	10002	This is easily the most underrated film inn th...	7
4	10003	This is not the typical Mel Brooks film. It wa...	8

Посмотрев на содержимое отзывов, можно заметить, что при парсинге захватывались различные теги, и предобработка данных не провалидась.

In [4]:
```
np.seed = 42
df.iloc[np.random.randint(50_000), 1]
```

Out[4]:
```
"Please -- if you haven't attempted to sit through this garbage and are considering viewing this flick/mini-series -- d
o yourself a favor and find anything else to do. Floss your teeth, start learning to play the cello, beat your dog -- a
nything you choose will be time better spent than watching this junk. This is not a bad movie that you can get a few ch
uckles out of -- it simply sucks in every way possible. Just boring from beginning to end.<br /><br />And for those ani
mal lovers out there that feel my comment above is insensitive -- if your dog could speak, he or she would beg for a be
ating rather than suffer through watching this mess."
```

Предварительная чистка данных

In [5]:
```
from bs4 import BeautifulSoup
import re
```

In [6]:
```
# Удаление тегов
def strip_html(text):
    soup = BeautifulSoup(text, 'html.parser')
    return soup.get_text()


# Удаление квадратных скобок
def remove_square_brackets(text):
    return re.sub('\[[^]]*\]', '', text)


# Общая чистка
def clean_text(text):
    text = strip_html(text)
    text = remove_square_brackets(text)
    return text

df['text'] = df['text'].apply(clean_text)
```

In [7]:
```
# Глубокая чистка
def deep_cleaning(text, remove_digits=True):
    pattern = r'[^a-zA-z0-9\s]'
    text = re.sub(pattern, '', text)
    return text

df['text'] = df['text'].apply(deep_cleaning)
```

Преобразование данных

Для начала стоит перемешать данные.
Векторизуем все наше множество слов с помощью ***TF-IDF*** преобразований.
Также используем **n-граммы** длинны два (*биграммы*).

In [8]:
```
df = df.sample(frac=1)
df.head()
```

Out[8]:

	ids	text	score
7716	5696	It kept my attention to the end however withou...	4
7218	5247	This film came recommended as a good action fi...	4
3788	215	The word classic is thrown around too loosely ...	8
1245	11120	Film follows a bunch of students in the NYC Hi...	7
9871	7635	This is one of the worst movie I have ever see...	1

In [9]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
```

In [10]:
```
train_score=df.score[:40000]
train_review=df.text[:40000]
test_score=df.score[40000:]
test_review=df.text[40000:]
```

In [11]:
```
tf_idf_vec = TfidfVectorizer(min_df = 20, max_df = 0.5, ngram_range=(1, 2))

matrix_train_review = tf_idf_vec.fit_transform(train_review)
matrix_test_reiew = tf_idf_vec.transform(test_review)

matrix_train_review.shape, matrix_test_reiew.shape
```

Out[11]:
```
((40000, 60536), (10000, 60536))
```

Обучение

Выбор пал на **логистическую регрессию** потому, что она хорошо спавляется с сильно разреженными данными. В виду размеров обучающей выборки можно не использовать кросс-валидацию.

In [12]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

In [13]:
```
logit = LogisticRegression(C = 1, random_state=42, max_iter=1000)
logit.fit(matrix_train_review, train_score)
tmp = logit.predict(matrix_train_review)
print("overfit?\naccuracy: ",accuracy_score(list(tmp), train_score))

predict = logit.predict(matrix_test_reiew)
```

```
overfit?
accuracy:  0.771225
```

Отчёт по качеству определения оценки отзыва

Стоит заметить, что изначальной задачей является классификация тональноси отзыва. Также оценка отзыва более субъективное понятие, чем тональность так, как границы у окенок одной и той же тональности очень размыты. Можно предположить, что при размечивании разчными ассесорами, схожие отзывы могли получить различные оценки, но иметь одинаковые тональности.

In [14]:
```
print(classification_report(test_score, predict))
```

```
              precision    recall  f1-score   support

           1       0.56      0.84      0.67      2066
          10       0.51      0.80      0.62      1996
           2       0.33      0.07      0.12       914
           3       0.27      0.16      0.20       997
           4       0.40      0.36      0.38      1079
           7       0.35      0.31      0.33       896
           8       0.31      0.27      0.29      1166
           9       0.27      0.07      0.11       886

    accuracy                           0.46     10000
   macro avg       0.38      0.36      0.34     10000
weighted avg       0.41      0.46      0.41     10000
```

In [15]:
```
print("Качество определения тональности\n")
y_true = pd.Series(test_score).apply(lambda x: 1*np.sign(int(x)-5))
y_pred = pd.Series(predict).apply(lambda x: 1*np.sign(int(x)-5))
print(classification_report(y_true, y_pred))
```

```
Качество определения тональности

              precision    recall  f1-score   support

          -1       0.91      0.88      0.90      5056
           1       0.88      0.91      0.90      4944

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```

Как можно заметить, различные метрики качества (полнота, точность, f- мера и доля правильных отетов) дают примерно одинаковое качество классификации, равное 0,90

In [16]:
```
df['score'].value_counts().plot(kind = 'bar')
plt.title("scores distribution")
plt.xlabel('scores')
plt.ylabel('count')
```

Out[16]:
```
Text(0, 0.5, 'count')
```

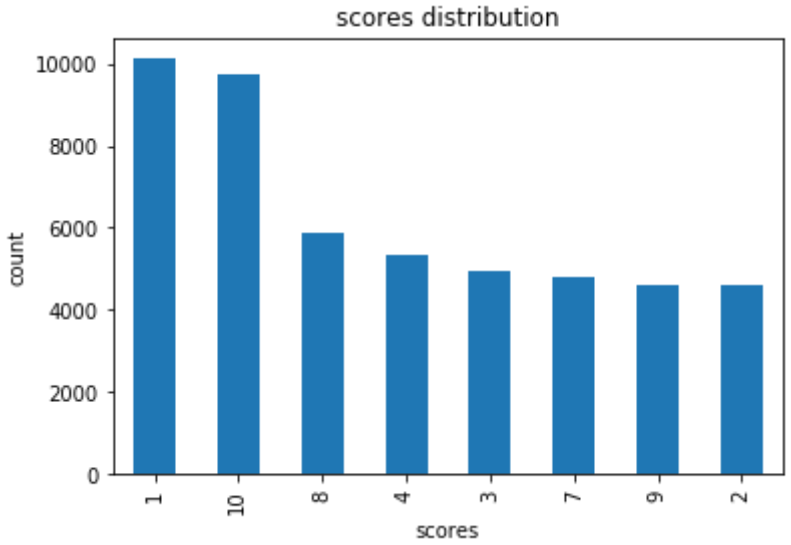

Рисунок выше наглядно отображает неравномерное распределение оценок, что может влиять на качество решения многоклассовой классификации, что в свою очередь косвенно может влиять на качество конечной модели.

Запекание (сериализация) модели

In [17]:
```
class Classifier():
    def __init__(self, model, tf_idf_vectorizer):
        self.logit = model
        self.tf_idf_vectorizer = tf_idf_vectorizer


    def strip_html(self, text):
        soup = BeautifulSoup(text, 'html.parser')
        return soup.get_text()


    # Удаление квадратных скобок
    def remove_square_brackets(self, text):
        return re.sub('\[[^]]*\]', '', text)


    # Общая чистка
    def clean_text(self, text):
        text = self.strip_html(text)
        text = self.remove_square_brackets(text)
        return text


    # Глубокая чистка
    def deep_cleaning(self, text, remove_digits=True):
        pattern = r'[^a-zA-z0-9\s]'
        text = re.sub(pattern, '', text)
        return text


    def processing (self, text):
        text = self.clean_text(text)
        text = self.deep_cleaning(text)
        text_vec = self.tf_idf_vectorizer.transform([text,])
        score  = int(self.logit.predict(text_vec))
        tone =  1*np.sign(score-5)
        return (tone, score)
```

In [18]:
```
import pickle

clf_model = Classifier(logit, tf_idf_vec)
filename = 'logit_model.sav'
pickle.dump(clf_model, open(filename, 'wb'))
```

In [ ]:
```

```