МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра информационных систем и цифровых технологий

ОТЧЕТ
по лабораторной работе № 9
на тему: «**Работа с текстовыми данными**»
по дисциплине: «Машинное обучение»

Выполнили: Шорин В.Д. Шифр: 171406
Кожухова О.А. Шифр: 170582
Институт приборостроения, автоматизации и информационных технологий
Направление: 09.03.04 «Программная инженерия»
Группа: 71-ПГ
Проверил: Митин А.А.

Отметка о зачете:

Дата: «____» _____ 2020 г.

Орел, 2020 г.

Задание

1 Книга Андреас Мюллер, Сара Гвидо «Введение в машинное обучение с помощью Python», стр. 345 – 378.

2 Изучить теоретический материал.

3 Построить модели работы с текстовыми данными и реализовать их средствами языка Python.

4 Проанализировать полученные результаты.

Ход работы

In [1]:
```
!tree -L 2 aclImdb
```

```
/bin/bash: tree: команда не найдена
```

In [2]:
```
from sklearn.datasets import load_files
reviews_train = load_files("aclImdb/train/")
# load_files возвращает коллекцию, содержащую обучающие тексты и обучающие метки
text_train, y_train = reviews_train.data, reviews_train.target
print("тип text_train: {}".format(type(text_train)))
print("длина text_train: {}".format(len(text_train)))
print("text_train[1]:\n{}".format(text_train[1]))
```

```
тип text_train: <class 'list'>
длина text_train: 25000
text_train[1]:
b'Words can\'t describe how bad this movie is. I can\'t explain it by writing only. You have too see it for yourse
lf to get at grip of how horrible a movie really can be. Not that I recommend you to do that. There are so many cl
ich\xc3\xa9s, mistakes (and all other negative things you can imagine) here that will just make you cry. To start
with the technical first, there are a LOT of mistakes regarding the airplane. I won\'t list them here, but just me
ntion the coloring of the plane. They didn\'t even manage to show an airliner in the colors of a fictional airlin
e, but instead used a 747 painted in the original Boeing livery. Very bad. The plot is stupid and has been done ma
ny times before, only much, much better. There are so many ridiculous moments here that i lost count of it really
early. Also, I was on the bad guys\' side all the time in the movie, because the good guys were so stupid. "Execut
ive Decision" should without a doubt be you\'re choice over this one, even the "Turbulence"-movies are better. In
fact, every other movie in the world is better than this one.'
```

In [3]:
```
text_train = [doc.replace(b"<br />", b" ") for doc in text_train]
```

In [6]:
```
import mglearn
import matplotlib.pyplot as plt
import numpy as np

print("Количество примеров на класс (обучение): {}".format(np.bincount(y_train)))
```

```
Количество примеров на класс (обучение): [12500 12500]
```

In [37]:
```
reviews_test = load_files("aclImdb/test/")
text_test, y_test = reviews_test.data, reviews_test.target
print("Количество документов в текстовых данных: {}".format(len(text_test)))
print("Количество примеров на класс (тест): {}".format(np.bincount(y_test)))
text_test = [doc.replace(b"<br />", b" ") for doc in text_test]
```

```
Количество документов в текстовых данных: 25000
Количество примеров на класс (тест): [12500 12500]
```

In [8]:
```
bards_words =["The fool doth think he is wise,",
"but the wise man knows himself to be a fool"]
```

In [9]:
```
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer()
vect.fit(bards_words)
```

Out[9]:
```
CountVectorizer()
```

In [10]:
```
print("Размер словаря: {}".format(len(vect.vocabulary_)))
print("Содержимое словаря:\n {}".format(vect.vocabulary_))
```

```
Размер словаря: 13
Содержимое словаря:
 {'the': 9, 'fool': 3, 'doth': 2, 'think': 10, 'he': 4, 'is': 6, 'wise': 12, 'but': 1, 'man': 8, 'knows': 7, 'hims
elf': 5, 'to': 11, 'be': 0}
```

In [11]:
```
bag_of_words = vect.transform(bards_words)
print("bag_of_words: {}".format(repr(bag_of_words)))
```

```
bag_of_words: <2x13 sparse matrix of type '<class 'numpy.int64'>'
        with 16 stored elements in Compressed Sparse Row format>
```

In [12]:
```
print("Плотное представление bag_of_words:\n{}".format(
bag_of_words.toarray()))
```

```
Плотное представление bag_of_words:
[[0 0 1 1 1 0 1 0 0 1 1 0 1]
 [1 1 0 1 0 1 0 1 1 1 0 1 1]]
```

In [13]:
```
vect = CountVectorizer().fit(text_train)
X_train = vect.transform(text_train)
print("X_train:\n{}".format(repr(X_train)))
```

```
X_train:
<25000x74849 sparse matrix of type '<class 'numpy.int64'>'
        with 3431196 stored elements in Compressed Sparse Row format>
```

In [14]:
```
feature_names = vect.get_feature_names()
print("Количество признаков: {}".format(len(feature_names)))
print("Первые 20 признаков:\n{}".format(feature_names[:20]))
print("Признаки с 20010 по 20030:\n{}".format(feature_names[20010:20030]))
print("Каждый 2000-й признак:\n{}".format(feature_names[::2000]))
```

```
Количество признаков: 74849
Первые 20 признаков:
['00', '000', '0000000000001', '00001', '00015', '000s', '001', '003830', '006', '007', '0079', '0080', '0083', '0
093638', '00am', '00pm', '00s', '01', '01pm', '02']
Признаки с 20010 по 20030:
['dratted', 'draub', 'draught', 'draughts', 'draughtswoman', 'draw', 'drawback', 'drawbacks', 'drawer', 'drawers',
'drawing', 'drawings', 'drawl', 'drawled', 'drawling', 'drawn', 'draws', 'draza', 'dre', 'drea']
Каждый 2000-й признак:
['00', 'aesir', 'aquarian', 'barking', 'blustering', 'bête', 'chicanery', 'condensing', 'cunning', 'detox', 'drape
r', 'enshrined', 'favorit', 'freezer', 'goldman', 'hasan', 'huitieme', 'intelligible', 'kantrowitz', 'lawful', 'ma
ars', 'megalunged', 'mostey', 'norrland', 'padilla', 'pincher', 'promisingly', 'receptionist', 'rivals', 'schnaa
s', 'shunning', 'sparse', 'subset', 'temptations', 'treatises', 'unproven', 'walkman', 'xylophonist']
```

In [16]:
```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
scores = cross_val_score(LogisticRegression(max_iter=100000), X_train, y_train, cv=5)
print("Средняя правильность перекр проверки: {:.2f}".format(np.mean(scores)))
```

```
Средняя правильность перекр проверки: 0.88
```

In [46]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}
grid = GridSearchCV(LogisticRegression(max_iter=100000), param_grid, cv=5)
grid.fit(X_train, y_train)
print("Наилучшее значение перекрестной проверки: {:.2f}".format(grid.best_score_))
print("Наилучшие параметры: ", grid.best_params_)
```

```
Наилучшее значение перекрестной проверки: 0.89
Наилучшие параметры:  {'C': 0.1}
```

```
X_train с min_df: <25000x27271 sparse matrix of type '<class 'numpy.int64'>'
        with 3354014 stored elements in Compressed Sparse Row format>
```

In [48]:
```
vect = CountVectorizer(min_df=5).fit(text_train)
X_train = vect.transform(text_train)
print("X_train с min_df: {}".format(repr(X_train)))
```

```
X_train с min_df: <25000x27271 sparse matrix of type '<class 'numpy.int64'>'
        with 3354014 stored elements in Compressed Sparse Row format>
```

In [49]:
```
feature_names = vect.get_feature_names()
print("Первые 50 признаков:\n{}".format(feature_names[:50]))
print("Признаки с 20010 по 20030:\n{}".format(feature_names[20010:20030]))
print("Каждый 700-й признак:\n{}".format(feature_names[::700]))
```

```
Первые 50 признаков:
['00', '000', '007', '00s', '01', '02', '03', '04', '05', '06', '07', '08', '09', '10', '100', '1000', '100th', '1
01', '102', '103', '104', '105', '107', '108', '10s', '10th', '11', '110', '112', '116', '117', '11th', '12', '12
0', '12th', '13', '135', '13th', '14', '140', '14th', '15', '150', '15th', '16', '160', '1600', '16mm', '16s', '16
th']
Признаки с 20010 по 20030:
['repentance', 'repercussions', 'repertoire', 'repetition', 'repetitions', 'repetitious', 'repetitive', 'rephras
e', 'replace', 'replaced', 'replacement', 'replaces', 'replacing', 'replay', 'replayable', 'replayed', 'replayin
g', 'replays', 'replete', 'replica']
Каждый 700-й признак:
['00', 'affections', 'appropriately', 'barbra', 'blurbs', 'butchered', 'cheese', 'commitment', 'courts', 'deconstr
ucted', 'disgraceful', 'dvds', 'eschews', 'fell', 'freezer', 'goriest', 'hauser', 'hungary', 'insinuate', 'juggl
e', 'leering', 'maelstrom', 'messiah', 'music', 'occasional', 'parking', 'pleasantville', 'pronunciation', 'recipi
ent', 'reviews', 'sas', 'shea', 'sneers', 'steiger', 'swastika', 'thrusting', 'tvs', 'vampyre', 'westerns']
```

In [103]:
```
feature_names = vect.get_feature_names()
print("Первые 50 признаков:\n{}".format(feature_names[:50]))
print("Признаки с 20010 по 20030:\n{}".format(feature_names[20010:20030]))
print("Каждый 700-й признак:\n{}".format(feature_names[::700]))
```

```
Первые 50 признаков:
['00', '00 and', '00 in', '000', '000 00', '000 000', '000 and', '000 budget', '000 feet', '000 for', '000 in', '0
00 people', '000 the', '000 to', '000 year', '000 year old', '000 years', '000 years ago', '007', '00s', '01', '0
2', '03', '04', '05', '06', '07', '08', '09', '10', '10 00', '10 000', '10 10', '10 10 for', '10 10 stars', '10 10
the', '10 15', '10 15 minutes', '10 30', '10 acting', '10 after', '10 all', '10 and', '10 and it', '10 and that',
'10 as', '10 at', '10 because', '10 because it', '10 because the']
Признаки с 20010 по 20030:
['are never really', 'are new', 'are nice', 'are nice and', 'are nice but', 'are nice people', 'are no', 'are no c
haracters', 'are no good', 'are no longer', 'are no more', 'are no other', 'are no real', 'are no redeeming', 'are
no special', 'are no words', 'are non', 'are non existent', 'are none', 'are not']
Каждый 700-й признак:
['00', '25 years and', 'about dozen', 'absolutely none', 'acting isn great', 'actresses and actors', 'adult film',
'again is', 'alien creature', 'all the colors', 'along to', 'although not as', 'american production', 'an entire m
ovie', 'and ability', 'and builds', 'and disappeared', 'and french', 'and how one', 'and less than', 'and oh yea
h', 'and rochester', 'and subject matter', 'and the viewer', 'and was able', 'animates', 'any more and', 'apparent
ly have', 'are eventually', 'are sent to', 'around this one', 'as fine', 'as remember', 'as williams', 'at his wor
st', 'attempt that', 'away her', 'bad guy in', 'barton', 'be dramatic', 'be serious', 'because he didn', 'been loo
king for', 'behold and', 'bereft of', 'better understanding of', 'bit that', 'body was', 'both versions', 'briefes
t', 'budget slasher', 'but don worry', 'but one of', 'but with all', 'by paul', 'came out it', 'can play the', 'ca
re of the', 'catapult', 'change to', 'characters which are', 'chillers', 'clampett', 'clue about', 'comedy if yo
u', 'compete for', 'congrats to', 'convinced by', 'could say the', 'crappy film', 'cruel world', 'dangerfield', 'd
ead you', 'definitely in the', 'deserves special mention', 'did better', 'different backgrounds', 'directors do',
'ditzy blonde', 'does his job', 'don be fooled', 'doubt the best', 'drinking and', 'dvds are', 'edition of', 'else
the', 'ending is predictable', 'ensues is', 'especially at', 'even show', 'every point', 'excellent performances f
rom', 'explains why', 'factory and', 'fans of this', 'fear but', 'fetus', 'film avoid', 'film lack', 'film was t
o', 'finally take', 'first off it', 'flighty', 'for but', 'for next', 'for truly', 'found out it', 'from all', 'fr
om their parents', 'further and further', 'generations and', 'gets attacked', 'give solid performances', 'goblin
s', 'good don', 'gorris', 'great example', 'grown up', 'had first', 'hand man', 'has absolutely no', 'has troubl
e', 'have hated', 'haven really', 'he happens', 'he tends', 'held responsible for', 'her lips', 'heroes', 'him not
to', 'his element', 'his pregnant', 'holds his', 'horror director', 'how much is', 'hundreds of years', 'if the au
dience', 'important it is', 'in chinatown', 'in his scenes', 'in places', 'in the ensuing', 'in tragic', 'infamy',
'intensity to', 'into the middle', 'is alternately', 'is curiously', 'is haunted by', 'is mostly made', 'is put in
to', 'is subjected to', 'is used in', 'it all bit', 'it didn show', 'it is absolutely', 'it never ever', 'it sort
of', 'it was slow', 'its type', 'jim corbett', 'julia ross', 'just plain silly', 'keira knightley', 'kind or anoth
er', 'knows why', 'last name', 'leaders of the', 'legs', 'life in general', 'like music', 'line really', 'liven u
p', 'longer in', 'lot better and', 'low to', 'made this is', 'make up', 'man whom', 'marks on', 'may make some',
'me watch it', 'mentality', 'miles away', 'mission was', 'mood which', 'more to say', 'mounties', 'movie four', 'm
ovie reminds', 'movies and', 'much or', 'mvp', 'myself wondering', 'need to tell', 'new films', 'no business', 'no
ll', 'not full', 'not taken', 'now do', 'odd minutes', 'of change', 'of fans', 'of human bondage', 'of money on',
'of revolution', 'of the bodies', 'of the set', 'of us have', 'office the', 'on but think', 'on stage play', 'once
you ve', 'one takes the', 'only small', 'or goes', 'original look', 'otherwise', 'out with one', 'own in the', 'pa
rsifal', 'pavement', 'perennial', 'persuades', 'pity on', 'please anyone', 'point when', 'possessed', 'presents it
self', 'probably will', 'prove that', 'puzzles', 'raised on', 're meant to', 'realize just', 'really wish', 'reec
e', 'remarkable thing', 'resorted', 'ribs', 'robert zemeckis', 'routh', 'said would', 'saw the trailer', 'scene wa
s so', 'screen at', 'security for', 'seems like an', 'sense unless', 'sets out', 'she could not', 'shifted', 'shou
ld watch this', 'side of her', 'singled', 'slow pacing', 'so good', 'soap opera like', 'some psychological', 'some
where around the', 'spanning', 'sports and', 'start going', 'still doesn', 'story even more', 'strictly', 'submiss
ive', 'sunday', 'survive to', 'take what', 'teacher is', 'terms it', 'that about all', 'that find', 'that lasts',
'that some', 'that what', 'the america', 'the better ones', 'the chainsaw', 'the courage of', 'the downside', 'the
face with', 'the first minutes', 'the groove tube', 'the intensity', 'the lock', 'the most confusing', 'the next b
est', 'the parents are', 'the previously mentioned', 'the role that', 'the shack', 'the story couldn', 'the time w
hat', 'the vile', 'the year best', 'them after', 'then surely', 'there was anything', 'they drive', 'thieves and',
'think we', 'this film has', 'this man and', 'this respect', 'those evil', 'threw it', 'time everything', 'title h
as', 'to be somewhat', 'to detail and', 'to get job', 'to kill an', 'to perform the', 'to seem', 'to the girls',
'to what we', 'too it', 'towards the end', 'trident', 'trying to protect', 'two from', 'underplayed', 'unreality',
'up young', 'valli', 'very charismatic', 'vicar', 'visually stimulating', 'wanted it to', 'was captivated', 'was l
ike watching', 'was spoiled', 'wasted watching', 'way he did', 'we say', 'well known for', 'were put', 'what mes
s', 'when see the', 'which does', 'whiney', 'who loved me', 'why are', 'will lose', 'with an odd', 'with leslie',
'with the latest', 'woman it', 'work perfectly', 'worth the price', 'wow could', 'years ago are', 'you can still',
'you re already', 'your age']
```

In [52]:
```
from sklearn.feature_extraction.text import ENGLISH_STOP_WORDS
print("Количество стоп-слов: {}".format(len(ENGLISH_STOP_WORDS)))
print("Каждое 10-е стоп-слово:\n{}".format(list(ENGLISH_STOP_WORDS)[::10]))
```

```
Количество стоп-слов: 318
Каждое 10-е стоп-слово:
['even', 'except', 'our', 'am', 'is', 'in', 'whereupon', 'my', 'couldnt', 'somehow', 'you', 'down', 'whither',
'name', 'becoming', 'and', 'on', 'ie', 'beforehand', 'upon', 'more', 'cannot', 'cant', 'towards', 'everywhere', 'p
art', 'see', 'but', 'describe', 'once', 'all', 'further']
```

In [53]:
```
# настройка stop_words="english" задает встроенный список стоп-слов.
# мы можем также расширить его и передать свой собственный.
vect = CountVectorizer(min_df=5, stop_words="english").fit(text_train)
X_train = vect.transform(text_train)
print("X_train с использованием стоп-слов:\n{}".format(repr(X_train)))
```

```
X_train с использованием стоп-слов:
<25000x26966 sparse matrix of type '<class 'numpy.int64'>'
        with 2149958 stored elements in Compressed Sparse Row format>
```

In [54]:
```
grid = GridSearchCV(LogisticRegression(max_iter=100000), param_grid, cv=5)
grid.fit(X_train, y_train)
print("Наилучшее значение перекр проверки: {:.2f}".format(grid.best_score_))
```

```
Наилучшее значение перекр проверки: 0.88
```

In [55]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline
pipe = make_pipeline(TfidfVectorizer(min_df=5, norm=None),
                     LogisticRegression(max_iter=100000))
param_grid = {'logisticregression__C': [0.001, 0.01, 0.1, 1, 10]}
grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(text_train, y_train)
print("Наилучшее значение перекр проверки: {:.2f}".format(grid.best_score_))
```

```
Наилучшее значение перекр проверки: 0.89
```

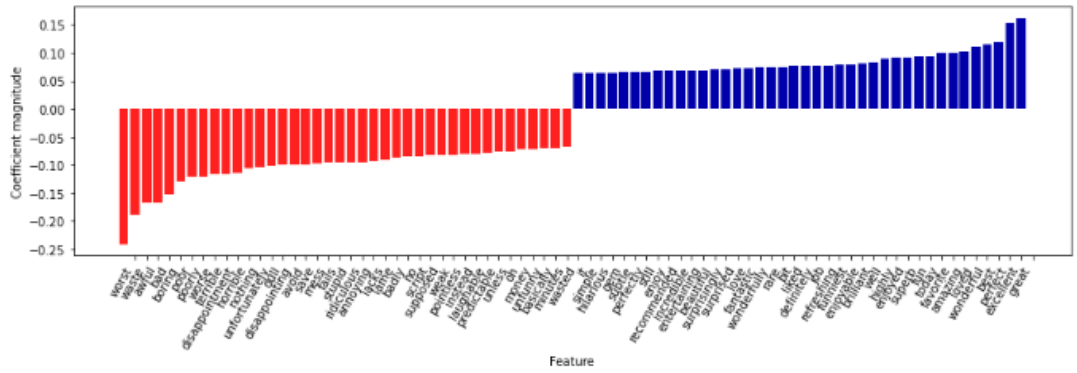
In [56]:
```
vectorizer = grid.best_estimator_.named_steps["tfidfvectorizer"]
# преобразуем обучающий набор данных
X_train = vectorizer.transform(text_train)
# находим максимальное значение каждого признака по набору данных
max_value = X_train.max(axis=0).toarray().ravel()
sorted_by_tfidf = max_value.argsort()
# получаем имена признаков
feature_names = np.array(vectorizer.get_feature_names())
print("Признаки с наименьшими значениями tfidf:\n{}".format(
    feature_names[sorted_by_tfidf[:20]]))
print("Признаки с наибольшими значениями tfidf: \n{}".format(
    feature_names[sorted_by_tfidf[-20:]]))
```

```
Признаки с наименьшими значениями tfidf:
['poignant' 'disagree' 'instantly' 'importantly' 'lacked' 'occurred'
 'currently' 'altogether' 'nearby' 'undoubtedly' 'directs' 'fond'
 'stinker' 'avoided' 'emphasis' 'commented' 'disappoint' 'realizing'
 'downhill' 'inane']
Признаки с наибольшими значениями tfidf:
['coop' 'homer' 'dillinger' 'hackenstein' 'gadget' 'taker' 'macarthur'
 'vargas' 'jesse' 'basket' 'dominick' 'the' 'victor' 'bridget' 'victoria'
 'khouri' 'zizek' 'rob' 'timon' 'titanic']
```

In [57]:
```
sorted_by_idf = np.argsort(vectorizer.idf_)
print("Признаки с наименьшими значениями idf:\n{}".format(
    feature_names[sorted_by_idf[:100]]))
```

```
Признаки с наименьшими значениями idf:
['the' 'and' 'of' 'to' 'this' 'is' 'it' 'in' 'that' 'but' 'for' 'with'
 'was' 'as' 'on' 'movie' 'not' 'have' 'one' 'be' 'film' 'are' 'you' 'all'
 'at' 'an' 'by' 'so' 'from' 'like' 'who' 'they' 'there' 'if' 'his' 'out'
 'just' 'about' 'he' 'or' 'has' 'what' 'some' 'good' 'can' 'more' 'when'
 'time' 'up' 'very' 'even' 'only' 'no' 'would' 'my' 'see' 'really' 'story'
 'which' 'well' 'had' 'me' 'than' 'much' 'their' 'get' 'were' 'other'
 'been' 'do' 'most' 'don' 'her' 'also' 'into' 'first' 'made' 'how' 'great'
 'because' 'will' 'people' 'make' 'way' 'could' 'we' 'bad' 'after' 'any'
 'too' 'then' 'them' 'she' 'watch' 'think' 'acting' 'movies' 'seen' 'its'
 'him']
```

In [58]:
```
mglearn.tools.visualize_coefficients(
    grid.best_estimator_.named_steps["logisticregression"].coef_,
    feature_names, n_top_features=40)
```



In [59]:
```
print("bards_words:\n{}".format(bards_words))
```

```
bards_words:
['The fool doth think he is wise,', 'but the wise man knows himself to be a fool']
```

In [60]:
```
cv = CountVectorizer(ngram_range=(1, 1)).fit(bards_words)
print("Размер словаря: {}".format(len(cv.vocabulary_)))
print("Словарь:\n{}".format(cv.get_feature_names()))
```

```
Размер словаря: 13
Словарь:
['be', 'but', 'doth', 'fool', 'he', 'himself', 'is', 'knows', 'man', 'the', 'think', 'to', 'wise']
```

In [61]:
```
cv = CountVectorizer(ngram_range=(2, 2)).fit(bards_words)
print("Размер словаря: {}".format(len(cv.vocabulary_)))
print("Словарь:\n{}".format(cv.get_feature_names()))
```

```
Размер словаря: 14
Словарь:
['be fool', 'but the', 'doth think', 'fool doth', 'he is', 'himself to', 'is wise', 'knows himself', 'man knows',
'the fool', 'the wise', 'think he', 'to be', 'wise man']
```

In [62]:
```
print("Преобразованные данные (плотн):\n{}".format(cv.transform(bards_words).toarray()))
```

```
Преобразованные данные (плотн):
[[0 0 1 1 1 0 1 0 0 1 0 1 0 0]
 [1 1 0 0 0 1 0 1 1 0 1 0 1 1]]
```

In [63]:
```
cv = CountVectorizer(ngram_range=(1, 3)).fit(bards_words)
print("Размер словаря: {}".format(len(cv.vocabulary_)))
print("Словарь:\n{}".format(cv.get_feature_names()))
```

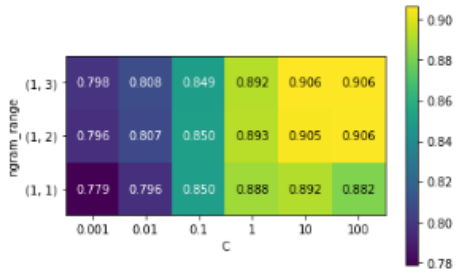
```
Размер словаря: 39
Словарь:
['be', 'be fool', 'but', 'but the', 'but the wise', 'doth', 'doth think', 'doth think he', 'fool', 'fool doth', 'f
ool doth think', 'he', 'he is', 'he is wise', 'himself', 'himself to', 'himself to be', 'is', 'is wise', 'knows',
'knows himself', 'knows himself to', 'man', 'man knows', 'man knows himself', 'the', 'the fool', 'the fool doth',
'the wise', 'the wise man', 'think', 'think he', 'think he is', 'to', 'to be', 'to be fool', 'wise', 'wise man',
'wise man knows']
```

In [72]:
```
pipe = make_pipeline(TfidfVectorizer(min_df=5), LogisticRegression(max_iter=100000))
# выполнение решетчатого поиска займет много времени из-за
# относительно большой сетки параметров и включения триграмм
param_grid = {"logisticregression__C": [0.001, 0.01, 0.1, 1, 10, 100],
              "tfidfvectorizer__ngram_range": [(1, 1), (1, 2), (1, 3)]}
grid = GridSearchCV(pipe, param_grid, cv=5)
grid.fit(text_train, y_train)
print("Наилучшее значение перекр проверки: {:.2f}".format(grid.best_score_))
print("Наилучшие параметры:\n{}".format(grid.best_params_))
```

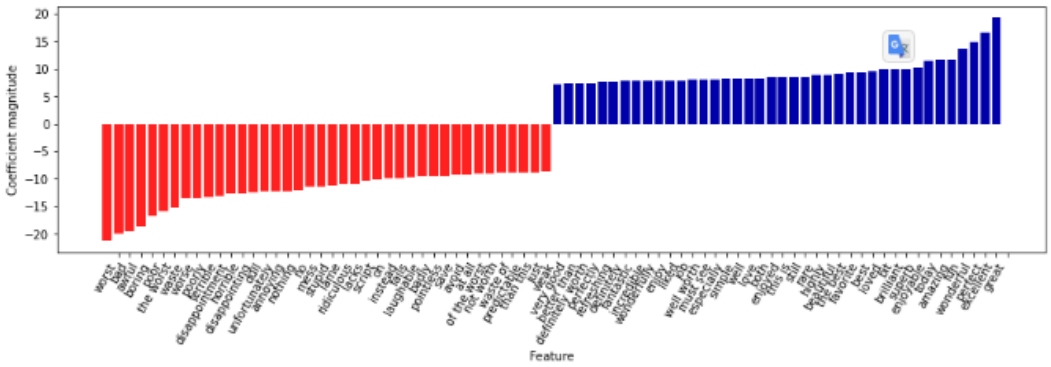
```
Наилучшее значение перекр проверки: 0.91
Наилучшие параметры:
{'logisticregression__C': 100, 'tfidfvectorizer__ngram_range': (1, 3)}
```

In [67]:
```
# излекаем значения правильности, найденные в ходе решетчатого поиска
scores = grid.cv_results_['mean_test_score'].reshape(-1, 3).T
# визуализируем теплокарту
heatmap = mglearn.tools.heatmap(
scores, xlabel="C", ylabel="ngram_range", cmap="viridis", fmt="%.3f",
xticklabels=param_grid['logisticregression__C'],
yticklabels=param_grid['tfidfvectorizer__ngram_range'])
plt.colorbar(heatmap)
```

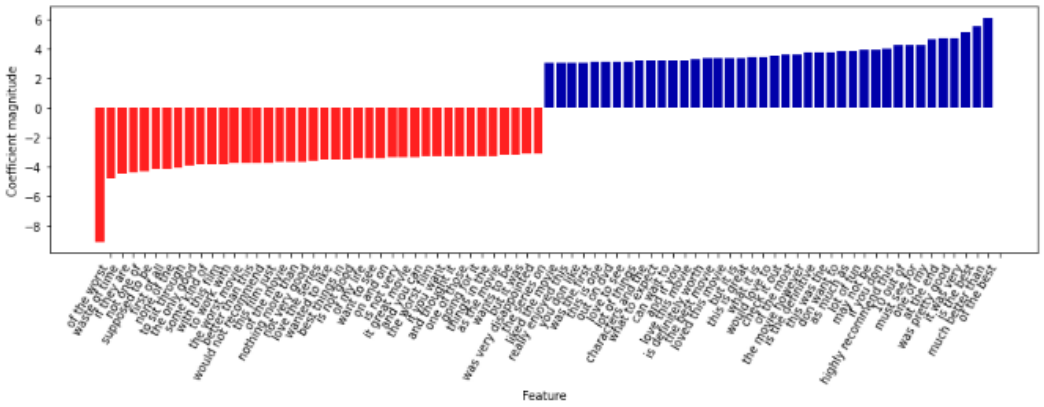
Out[67]: <matplotlib.colorbar.Colorbar at 0x7f62d17a6b20>



In [68]:
```
# извлекаем названия признаков и коэффициенты
vect = grid.best_estimator_.named_steps['tfidfvectorizer']
feature_names = np.array(vect.get_feature_names())
coef = grid.best_estimator_.named_steps['logisticregression'].coef_
mglearn.tools.visualize_coefficients(coef, feature_names, n_top_features=40)
```



In [69]:
```
# находим триграммные признаки
mask = np.array([len(feature.split(" ")) for feature in feature_names]) == 3
# визуализируем только 3-граммные признаки
mglearn.tools.visualize_coefficients(coef.ravel()[mask],
                                     feature_names[mask], n_top_features=40)
```



In [91]:
```
import spacy
import nltk
# загружаем модели пакета spacy для английского языка
en_nlp = spacy.load('en')
# создаем экземпляр стеммера Портера из пакета nltk
stemmer = nltk.stem.PorterStemmer()
# задаем функцию, сравнивающую лемматизацию в spacy со стеммингом в nltk
def compare_normalization(doc):
    # токенизируем документ в spacy
    doc_spacy = en_nlp(doc)
    # печатаем леммы, найденные с помощью spacy
    print("Лемматизация:")
    print([token.lemma_ for token in doc_spacy])
    # печатаем токены, найденные с помощью стеммера Портера
    print("Стемминг:")
    print([stemmer.stem(token.norm_.lower()) for token in doc_spacy])
```

In [92]:
```
compare_normalization(u"Our meeting today was worse than yesterday, "
"I'm scared of meeting the clients tomorrow.")
```

```
Лемматизация:
['-PRON-', 'meeting', 'today', 'be', 'bad', 'than', 'yesterday', ',', '-PRON-', 'be', 'scared', 'of', 'meet', 'th
e', 'client', 'tomorrow', '.']
Стемминг:
['our', 'meet', 'today', 'wa', 'wors', 'than', 'yesterday', ',', 'i', 'am', 'scare', 'of', 'meet', 'the', 'clien
t', 'tomorrow', '.']
```

In [93]:
```
#С технической точки зрения мы хотим применить токенизатор на основе
#регулярных выражений (regexp), который используется в CountVectorizer, а
#пакет spacy использовать лишь для лемматизации. Для этого мы
#заменим en_nlp.tokenizer (токенизатор пакета spacy)
#токенизатором на основе регулярных выражений.
import re
# regexp, используемые в CountVectorizer
regexp = re.compile('(?u)\\b\\w\\w+\\b')
# загружаем языковую модель spacy и сохраняем старый токенизатор
en_nlp = spacy.load('en')
old_tokenizer = en_nlp.tokenizer
# заменяем токенизатор старым на основе регулярных выражений
en_nlp.tokenizer = lambda string: old_tokenizer.tokens_from_list(
regexp.findall(string))
# создаем пользовательский токенизатор с помощью конвейера обработки документов spacy
# (теперь используем наш собственный токенизатор)
def custom_tokenizer(document):
    doc_spacy = en_nlp(document)
    return [token.lemma_ for token in doc_spacy]
# задаем countvectorizer с пользовательским токенизатором
lemma_vect = CountVectorizer(tokenizer=custom_tokenizer, min_df=5)
```

In [94]:
```
# преобразуем text_train, используя CountVectorizer с лемматизацией
X_train_lemma = lemma_vect.fit_transform(text_train)
print("форма X_train_lemma: {}".format(X_train_lemma.shape))
# стандартный CountVectorizer для сравнения
vect = CountVectorizer(min_df=5).fit(text_train)
X_train = vect.transform(text_train)
print("форма X_train: {}".format(X_train.shape))
```

```
форма X_train_lemma: (25000, 21825)
форма X_train: (25000, 27271)
```

In [ ]:
```
# строим модель решетчатого поиска, используя 1% данных в качестве обучающего набора
from sklearn.model_selection import StratifiedShuffleSplit
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}
cv = StratifiedShuffleSplit(test_size=0.99, train_size=0.01, random_state=0)
grid = GridSearchCV(LogisticRegression(max_iter=10000), param_grid, cv=cv)
# выполняем решетчатый поиск, используя данные, к которым был
# применен стандартный CountVectorizer
grid.fit(X_train, y_train)
print("Наилучшее значение перекрестной проверки "
     "(стандартный CountVectorizer): {:.3f}".format(grid.best_score_))
# выполняем решетчатый поиск, используя данные, к которым была
# применена лемматизация
grid.fit(X_train_lemma, y_train)
print("Наилучшее значение перекрестной проверки "
     "(лемматизация): {:.3f}".format(grid.best_score_))
```

In [85]:
```
vect = CountVectorizer(max_features=10000, max_df=.15)
X = vect.fit_transform(text_train)
```

In [97]:
```
from sklearn.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(learning_method="batch", max_iter=25, random_state=0)
# Мы строим модель и преобразуем данные в один этап
# Преобразование займет некоторое время,
# и мы можем сэкономить время, выполнив обе операции сразу
document_topics = lda.fit_transform(X)
```

In [98]:
```
lda.components_.shape
```

Out[98]:
```
(10, 10000)
```

In [100]:
```
#Для каждой темы (строки в components_) сортируем признаки (по возрастанию)
# Инвертируем строки с помощью [:, ::-1], чтобы получить сортировку по убыванию
sorting = np.argsort(lda.components_, axis=1)[:, ::-1]
# Получаем имена признаков из векторизатора
feature_names = np.array(vect.get_feature_names())
```

In [101]:
```
# Выводим 10 тем:
mglearn.tools.print_topics(topics=range(10), feature_names=feature_names,
                           sorting=sorting, topics_per_chunk=5, n_words=10)
```

```
topic 0       topic 1       topic 2       topic 3       topic 4
--------      --------      --------      --------      --------
altering      forrest       cargo         emotive       birds
gab           fugitive      fullest       electrical    earrings
bulimia       flooded       avoiding      break         accidently
diverts       defence       fatally       firefighter   fault
dept          accomplishing chapter       breakdancing  fuses
alien         blockbusters  ditsy         emphasized    ancestral
fudd          choosing      exhilarating  edmond        found
boogeyman     dancer        50s           dancer        davis
anecdotes     dehumanizing  data          fanaticism    bono
blackmail     devious       formulaic     fuses         celebratory


topic 5       topic 6       topic 7       topic 8       topic 9
--------      --------      --------      --------      --------
cigars        comprise      aristocratic  dept          cinematograph
43            43            du            du            frothy
borough       acoustic      committed     committed     castles
antonio       carney        foibles       48            comprising
data          carelessly    daves         deepti        categorized
deeds         bleed         anecdotes     aristocratic  frequency
blackmail     assignment    blackmail     deux          cigars
criticizing   007           detriment     combination   gab
dint          compounded    dept          commit        caveman
blog          decayed       cupid         depth         adolescents
```

In [105]:
```
lda100 = LatentDirichletAllocation(learning_method="batch",
    max_iter=25, random_state=0)
document_topics100 = lda100.fit_transform(X)
```

In [106]:
```
topics = np.array([7, 16, 24, 25, 28, 36, 37, 45, 51, 53, 54, 63, 89, 97])
sorting = np.argsort(lda100.components_, axis=1)[:, ::-1]
feature_names = np.array(vect.get_feature_names())
mglearn.tools.print_topics(topics=topics, feature_names=feature_names,
                           sorting=sorting, topics_per_chunk=7, n_words=20)
```

```
topic 7       topic 16      topic 24      topic 25      topic 28      topic 36      topic 37
--------      --------      --------      --------      --------      --------      --------


topic 45      topic 51      topic 53      topic 54      topic 63      topic 89      topic 97
--------      --------      --------      --------      --------      --------      --------
```

In [117]:
```
# сортируем документы по весу темы 45 "музыка"
music = np.argsort(document_topics100[:, 1])[::-1]
# печатаем пять документов, в которых данная тема является наиболее важной
for i in music[:10]:
    # выводим первые два предложения
    print(b".".join(text_train[i].split(b".")[:2]) + b".\n")
```

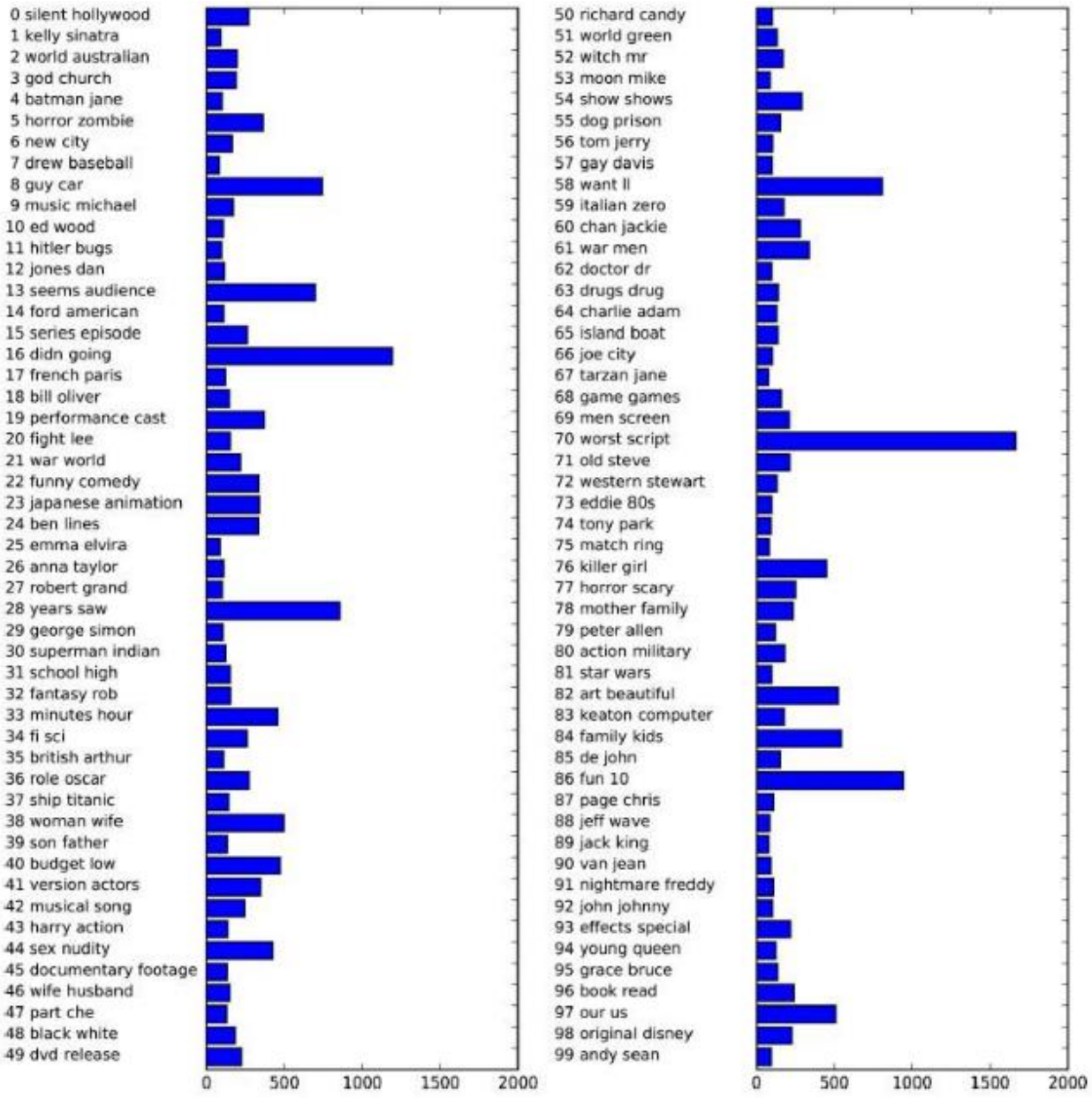
```
b"Adolf Hitler's maniacal desire to impose his will on the rest of the world is the subject of this second in a se
ven part series of films produced by the U.S.\n"
b"In short:  Spike Lee clearly has a lot on his mind. He's thinking about racism color-ism, media and hegemony, co
nsumerism and capitalism, religion, sexism, 'hetero-sexism', politics of the drug war etc etc.\n"
b'This documentary explores a story covered in Pilger\'s latest book "Freedom Next Time", which was published in 2
006. It reveals the shocking expulsion of the natives of Diego Garcia, one of the Chagos Islands in the Indian Oce
an.\n'
b"It's sad to view this film now that we know how the ANC got shafted by international capitalism. Biko died for n
othing much.\n"
b'The second of the Why We Fight Series concentrates on Hitler\'s grab of the Sudetanland and beyond as he makes a
chump out of Neville Chamberlain and embarks on his conquest of Europe.   Clearly meant as propaganda in its day t
his series over the test of time has become an informative documentary as well with most of the "Allied bias" turn
ing out to be historical fact.\n'
b'Do not expect a depiction of the "truth". However, the accounts of these veterans of the Iraqi & Afghanistan war
s demand thoughtful consideration.\n'
b'Probably the first Portuguese film I have seen in my life, and I enjoyed it. The plot is related of how the youn
g army officers took the power in Portugal in 1974, to finally defeat the fascist government of Caetano and to als
o finalize the wars in the colonies, i.\n'
b'Gojoe is part of a new wave of Japanese cinema, taking very creative directors, editors and photographers and wo
rking on historic themes, what the Japanese call "period pieces". Gojoe is extremely creative in terms of color, p
hotography, and editing.\n'
b"A true story about a true revolution, 25 of April ; a revolution against a repressive regime of 41 years, that w
as imposing a colonial war on it's military's, for maintaining an empire (Angola, Mozambique, Guine-Bissau, Cabo V
erde, S. Tom\xc3\xa9 e Principe; the first and the last of the great colonial empire's of Europe) of 600 years, si
nce it's beginning in the conquest of Ceuta in 1415; a revolution by the army for the people, and for a democratic
Portugal; the most's surprising fact in this revolution is that it were no people killed in it (except those that
died in the hand's of PIDE, the political police of the State, during a brutal gunfire against an unarmed crowd pr
otesting in front of it's headquarters in the day of the revolution, in 25 of April 1974, has it show's on the fil
m).\n"
b'This is an astounding film. As well as showing actual footage of key events in the failed coup to oust Chavez, w
e are given the background picture which describes a class-divided society.\n'
```

In [118]:
```
fig, ax = plt.subplots(1, 2, figsize=(10, 10))
topic_names = ["{:>2} ".format(i) + " ".join(words)
               for i, words in enumerate(feature_names[sorting[:, :2]])]
# две столбиковые диаграммы:
for col in [0, 1]:
    start = col * 50
    end = (col + 1) * 50
    ax[col].barh(np.arange(50), np.sum(document_topics100, axis=0)[start:end])
    ax[col].set_yticks(np.arange(50))
    ax[col].set_yticklabels(topic_names[start:end], ha="left", va="top")
    ax[col].invert_yaxis()
    ax[col].set_xlim(0, 2000)
    yax = ax[col].get_yaxis()
    yax.set_tick_params(pad=130)
plt.tight_layout()
```

In [117]:
```
# сортируем документы по весу темы 45 "музыка"
music = np.argsort(document_topics100[:, 1])[::-1]
# печатаем пять документов, в которых данная тема является наиболее важной
for i in music[:10]:
    # выводим первые два предложения
    print(b".".join(text_train[i].split(b".")[:2]) + b".\n")
```

```
b"Adolf Hitler's maniacal desire to impose his will on the rest of the world is the subject of this second in a se
ven part series of films produced by the U.S.\n"
b"In short:  Spike Lee clearly has a lot on his mind. He's thinking about racism color-ism, media and hegemony, co
nsumerism and capitalism, religion, sexism, 'hetero-sexism', politics of the drug war etc etc.\n"
b'This documentary explores a story covered in Pilger\'s latest book "Freedom Next Time", which was published in 2
006. It reveals the shocking expulsion of the natives of Diego Garcia, one of the Chagos Islands in the Indian Oce
an.\n'
b"It's sad to view this film now that we know how the ANC got shafted by international capitalism. Biko died for n
othing much.\n"
b'The second of the Why We Fight Series concentrates on Hitler\'s grab of the Sudetanland and beyond as he makes a
chump out of Neville Chamberlain and embarks on his conquest of Europe.   Clearly meant as propaganda in its day t
his series over the test of time has become an informative documentary as well with most of the "Allied bias" turn
ing out to be historical fact.\n'
b'Do not expect a depiction of the "truth". However, the accounts of these veterans of the Iraqi & Afghanistan war
s demand thoughtful consideration.\n'
b'Probably the first Portuguese film I have seen in my life, and I enjoyed it. The plot is related of how the youn
g army officers took the power in Portugal in 1974, to finally defeat the fascist government of Caetano and to als
o finalize the wars in the colonies, i.\n'
b'Gojoe is part of a new wave of Japanese cinema, taking very creative directors, editors and photographers and wo
rking on historic themes, what the Japanese call "period pieces". Gojoe is extremely creative in terms of color, p
hotography, and editing.\n'
b"A true story about a true revolution, 25 of April ; a revolution against a repressive regime of 41 years, that w
as imposing a colonial war on it's military's, for maintaining an empire (Angola, Mozambique, Guine-Bissau, Cabo V
erde, S. Tom\xc3\xa9 e Principe; the first and the last of the great colonial empire's of Europe) of 600 years, si
nce it's beginning in the conquest of Ceuta in 1415; a revolution by the army for the people, and for a democratic
Portugal; the most's surprising fact in this revolution is that it were no people killed in it (except those that
died in the hand's of PIDE, the political police of the State, during a brutal gunfire against an unarmed crowd pr
otesting in front of it's headquarters in the day of the revolution, in 25 of April 1974, has it show's on the fil
m).\n"
b'This is an astounding film. As well as showing actual footage of key events in the failed coup to oust Chavez, w
e are given the background picture which describes a class-divided society.\n'
```

In [118]:
```
fig, ax = plt.subplots(1, 2, figsize=(10, 10))
topic_names = ["{:>2} ".format(i) + " ".join(words)
               for i, words in enumerate(feature_names[sorting[:, :2]])]
# две столбиковые диаграммы:
for col in [0, 1]:
    start = col * 50
    end = (col + 1) * 50
    ax[col].barh(np.arange(50), np.sum(document_topics100, axis=0)[start:end])
    ax[col].set_yticks(np.arange(50))
    ax[col].set_yticklabels(topic_names[start:end], ha="left", va="top")
    ax[col].invert_yaxis()
    ax[col].set_xlim(0, 2000)
    yax = ax[col].get_yaxis()
    yax.set_tick_params(pad=130)
plt.tight_layout()
```



Вывод

Таким образом, мы изучили с помощью языка Python и средств Jupyter Notebook возможности работы с текстовыми данными. Текстовые данные обычно представлены в виде строк, состоящих из символов. На практике можно встретить четыре типа строковых данных:

• Категориальные данные

• Неструктурированные строки, которые по смыслу можно сгруппировать в категории

• Структурированные строки

• Текстовые данные

Категориальные данные (categorical data) представляют собой данные, которые берутся из фиксированного списка. Ответы, записанные в текстовом поле, относятся ко второй категории списка, неструктурированным строкам, которые по смыслу можно сгруппировать в категории (free strings that can be semantically mapped to categories). Лучшее всего закодировать их в виде категориальной переменной. Строковые значения, введенные вручную, не соответствуют фиксированным категориям, но при этом все же имеют некоторую базовую структуру (structure), например, адреса, названия мест, имена и фамилии людей, даты, номера телефонов и другие идентификаторы. Этот тип строк очень трудно спарсить и их обработка сильно зависит от контекста и предметной области. Последняя категория строковых данных – это текстовые данные (text data), которые состоят из фраз или предложений. Примерами таких данных могут быть твиты, логи чата или отзывы о гостинице, а также собрание сочинений Шекспира, содержание Википедии или проекта «Гутенберг», включающего 50000 электронных книг. Все эти коллекции содержат информацию, представленную преимущественно в виде предложений, составленных из слов. С точки зрения анализа текста набор данных часто называют корпусом (corpus) и каждая точка данных, представленная в виде отдельного текста, называется документом (document).

Представление текстовых данных в виде «мешка слов». Один из самых простых, но эффективных и широко используемых способов подготовки текста для машинного обучения – это представление текстовой информации в виде «мешка слов» (bag-of-words). Используя это представление, мы удаляем структуру исходного текста, например, главы, параграфы, предложения, форматирование, и лишь подсчитываем частоту встречаемости каждого слова в каждом документе корпуса. Удаление структуры и подсчет частоты каждого слова позволяет получить образное представление текста в виде «мешка слов». Получение представления «мешок слов» включает следующие три этапов:

1. Токенизация (tokenization). Разбиваем каждый документ на слова, которые встречаются в нем (токены), например, с помощью пробелов и знаков пунктуации.

2. Построение словаря (vocabulary building). Собираем словарь всех слов, которые появляются в любом из документов, и пронумеровываем их (например, в алфавитном порядке).

3. Создание разреженной матрицы (sparse matrix encoding). Для каждого документа подсчитываем, как часто каждое из слов, занесенное в словарь, встречается в документе.

Способ, с помощью которого мы можем избавиться от неинформативных слов – исключение слов, которые встречаются слишком часто, чтобы быть информативными. Существуют два основных подхода: использование списка стоп-слов (на основе соответствующего языка), или удаление слов, которые встречаются слишком часто.

Следующий подход вместо исключения несущественных признаков пытается масштабировать признаки в зависимости от степени их информативности. Одним из наиболее распространенных способов такого масштабирования является метод частота термина-обратная частота документа (term frequency-inverse document frequency, tf-idf). Идея этого метода заключается в том, чтобы присвоить большой вес термину, который часто встречается в конкретном документе, но при этом редко встречается в остальных документах корпуса. Если слово часто появляется в конкретном документе, но при этом редко встречается в остальных документах, оно, вероятно, будет описывать содержимое этого документа лучше.

Исследование коэффициентов модели. Строится гистограмма. Отрицательные коэффициенты, расположенные в левой части гистограммы, относятся к словам, которые в соответствии с моделью указывают на негативные отзывы, а положительные коэффициенты, расположенные в правой части гистограммы, принадлежат словам, которые означают положительные отзывы.

Модель «мешка слов» для последовательностей из нескольких слов. Один из главных недостатков представления «мешок слов» заключается в полном игнорировании порядка слов. Таким образом, две строки «it's bad, not good at all» и «it's good, not bad at all» будут иметь одинаковое представление, хотя противоположны по смыслу. Употребление частицы «not» перед словом – это лишь один из примеров того, какое важное значение имеет контекст. К счастью, существует способ, позволяющий учитывать контекст при использовании представления «мешок слов», фиксируя не только частоты одиночных токенов, но и пары, тройки токенов, которые появляются рядом друг с другом. Пары токенов называют биграммами (bigrams), тройки токенов известны как триграммы (trigrams), а в более широком смысле последовательности токенов известны как n-граммы (n-grams). Мы можем изменить диапазон токенов, которые рассматриваются в качестве признаков, изменив параметр ngram_range для CountVectorizer или TfidfVectorizer. Параметр ngram_range задает нижнюю и верхнюю границы диапазона n-значений для различных извлекаемых n-грамм. Таким образом, будут использованы все значения n, которые удовлетворяют условию min_n <= n <= max_n .

Продвинутая токенизация, стемминг и лемматизация. В более сложных задачах обработки текста часто возникает необходимость улучшить токенизацию, которая является первым этапом создания модели «мешка слов». Этот этап определяет, что представляет собой слово в плане извлечения признаков. Как и в случае с единственным и множественным числом, обработка различных глагольных форм и взаимосвязанных слов как отдельных токенов является препятствием, не позволяющим добиться хорошей обобщающей способности модели. Эту проблему можно решить, найдя для каждого слова его основу (word stem). Это подразумевает идентификацию или объединение (conflating) всех слов с одной и той же основой. Если этот процесс выполняется с помощью эвристик на основе правил (например, удаление общих суффиксов), его обычно называют стеммингом (stemming). Если вместо этого используется словарь с заранее заданными формами слов

(явный процесс, контролируемый человеком) и учитывается роль слова в предложении (то есть принимаем во внимание, к какой части речи относится слово), то этот процесс называется лемматизацией (lemmatization), а стандартизированная форма слова называется леммой (lemma). Лемматизация и стемминг являются способами нормализации (normalization), которые пытаются извлечь определенную нормальную (то есть начальную) форму слова. Еще один интересный случай нормализации – это исправление орфографических ошибок.

Моделирование тем и кластеризация документов. Еще один метод, который часто применяется к текстовым данным – моделирование тем (topic modeling). Моделирование тем – это зонтичный термин, описывающий процедуру присвоения каждому документу одной или нескольких тем, которая осуществляется, как правило, без учителя.Хорошим примером моделирования тем является новостные данные, которые можно сгруппировать по таким темам, как «политика», «спорт», «финансы» и так далее. Если каждый документ может иметь только одну тему, то речь идет о задаче кластеризации документов, которая рассматривалась в главе 3. Если каждый документ может иметь несколько тем, эта задача относится к декомпозиционным методам, освещавшимся в главы 3. Каждая полученная компонента соответствует одной теме, а коэффициенты компонент, которые описывают документ, позволяют нам судить о том, насколько тесно данный документ связан с конкретной темой. Часто, когда люди говорят о моделирования тем, они имеют в виду конкретный декомпозиционный метод под названием латентное размещение Дирихле(Latent Dirichlet Allocation , LDA). Говоря простым языком, модель LDA пытается найти группы слов (темы или топики), которые часто появляются вместе. LDA также подразумевает, что каждый документ можно интерпретировать как «смесь» из нескольких тем. Важно понимать, что для модели машинного обучения «тема» - это далеко не то же самое, что мы подразумеваем под «темой» в повседневной речи. В данном случае «тема» больше напоминает извлекаемые с помощью PCA или NMF компоненты. Даже

если «тема», полученная с помощью LDA, и имеет смысловое значение, все равно она не тождественна «теме» в ее традиционном понимании.

Модели, получаемые с помощью LDA, представляют собой интересные методы, позволяющие интерпретировать огромные корпусы текстов, когда метки классов отсутствуют или когда они имеются, как в данном случае. Однако алгоритм LDA является рандомизированным и разные значения параметра random_state могут привести к совершенно различным результаты. Несмотря на то что выделение тем может быть полезным, любые выводы, которые можно сделать, исходя из результатов модели неконтролируемого обучения, нужно принимать с определенной долей сомнения и рекомендуется проверять выводы, анализируя документы, присвоенные определенной теме. Кроме того, темы, полученные с помощью метода LDA.transform, можно иногда использовать в качестве входного признака для машинного обучения с учителем.