In [2]:
```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_repo
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_er
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
import seaborn as sns
from collections import Counter
from sklearn.datasets import fetch_20newsgroups
import matplotlib.pyplot as plt

%matplotlib inline
sns.set(style="ticks")
```

Векторизация текста на основе модели "мешка слов"

In [3]:
```
categories = ["rec.sport.hockey", "rec.sport.baseball", "sci.crypt","sci.space"]
newsgroups = fetch_20newsgroups(subset='train', categories=categories)
data = newsgroups['data']
```

In [4]:
```
def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray) -> Dict[int, float]:
    """
    Вычисление метрики accuracy для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
    Возвращает словарь: ключ - метка класса,
    значение - Accuracy для данного класса
    """
    # Для удобства фильтрации сформируем Pandas DataFrame
    d = {'t': y_true, 'p': y_pred}
    df = pd.DataFrame(data=d)
    # Метки классов
    classes = np.unique(y_true)
    # Результирующий словарь
    res = dict()
    # Перебор меток классов
    for c in classes:
        # отфильтруем данные, которые соответствуют
        # текущей метке класса в истинных значениях
        temp_data_flt = df[df['t']==c]
        # расчет accuracy для заданной метки класса
        temp_acc = accuracy_score(
            temp_data_flt['t'].values,
```

```
            temp_data_flt['p'].values)
        # сохранение результата в словарь
        res[c] = temp_acc
    return res

def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики accuracy для каждого класса
    """
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
    for i in accs:
        print('{} \t {}'.format(i, accs[i]))
```

In [5]:
```
vocabVect = CountVectorizer()
vocabVect.fit(data)
corpusVocab = vocabVect.vocabulary_
print('Количество сформированных признаков - {}'.format(len(corpusVocab)))
```

```
Количество сформированных признаков - 36053
```

In [6]:
```
for i in list(corpusVocab)[1:15]:
    print('{}={}'.format(i, corpusVocab[i]))
```

```
eastgate=13606
world=35502
std=31184
com=10437
mark=21937
bernstein=7838
subject=31563
re=27488
jewish=19518
baseball=7514
players=26024
organization=24729
the=32523
public=26947
```

In [8]:
```
test_features = vocabVect.transform(data)
test_features
```

Out[8]:
```
<2385x36053 sparse matrix of type '<class 'numpy.int64'>'
        with 390795 stored elements in Compressed Sparse Row format>
```

In [9]:
```
test_features.todense()
```

Out[9]:
```
matrix([[0, 0, 0, ..., 0, 0, 0],
        [2, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

In [ ]:
```
# Размер нулевой строки
len(test_features.todense()[0].getA1())
```

In [11]:
```
vocabVect.get_feature_names()[0:30]
```

Out[11]:
```
['00',
 '000',
 '0000',
```

```
 '00000',
 '000000',
 '00000000',
 '00000000b',
 '00000001',
 '00000001b',
 '00000010',
 '00000010b',
 '00000011',
 '00000011b',
 '00000100',
 '00000100b',
 '00000101',
 '00000101b',
 '00000110',
 '00000110b',
 '00000111',
 '00000111b',
 '00001000',
 '00001000b',
 '00001001',
 '00001001b',
 '00001010',
 '00001010b',
 '00001011',
 '00001011b',
 '00001100']
```

In [12]:
```
def VectorizeAndClassify(vectorizers_list, classifiers_list):
    for v in vectorizers_list:
        for c in classifiers_list:
            pipeline1 = Pipeline([("vectorizer", v), ("classifier", c)])
            score = cross_val_score(pipeline1, newsgroups['data'], newsgroups['target'],
            print('Векторизация - {}'.format(v))
            print('Модель для классификации - {}'.format(c))
            print('Accuracy = {}'.format(score))
            print('============================')
```

In [19]:
```
vectorizers_list = [CountVectorizer(vocabulary = corpusVocab),TfidfVectorizer(vocabulary
classifiers_list = [LogisticRegression(C=3.0), MultinomialNB(alpha=0.3)]
VectorizeAndClassify(vectorizers_list, classifiers_list)
```

```
D:\ml\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs
failed to converge (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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
D:\ml\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs
failed to converge (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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
D:\ml\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs
failed to converge (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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,
                               '000000': 4, '00000000': 5, '00000000b': 6,
                               '00000001': 7, '00000001b': 8, '00000010': 9,
                               '00000010b': 10, '00000011': 11, '00000011b': 12,
                               '00000100': 13, '00000100b': 14, '00000101': 15,
                               '00000101b': 16, '00000110': 17, '00000110b': 18,
                               '00000111': 19, '00000111b': 20, '00001000': 21,
                               '00001000b': 22, '00001001': 23, '00001001b': 24,
                               '00001010': 25, '00001010b': 26, '00001011': 27,
                               '00001011b': 28, '00001100': 29, ...})
Модель для классификации - LogisticRegression(C=3.0)
Accuracy = 0.9639412997903564
============================
Векторизация - CountVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,
                               '000000': 4, '00000000': 5, '00000000b': 6,
                               '00000001': 7, '00000001b': 8, '00000010': 9,
                               '00000010b': 10, '00000011': 11, '00000011b': 12,
                               '00000100': 13, '00000100b': 14, '00000101': 15,
                               '00000101b': 16, '00000110': 17, '00000110b': 18,
                               '00000111': 19, '00000111b': 20, '00001000': 21,
                               '00001000b': 22, '00001001': 23, '00001001b': 24,
                               '00001010': 25, '00001010b': 26, '00001011': 27,
                               '00001011b': 28, '00001100': 29, ...})
Модель для классификации - MultinomialNB(alpha=0.3)
Accuracy = 0.9815513626834381
============================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,
                               '000000': 4, '00000000': 5, '00000000b': 6,
                               '00000001': 7, '00000001b': 8, '00000010': 9,
                               '00000010b': 10, '00000011': 11, '00000011b': 12,
                               '00000100': 13, '00000100b': 14, '00000101': 15,
                               '00000101b': 16, '00000110': 17, '00000110b': 18,
                               '00000111': 19, '00000111b': 20, '00001000': 21,
                               '00001000b': 22, '00001001': 23, '00001001b': 24,
                               '00001010': 25, '00001010b': 26, '00001011': 27,
                               '00001011b': 28, '00001100': 29, ...})
Модель для классификации - LogisticRegression(C=3.0)
Accuracy = 0.9786163522012578
============================
Векторизация - TfidfVectorizer(vocabulary={'00': 0, '000': 1, '0000': 2, '00000': 3,
                               '000000': 4, '00000000': 5, '00000000b': 6,
                               '00000001': 7, '00000001b': 8, '00000010': 9,
                               '00000010b': 10, '00000011': 11, '00000011b': 12,
                               '00000100': 13, '00000100b': 14, '00000101': 15,
                               '00000101b': 16, '00000110': 17, '00000110b': 18,
                               '00000111': 19, '00000111b': 20, '00001000': 21,
                               '00001000b': 22, '00001001': 23, '00001001b': 24,
                               '00001010': 25, '00001010b': 26, '00001011': 27,
                               '00001011b': 28, '00001100': 29, ...})
Модель для классификации - MultinomialNB(alpha=0.3)
Accuracy = 0.979874213836478
============================
```

С учетом того, что при параметрах по умолчанию у нас были крайне высокие результаты(примерно 0.95), то было принято решение о тестировании с разными гипе-параметрами.

В результате при снижении alpha для MultinomialNB был получен лучший результат 0.982(Векторизация - CountVectorizer)