1. Решаемая задача.
Применить 3-4 классификатора библиотеки scikit-learn для классификации документов, имеющихся в полученном файле.

2. Перечень классов документов.
13 классов:
автомобиль
здоровье
культура
наука
недвижимость
политика
происшествие
реклама
семья
спорт
страна
техника
экономика

3. Число документов в каждом классе.
Класс автомобиль
: 249
Класс здоровье
: 157
Класс культура
: 358
Класс наука
: 227
Класс недвижимость
: 98
Класс политика
: 600
Класс происшествие
: 436
Класс реклама
: 94

Класс семья
: 101
Класс спорт
: 373
Класс страна
: 146
Класс техника
: 289
Класс экономика
: 272

4. Порядок предварительной обработки корпуса.
Подготовка набора данных.

Вариант *токенизации*:

- приводим в нижний регистр;

```
#делаем все слова нижнего регистра
a = [x.lower() for x in dataset]
```

- иностранные слова заменяем на FRGN;
- элементы с цифрами заменяем на NUMB;

```
# иностранные слова заменяем на FRGN; элементы с цифрами заменяем на NUMB
b = []
n = len(a)
for i in range (n):
    b.append(a[i].split(" "))
    for j in range (0, len(b[i])):
        if (re.search('[a-z]', b[i][j])):
            b[i][j]= "frgn"#"FRGN"
        if (re.search('[0-9]', b[i][j])):
            b[i][j]= "numb"#"NUMB"
    b[i]=" ".join(b[i])
```

- удаляем знаки препинания;

```
out = []
for x in b:
     out.append (re.sub(r'[^\w\s]','', x))

dataset = out
```

- Лемматизация русских слов. Её *цель* - сокращение словаря;

```
# лемматизация
# Заменяет  слова леммами
def to_normal_form(morph, s):
    s2 = s.split() # Список слов предложения s
    s = ''
    for w in s2:
        w = morph.parse(w)[0].normal_form
        s += (' ' + w)
    return s.lstrip()
morph = pymorphy2.MorphAnalyzer()
for i in range (len(dataset)):
    dataset[i]=to_normal_form(morph, dataset[i])
print (dataset)
```

- разделение данных и меток в разные файлы (x.txt, y.txt).

```
dataset = out

#извлекаем метки
print("---------------------------------------")
y_data = []
n = len (dataset)
for i in range (n):
    y_words = dataset[i].split(" ")
    y_data.append(y_words[0])

#print("Разбиение предложения по словам y_words", y_words,"\n")
print("Метки y_label, \n", y_data,"\n")
#извлекаем предложения без меток
print("---------------------------------------")
x_data = []
n = len (dataset)
for i in range (n):
    ind_begin = dataset[i].index(" ")
    ind_begin +=1
    x_data.append(dataset[i][ind_begin:])

print("Предложения без меток: \n", x_data,"\n")

f_x = open("f_x.txt", "w")
f_y = open("f_y.txt", "w")

np.savetxt(f_x, x_data, fmt='%s')
np.savetxt(f_y, y_data, fmt='%s')

f_x.close()
f_y.close()
```

Лемматизация – это приведение словоформы к лемме, или нормальной форме.

В русском языке леммами считаются следующие морфологические формы:

- имя существительное в именительном падеже и единственном числе;

- имя прилагательное в именительном падеже, единственном числе, мужского рода;

- глагол, причастие, деепричастие – глагол несовершенного вида в инфинитиве.

5. Размер словаря корпуса до и после предварительной обработки.
До:
После: 34707
6. Примененные классификаторы и их параметры.

SGDClassifier.

```
from sklearn.linear_model import SGDClassifier
doc_clf = SGDClassifier(loss = 'hinge', max_iter = 1000, tol = 1e-3)

start_time = time.time()
#print(y_trn)
doc_clf.fit(x_trn, y_trn) # Обучение классификатора
end_time = time.time() - start_time
print("Время выполнения :  {:.8f} секунд " .format(end_time) )
print('Оценка точности классификации')
score = doc_clf.score(x_vl, y_vl)
print('Точность на проверочном множестве:', round(score, 4))
score = doc_clf.score(x_trn, y_trn)
print('Точность на обучающем множестве:', round(score, 4))
```

```
Время выполнения :  36.82605052 секунд
Оценка точности классификации
Точность на проверочном множестве: 0.8279
Точность на обучающем множестве: 0.9901
```

```python
from sklearn.linear_model import LogisticRegression
doc_clf = LogisticRegression(solver = 'lbfgs', # newton-cg
                             max_iter = 500, multi_class = 'auto')
start_time = time.time()
#print(y_trn)
doc_clf.fit(x_trn, y_trn) # Обучение классификатора
end_time = time.time() - start_time
print("Время выполнения :  {:.8f} секунд " .format(end_time) )
print('Оценка точности классификации')
score = doc_clf.score(x_vl, y_vl)
print('Точность на проверочном множестве:', round(score, 4))
score = doc_clf.score(x_trn, y_trn)
print('Точность на обучающем множестве:', round(score, 4))
```

```
Время выполнения :  156.43311954 секунд
Оценка точности классификации
Точность на проверочном множестве: 0.8926
Точность на обучающем множестве: 1.0
```

```python
#---------------------
from sklearn.svm import SVC
doc_clf_svc = SVC()
#doc_clf = SGDClassifier(loss = 'hinge', max_iter = 1000, tol = 1e-3)

start_time = time.time()
#print(y_trn)
doc_clf_svc.fit(x_trn, y_trn) # Обучение классификатора
end_time = time.time() - start_time
print("Время выполнения :  {:.8f} секунд " .format(end_time) )
print('Оценка точности классификации')
score = doc_clf_svc.score(x_vl, y_vl)
print('Точность на проверочном множестве:', round(score, 4))
score = doc_clf_svc.score(x_trn, y_trn)
print('Точность на обучающем множестве:', round(score, 4))
```

```
Время выполнения :  451.74127626 секунд
Оценка точности классификации
Точность на проверочном множестве: 0.6956
Точность на обучающем множестве: 0.8081
```

7. Точность классификации документов обучающего и проверочного множеств (общая и по классам).

SGDClassifier:

Время выполнения : 41.59354258 секунд

Оценка точности классификации

Точность на проверочном множестве: 0.7926

Точность на обучающем множестве: 0.9728

```
Время выполнения :  41.59354258 секунд
Оценка точности классификации
Точность на проверочном множестве: 0.7926
Точность на обучающем множестве: 0.9728
              precision    recall  f1-score   support

  автомобиль
      0.9750    0.7647    0.8571        51
    здоровье
      0.3729    0.8800    0.5238        25
    культура
      0.6860    0.9077    0.7815        65
       наука
      0.9630    0.9455    0.9541        55
недвижимость
      0.6500    0.5652    0.6047        23
    политика
      0.9279    0.8729    0.8996       118
происшествие
      0.9053    0.9247    0.9149        93
     реклама
      0.7143    0.2381    0.3571        21
       семья
      0.7692    0.7143    0.7407        14
       спорт
      1.0000    0.8143    0.8976        70
      страна
      0.3333    0.0541    0.0930        37
     техника
      0.5833    0.8305    0.6853        59
   экономика
      0.8750    0.8571    0.8660        49

    accuracy                         0.7926       680
   macro avg     0.7504    0.7207    0.7058       680
weighted avg     0.8098    0.7926    0.7827       680
```

SVC:

Точность на проверочном множестве: 0.6956

Точность на обучающем множестве: 0.8081

Logistic Regression

Время выполнения : 157.71556211 секунд

Оценка точности классификации

Точность на проверочном множестве: 0.8324

Точность на обучающем множестве: 1.0

```
Время выполнения :  157.71556211 секунд
Оценка точности классификации
Точность на проверочном множестве: 0.8324
Точность на обучающем множестве: 1.0
              precision    recall  f1-score   support

  автомобиль
      0.9535    0.8039    0.8723        51
    здоровье
      0.6176    0.8400    0.7119        25
    культура
      0.7160    0.8923    0.7945        65
       наука
      0.8814    0.9455    0.9123        55
недвижимость
      0.6842    0.5652    0.6190        23
    политика
      0.9211    0.8898    0.9052       118
происшествие
      0.9072    0.9462    0.9263        93
     реклама
      0.5000    0.3333    0.4000        21
       семья
      0.7059    0.8571    0.7742        14
       спорт
      0.9701    0.9286    0.9489        70
      страна
      0.6000    0.3243    0.4211        37
     техника
      0.7576    0.8475    0.8000        59
   экономика
      0.8571    0.8571    0.8571        49

    accuracy                         0.8324       680
   macro avg     0.7748    0.7716    0.7648       680
weighted avg     0.8310    0.8324    0.8261       680
```

8. Время обучения каждого классификатора.

SGDClassifier:

Время выполнения : 41.59354258 секунд

SVC:

Время выполнения : 451.74127626 секунд

Logistic Regression

Время выполнения : 157.71556211 секунд