`Дисциплина: Методы и технологии машинного обучения`

`Уровень подготовки: бакалавриат`

`Направление подготовки: 01.03.02 Прикладная математика и информатика`

`Семестр: осень 2021/2022`

In [1]:
```
# настройка ширины страницы блокнота .....................................
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:80% !important; }</style>"))

# расширение watermark для вывода информации о версиях пакетов
#  https://github.com/rasbt/watermark
%load_ext watermark
```

Лабораторная работа №6: Машины опорных векторов

В практических примерах ниже показано:

- как классифицировать данные с помощью модели SVM
- как использовать конвейеры для подгонки модели и применения её к новым данным

Точность всех моделей оценивается методом перекрёстной проверки по 5 блокам.

Модели: SVM
Данные: `wdbc.data` (Breast Cancer Wisconsin). Источник: сайт Калифорнийского университета в Ирвине

In [2]:
```
# выводим информацию о версиях python и пакетов
%watermark -a "aksyuk@github.com" -d -v -p numpy,pandas,matplotlib,sklearn
```

```
Author: aksyuk@github.com

Python implementation: CPython
Python version       : 3.8.8
IPython version      : 7.22.0

numpy     : 1.20.1
pandas    : 1.2.4
matplotlib: 3.3.4
sklearn   : 0.24.1
```

Указания к выполнению

Загружаем пакеты

In [3]:
```
# загрузка пакетов: инструменты ----------------------------------------------
#  работа с массивами
```

```
import numpy as np
#  фреймы данных
import pandas as pd
#  графики
import matplotlib as mpl
#  стили и шаблоны графиков на основе matplotlib
import seaborn as sns
# перекодировка символьных показателей
from sklearn.preprocessing import LabelEncoder
#  для таймера
import time

# загрузка пакетов: модели ------------------------------------------------
#  SVM
from sklearn.svm import SVC
# логистическая рагрессия
from sklearn.linear_model import LogisticRegression
# стандартизация
from sklearn.preprocessing import StandardScaler
# метод главных компонент
from sklearn.decomposition import PCA
# конвейеры
from sklearn.pipeline import make_pipeline
# перекрёстная проверка и метод проверочной выборки
from sklearn.model_selection import cross_val_score, train_test_split
# для перекрёстной проверки и сеточного поиска
from sklearn.model_selection import KFold, GridSearchCV
#  сводка по точности классификации
from sklearn.metrics import classification_report
```

In [4]:

```
# константы
#  ядро для генератора случайных чисел
my_seed = 9212
#  создаём псевдоним для короткого обращения к графикам
plt = mpl.pyplot
# настройка стиля и отображения графиков
#  примеры стилей и шаблонов графиков:
#  http://tonysyu.github.io/raw_content/matplotlib-style-gallery/gallery.html
mpl.style.use('seaborn-whitegrid')
sns.set_palette("Set2")
# раскомментируйте следующую строку, чтобы посмотреть палитру
# sns.color_palette("Set2")
```

Загружаем данные

Набор данных можно загрузить напрямую по ссылке: https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data (копия: https://raw.githubusercontent.com/aksyuk/MTML/main/Labs/data/wdbc.data). Справочник к данным доступен по адресу: https://github.com/aksyuk/MTML/blob/main/Labs/data/CodeBook_wdbc.md.

Загружаем данные во фрейм и выясняем их размерность. Смотрим первые строки таблицы.

In [5]:

```
# загружаем данные
DF_raw = pd.read_csv('https://archive.ics.uci.edu/ml/'
```

```
                 'machine-learning-databases'
                 '/breast-cancer-wisconsin/wdbc.data', header=None)

# выясняем размерность фрейма
print('Число строк и столбцов в наборе данных:\n', DF_raw.shape)
```

```
Число строк и столбцов в наборе данных:
 (569, 32)
```

In [6]:

```
# первые строки
DF_raw.head()
```

Out[6]:

	0	1	2	3	4	5	6	7	8	9	...	22	23	24
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20

5 rows × 32 columns

Все столбцы кроме первого количественные.

In [7]:

```
# типы столбцов
DF_raw.dtypes
```

Out[7]:

```
0       int64
1      object
2     float64
3     float64
4     float64
5     float64
6     float64
7     float64
8     float64
9     float64
10    float64
11    float64
12    float64
13    float64
14    float64
15    float64
16    float64
17    float64
18    float64
19    float64
20    float64
21    float64
22    float64
23    float64
24    float64
25    float64
26    float64
27    float64
28    float64
29    float64
```

```
30    float64
31    float64
dtype: object
```

Классы отклика: **B** – доброкачественная опухоль, **M** – злокачественная. Частоты классов:

In [8]:
```
DF_raw[1].value_counts(dropna=False)
```

Out[8]:
```
B    357
M    212
Name: 1, dtype: int64
```

Оцифровываем классы, кодируя **B** как 0, а **M** как 1.

In [9]:
```
le = LabelEncoder()
DF_raw.loc[:, 1] = le.fit_transform(DF_raw.loc[:, 1].values)
le.classes_
```

Out[9]:
```
array(['B', 'M'], dtype=object)
```

In [10]:
```
le.transform(['M', 'B'])
```

Out[10]:
```
array([1, 0])
```

In [11]:
```
# проверка
DF_raw[1].value_counts(dropna=False)
```

Out[11]:
```
0    357
1    212
Name: 1, dtype: int64
```

Отложим 10% наблюдений для прогноза.

In [12]:
```
# наблюдения для моделирования
DF = DF_raw.sample(frac=0.9, random_state=my_seed)
# отложенные наблюдения
DF_predict = DF_raw.drop(DF.index)
```

Преобразование исходных данных и построение моделей

Как указано в справочнике к данным, объясняющие переменные уже являются производными от описательных статистик 10 показателей, рассчитанных по визуализации клеток опухоли. Все эти переменные количественные, и как видно из отчёта ниже, их значения неотрицательные, а разброс различается.

In [13]:
```
DF.loc[:, 2:].describe()
```

Out[13]:

	2	3	4	5	6	7	8	9

	2	3	4	5	6	7	8	9
count	512.000000	512.000000	512.000000	512.000000	512.000000	512.000000	512.000000	512.000000
mean	14.157576	19.245117	92.204766	657.957031	0.096156	0.104752	0.089560	0.049236
std	3.528499	4.324990	24.361272	353.814905	0.013967	0.052800	0.079642	0.038963
min	7.691000	9.710000	47.920000	170.400000	0.052630	0.019380	0.000000	0.000000
25%	11.710000	16.157500	75.200000	420.875000	0.086508	0.065175	0.029640	0.020348
50%	13.390000	18.770000	86.415000	552.050000	0.095865	0.094035	0.061880	0.033455
75%	16.040000	21.802500	105.250000	798.050000	0.104950	0.130525	0.131950	0.074320
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200

8 rows × 30 columns

В качестве альтернативных моделей рассмотрим SVM с различными вариантами ядер и логистическую регрессию. Причём предварительно преобразуем пространство исходных показателей с помощью метода главных компонент.

Стандартизация и переход к главным компонентам

In [14]:
```
# стандартизация
sc = StandardScaler()
X_train_std = sc.fit_transform(DF.iloc[:, 2:].values)

# оцениваем объяснённую главными компонентами дисперсию
pca = PCA()
X_train_pca = pca.fit_transform(X_train_std)

# считаем доли объяснённой дисперсии
frac_var_expl = pca.explained_variance_ratio_
print('Доли объяснённой дисперсии по компонентам в PLS:\n',
      np.around(frac_var_expl, 3),
      '\nОбщая сумма долей:', np.around(sum(frac_var_expl), 3))
```

```
Доли объяснённой дисперсии по компонентам в PLS:
 [0.441 0.188 0.097 0.064 0.056 0.041 0.023 0.016 0.014 0.012 0.01  0.009
 0.008 0.005 0.003 0.003 0.002 0.002 0.002 0.001 0.001 0.001 0.001 0.001
 0.001 0.    0.    0.    0.    0.   ]
Общая сумма долей: 1.0
```

Таким образом, первые две главные компоненты объясняют 62.9% разброса 30 объясняющих переменных.
Теперь объединим функции-преобразователи и оценщики в конвейер с помощью `Pipeline` и оценим точность логистической регрессии с помощью перекрёстной проверки.

Модель логистической регрессии с перекрёстной проверкой

In [15]:
```
# данные для обучения моделей
```

```
X_train = DF.loc[:, 2:]
y_train = DF.loc[:, 1]

# объединяем в конвейер шкалирование, ГК с 2 компонентами и логит
pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=2),
                        LogisticRegression(random_state=my_seed, solver='lbfgs'))

# будем сохранять точность моделей в один массив:
score = list()
score_models = list()

# считаем точность с перекрёстной проверкой, показатель Acc
cv = cross_val_score(estimator=pipe_lr, X=X_train, y=y_train, cv=5,
                     scoring='accuracy')

# записываем точность
score.append(np.around(np.mean(cv), 3))
score_models.append('sc_pca_logit')

print('Acc с перекрёстной проверкой',
      '\nдля модели', score_models[0], ':', score[0])
```

```
Acc с перекрёстной проверкой
для модели sc_pca_logit : 0.949
```

SVM с перекрёстной проверкой

Построим несколько вариантов модели SVM с различными ядерными функциями.

In [16]:

```
pipe_svc = make_pipeline(StandardScaler(),
                         SVC(random_state=my_seed))

# настроим параметры SVM с помощью сеточного поиска
param_range = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
param_grid = [{'svc__C': param_range,
               'svc__kernel': ['linear']},
              {'svc__C': param_range,
               'svc__gamma': param_range,
               'svc__kernel': ['rbf']},
              {'svc__C': param_range,
               'svc__gamma': param_range,
               'svc__degree' : [2, 3],
               'svc__kernel': ['poly']}]

# разбиения для перекрёстной проверки
kfold = KFold(n_splits=5, random_state=my_seed, shuffle=True)

gs = GridSearchCV(estimator=pipe_svc, param_grid=param_grid,
                  scoring='accuracy', refit=True, cv=kfold,
                  n_jobs=-1)

# таймер
tic = time.perf_counter()
# запускаем сеточный поиск
gs = gs.fit(X_train, y_train)
# таймер
toc = time.perf_counter()
print(f"Сеточный поиск занял {toc - tic:0.2f} секунд", sep='')
```

```
Сеточный поиск занял 4.93 секунд
```

In [17]:
```
# точность лучшей модели
np.around(gs.best_score_, 3)
```

Out[17]:
```
0.975
```

In [18]:
```
# параметры лучшей модели
#  * ядерная функция
gs.best_estimator_.get_params()['svc__kernel']
```

Out[18]:
```
'linear'
```

In [19]:
```
#  * параметр регуляризации
gs.best_estimator_.get_params()['svc__C']
```

Out[19]:
```
1.0
```

In [20]:
```
#  * коэффициент ядерной функции (для ядер 'rbf', 'poly' и 'sigmoid')
gs.best_estimator_.get_params()['svc__gamma']
```

Out[20]:
```
'scale'
```

In [21]:
```
#  * степень полинома (для ядра 'poly')
gs.best_estimator_.get_params()['svc__degree']
```

Out[21]:
```
3
```

Подробности сеточного поиска

Посмотреть результаты сеточного поиска можно в объектах:

- `gs.cv_results_['params']` – список сочетаний параметров;
- `gs.cv_results_['mean_test_score']` – значения Acc для сочетаний параметров (средние по блокам перекрёстной проверки).

In [22]:
```
# записываем точность
score.append(np.around(gs.best_score_, 3))
score_models.append('sc_pca_svc')

print('Acc с перекрёстной проверкой',
      '\nдля модели', score_models[1], ':', score[1])
```

```
Acc с перекрёстной проверкой
для модели sc_pca_svc : 0.975
```

Прогноз на отложенные наблюдения по

лучшей модели

В данном случае модель SVM показывает большую точность, чем модель логистической регрессии.

In [23]:
```
# сводка по точности моделей
pd.DataFrame({'Модель' : score_models, 'Acc' : score})
```

Out[23]:

	Модель	Acc
0	sc_pca_logit	0.949
1	sc_pca_svc	0.975

Сделаем прогноз на отложенные наблюдения с помощью второго ансамбля.

In [24]:
```
# прогноз с помощью лучшей модели ансамбля с SVC
y_hat = gs.best_estimator_.predict(X=DF_predict.loc[:, 2:])

# точность
# характеристики точности
print(classification_report(DF_predict.loc[:, 1], y_hat))
```

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        37
           1       1.00      0.90      0.95        20

    accuracy                           0.96        57
   macro avg       0.97      0.95      0.96        57
weighted avg       0.97      0.96      0.96        57
```

Упражнение 5

1. Данные своего варианта из упражнения 3 (на регуляризацию и снижение размерности) разделить на выборку для построения моделей (85%) и отложенные наблюдения (15%). Отложенные наблюдения использовать только для прогноза по лучшей модели.

2. Построить два ансамблей моделей, каждый из которых включает преобразование объясняющих переменных (например: шкалирование, PCA, PLS) и модель классификации (например: LDA, QDA, логит, случайный лес, svm). Построить третий ансамбль, который включает преобразование объясняющих переменных и модель kNN. Провести настройку параметров каждой модели, которая содержит настроечные параметры, с помощью сеточного поиска.

3. Довести точность Acc лучшего ансамбля (на обучающих данных, с перекрёстной проверкой) до 96% и выше.

4. Сделать прогноз по лучшей модели на отложенные наблюдения и оценить его точность.

Обратите внимание: при наличии среди объясняющих переменных категориальных из них необходимо сделать фиктивные переменные.

В качестве ядра генератора случайных чисел (в частности, для разделения данных на выборку для построения моделей и отложенные наблюдения) используйте номер своего варианта.

Источники

1. Джеймс Г., Уиттон Д., Хасти Т., Тибширани Р. Введение в статистическое обучение с примерами на языке R. Пер. с англ. С.Э. Мастицкого – М.: ДМК Пресс, 2016 – 450 с.
2. *Рашка С.* Python и машинное обучение: крайне необходимое пособие по новейшей предсказательной аналитике, обязательное для более глубокого понимания методологии машинного обучения / пер. с англ. А.В. Логунова. – М.: ДМК Пресс, 2017. – 418 с.: ил.