Демьянчук Г.В. ИУ5-22М Л/Р №4

Цель лабораторной работы

Изучить сложные способы подготовки выборки и подбора гиперпараметров на примере мето

Задание

Требуется выполнить следующие действия:

1. Выбрать набор данных (датасет) для решения задачи классификации или регресии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование к
3. С использованием метода `train_test_split` разделите выборку на обучающую и тестову
4. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оц трех подходящих для задачи метрик.
5. Постройте модель и оцените качество модели с использованием кросс-валидации. Прове различными стратегиями кросс-валидации.
6. Произведите подбор гиперпараметра K с использованием `GridSearchCV` и кросс-валида
7. Повторите пункт 4 для найденного оптимального значения гиперпараметра K. Сравните с качеством модели, полученной в пункте 4.
8. Постройте кривые обучения и валидации.

Ход выполнения работы

Подключим все необходимые библиотеки и настроим отображение графиков:

```
from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, RepeatedKFold, ShuffleSplit
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.model_selection import learning_curve, validation_curve
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
```

```
# Enable inline plots
%matplotlib inline

# Set plots formats to save high resolution PNG
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("retina")
```

Зададим ширину текстового представления данных, чтобы в дальнейшем текст в отчёте влеза

```
pd.set_option("display.width", 70)
```

Double-click (or enter) to edit

Предварительная подготовка данных

В качестве набора данных используются оценка качества белых вин по шкале с учетом химич

```
data = pd.read_csv("/content/whitew.csv")
```

Проверим полученные типы:

```
data.dtypes
```

```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

Посмотрим на данные в данном наборе данных:

```
data.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free su dio
0	7.4	0.70	0.00	1.9	0.076	
1	7.8	0.88	0.00	2.6	0.098	
2	7.8	0.76	0.04	2.3	0.092	

```
df = data.copy()
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free su dio
0	7.4	0.70	0.00	1.9	0.076	
1	7.8	0.88	0.00	2.6	0.098	
2	7.8	0.76	0.04	2.3	0.092	
3	11.2	0.28	0.56	1.9	0.075	
4	7.4	0.70	0.00	1.9	0.076	

```
df.dtypes
```

```
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

Проверим размер набора данных:

```
df.shape
```

```
(1599, 12)
```

Проверим основные статистические характеристики набора данных:

```
df.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	s di
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.4
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.8
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.0
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.0
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.0
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.0
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.0

Проверим наличие пропусков в данных:

```
df.isnull().sum()
```

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

Разделение данных

Разделим данные на целевой столбец и признаки:

```
X = df.drop("density", axis=1)
y = df["density"]


print(X.head(), "\n")
print(y.head())
```

```
   fixed acidity  volatile acidity  citric acid  ...  sulphates  alcohol  quality
0            7.4              0.70         0.00  ...       0.56      9.4        5
1            7.8              0.88         0.00  ...       0.68      9.8        5
2            7.8              0.76         0.04  ...       0.65      9.8        5
3           11.2              0.28         0.56  ...       0.58      9.8        6
4            7.4              0.70         0.00  ...       0.56      9.4        5

[5 rows x 11 columns]

0    0.9978
1    0.9968
```

```
print(X.shape)
print(y.shape)
```

```
(1599, 11)
(1599,)
```

Предобработаем данные, чтобы методы работали лучше:

```
columns = X.columns
scaler = StandardScaler()
X = scaler.fit_transform(X)
pd.DataFrame(X, columns=columns).describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sul diox
count	1.599000e+03	1.599000e+03	1.599000e+03	1.599000e+03	1.599000e+03	1.599000e
mean	3.435512e-16	1.699704e-16	4.335355e-16	-1.905223e-16	4.838739e-16	1.432042e
std	1.000313e+00	1.000313e+00	1.000313e+00	1.000313e+00	1.000313e+00	1.000313e
min	-2.137045e+00	-2.278280e+00	-1.391472e+00	-1.162696e+00	-1.603945e+00	-1.422500e
25%	-7.007187e-01	-7.699311e-01	-9.293181e-01	-4.532184e-01	-3.712290e-01	-8.487156e
50%	-2.410944e-01	-4.368911e-02	-5.636026e-02	-2.403750e-01	-1.799455e-01	-1.793002e
75%	5.057952e-01	6.266881e-01	7.652471e-01	4.341614e-02	5.384542e-02	4.901152e
max	4.355149e+00	5.877976e+00	3.743574e+00	9.195681e+00	1.112703e+01	5.367284e

Разделим выборку на тренировочную и тестовую:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.25, random_state=346705925)
```

```
print(X_train.shape)
print(X_test.shape)
```

```
print(y_train.shape)
print(y_test.shape)
```

```
(1199, 11)
(400, 11)
(1199,)
(400,)
```

Модель ближайших соседей для произвольно заданного гиперпа

Напишем функцию, которая считает метрики построенной модели:

```
def test_model(model):
    print("mean_absolute_error:",
          mean_absolute_error(y_test, model.predict(X_test)))
    print("median_absolute_error:",
          median_absolute_error(y_test, model.predict(X_test)))
    print("r2_score:",
          r2_score(y_test, model.predict(X_test)))
```

Попробуем метод ближайших соседей с гиперпараметром $K = 5$:

```
reg_5 = KNeighborsRegressor(n_neighbors=5)
reg_5.fit(X_train, y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

Проверим метрики построенной модели:

```
test_model(reg_5)
```

```
mean_absolute_error: 0.0006631799999999915
median_absolute_error: 0.0004889999999999617
r2_score: 0.7565853831960054
```

Видно, что средние ошибки не очень показательны для одной модели, они больше подходят д
время коэффициент детерминации неплох сам по себе, в данном случае модель более-менее

Использование кросс-валидации

Проверим различные стратегии кросс-валидации. Для начала посмотрим классический K-fold

```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=5), X, y,
                         cv=KFold(n_splits=10), scoring="r2")
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.54201038 0.62530238 0.60103483 0.42462355 0.16800613 0.67138819
 0.69157226 0.54801332 0.58253055 0.60540914]
0.5459890743612823 ± 0.14447376628869613
```

```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=5), X, y,
                         cv=RepeatedKFold(n_splits=5, n_repeats=2),
                         scoring="r2")
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.73450664 0.75859279 0.71236101 0.75826791 0.73064159 0.73583826
 0.73708952 0.72441664 0.76320957 0.73085156]
0.7385775486996173 ± 0.01559440793882926
```

```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=5), X, y,
                         cv=ShuffleSplit(n_splits=10), scoring="r2")
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.76101624 0.74486624 0.78022773 0.74529884 0.72949467 0.73281696
 0.75714985 0.75469655 0.75203314 0.80103417]
0.7558634383711698 ± 0.02034024152887018
```

Подбор гиперпараметра K

Введем список настраиваемых параметров:

```
n_range = np.array(range(1, 50, 2))
tuned_parameters = [{'n_neighbors': n_range}]
n_range
```

```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33,
       35, 37, 39, 41, 43, 45, 47, 49])
```

Запустим подбор параметра:

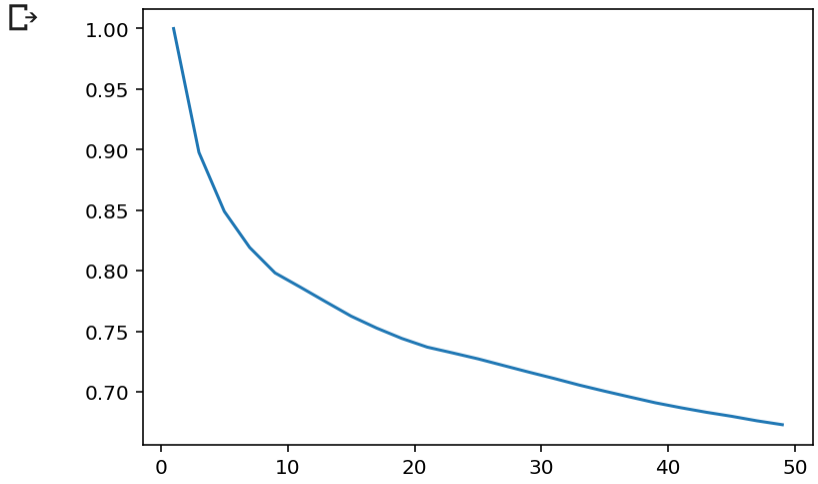
```
gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2"
```

```
                    return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_params_
```

```
{'n_neighbors': 3}
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данны

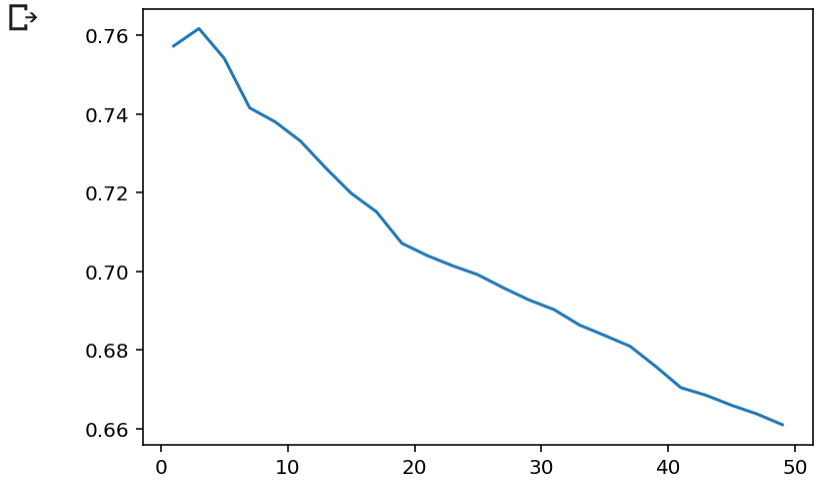
```
plt.plot(n_range, gs.cv_results_["mean_train_score"]);
```



Очевидно, что для $K = 1$ на тренировочном наборе данных мы находим ровно ту же точку, чт
её соседей мы берём — тем меньше точность.

На тестовом наборе данных картина сильно интереснее:

```
plt.plot(n_range, gs.cv_results_["mean_test_score"]);
```



Выходит, что сначала соседей слишком мало (высоко влияние выбросов), а затем количество слишком велико, и среднее значение по этим соседям всё больше и больше оттягивает значе

Проверим получившуюся модель:

```
reg = KNeighborsRegressor(**gs.best_params_)
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 0.0006740416666666596
median_absolute_error: 0.0005033333333333001
r2_score: 0.7550997432386466
```

В целом получили примерно тот же результат. Очевидно, что проблема в том, что данный мет результат для данной выборки.

Построим кривую обучения:

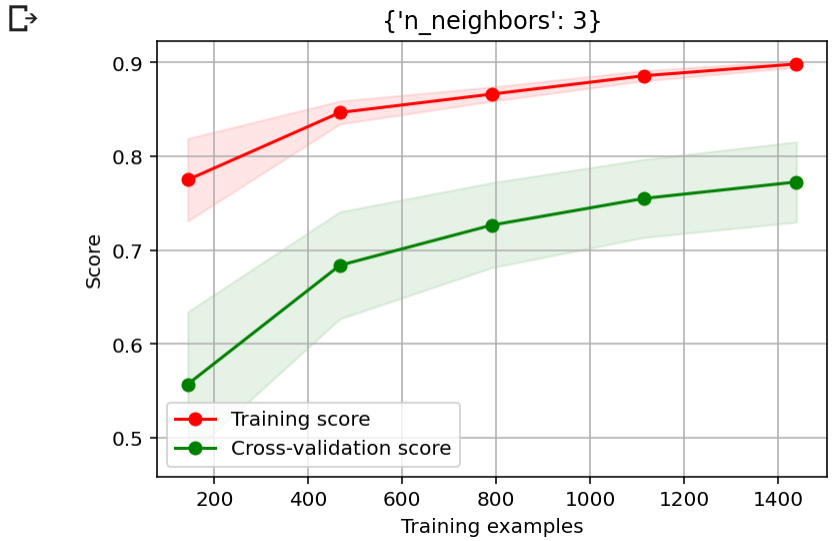
```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None):
    train_sizes=np.linspace(.1, 1.0, 5)

    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=-1, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,
                     color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1,
                     color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")
```

```
    plt.legend(loc="best")
    return plt


plot_learning_curve(reg, str(gs.best_params_), X, y,
                    cv=ShuffleSplit(n_splits=10));
```



Построим кривую валидации:

```
def plot_validation_curve(estimator, title, X, y,
                          param_name, param_range, cv,
                          scoring="accuracy"):

    train_scores, test_scores = validation_curve(
        estimator, X, y, param_name=param_name,
        param_range=param_range,
        cv=cv, scoring=scoring, n_jobs=-1)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.title(title)
    plt.xlabel(param_name)
    plt.ylabel("Score")
    plt.ylim(0.0, 1.1)
    lw = 2
    plt.plot(param_range, train_scores_mean, label="Training score",
                 color="darkorange", lw=lw)
    plt.fill_between(param_range, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.2,
                     color="darkorange", lw=lw)
    plt.plot(param_range, test_scores_mean,
                 label="Cross-validation score",
                 color="navy", lw=lw)
```

```
    plt.fill_between(param_range, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.2,
                     color="navy", lw=lw)
    plt.legend(loc="best")
    return plt


plot_validation_curve(KNeighborsRegressor(), "knn", X, y,
                      param_name="n_neighbors", param_range=n_range,
                      cv=ShuffleSplit(n_splits=10), scoring="r2");
```