Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Лабораторная работа №4
по дисциплине
«Методы машинного обучения»
на тему
«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей»

Выполнил:
студент группы РТ5-61Б
Корякин Д.

Москва — 2020 г.

1. Цель лабораторной работы

Изучить сложные способы подготовки выборки и подбора гиперпараметров на примере метода ближайших соседей [**?**].

2. 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 категориальных признаков.
3. С использованием метода `train_test_split` разделите выборку на обучающую и тестовую.
4. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью трех подходящих для задачи метрик.
5. Постройте модель и оцените качество модели с использованием кросс-валидации. Проведите эксперименты с тремя различными стратегиями кросс-валидации.
6. Произведите подбор гиперпараметра K с использованием `GridSearchCV` и кросс-валидации.
7. Повторите пункт 4 для найденного оптимального значения гиперпараметра K. Сравните качество полученной модели с качеством модели, полученной в пункте 4.
8. Постройте кривые обучения и валидации.

3. Ход выполнения работы

Подключим все необходимые библиотеки и настроим отображение графиков [**?**, **?**]:

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator
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 влезал на A4 [**?**]:

[2]:
```
pd.set_option("display.width", 70)
```

3.1. Предварительная подготовка данных

В качестве набора данных используются датасет показаателей недоедания по странам мира- https://www.kaggle.com/ruchi798/malnutrition-across-the-globe

[3]:
```
data = pd.read_csv("./data/country-wise-average.csv")
```

Проверим полученные типы:

[4]:
```
data.dtypes
```

[4]:
```
Country                    object
Income Classification     float64
Severe Wasting            float64
Wasting                   float64
Overweight                float64
Stunting                  float64
Underweight               float64
U5 Population ('000s)     float64
dtype: object
```

Посмотрим на данные в данном наборе данных:

[5]:
```
data.head()
```

[5]:
```
       Country  Income Classification  Severe Wasting    Wasting ▫
  ↪\
0  AFGHANISTAN                    0.0        3.033333  10.350000
1      ALBANIA                    2.0        4.075000   7.760000
2      ALGERIA                    2.0        2.733333   5.942857
3       ANGOLA                    1.0        2.400000   6.933333
4    ARGENTINA                    2.0        0.200000   2.150000

   Overweight   Stunting  Underweight  U5 Population ('000s)
0    5.125000  47.775000    30.375000             4918.561500
1   20.800000  24.160000     7.700000              232.859800
2   12.833333  19.571429     7.342857             3565.213143
3    2.550000  42.633333    23.600000             3980.054000
4   11.125000  10.025000     2.600000             3613.651750
```

Проверим размер набора данных:

[6]:
```
data.shape
```

[6]:
```
(152, 8)
```

Проверим основные статистические характеристики набора данных:

[7]:
```
data.describe()
```

[7]:
```
       Income Classification  Severe Wasting     Wasting  \
count             152.000000      140.000000  150.000000
mean                1.427632        2.168650    6.599257
std                 0.967019        1.708939    4.481723
min                 0.000000        0.000000    0.000000
25%                 1.000000        0.900000    3.262500
50%                 1.000000        1.872500    5.710714
75%                 2.000000        2.822727    8.740476
max                 3.000000       11.400000   23.650000

       Overweight    Stunting  Underweight  U5 Population ('000s)
count  149.000000  151.000000   150.000000              152.000000
mean     7.201638   25.814728    13.503047             4042.927052
std      4.649144   14.686807    10.895839            13164.191927
min      0.962500    1.000000     0.100000                1.000000
25%      3.850000   13.485000     4.305000              241.765813
50%      6.300000   24.160000    10.380000              981.233486
75%      9.080000   36.564935    19.496875             3002.433080
max     26.500000   57.600000    46.266667           123014.491000
```

Проверим наличие пропусков в данных:

[8]:
```
data.isnull().sum()
```

[8]:
```
Country                    0
Income Classification      0
Severe Wasting            12
Wasting                    2
Overweight                 3
Stunting                   1
Underweight                2
U5 Population ('000s)      0
dtype: int64
```

Уберем пропуски используя функции из прошлых лабораторных.

[9]:
```
from sklearn.impute import SimpleImputer


def impute_num_col(dataset, column, strategy_param):
    temp_data = dataset[[column]]
    imp_num = SimpleImputer(strategy=strategy_param)
    data_num_imp = imp_num.fit_transform(temp_data)
    return data_num_imp


def get_empty_num_columns(data):
    num_cols = []
    for col in data.columns:
        # Количество пустых значений
```

```
        temp_null_count = data[data[col].isnull()].shape[0]
        dt = str(data[col].dtype)
        if temp_null_count>0 and (dt=='float64' or dt=='int64'):
            num_cols.append(col)
    return num_cols


def impute_num_cols(data):
    num_cols = get_empty_num_columns(data)
    for col in num_cols:
        cols_data_imp = impute_num_col(dataset=data, column=col,
 ↪strategy_param='mean')
        data[col] = cols_data_imp
    return data

data = impute_num_cols(data)
data.isnull().sum()
```

[9]:
```
Country                   0
Income Classification     0
Severe Wasting            0
Wasting                   0
Overweight                0
Stunting                  0
Underweight               0
U5 Population ('000s)     0
dtype: int64
```

3.2. Разделение данных

Разделим данные на целевой столбец и признаки:

[10]:
```
#убираем категориальный признак
X = data.drop("Country", axis=1)
# целевой столбец - Severe Wasting
X = X.drop("Severe Wasting", axis = 1)
y = data["Severe Wasting"]
```

[11]:
```
print(X.head(), "\n")
print(y.head())
```

```
   Income Classification    Wasting  Overweight   Stunting  \
0                    0.0  10.350000    5.125000  47.775000
1                    2.0   7.760000   20.800000  24.160000
2                    2.0   5.942857   12.833333  19.571429
3                    1.0   6.933333    2.550000  42.633333
4                    2.0   2.150000   11.125000  10.025000

   Underweight  U5 Population ('000s)
0    30.375000            4918.561500
```

```
1     7.700000             232.859800
2     7.342857            3565.213143
3    23.600000            3980.054000
4     2.600000            3613.651750

0    3.033333
1    4.075000
2    2.733333
3    2.400000
4    0.200000
Name: Severe Wasting, dtype: float64
```

[12]:
```
print(X.shape)
print(y.shape)
```

```
(152, 6)
(152,)
```

Предобработаем данные, чтобы методы работали лучше:

[13]:
```
columns = X.columns
scaler = StandardScaler()
X = scaler.fit_transform(X)
pd.DataFrame(X, columns=columns).describe()
```

[13]:

	Income Classification	Wasting	Overweight	\
count	1.520000e+02	1.520000e+02	1.520000e+02	
mean	5.331992e-17	2.191230e-18	-7.632783e-17	
std	1.003306e+00	1.003306e+00	1.003306e+00	
min	-1.481203e+00	-1.487232e+00	-1.360011e+00	
25%	-4.436783e-01	-7.463493e-01	-7.224179e-01	
50%	-4.436783e-01	-1.868843e-01	-1.949826e-01	
75%	5.938463e-01	4.707480e-01	4.072667e-01	
max	1.631371e+00	3.842616e+00	4.206669e+00	

	Stunting	Underweight	U5 Population ('000s)
count	1.520000e+02	1.520000e+02	1.520000e+02
mean	-2.136449e-16	-6.391086e-17	2.191230e-18
std	1.003306e+00	1.003306e+00	1.003306e+00
min	-1.700820e+00	-1.242429e+00	-3.080545e-01
25%	-8.360070e-01	-8.517089e-01	-2.897046e-01
50%	-1.012748e-01	-2.485572e-01	-2.333463e-01
75%	7.268130e-01	5.411289e-01	-7.930100e-02
max	2.178586e+00	3.037105e+00	9.067390e+00

Разделим выборку на тренировочную и тестовую:

[14]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size=0.25,
 ↪random_state=346705925)
```

[15]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(114, 6)
(38, 6)
(114,)
(38,)
```

3.3. Модель ближайших соседей для произвольно заданного гиперпараметра K

Напишем функцию, которая считает метрики построенной модели:

[16]:
```
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 $K = 5$:

[17]:
```
reg_5 = KNeighborsRegressor(n_neighbors=5)
reg_5.fit(X_train, y_train)
```

[17]:
```
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:

[18]:
```
test_model(reg_5)
```

```
mean_absolute_error: 0.612842944974111
median_absolute_error: 0.41828780827046796
r2_score: 0.5723024044220422
```

Видно, что средние ошибки не очень показательны для одной модели, они больше подходят для сравнения разных моделей. В тоже время коэффициент детерминации неплох сам по себе, в данном случае модель более-менее состоятельна.

3.4. Использование кросс-валидации

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

[19]:
```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=5), X, y,
                         cv=KFold(n_splits=10), scoring="r2")
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.28427162 0.61257968 0.39848236 0.57720233 0.60681    0.75059246
 0.80958992 0.29238318 0.61542126 0.81120914]
0.5758541942941516 ± 0.18473472273887798
```

[20]:
```
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())
```

```
[0.60363622 0.57743569 0.64639056 0.76904062 0.55872881 0.61860221
 0.70638217 0.52101771 0.65235021 0.5753232 ]
0.6228907401736996 ± 0.06997395218921447
```

[21]:
```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=5), X, y,
                         cv=ShuffleSplit(n_splits=10),
  ↪scoring="r2")
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.8895004  0.39002193 0.53725253 0.84620712 0.27306257 0.41792732
 0.56898865 0.61880737 0.6838273  0.78862298]
0.6014218162391479 ± 0.1940092416154486
```

4. Подбор гиперпараметра K

Введем список настраиваемых параметров:

[22]:
```
n_range = np.array(range(1, 50, 2))
tuned_parameters = [{'n_neighbors': n_range}]
n_range
```

[22]:
```
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])
```

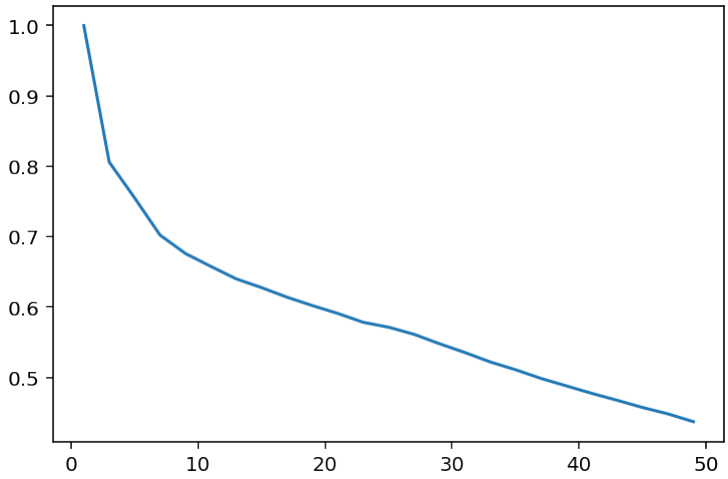
Запустим подбор параметра:

[23]:
```
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_
```

[23]:
```
{'n_neighbors': 11}
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

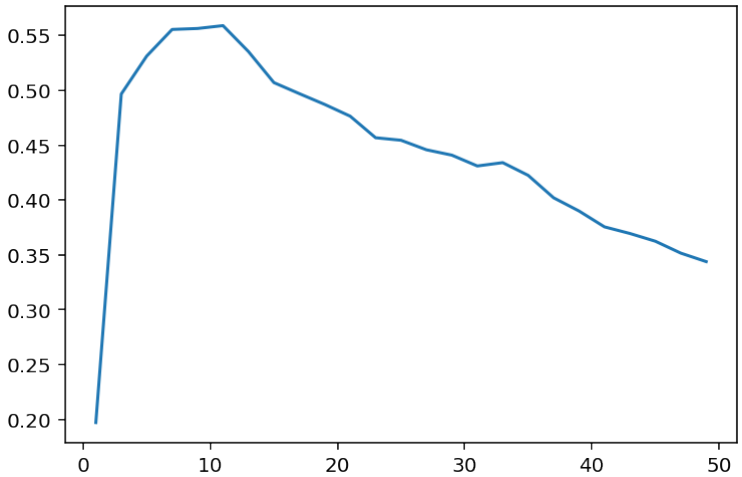
[24]:
```
plt.plot(n_range, gs.cv_results_["mean_train_score"]);
```



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

На тестовом наборе данных картина сильно интереснее:

[25]:
```
plt.plot(n_range, gs.cv_results_["mean_test_score"]);
```



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

Проверим получившуюся модель:

[26]:
```
reg = KNeighborsRegressor(**gs.best_params_)
reg.fit(X_train, y_train)
```

```
test_model(reg)
```

```
mean_absolute_error: 0.6383881515731873
median_absolute_error: 0.5252921494730495
r2_score: 0.6295001558140205
```

В целом получили примерно тот же результат. Очевидно, что проблема в том, что данный метод и так показал достаточно хороший результат для данной выборки.

Построим кривую обучения [**?**]:

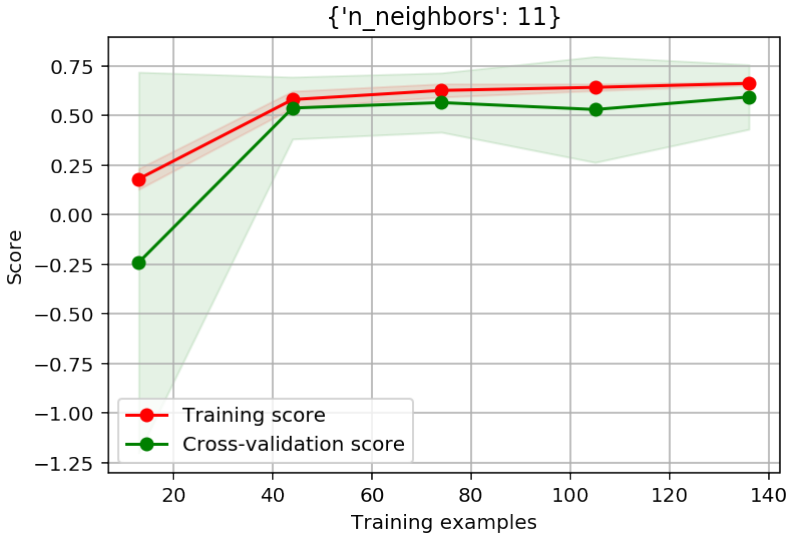
[27]:
```
def plot_learning_curve(estimator, title, X, y, ylim=None,
↪cv=None):
    train_sizes=np.linspace(.1, 1.0, 5)

    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
```

[28]:
```
plot_learning_curve(reg, str(gs.best_params_), X, y,
                    cv=ShuffleSplit(n_splits=10));
```



Построим кривую валидации:

[29]:
```
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
```

[30]:
```
plot_validation_curve(KNeighborsRegressor(), "knn", X, y,
                      param_name="n_neighbors",
↪param_range=n_range,
                      cv=ShuffleSplit(n_splits=10), scoring="r2");
```