Машинное обучение, ФКН ВШЭ

Практическое задание 2

Общая информация

Дата выдачи: 17.09.2018

Мягкий дедлайн: 2:59MSK 24.09.2018

Жесткий дедлайн: 23:59MSK 26.09.2018

О задании

Задание состоит из двух частей:

Первая часть задания посвящена получению студентами навыков работы с библиотекой векторного вычисления numpy и библиотеками для построения графиков matplotlib/plotly. Это задание сдается частично в Яндекс.Контест (см. информацию ниже). В случае проблем с доступом к Яндекс.Контесту обращайтесь к своему семинаристу(ке).

Вторая часть задания посвящена обучению линейной регрессии, подбору гиперпараметров и работе с данными.

Оценивание и штрафы

Каждая из задач имеет определенную «стоимость» (указана в скобках около задачи). Максимально допустимая оценка за работу — 10 баллов.

Сдавать задание после указанного срока сдачи нельзя. При выставлении неполного балла за задание в связи с наличием ошибок на усмотрение проверяющего предусмотрена возможность исправить работу на указанных в ответном письме условиях.

Задание выполняется самостоятельно. «Похожие» решения считаются плагиатом и все задействованные студенты (в том числе те, у кого списали) не могут получить за него больше 0 баллов (подробнее о плагиате см. на странице курса). Если вы нашли решение какого-то из заданий (или его часть) в открытом источнике, необходимо указать ссылку на этот источник в отдельном блоке в конце Вашей работы (скорее всего вы будете не единственным, кто это нашел, поэтому чтобы исключить подозрение в плагиате, необходима ссылка на источник).

Неэффективная реализация кода может негативно отразиться на оценке.

Формат сдачи

Ipython Notebook с выполненым заданием необходимо сдать в Anytask (lab_02). Также обратите внимание, что в первой части про Numpy также необходимо сделать две посылки в Яндекс.Контест.

Часть 1. Numpy

Задачи 1-6

(2.4 балла)

Ниже приведены задачи на работу с numpy-массивами. Для каждой из задач нужно привести 2 реализации: одна без использования numpy (считайте, что там, где на входе или выходе должны быть numpy array, будут просто списки), а вторая полностью векторизованная (без использования питоновских циклов/map/list comprehension). Невекторизованная реализация каждой из задач оценивается в **0.15 балла**, векторизованная – в **0.25 балла**.

Реализации без использования векторизации нужно записать в файл functions.py, а векторизованные — в файл functions_vectorized.py (см. шаблоны). Далее эти файлы необходимо сдать в Яндекс.Контест: https://official.contest.yandex.ru/contest/9148/enter/ (https://official.contest.yandex.ru/contest/9148/enter/) в соответствующие задачи. По техническим причинам тестирование проводится на этапе компиляции, поэтому в случае любой ошибки вы будете получать вердикт CE, и в логе компиляции можно будет посмотреть, в чем проблема. Частичное выполнение задания (не все задачи) будет оцениваться, хотя и будет получать вердикт CE. Для удобства проверки приложите в ячейке ниже ссылки на самые успешные посылки.

- **Задача 1**: Подсчитать произведение ненулевых элементов на диагонали прямоугольной матрицы.
  Например, для X = np.array([[1, 0, 1], [2, 0, 2], [3, 0, 3], [4, 4, 4]]) ответ – 3.

- **Задача 2**: Даны два вектора x и y. Проверить, задают ли они одно и то же мультимножество.
  Например, для x = np.array([1, 2, 2, 4]), y = np.array([4, 2, 1, 2]) ответ – True.

- **Задача 3**: Найти максимальный элемент в векторе x среди элементов, перед которыми стоит нулевой.
  Например, для x = np.array([6, 2, 0, 3, 0, 0, 5, 7, 0]) ответ – 5.

- __ Задача 4__: Дан трёхмерный массив, содержащий изображение, размера (height, width, numChannels), а также вектор длины numChannels. Сложить каналы изображения с указанными весами, и вернуть результат в виде матрицы размера (height, width). В ноутбуке приведите пример работы функции – преобразуйте цветное изображение в оттенки серого, использовав коэффициенты np.array([0.299, 0.587, 0.114]). Считать реальное изображение можно при помощи функции scipy.misc.imread (если изображение не в формате png, установите пакет pillow). **Обратите внимание, что в изображении может быть не три канала! За решения, явно или неявно использующие трёхканальность изображения будет присуждена половина баллов.**

- **Задача 5**: Реализовать кодирование длин серий (Run-length encoding). Для некоторого вектора x необходимо вернуть кортеж из двух векторов одинаковой длины. Первый содержит числа, а второй - сколько раз их нужно повторить.
  Например, для x = np.array([2, 2, 2, 3, 3, 3, 5]) ответ – (np.array([2, 3, 5]), np.array([3, 3, 1])).

- **Задача 6**: Даны две выборки объектов - X и Y. Вычислить матрицу евклидовых

расстояний между объектами. Сравните с функцией scipy.spatial.distance.cdist по скорости работы (сравнения приведите ниже в ноутбуке).

Замечание. Можно считать, что все указанные объекты непустые (к примеру, в **задаче 1** на диагонали матрицы есть ненулевые элементы) и корректные.

Укажите ссылки на посылки

Посылка по невекторизованным функциям: https://contest.yandex.ru/contest/9148/run-report/12055442/ (https://contest.yandex.ru/contest/9148/run-report/12055442/)

Посылка по векторизованным функциям: https://contest.yandex.ru/contest/9148/run-report/12055411/ (https://contest.yandex.ru/contest/9148/run-report/12055411/)

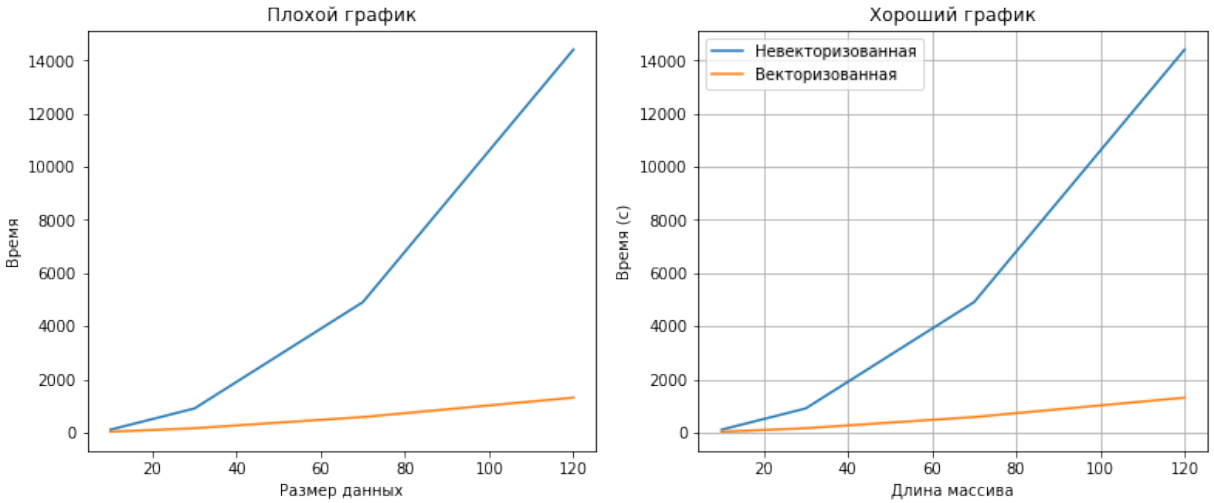
Задача 7

(1.6 балла)
Для каждой задачи сравните скорость работы невекторизованной и векторизованной реализации. С помощью пакета matplotlib или plotly постройте графики времени работы в зависимости от размера данных. **Графики должны выглядеть опрятно!** То есть должны быть подписаны оси, названия графиков, и т.д. Например, ниже представлены хороший и плохой графики:

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline

data_size = np.array([10, 30, 70, 120])
time_non_vectorized = data_size ** 2 + 10
time_vectorized = data_size ** 1.5

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(13, 5))

ax1.plot(data_size, time_non_vectorized)
ax1.plot(data_size, time_vectorized)
ax1.set_title(u"Плохой график")
ax1.set_xlabel(u"Размер данных")
ax1.set_ylabel(u"Время")

ax2.plot(data_size, time_non_vectorized, label=u"Невекторизованная
ax2.plot(data_size, time_vectorized, label=u"Векторизованная")
ax2.set_title(u"Хороший график")
ax2.set_xlabel(u"Длина массива")
ax2.set_ylabel(u"Время (с)")
ax2.grid()
ax2.legend()

f.show()
```

```
/usr/local/lib/python3.7/site-packages/matplotlib/figure.py:448: Use
rWarning: Matplotlib is currently using module://ipykernel.pylab.bac
kend_inline, which is a non-GUI backend, so cannot show the figure.
  % get_backend())
```



In [2]:
```
def build_execution_time_graph(time_non_vectorized, time_vectorize
    f, (ax2) = plt.subplots(1, 1, figsize=(13, 5))

    ax2.plot(data_sizes, time_non_vectorized, label=u"non_vectoriz
    ax2.plot(data_sizes, time_vectorized, label=u"vectorized")
    ax2.set_title(function_name + " execution time")
    ax2.set_xlabel(u"data size")
    ax2.set_ylabel(u"execution time, seconds")
    ax2.grid()
    ax2.legend()

    f.show()
```

In [3]:
```
import timeit

def calculate_execution_time(function, parameters):
    start_timer = timeit.default_timer()
    function(*parameters)
    return (timeit.default_timer() - start_timer)
```

In [4]:
```
import functions as funct
import functions_vectorized as vfunct
```

In [7]:
```
def compare_prod_non_zero_diag():
    data_sizes = [700 * step for step in range(1, 20)]
    time_non_vectorized = [
        calculate_execution_time(funct.prod_non_zero_diag, [np.rar
        for data_size in data_sizes
    ]
    time_vectorized = [
        calculate_execution_time(vfunct.prod_non_zero_diag, [np.ra
        for data_size in data_sizes
    ]
    build_execution_time_graph(
        time_non_vectorized,
        time_vectorized,
        data_sizes,
        u"prod_non_zero_diag"
    )
compare_prod_non_zero_diag()
```

```
/usr/local/lib/python3.7/site-packages/matplotlib/figure.py:448: Use
rWarning: Matplotlib is currently using module://ipykernel.pylab.bac
kend_inline, which is a non-GUI backend, so cannot show the figure.
  % get_backend())
```

In [8]:
```
def compare_are_multisets_equal():
    data_sizes = [100000 * step for step in range(1, 50)]
    time_non_vectorized = [
        calculate_execution_time(
            funct.are_multisets_equal, [np.random.rand(data_size),
        )
        for data_size in data_sizes
    ]
    time_vectorized = [
        calculate_execution_time(
            vfunct.are_multisets_equal, [np.random.rand(data_size)
        )
        for data_size in data_sizes
    ]
    build_execution_time_graph(
        time_non_vectorized,
        time_vectorized,
        data_sizes,
        u"are_multisets_equal"
    )
compare_are_multisets_equal()
```

```
/usr/local/lib/python3.7/site-packages/matplotlib/figure.py:448: Use
rWarning: Matplotlib is currently using module://ipykernel.pylab.bac
kend_inline, which is a non-GUI backend, so cannot show the figure.
  % get_backend())
```

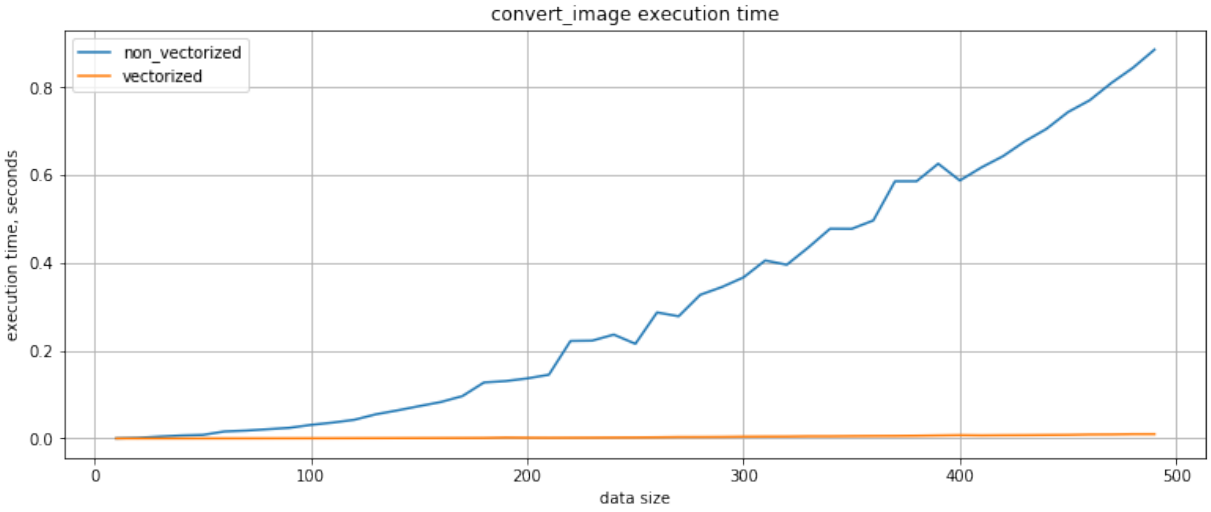
In [9]:
```
def compare_max_after_zero():
    data_sizes = [100000 * step for step in range(1, 30)]
    time_non_vectorized = [
        calculate_execution_time(
            funct.max_after_zero, [np.random.randint(10, size = da
        )
        for data_size in data_sizes
    ]
    time_vectorized = [
        calculate_execution_time(
            vfunct.max_after_zero, [np.random.randint(10, size = d
        )
        for data_size in data_sizes
    ]
    build_execution_time_graph(
        time_non_vectorized,
        time_vectorized,
        data_sizes,
        u"max_after_zero"
    )
compare_max_after_zero()
```

```
/usr/local/lib/python3.7/site-packages/matplotlib/figure.py:448: Use
rWarning: Matplotlib is currently using module://ipykernel.pylab.bac
kend_inline, which is a non-GUI backend, so cannot show the figure.
  % get_backend())
```

In [10]:

```
def compare_convert_image():
    data_sizes = [10 * step for step in range(1, 50)]
    data_size = 3
    import math
    time_non_vectorized = [
        calculate_execution_time(
            funct.convert_image,
            [
                np.random.rand(data_size, data_size, int(math.log(
                np.random.rand(int(math.log(data_size, 2)))
            ]
        )
        for data_size in data_sizes
    ]
    time_vectorized = [
        calculate_execution_time(
            vfunct.convert_image,
            [
                np.random.rand(data_size, data_size, int(math.log(
                np.random.rand(int(math.log(data_size, 2)))
            ]
        )
        for data_size in data_sizes
    ]
    build_execution_time_graph(
        time_non_vectorized,
        time_vectorized,
        data_sizes,
        u"convert_image"
    )
compare_convert_image()
```

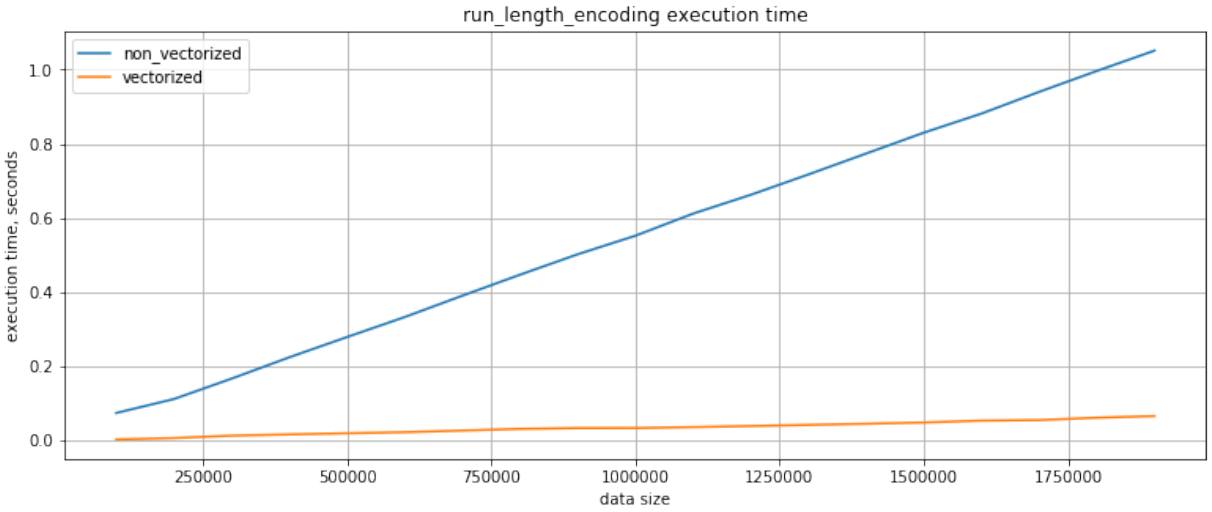
```
/usr/local/lib/python3.7/site-packages/matplotlib/figure.py:448: Use
rWarning: Matplotlib is currently using module://ipykernel.pylab.bac
kend_inline, which is a non-GUI backend, so cannot show the figure.
  % get_backend())
```



In [11]:

```
def compare_run_length_encoding():
    data_sizes = [100000 * step for step in range(1, 20)]
    time_non_vectorized = [
        calculate_execution_time(
            funct.run_length_encoding, [np.random.randint(10, size
        )
        for data_size in data_sizes
    ]
    time_vectorized = [
        calculate_execution_time(
            vfunct.run_length_encoding, [np.random.randint(10, siz
        )
        for data_size in data_sizes
    ]
    build_execution_time_graph(
        time_non_vectorized,
        time_vectorized,
        data_sizes,
        u"run_length_encoding"
    )
compare_run_length_encoding()
```

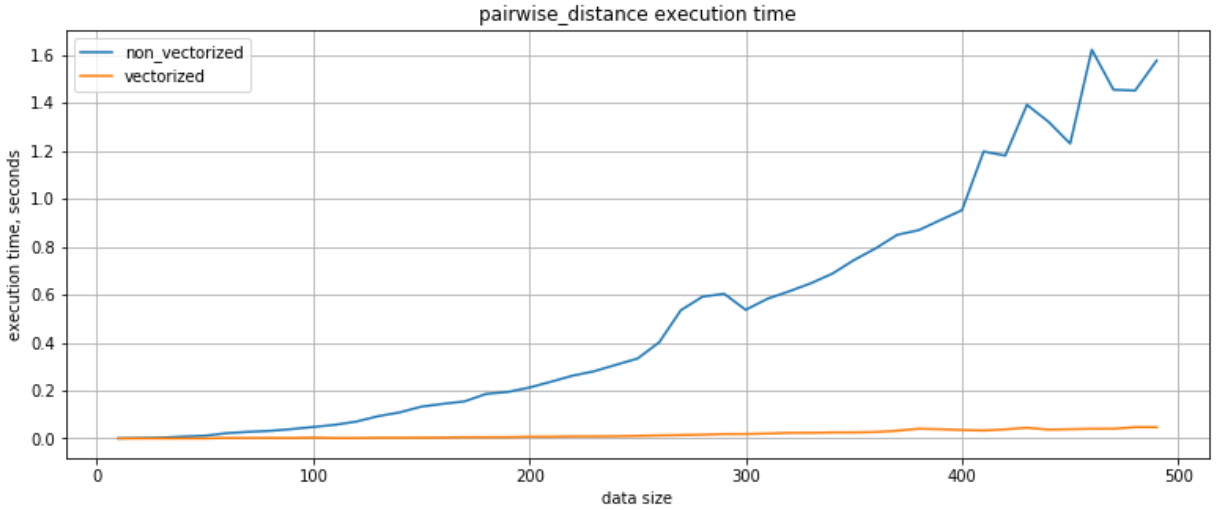
```
/usr/local/lib/python3.7/site-packages/matplotlib/figure.py:448: Use
rWarning: Matplotlib is currently using module://ipykernel.pylab.bac
kend_inline, which is a non-GUI backend, so cannot show the figure.
  % get_backend())
```



In [13]:

```
import math
def compare_pairwise_distance():
    data_sizes = [10 * step for step in range(1, 50)]
    time_non_vectorized = [
        calculate_execution_time(
            funct.pairwise_distance,
            [
                np.random.rand(data_size, int(math.log(data_size,
                np.random.rand(data_size, int(math.log(data_size,
            ]
        )
        for data_size in data_sizes
    ]
    time_vectorized = [
        calculate_execution_time(
            vfunct.pairwise_distance,
            [
                np.random.rand(data_size, int(math.log(data_size,
                np.random.rand(data_size, int(math.log(data_size,
            ]
        )
        for data_size in data_sizes
    ]
    build_execution_time_graph(
        time_non_vectorized,
        time_vectorized,
        data_sizes,
        u"pairwise_distance"
    )
compare_pairwise_distance()
```

```
/usr/local/lib/python3.7/site-packages/matplotlib/figure.py:448: Use
rWarning: Matplotlib is currently using module://ipykernel.pylab.bac
kend_inline, which is a non-GUI backend, so cannot show the figure.
  % get_backend())
```



Часть 2. Линейная регрессия

In [14]:
```
%pylab inline

import seaborn as sns
sns.set(style="whitegrid")

from sklearn import datasets
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import pandas as pd
```

```
Populating the interactive namespace from numpy and matplotlib

/usr/local/lib/python3.7/site-packages/IPython/core/magics/pylab.py:
160: UserWarning: pylab import has clobbered these variables: ['f']
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```

In [15]:
```
def pairplot(df, target):
    ncol, nrow = 7, df.shape[1] // 7 + (df.shape[1] % 7 > 0)
    plt.figure(figsize=(ncol * 4, nrow * 4))

    for i, feature in enumerate(df.columns):
        plt.subplot(nrow, ncol, i + 1)
        plt.scatter(df[feature], target, s=10, marker='o', alpha=.
        plt.xlabel(feature)
        if i % ncol == 0:
            plt.ylabel('target')
```

В данном задании мы рассмотрим стандартный датасет для задачи регрессии Boston Housing, в котором необходимо предсказать стоимость недвижимости по 13 признакам.

Датасет достаточно просто загрузить из библиотеки scikit-learn:

In [16]:
```
data = datasets.load_boston()
attributes = pd.DataFrame(data.data)
attributes.columns = data.feature_names
target = data.target
```

In [17]:
```
print(data['DESCR'])
```

```
Boston House Prices dataset
```

```
==========================


Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots ove
r 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds
river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million
)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to
1940
        - DIS      weighted distances to five Boston employment cent
res
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of b
lacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
http://archive.ics.uci.edu/ml/datasets/Housing
(http://archive.ics.uci.edu/ml/datasets/Housing)


This dataset was taken from the StatLib library which is maintained
at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hed
onic
prices and the demand for clean air', J. Environ. Economics & Manage
ment,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression di
agnostics
...', Wiley, 1980.   N.B. Various transformations are used in the ta
```

```
ble on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning p
apers that address regression
problems.

**References**

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Inf
luential Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Lea
rning. In Proceedings on the Tenth International Conference of Machi
ne Learning, 236-243, University of Massachusetts, Amherst. Morgan K
aufmann.
   - many more! (see http://archive.ics.uci.edu/ml/datasets/Housing)
(http://archive.ics.uci.edu/ml/datasets/Housing))
```

In [18]:
```
1 attributes.head()
```

Out[18]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	

In [19]:
```
1 print(target[:4])
```

```
[24.  21.6 34.7 33.4]
```

Разделим выборку на обучающую и тестовую в отношении 8/2:

In [20]:
```
1 attributes_train, attributes_test, target_train, target_test = tra
```

```
/usr/local/lib/python3.7/site-packages/sklearn/model_selection/_spli
t.py:2026: FutureWarning: From version 0.21, test_size will always c
omplement train_size unless both are specified.
  FutureWarning)
```

Задание 1. Обучение линейной регрессии.

(0.5 балл)

Обучите стандартную линейную регрессию, а также с L_1 и L_2 регуляризаторами (используйте параметры по умолчанию). Посчитайте метрику R^2 для каждого метода (метод score).

In [21]:
```
import sklearn.linear_model as lm
import sklearn.preprocessing as pr
def calculate_standart_regression():
    regression = lm.LinearRegression()
    regression.fit(attributes_train, target_train)
    print("standart regulization:")
    print("score on train sample:", regression.score(attributes_tr
    print("score on test sample:", regression.score(attributes_tes
calculate_standart_regression()
```

```
standart regulization:
score on train sample: 0.7508837786732915
score on test sample: 0.6684825753971597

/usr/local/lib/python3.7/site-packages/sklearn/linear_model/base.py:
509: RuntimeWarning: internal gelsd driver lwork query error, requir
ed iwork dimension not returned. This is likely the result of LAPACK
bug 0038, fixed in LAPACK 3.2.2 (released July 21, 2010). Falling ba
ck to 'gelss' driver.
  linalg.lstsq(X, y)
```

In [22]:
```
def calculate_l1_regression():
    regression = lm.Lasso()
    regression.fit(attributes_train, target_train)
    print("l1 regulization:")
    print("score on train sample:", regression.score(attributes_tr
    print("score on test sample:", regression.score(attributes_tes
calculate_l1_regression()
```

```
l1 regulization:
score on train sample: 0.6958999474620655
score on test sample: 0.6668687223368213
```

In [23]:
```
def calculate_l2_regression():
    regression = lm.Ridge()
    regression.fit(attributes_train, target_train)
    print("l2 regulization:")
    print("score on train sample:", regression.score(attributes_tr
    print("score on test sample:", regression.score(attributes_tes
calculate_l2_regression()
```

```
l2 regulization:
score on train sample: 0.7487713711418569
score on test sample: 0.6659608075261689
```

Задание 2. Подбор гиперпараметров.

(1 балл)

Для Lasso- и Ridge-регресий подберите коэффициент регуляризации по обучающей выборке с помощью кросс-валидации. Параметры для перебора возьмите по логарифмической сетке от 10^{-6} до 10^{6}). Также посчитайте метрику R^2 для тестовой выборки и сравните с предыдущими результатами. Заметно ли изменилось качество?

Useful: GridSearchCV, RidgeCV, LassoCV

In [24]:
```
def calculate_ridge_regularization_coefficient():
    regression = lm.RidgeCV(
        alphas=np.log2(np.arange(1e-6, 1e6, 100)),
        store_cv_values=True,
    )
    regression.fit(attributes_train, target_train)
    print("RidgeCV regression:")
    print("regularization coefficient:", regression.alpha_)
    print("score on train sample:", regression.score(attributes_tr
    print("score on test sample:", regression.score(attributes_tes
calculate_ridge_regularization_coefficient()
```

```
RidgeCV regression:
regularization coefficient: 6.643856204201675
score on train sample: 0.7429713402782756
score on test sample: 0.6627401932541358
```

In [25]:
```
def calculate_lasso_regularization_coefficient():
    regression = lm.LassoCV(
        alphas=np.log2(np.arange(1e-6, 1e6, 100))
    )
    regression.fit(attributes_train, target_train)
    print("LassoCV regression:")
    print("regularization coefficient:", regression.alpha_)
    print("score on train sample:", regression.score(attributes_tr
    print("score on test sample:", regression.score(attributes_tes
calculate_lasso_regularization_coefficient()
```

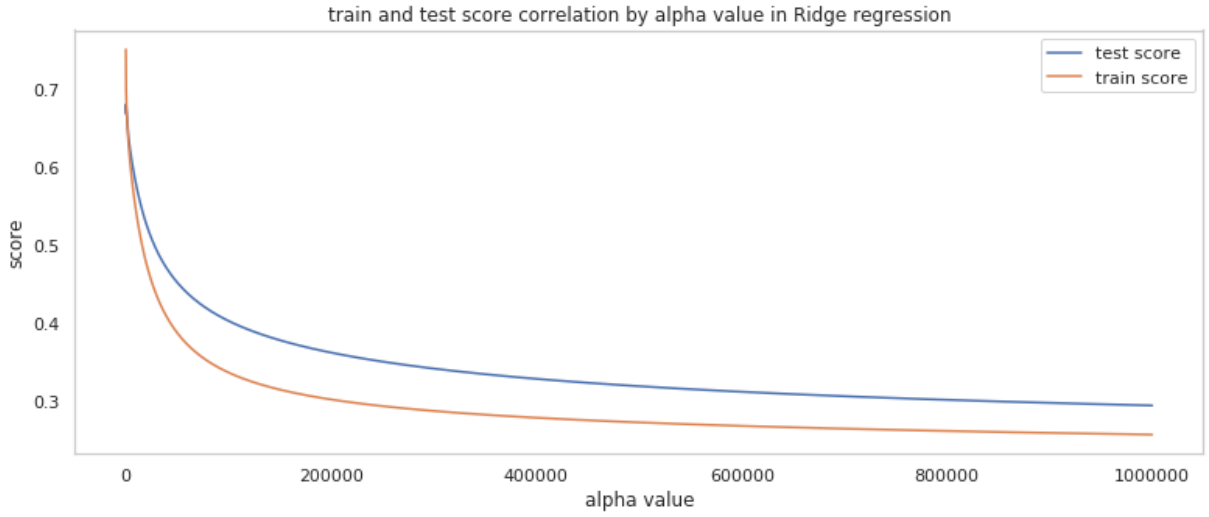
```
LassoCV regression:
regularization coefficient: 6.643856204201675
score on train sample: 0.5510142723812895
score on test sample: 0.549865805812861
```

Постройте график зависимости R^2 для обучающей и тестовой (на кросс-валидации) выборок в зависимости от значения гиперпараметра. Для Lasso также постройте график зависимости количества ненулевых весов.

In [26]:

```
def draw_ridge_alpha_correlation():
    scores_train = []
    scores_test = []
    alpha_values = np.arange(1e-6, 1e6, 100)
    for alpha_value in alpha_values:
        regression = lm.Ridge(alpha=alpha_value)
        regression.fit(attributes_train, target_train)
        scores_train.append(regression.score(attributes_train, tar
        scores_test.append(regression.score(attributes_test, targe

    f, (ax2) = plt.subplots(1, 1, figsize=(13, 5))

    ax2.plot(alpha_values, scores_test, label=u"test score")
    ax2.plot(alpha_values, scores_train, label=u"train score")
    ax2.set_title(u"train and test score correlation by alpha valu
    ax2.set_xlabel(u"alpha value")
    ax2.set_ylabel(u"score")
    ax2.grid()
    ax2.legend()

    f.show()
draw_ridge_alpha_correlation()
```

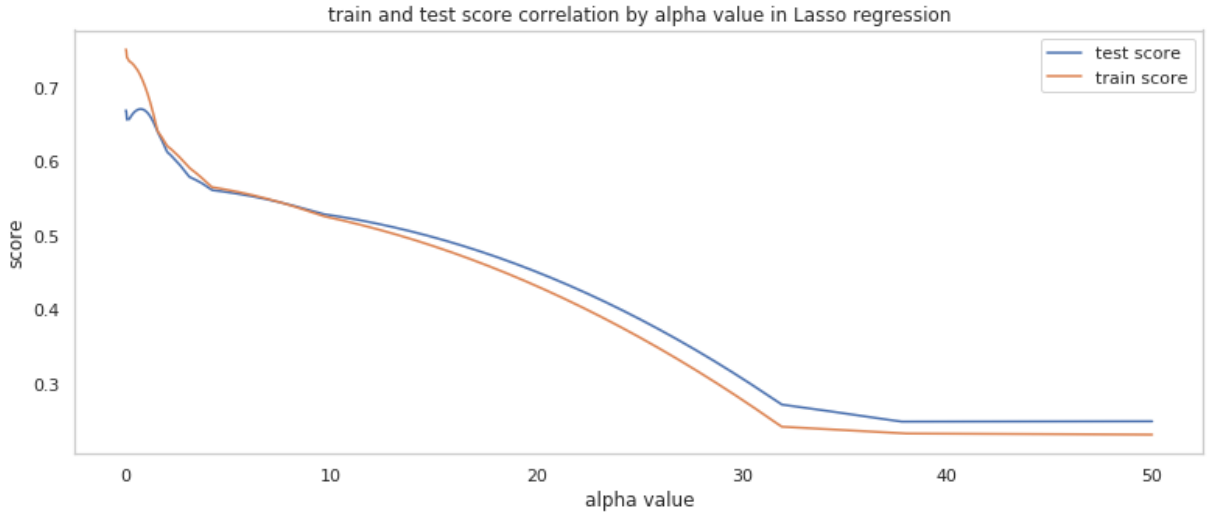
```
/usr/local/lib/python3.7/site-packages/matplotlib/figure.py:448: Use
rWarning: Matplotlib is currently using module://ipykernel.pylab.bac
kend_inline, which is a non-GUI backend, so cannot show the figure.
  % get_backend())
```



In [27]:

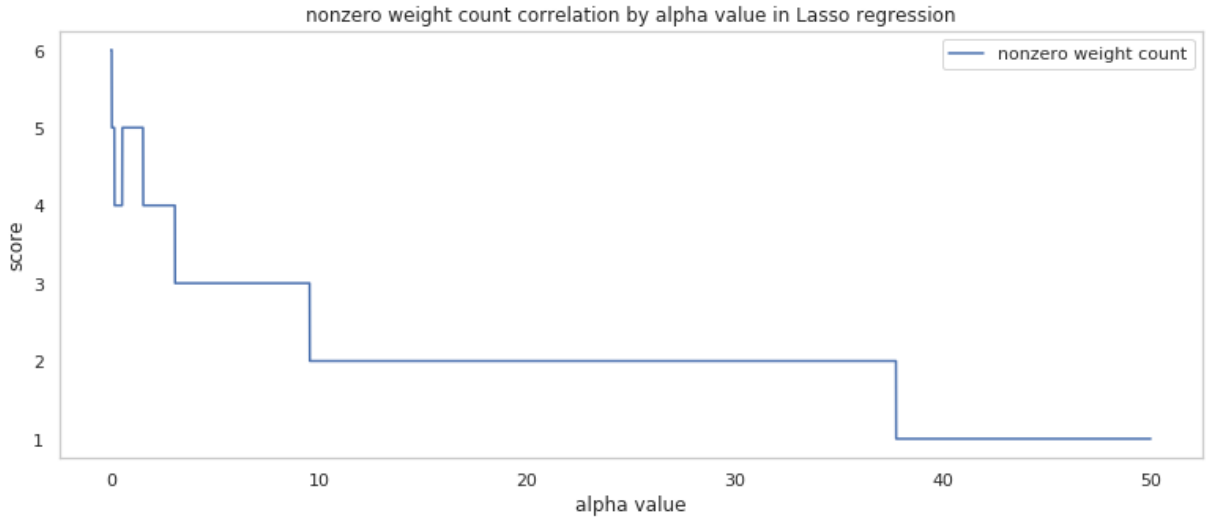
```
def draw_lasso_alpha_correlation():
    scores_train = []
    scores_test = []
    alpha_values = np.arange(1e-6, 50, 0.005)
    for alpha_value in alpha_values:
        regression = lm.Lasso(alpha=alpha_value)
        regression.fit(attributes_train, target_train)
        scores_train.append(regression.score(attributes_train, tar
        scores_test.append(regression.score(attributes_test, targe

    f, (ax2) = plt.subplots(1, 1, figsize=(13, 5))

    ax2.plot(alpha_values, scores_test, label=u"test score")
    ax2.plot(alpha_values, scores_train, label=u"train score")
    ax2.set_title(u"train and test score correlation by alpha valu
    ax2.set_xlabel(u"alpha value")
    ax2.set_ylabel(u"score")
    ax2.grid()
    ax2.legend()

    f.show()
draw_lasso_alpha_correlation()
```

```
/usr/local/lib/python3.7/site-packages/matplotlib/figure.py:448: Use
rWarning: Matplotlib is currently using module://ipykernel.pylab.bac
kend_inline, which is a non-GUI backend, so cannot show the figure.
  % get_backend())
```



In [28]:
```
def draw_lasso_nonzero_weight_count_correlation():
    nonzero_weight_count = []
    alpha_values = np.arange(1e-6, 50, 0.005)
    for alpha_value in alpha_values:
        regression = lm.Lasso(alpha=alpha_value)
        regression.fit(attributes_train, target_train)
        nonzero_weight_count.append(sum(regression.coef_ > 0))

    f, (ax2) = plt.subplots(1, 1, figsize=(13, 5))

    ax2.plot(alpha_values, nonzero_weight_count, label=u"nonzero w
    ax2.set_title(u"nonzero weight count correlation by alpha valu
    ax2.set_xlabel(u"alpha value")
    ax2.set_ylabel(u"score")
    ax2.grid()
    ax2.legend()

    f.show()
draw_lasso_nonzero_weight_count_correlation()
```

```
/usr/local/lib/python3.7/site-packages/matplotlib/figure.py:448: Use
rWarning: Matplotlib is currently using module://ipykernel.pylab.bac
kend_inline, which is a non-GUI backend, so cannot show the figure.
  % get_backend())
```



В какой момент происходит недообучение? Почему?

Ответ: По графику видно, что переобучение Ridge происходит при коэффициенте 0.7. Lasso переобучается при коэффициенте 6 -- 10, видимо из-за изменения количества ненулевых векторов (какой-то значимый вектор перестает влиять на результат).

Подготовка данных

Как вы могли заметить, большого прироста качества с помощью подбора коэфициента регуляризации добиться не удалось. Поэтому прежде чем бросаться обучать модели, необходимо изучить и предобработать данные.

Задание 3. Масштабирование.

(1 балл)

Зачастую признаки в сырых данных имеют разный масштаб. Попробуйте применить масштабирование к данным, сравните качество. Заметно ли изменение?

Pipeline, StandardScaler, MinMaxScaler

In [29]:
```
def calculate_regressions_with_standart_scaler():
    scaler = pr.StandardScaler()
    scaled_attributes_train = scaler.fit_transform(attributes_trai
    scaled_attributes_test = scaler.fit_transform(attributes_test,

    linear_regression = lm.LinearRegression()
    linear_regression.fit(scaled_attributes_train, target_train)
    ridge_regression = lm.Ridge()
    ridge_regression.fit(scaled_attributes_train, target_train)
    lasso_regression = lm.Lasso()
    lasso_regression.fit(scaled_attributes_train, target_train)
    print("standart scaler:")
    print("scaled simple linear regression score:", linear_regress
    print("scaled ridge regression score:", ridge_regression.score
    print("scaled lasso regression score:", lasso_regression.score

calculate_regressions_with_standart_scaler()
```

```
standart scaler:
scaled simple linear regression score: 0.6260120845037371
scaled ridge regression score: 0.6260848349266783
scaled lasso regression score: 0.6068805552998523
```

In [30]:
```
def calculate_regressions_with_minmax_scaler():
    scaler = pr.MinMaxScaler()
    scaled_attributes_train = scaler.fit_transform(attributes_trai
    scaled_attributes_test = scaler.fit_transform(attributes_test,

    linear_regression = lm.LinearRegression()
    linear_regression.fit(scaled_attributes_train, target_train)
    ridge_regression = lm.Ridge()
    ridge_regression.fit(scaled_attributes_train, target_train)
    lasso_regression = lm.Lasso()
    lasso_regression.fit(scaled_attributes_train, target_train)
    print("minmax scaler:")
    print("scaled simple linear regression score:", linear_regress
    print("scaled ridge regression score:", ridge_regression.score
    print("scaled lasso regression score:", lasso_regression.score

calculate_regressions_with_minmax_scaler()
```

```
minmax scaler:
scaled simple linear regression score: 0.569764132570967
scaled ridge regression score: 0.6013620981305376
scaled lasso regression score: 0.26666070646435747
```

Сравните оптимальное значение коэффициента регуляризации для Ridge-регрессии до и после масштабирования. Изменилось ли оно? Предположите, почему так могло произойти.

In [31]:
```
def calculate_regression_coeficients_after_standart_scaling():
    scaler = pr.MinMaxScaler()
    scaled_attributes_train = scaler.fit_transform(attributes_trai

    ridge_regression = lm.RidgeCV(
        alphas=np.log2(np.arange(1e-6, 1e6, 100)),
        store_cv_values=True,
    )
    ridge_regression.fit(scaled_attributes_train, target_train)

    print("new ridge coefficient:", ridge_regression.alpha_)

calculate_regression_coeficients_after_standart_scaling()
```

```
new ridge coefficient: 6.643856204201675
```

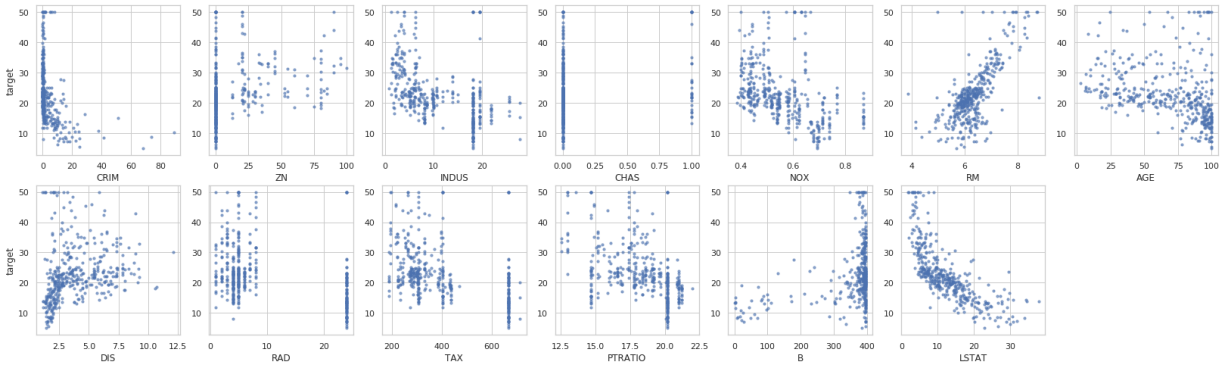
Ответ: Значение коэффициента не изменилось, поскольку MinMax масштабирование нормирует данные относительно целевого вектора.

Задание 4. Новые признаки.

(1.5 балла)

Полезным также бывает посмотреть как целевая переменная зависит от каждого признака.

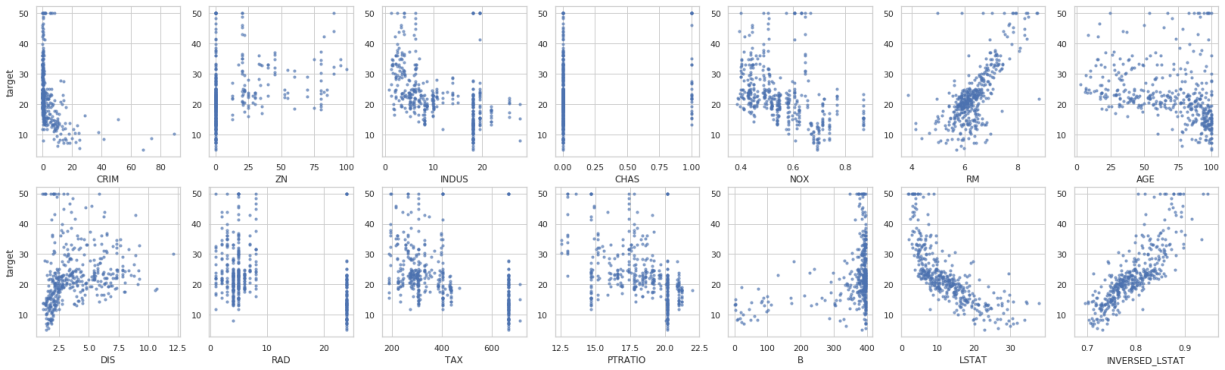
In [32]:
```
1 pairplot(attributes_train, target_train)
```



Обратите внимание на зависимость цены от признака LSTAT. Является ли эта зависимость линейной? А какой? Попробуйте выбрать преобразование для этого признака так, чтобы получившаяся зависимость была более линейной. Добейтесь R^2 на тестовой выборке не меньше 0.71.

Ответ: Прослеживается гиперболическая зависимость. Для линейности можно попробовать взять $1/\sqrt[10]{LSTAT}$.

In [43]:
```
1 extended_attributes_train = attributes_train
2 attributes_train = extended_attributes_train.drop(columns = ["INVE
3 pairplot(extended_attributes_train, target_train)
4
```



Помимо преобразований отдельный признаков полезными бывают их попарные взаимодействия. Воспользуйтесь PolynomialFeatures, чтобы добавить попарные произведения и квадраты всех признаков. Обучите Ridge-регрессию (подберите гиперпараметр!) и посчитайте R^2 на тесте. Сильно ли изменилось качество?

In [47]:
```
polynomial_features = pr.PolynomialFeatures(2)
ridge_regression = lm.RidgeCV(alphas=np.log2(np.arange(1e-6, 1e6,
ridge_regression.fit(polynomial_features.fit_transform(attributes_
print(ridge_regression.score(polynomial_features.fit_transform(att
```

```
0.799489615663477
```

Да, при попарном взаимодействии качество увеличивается на порядок.

Задание 5. Оптимальная архитектура, анализ.

(2 балла)

При помощи `Pipeline` и `GridSearchSCV` выберите оптимальную архитектуру, комбинируя различные методы масштабирования, степень полинома в `PolynomialFeatures`, а также регуляризаторы. Для `Lasso` поставьте максимальное количество итераций больше значения по умолчанию, чтобы оптимизация сошлась.

In [ ]:
```
#  Your code here
# ...
```

Для одной комбинаций метода масштабирования и линейной регрессии постройте зависимость метрики R^2 на кросс-валидации (`GridSearchCV.cv_results_['mean_test_score']`) от значения параметра регуляризации для различных степеней полиномов в `PolynomialFeatures`.

In [ ]:
```
#  Your code here
# ...
```

При каких значениях происходит переобучение? А недообучение? Почему?

Ответ:

Оценка: 8