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

Курсовая работа
по дисциплине
«Методы машинного обучения»
на тему
«Построение моделей машинного обучения»

Выполнил:
студент группы РТ5-61Б
Корякин Д.

_____

Москва — 2020 г.

1. Домашнее задание

Корякин Данила Алексеевич, группа РТ5-61Б.

1.1. Задание

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее двух метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее трех моделей, хотя бы одна из которых должна быть ансамблевой.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
9. Подбор гиперпараметров для выбранных моделей. Рекомендуется подбирать не более 1-2 гиперпараметров. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик.

2. Ход работы

[3]:
```
from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import accuracy_score,
 ↪balanced_accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler


from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import accuracy_score,
 ↪balanced_accuracy_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 KNeighborsClassifier
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import Lasso, LinearRegression,
 ↪LogisticRegression
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error, r2_score
from sklearn.metrics import accuracy_score,
 ↪balanced_accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import NuSVR, LinearSVC, SVC
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import export_graphviz, plot_tree

import seaborn as sns

# Enable inline plots
%matplotlib inline
```

[4]:
```
data = pd.read_csv('data/heart.csv', header=0)
```

[5]:
```
df = data.copy()
```

[6]:
```
df.describe()
```

[6]:
```
              age         sex          cp    trestbps        chol
 ↪        fbs  \
count  303.000000  303.000000  303.000000  303.000000  303.000000
 ↪ 303.000000
mean    54.366337    0.683168    0.966997  131.623762  246.264026
 ↪   0.148515
std      9.082101    0.466011    1.032052   17.538143   51.830751
 ↪   0.356198
```

```
min     29.000000    0.000000    0.000000   94.000000  126.000000
  ↪   0.000000
25%     47.500000    0.000000    0.000000  120.000000  211.000000
  ↪   0.000000
50%     55.000000    1.000000    1.000000  130.000000  240.000000
  ↪   0.000000
75%     61.000000    1.000000    2.000000  140.000000  274.500000
  ↪   0.000000
max     77.000000    1.000000    3.000000  200.000000  564.000000
  ↪   1.000000

          restecg     thalach       exang     oldpeak       slope
  ↪         ca  \
count  303.000000  303.000000  303.000000  303.000000  303.000000
  ↪ 303.000000
mean     0.528053  149.646865    0.326733    1.039604    1.399340
  ↪   0.729373
std      0.525860   22.905161    0.469794    1.161075    0.616226
  ↪   1.022606
min      0.000000   71.000000    0.000000    0.000000    0.000000
  ↪   0.000000
25%      0.000000  133.500000    0.000000    0.000000    1.000000
  ↪   0.000000
50%      1.000000  153.000000    0.000000    0.800000    1.000000
  ↪   0.000000
75%      1.000000  166.000000    1.000000    1.600000    2.000000
  ↪   1.000000
max      2.000000  202.000000    1.000000    6.200000    2.000000
  ↪   4.000000

             thal      target
count  303.000000  303.000000
mean     2.313531    0.544554
std      0.612277    0.498835
min      0.000000    0.000000
25%      2.000000    0.000000
50%      2.000000    1.000000
75%      3.000000    1.000000
max      3.000000    1.000000
```

[7]:
```
df.corr()
```

[7]:
```
                age       sex        cp  trestbps      chol       
  ↪fbs  \
age        1.000000 -0.098447 -0.068653  0.279351  0.213678  0.
  ↪121308
sex       -0.098447  1.000000 -0.049353 -0.056769 -0.197912  0.
  ↪045032
```

```
cp       -0.068653 -0.049353  1.000000  0.047608 -0.076904  0.
 ↪094444
trestbps  0.279351 -0.056769  0.047608  1.000000  0.123174  0.
 ↪177531
chol      0.213678 -0.197912 -0.076904  0.123174  1.000000  0.
 ↪013294
fbs       0.121308  0.045032  0.094444  0.177531  0.013294  1.
 ↪000000
restecg  -0.116211 -0.058196  0.044421 -0.114103 -0.151040 -0.
 ↪084189
thalach  -0.398522 -0.044020  0.295762 -0.046698 -0.009940 -0.
 ↪008567
exang     0.096801  0.141664 -0.394280  0.067616  0.067023  0.
 ↪025665
oldpeak   0.210013  0.096093 -0.149230  0.193216  0.053952  0.
 ↪005747
slope    -0.168814 -0.030711  0.119717 -0.121475 -0.004038 -0.
 ↪059894
ca        0.276326  0.118261 -0.181053  0.101389  0.070511  0.
 ↪137979
thal      0.068001  0.210041 -0.161736  0.062210  0.098803 -0.
 ↪032019
target   -0.225439 -0.280937  0.433798 -0.144931 -0.085239 -0.
 ↪028046

            restecg   thalach     exang   oldpeak     slope
 ↪ca  \
age      -0.116211 -0.398522  0.096801  0.210013 -0.168814  0.
 ↪276326
sex      -0.058196 -0.044020  0.141664  0.096093 -0.030711  0.
 ↪118261
cp        0.044421  0.295762 -0.394280 -0.149230  0.119717 -0.
 ↪181053
trestbps -0.114103 -0.046698  0.067616  0.193216 -0.121475  0.
 ↪101389
chol     -0.151040 -0.009940  0.067023  0.053952 -0.004038  0.
 ↪070511
fbs      -0.084189 -0.008567  0.025665  0.005747 -0.059894  0.
 ↪137979
restecg   1.000000  0.044123 -0.070733 -0.058770  0.093045 -0.
 ↪072042
thalach   0.044123  1.000000 -0.378812 -0.344187  0.386784 -0.
 ↪213177
exang    -0.070733 -0.378812  1.000000  0.288223 -0.257748  0.
 ↪115739
oldpeak  -0.058770 -0.344187  0.288223  1.000000 -0.577537  0.
 ↪222682
```

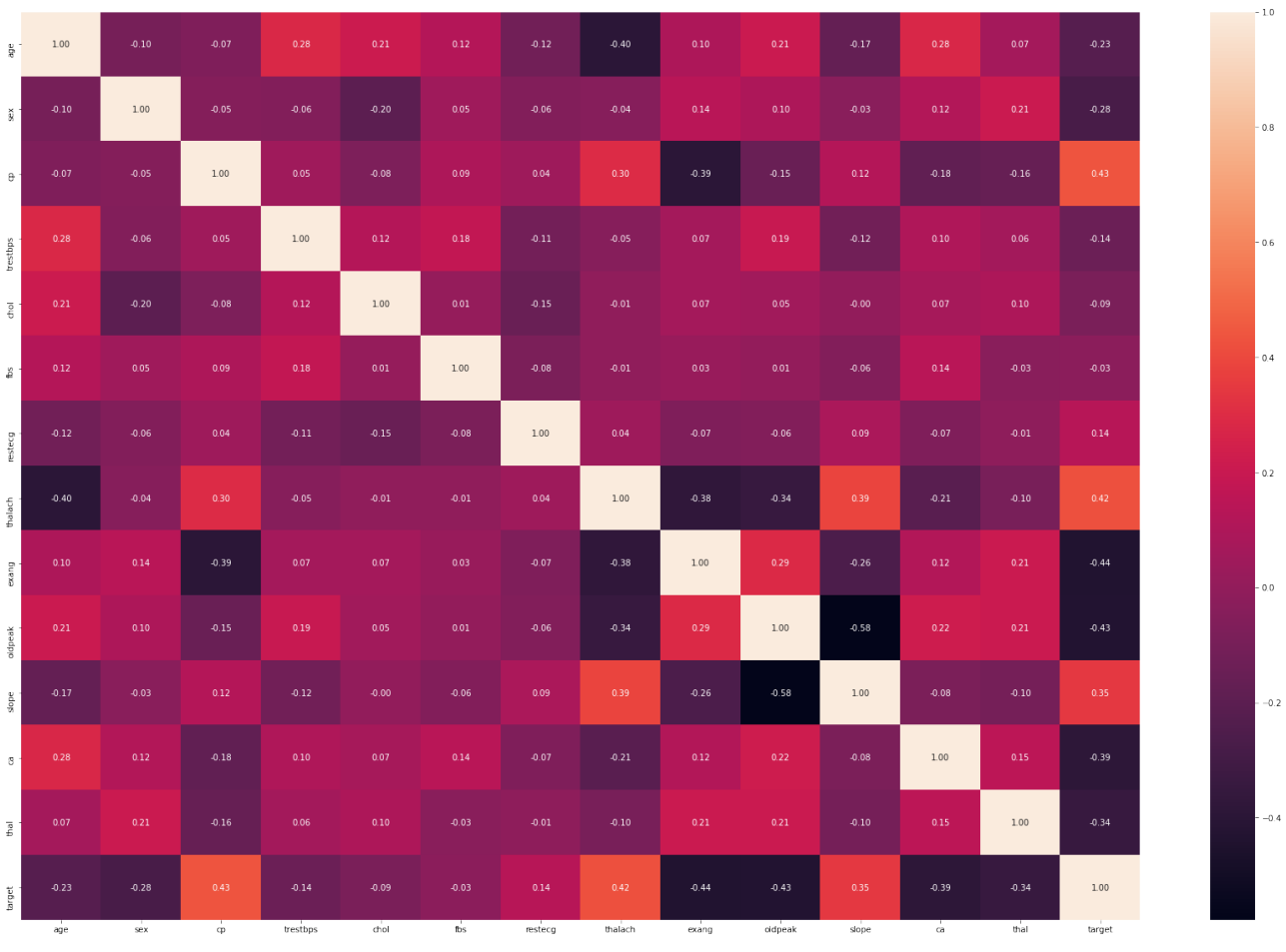
```
slope     0.093045  0.386784 -0.257748 -0.577537  1.000000 -0.
  ↪080155
ca       -0.072042 -0.213177  0.115739  0.222682 -0.080155  1.
  ↪000000
thal     -0.011981 -0.096439  0.206754  0.210244 -0.104764  0.
  ↪151832
target    0.137230  0.421741 -0.436757 -0.430696  0.345877 -0.
  ↪391724
```

```
              thal    target
age       0.068001 -0.225439
sex       0.210041 -0.280937
cp       -0.161736  0.433798
trestbps  0.062210 -0.144931
chol      0.098803 -0.085239
fbs      -0.032019 -0.028046
restecg  -0.011981  0.137230
thalach  -0.096439  0.421741
exang     0.206754 -0.436757
oldpeak   0.210244 -0.430696
slope    -0.104764  0.345877
ca        0.151832 -0.391724
thal      1.000000 -0.344029
target   -0.344029  1.000000
```

[8]:
```
%pylab inline
pylab.rcParams['figure.figsize'] = (30, 20)
```

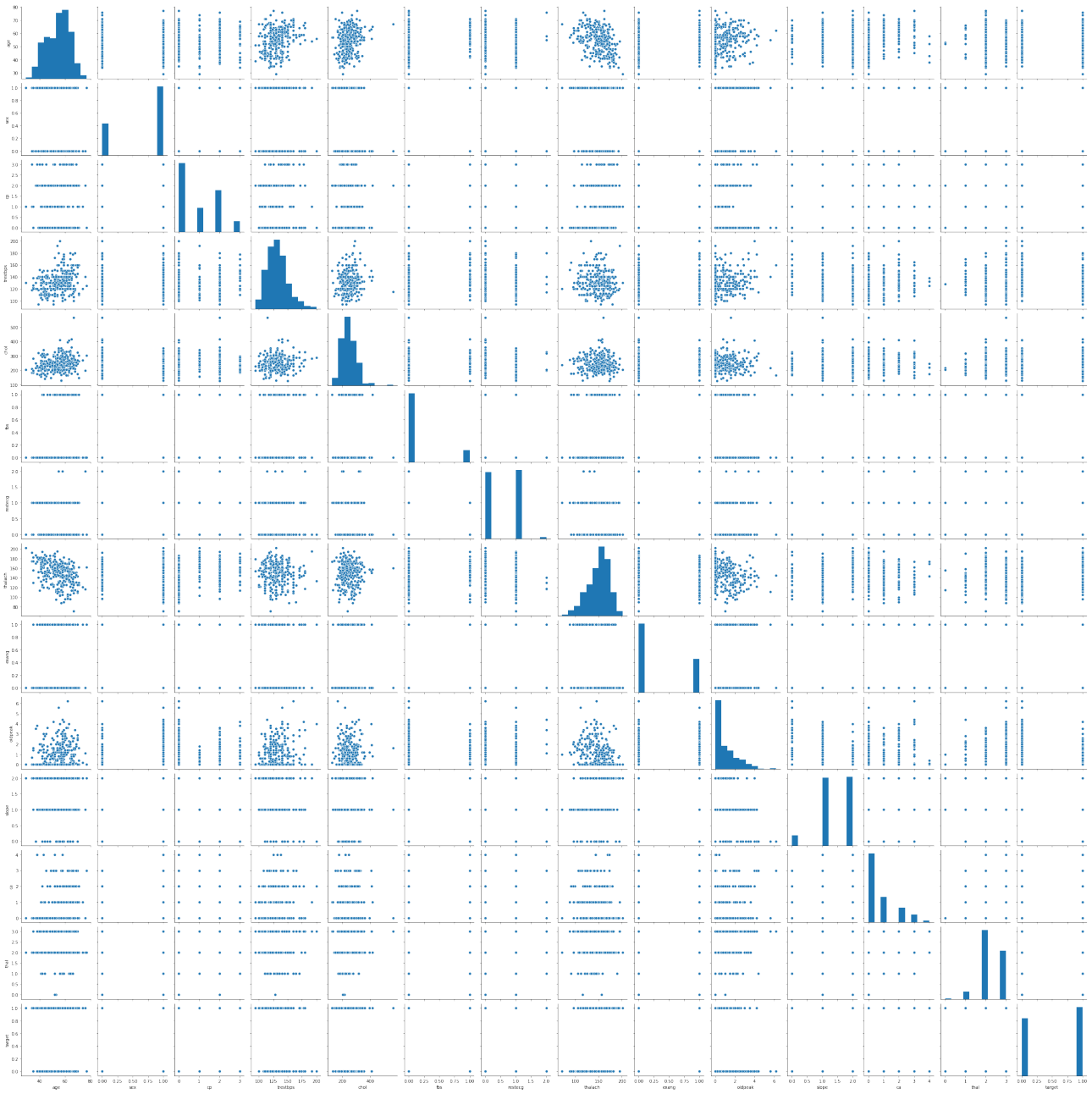
```
Populating the interactive namespace from numpy and matplotlib

/home/dan/anaconda3/lib/python3.7/site-
packages/IPython/core/magics/pylab.py:160: UserWarning: pylab
  ↪import has
clobbered these variables: ['datetime']
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```

[9]:
```
sns.heatmap(data=df.corr(), annot=True, fmt='.2f');
```



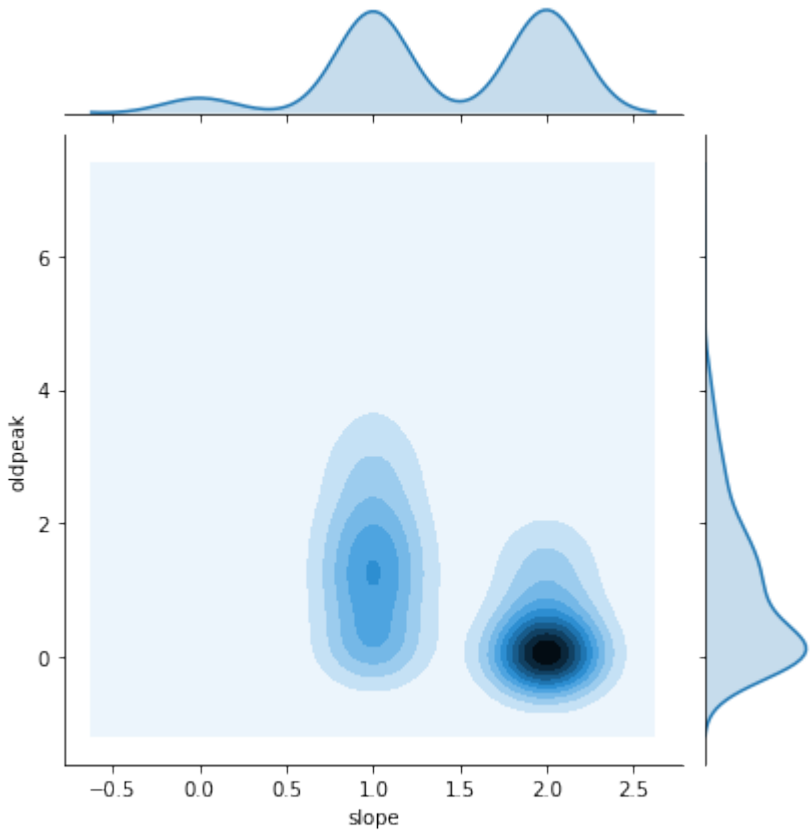
[10]:
```
sns.pairplot(df)
```

[10]: <seaborn.axisgrid.PairGrid at 0x7f759afd1c90>



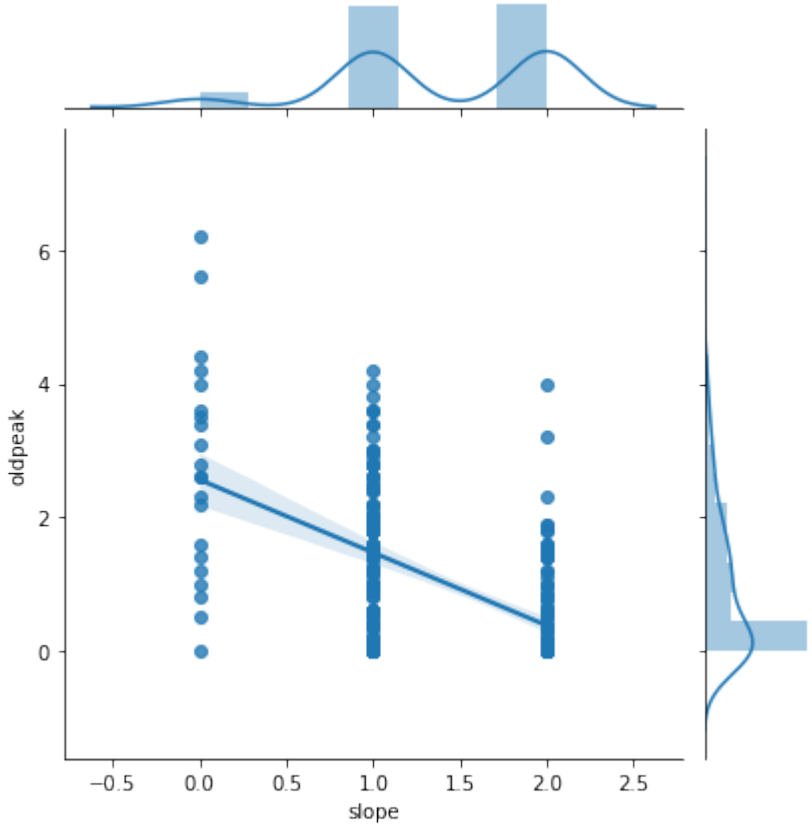
[11]:
```
sns.jointplot(data=df, x='slope', y='oldpeak', kind='kde')
```

[11]: <seaborn.axisgrid.JointGrid at 0x7f758f6183d0>



[12]:
```
sns.jointplot(data=df, x='slope', y='oldpeak', kind='reg')
```

[12]:
```
<seaborn.axisgrid.JointGrid at 0x7f758f470550>
```



2.1. Разделение данных

Разделим данные на целевой столбец и признаки:

[13]:
```
X = df.drop("target", axis=1)
y = df["target"]
```

[14]:
```
print(X.head(), "\n")
print(y.head())
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang ▫
  ↪oldpeak  slope  \
0   63    1   3       145   233    1        0      150      0     ▫
  ↪2.3      0
1   37    1   2       130   250    0        1      187      0     ▫
  ↪3.5      0
2   41    0   1       130   204    0        0      172      0     ▫
  ↪1.4      2
3   56    1   1       120   236    0        1      178      0     ▫
  ↪0.8      2
4   57    0   0       120   354    0        1      163      1     ▫
  ↪0.6      2

   ca  thal
0   0     1
1   0     2
2   0     2
3   0     2
4   0     2

0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

[15]:
```
print(X.shape)
print(y.shape)
```

```
(303, 13)
(303,)
```

Предобработаем данные, чтобы методы работали лучше:

[16]:
```
columns = X.columns
scaler = StandardScaler()
```

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

[16]:

```
                age           sex            cp      trestbps     ▫
  ↪      chol  \
count  3.030000e+02  3.030000e+02  3.030000e+02  3.030000e+02  3.
  ↪030000e+02
mean   5.825923e-17 -1.319077e-17 -5.562565e-17 -7.146832e-16 -9.
  ↪828955e-17
std    1.001654e+00  1.001654e+00  1.001654e+00  1.001654e+00  1.
  ↪001654e+00
min   -2.797624e+00 -1.468418e+00 -9.385146e-01 -2.148802e+00 -2.
  ↪324160e+00
25%   -7.572802e-01 -1.468418e+00 -9.385146e-01 -6.638668e-01 -6.
  ↪814943e-01
50%    6.988599e-02  6.810052e-01  3.203122e-02 -9.273778e-02 -1.
  ↪210553e-01
75%    7.316189e-01  6.810052e-01  1.002577e+00  4.783913e-01  5.
  ↪456738e-01
max    2.496240e+00  6.810052e-01  1.973123e+00  3.905165e+00  6.
  ↪140401e+00

                fbs       restecg       thalach         exang     ▫
  ↪  oldpeak  \
count  3.030000e+02  3.030000e+02  3.030000e+02  3.030000e+02  3.
  ↪030000e+02
mean  -3.664102e-19  2.652810e-16 -5.203025e-16 -5.203025e-16 -3.
  ↪140136e-16
std    1.001654e+00  1.001654e+00  1.001654e+00  1.001654e+00  1.
  ↪001654e+00
min   -4.176345e-01 -1.005832e+00 -3.439267e+00 -6.966305e-01 -8.
  ↪968617e-01
25%   -4.176345e-01 -1.005832e+00 -7.061105e-01 -6.966305e-01 -8.
  ↪968617e-01
50%   -4.176345e-01  8.989622e-01  1.466343e-01 -6.966305e-01 -2.
  ↪067053e-01
75%   -4.176345e-01  8.989622e-01  7.151309e-01  1.435481e+00  4.
  ↪834512e-01
max    2.394438e+00  2.803756e+00  2.289429e+00  1.435481e+00  4.
  ↪451851e+00

               slope            ca          thal
count  3.030000e+02  3.030000e+02  3.030000e+02
mean   1.355718e-16  4.752341e-16  3.484561e-16
std    1.001654e+00  1.001654e+00  1.001654e+00
min   -2.274579e+00 -7.144289e-01 -3.784824e+00
25%   -6.491132e-01 -7.144289e-01 -5.129219e-01
50%   -6.491132e-01 -7.144289e-01 -5.129219e-01
75%    9.763521e-01  2.650822e-01  1.123029e+00
```

```
max    9.763521e-01  3.203615e+00  1.123029e+00
```

[17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size=0.25,
 ↪random_state=346705925)
```

[18]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(227, 13)
(76, 13)
(227,)
(76,)
```

2.2. Обучение моделей

Напишем функцию, которая считает метрики построенной модели:

[19]:
```
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("accuracy:",
          accuracy_score(y_test, model.predict(X_test).round()))
    print("balanced_accuracy:",
          balanced_accuracy_score(y_test, model.predict(X_test).
 ↪round()))
```

2.3. Метод ближайших соседей

Напишем функцию, которая считает метрики построенной модели:

[20]:
```
reg_5 = KNeighborsClassifier(n_neighbors=5)
reg_5.fit(X_train, y_train)
```

[20]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30,
 ↪metric='minkowski',
                     metric_params=None, n_jobs=None,
 ↪n_neighbors=5, p=2,
                     weights='uniform')
```

Проверим метрики построенной модели:

[21]:
```
test_model(reg_5)
```

```
mean_absolute_error: 0.17105263157894737
median_absolute_error: 0.0
accuracy: 0.8289473684210527
balanced_accuracy: 0.8204301075268817
```

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.

2.4. Использование кросс-валидации

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

[22]:
```
scores = cross_val_score(KNeighborsClassifier(n_neighbors=5), X,
 ↪y,
                         cv=KFold(n_splits=10))
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.87096774 0.83870968 0.83870968 0.8        0.8        0.76666667
 0.66666667 0.9        0.76666667 0.53333333]
0.7781720430107527 ± 0.10216643905662431
```

[23]:
```
scores = cross_val_score(KNeighborsClassifier(n_neighbors=5), X,
 ↪y,
                         cv=RepeatedKFold(n_splits=5,
 ↪n_repeats=2))
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.78688525 0.7704918  0.81967213 0.76666667 0.85       0.75409836
 0.80327869 0.81967213 0.85       0.85      ]
0.8070765027322404 ± 0.03470557366104938
```

[24]:
```
scores = cross_val_score(KNeighborsClassifier(n_neighbors=5), X,
 ↪y,
                         cv=ShuffleSplit(n_splits=10))
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.70967742 0.83870968 0.80645161 0.87096774 0.77419355 0.83870968
 0.83870968 0.77419355 0.83870968 0.87096774]
0.8161290322580644 ± 0.04795506047522099
```

2.5. Подбор гиперпараметра K

Введем список настраиваемых параметров:

[25]:
```
n_range = np.array(range(1, 50, 1))
tuned_parameters = [{'n_neighbors': n_range}]
n_range
```

[25]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
       15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
       32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
       49])
```

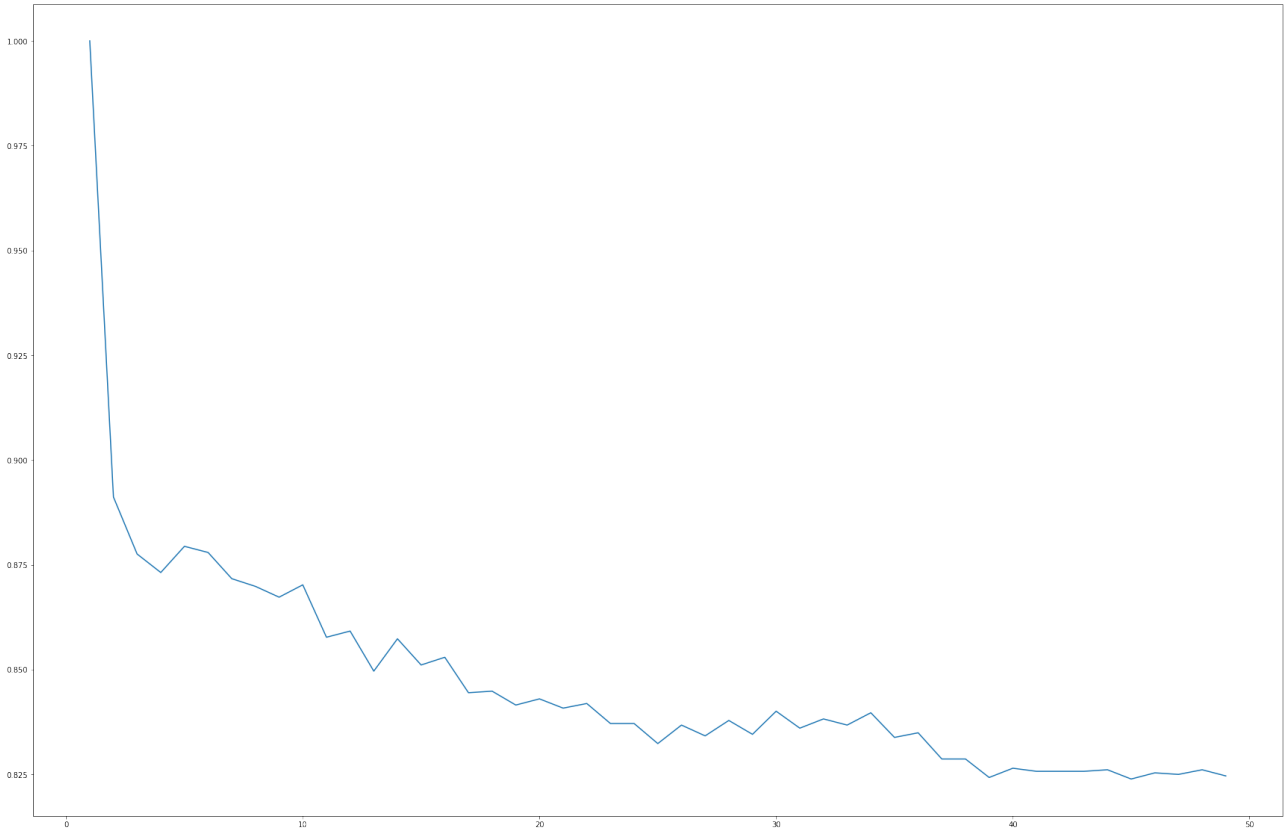
Запустим подбор параметра:

[26]:
```
gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10),
                  return_train_score=True)
gs.fit(X, y)
gs.best_params_
```

[26]:
```
{'n_neighbors': 32}
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

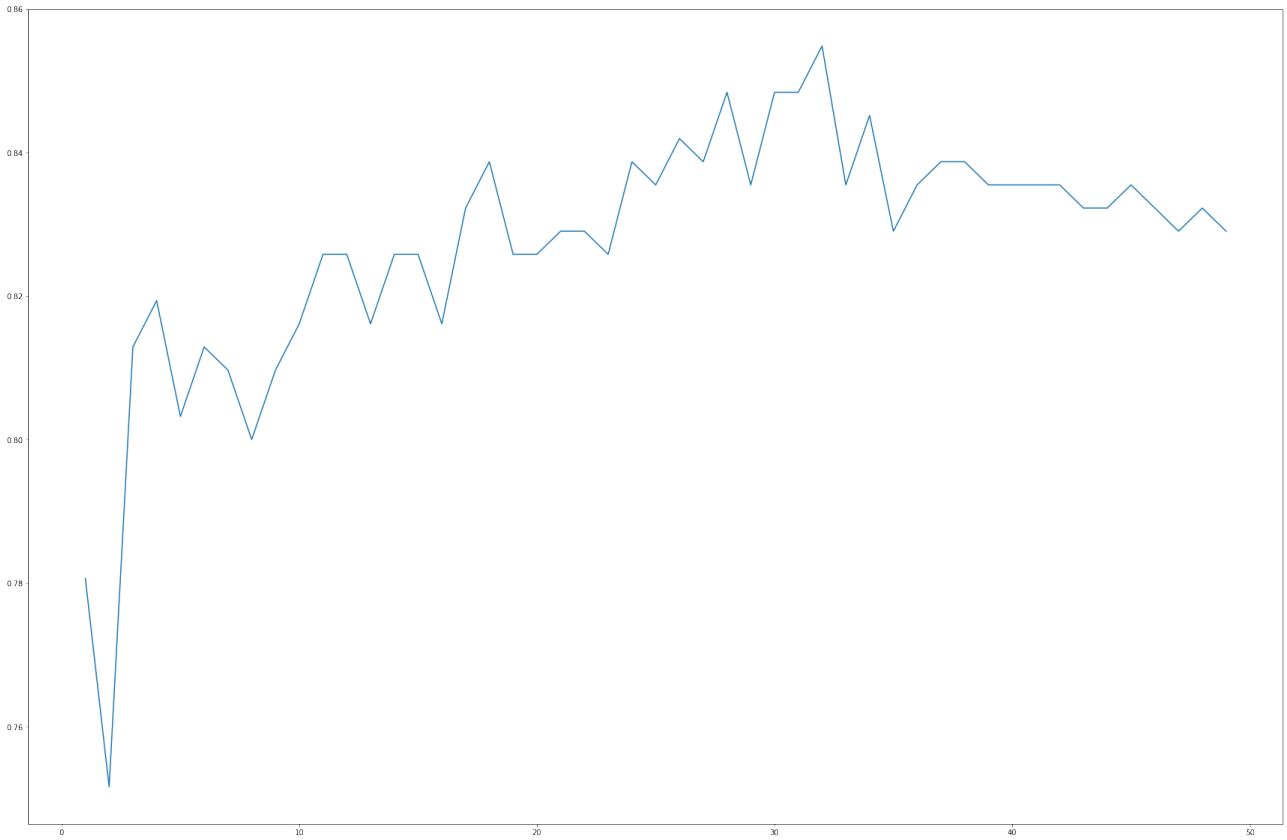
[27]:
```
plt.plot(n_range, gs.cv_results_["mean_train_score"]);
```



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:

[28]:
```
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:

[29]:
```
reg = KNeighborsClassifier(**gs.best_params_)
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 0.13157894736842105
median_absolute_error: 0.0
accuracy: 0.868421052631579
balanced_accuracy: 0.853763440860215
```

В целом получили примерно тот же результат. Очевидно, что проблема в том, что данный метод не может дать хороший результат для данной выборки.

Построим кривую обучения [**?**]:

[30]:
```
def plot_learning_curve(estimator, title, X, y, ylim=None,
 ↪cv=None,
                        n_jobs=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=n_jobs,
↪train_sizes=train_sizes, scoring='accuracy')
    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
```

[31]:
```
plot_learning_curve(reg, str(gs.best_params_), X, y,
                    cv=ShuffleSplit(n_splits=10));
```

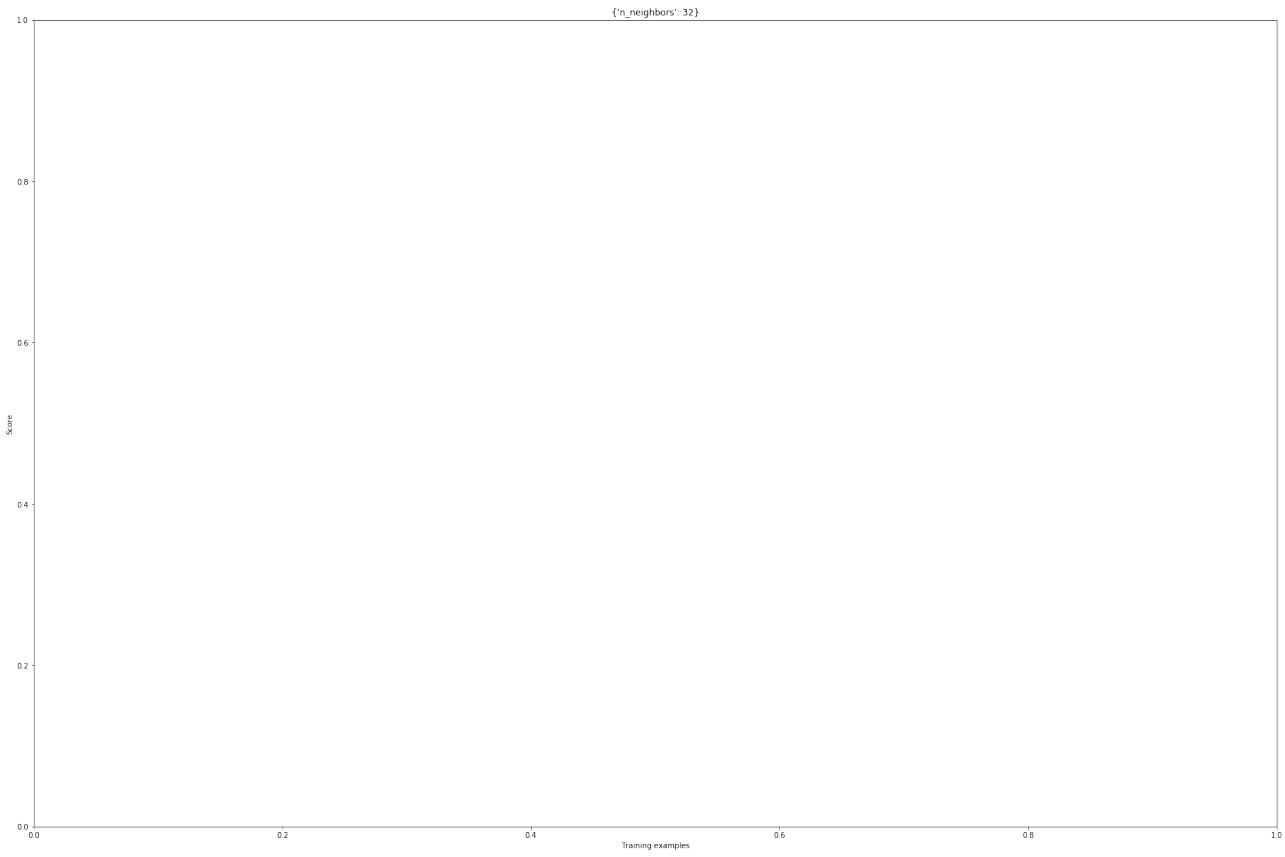
```
      
↪---------------------------------------------------------------------------

      ValueError                                Traceback (most
↪recent call last)

      <ipython-input-31-67c047eb9523> in <module>
        1 plot_learning_curve(reg, str(gs.best_params_), X, y,
  ----> 2                     cv=ShuffleSplit(n_splits=10));


      <ipython-input-30-77f0e588a046> in
↪plot_learning_curve(estimator, title, X, y, ylim, cv, n_jobs)
       10     plt.ylabel("Score")
       11     train_sizes, train_scores, test_scores =
↪learning_curve(
```

```
   ---> 12         estimator, X, y, cv=cv, n_jobs=n_jobs,▫
↪train_sizes=train_sizes, scoring='accuracy')
        13     train_scores_mean = np.mean(train_scores, axis=1)
        14     train_scores_std = np.std(train_scores, axis=1)


      ~/anaconda3/lib/python3.7/site-packages/sklearn/
↪model_selection/_validation.py in learning_curve(estimator, X,▫
↪y, groups, train_sizes, cv, scoring,▫
↪exploit_incremental_learning, n_jobs, pre_dispatch, verbose,▫
↪shuffle, random_state, error_score, return_times)
     1254             parameters=None, fit_params=None,▫
↪return_train_score=True,
     1255             error_score=error_score,▫
↪return_times=return_times)
   -> 1256             for train, test in train_test_proportions)
     1257         out = np.array(out)
     1258         n_cv_folds = out.shape[0] // n_unique_ticks


      ~/anaconda3/lib/python3.7/site-packages/joblib/parallel.py▫
↪in __call__(self, iterable)
     1002             # remaining jobs.
     1003             self._iterating = False
   -> 1004             if self.dispatch_one_batch(iterator):
     1005                 self._iterating = self.
↪_original_iterator is not None
     1006


      ~/anaconda3/lib/python3.7/site-packages/joblib/parallel.py▫
↪in dispatch_one_batch(self, iterator)
      833                 return False
      834             else:
   --> 835                 self._dispatch(tasks)
      836                 return True
      837


      ~/anaconda3/lib/python3.7/site-packages/joblib/parallel.py▫
↪in _dispatch(self, batch)
      752         with self._lock:
      753             job_idx = len(self._jobs)
   --> 754             job = self._backend.apply_async(batch,▫
↪callback=cb)
      755             # A job can complete so quickly than its▫
↪callback is
      756             # called before we get here, causing self.
↪_jobs to
```

```
      ~/anaconda3/lib/python3.7/site-packages/joblib/
↪_parallel_backends.py in apply_async(self, func, callback)
      207     def apply_async(self, func, callback=None):
      208         """Schedule a func to be run"""
  --> 209         result = ImmediateResult(func)
      210         if callback:
      211             callback(result)


      ~/anaconda3/lib/python3.7/site-packages/joblib/
↪_parallel_backends.py in __init__(self, batch)
      588         # Don't delay the application, to avoid▫
↪keeping the input
      589         # arguments in memory
  --> 590         self.results = batch()
      591
      592     def get(self):


      ~/anaconda3/lib/python3.7/site-packages/joblib/parallel.py▫
↪in __call__(self)
      254         with parallel_backend(self._backend,▫
↪n_jobs=self._n_jobs):
      255             return [func(*args, **kwargs)
  --> 256                     for func, args, kwargs in self.
↪items]
      257
      258     def __len__(self):


      ~/anaconda3/lib/python3.7/site-packages/joblib/parallel.py▫
↪in <listcomp>(.0)
      254         with parallel_backend(self._backend,▫
↪n_jobs=self._n_jobs):
      255             return [func(*args, **kwargs)
  --> 256                     for func, args, kwargs in self.
↪items]
      257
      258     def __len__(self):


      ~/anaconda3/lib/python3.7/site-packages/sklearn/
↪model_selection/_validation.py in _fit_and_score(estimator, X,▫
↪y, scorer, train, test, verbose, parameters, fit_params,▫
↪return_train_score, return_parameters, return_n_test_samples,▫
↪return_times, return_estimator, error_score)
      542     else:
```

```
      543         fit_time = time.time() - start_time
   --> 544         test_scores = _score(estimator, X_test,▫
↪y_test, scorer)
      545         score_time = time.time() - start_time -▫
↪fit_time
      546         if return_train_score:


      ~/anaconda3/lib/python3.7/site-packages/sklearn/
↪model_selection/_validation.py in _score(estimator, X_test,▫
↪y_test, scorer)
      589         scores = scorer(estimator, X_test)
      590     else:
   --> 591         scores = scorer(estimator, X_test, y_test)
      592
      593     error_msg = ("scoring must return a number, got %s▫
↪(%s) "


      ~/anaconda3/lib/python3.7/site-packages/sklearn/metrics/
↪_scorer.py in __call__(self, estimator, X, y_true,▫
↪sample_weight)
      167                                stacklevel=2)
      168         return self._score(partial(_cached_call,▫
↪None), estimator, X, y_true,
   --> 169                            sample_weight=sample_weight)
      170
      171     def _factory_args(self):


      ~/anaconda3/lib/python3.7/site-packages/sklearn/metrics/
↪_scorer.py in _score(self, method_caller, estimator, X, y_true,▫
↪sample_weight)
      203         """
      204
   --> 205         y_pred = method_caller(estimator, "predict", X)
      206         if sample_weight is not None:
      207             return self._sign * self.
↪_score_func(y_true, y_pred,


      ~/anaconda3/lib/python3.7/site-packages/sklearn/metrics/
↪_scorer.py in _cached_call(cache, estimator, method, *args,▫
↪**kwargs)
       50     """Call estimator with method and args and kwargs.
↪"""
       51     if cache is None:
   ---> 52         return getattr(estimator, method)(*args,▫
↪**kwargs)
```

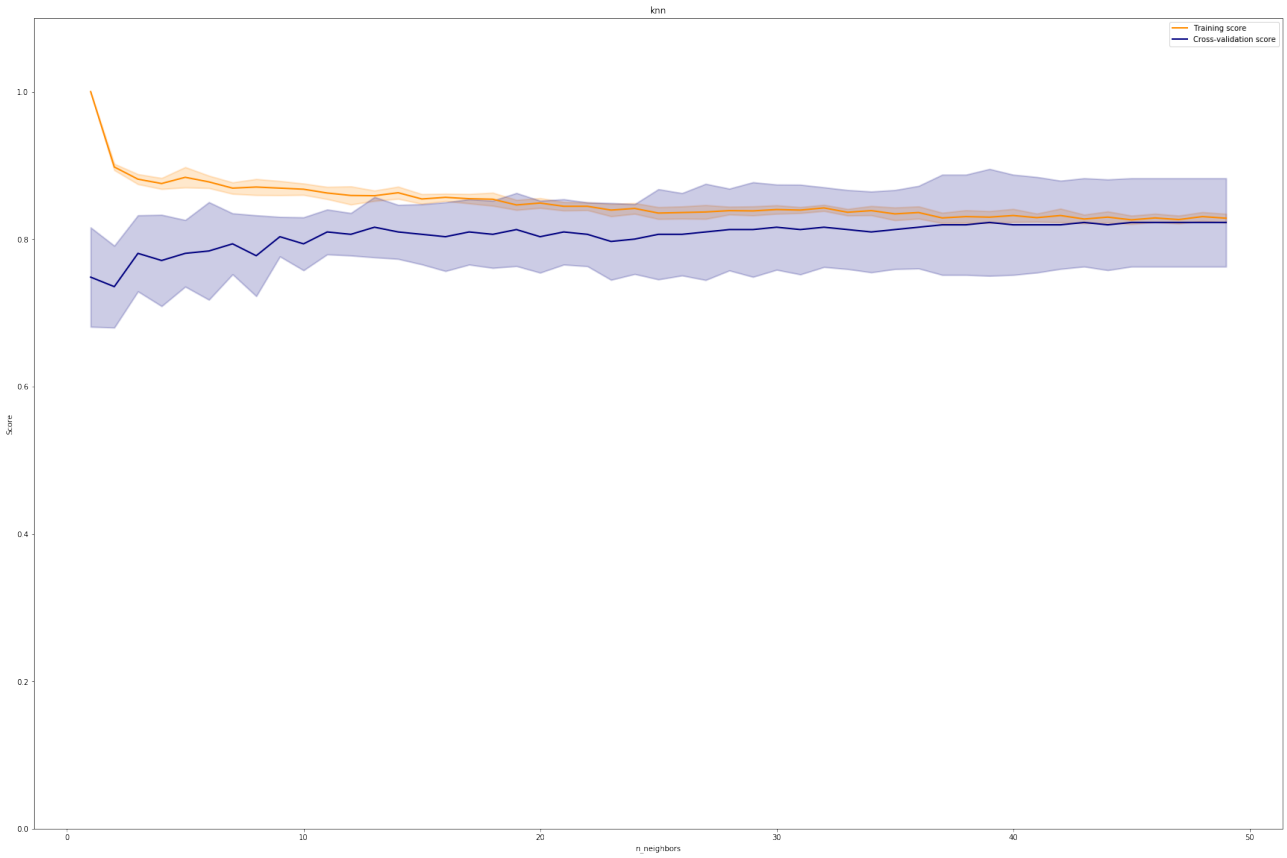
```
        53
        54     try:


      ~/anaconda3/lib/python3.7/site-packages/sklearn/neighbors/
↪_classification.py in predict(self, X)
      171         X = check_array(X, accept_sparse='csr')
      172
  --> 173         neigh_dist, neigh_ind = self.kneighbors(X)
      174         classes_ = self.classes_
      175         _y = self._y


      ~/anaconda3/lib/python3.7/site-packages/sklearn/neighbors/
↪_base.py in kneighbors(self, X, n_neighbors, return_distance)
      615                     "Expected n_neighbors <= n_samples, "
      616                     " but n_samples = %d, n_neighbors =
↪%d" %
  --> 617                     (n_samples_fit, n_neighbors)
      618                 )
      619


      ValueError: Expected n_neighbors <= n_samples,  but
↪n_samples = 27, n_neighbors = 32
```



Построим кривую валидации:

[32]:
```
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=4)
    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
```

[33]:
```
plot_validation_curve(KNeighborsClassifier(), "knn", X, y,
                      param_name="n_neighbors",
↪param_range=n_range,
                      cv=ShuffleSplit(n_splits=10));
```



2.5.1. Модель бинарной классификации — LogisticRegression

Попробуем метод LogisticRegression с гиперпараметром $\alpha = 1$:

[34]:
```
cl1 = LogisticRegression()
cl1.fit(X_train, y_train)
```

[34]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False,
↪fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None,
↪max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001,
↪verbose=0,
                   warm_start=False)
```

[35]:
```
test_model(cl1)
```

```
mean_absolute_error: 0.15789473684210525
median_absolute_error: 0.0
accuracy: 0.8421052631578947
balanced_accuracy: 0.8315412186379928
```

Видно, что данный метод без настройки гиперпараметров несколько хуже, чем метод ближайших соседей.

2.5.2. SVC

Попробуем метод SVC:

[36]:
```
linear_1 = SVC(C=1.0, gamma='auto')
linear_1.fit(X_train, y_train)
```

[36]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None,
↪coef0=0.0,
   decision_function_shape='ovr', degree=3, gamma='auto',
↪kernel='rbf',
   max_iter=-1, probability=False, random_state=None,
↪shrinking=True,
   tol=0.001, verbose=False)
```

[37]:
```
test_model(linear_1)
```

```
mean_absolute_error: 0.19736842105263158
median_absolute_error: 0.0
accuracy: 0.8026315789473685
balanced_accuracy: 0.7982078853046595
```

Внезапно LinearSVC показал результаты хуже по средней абсолютной ошибке и коэффициенте детерминации.

2.5.3. Дерево решений

Попробуем дерево решений с неограниченной глубиной дерева:

[38]:
```
dt_none = DecisionTreeRegressor(max_depth=None)
dt_none.fit(X_train, y_train)
```

[38]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse',
↪max_depth=None,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0,
↪min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0,
↪presort='deprecated',
                      random_state=None, splitter='best')
```

Проверим метрики построенной модели:

[39]:
```
test_model(dt_none)
```

```
mean_absolute_error: 0.2894736842105263
median_absolute_error: 0.0
accuracy: 0.7105263157894737
balanced_accuracy: 0.7103942652329749
```

Дерево решений показало прямо-таки очень хороший результат по сравнению с рассмотренными раньше методами. Оценим структуру получившегося дерева решений:

[40]:
```
def stat_tree(estimator):
    n_nodes = estimator.tree_.node_count
    children_left = estimator.tree_.children_left
    children_right = estimator.tree_.children_right

    node_depth = np.zeros(shape=n_nodes, dtype=np.int64)
    is_leaves = np.zeros(shape=n_nodes, dtype=bool)
    stack = [(0, -1)]  # seed is the root node id and its parent
    depth
    while len(stack) > 0:
        node_id, parent_depth = stack.pop()
        node_depth[node_id] = parent_depth + 1

        # If we have a test node
        if (children_left[node_id] != children_right[node_id]):
            stack.append((children_left[node_id], parent_depth +
    1))
            stack.append((children_right[node_id], parent_depth +
    1))
        else:
            is_leaves[node_id] = True

    print("Всего узлов:", n_nodes)
    print("Листовых узлов:", sum(is_leaves))
    print("Глубина дерева:", max(node_depth))
    print("Минимальная глубина листьев дерева:",
    min(node_depth[is_leaves]))
    print("Средняя глубина листьев дерева:",
    node_depth[is_leaves].mean())
```

[41]:
```
stat_tree(dt_none)
```

```
Всего узлов: 71
Листовых узлов: 36
Глубина дерева: 9
Минимальная глубина листьев дерева: 3
Средняя глубина листьев дерева: 6.083333333333333
```

2.6. Подбор гиперпараметра K

2.6.1. Модель бинарной классификации — LogisticRegression

Введем список настраиваемых параметров:

[42]:
```
param_range = np.arange(0.001, 2.01, 0.1)
tuned_parameters = [{'C': param_range}]
tuned_parameters
```

[42]:

```
[{'C': array([1.000e-03, 1.010e-01, 2.010e-01, 3.010e-01, 4.
 ↪010e-01, 5.010e-01,
        6.010e-01, 7.010e-01, 8.010e-01, 9.010e-01, 1.001e+00, 1.
 ↪101e+00,
        1.201e+00, 1.301e+00, 1.401e+00, 1.501e+00, 1.601e+00, 1.
 ↪701e+00,
        1.801e+00, 1.901e+00, 2.001e+00])}]
```

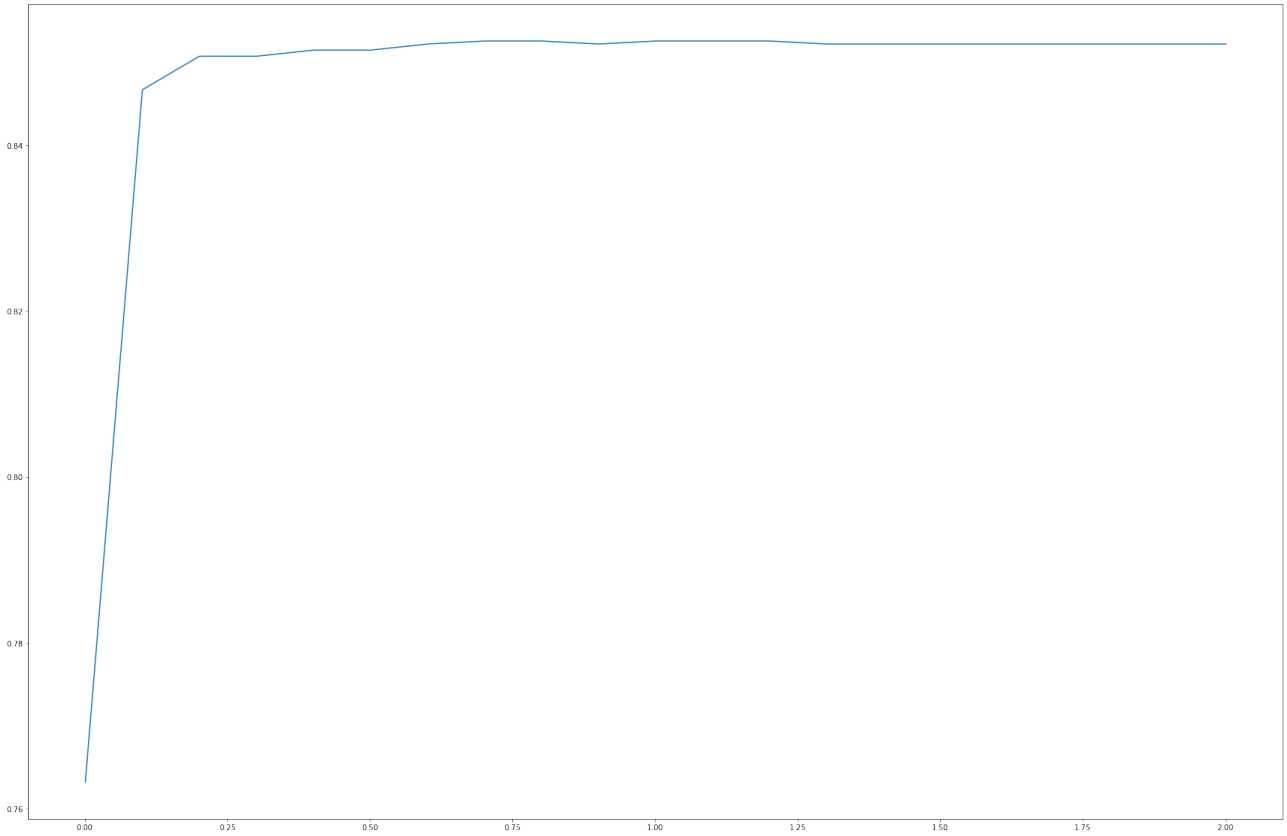
Запустим подбор параметра:

[43]:
```
gs = GridSearchCV(LogisticRegression(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10),
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

[43]:
```
LogisticRegression(C=0.101, class_weight=None, dual=False,
 ↪fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None,
 ↪max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001,
 ↪verbose=0,
                   warm_start=False)
```

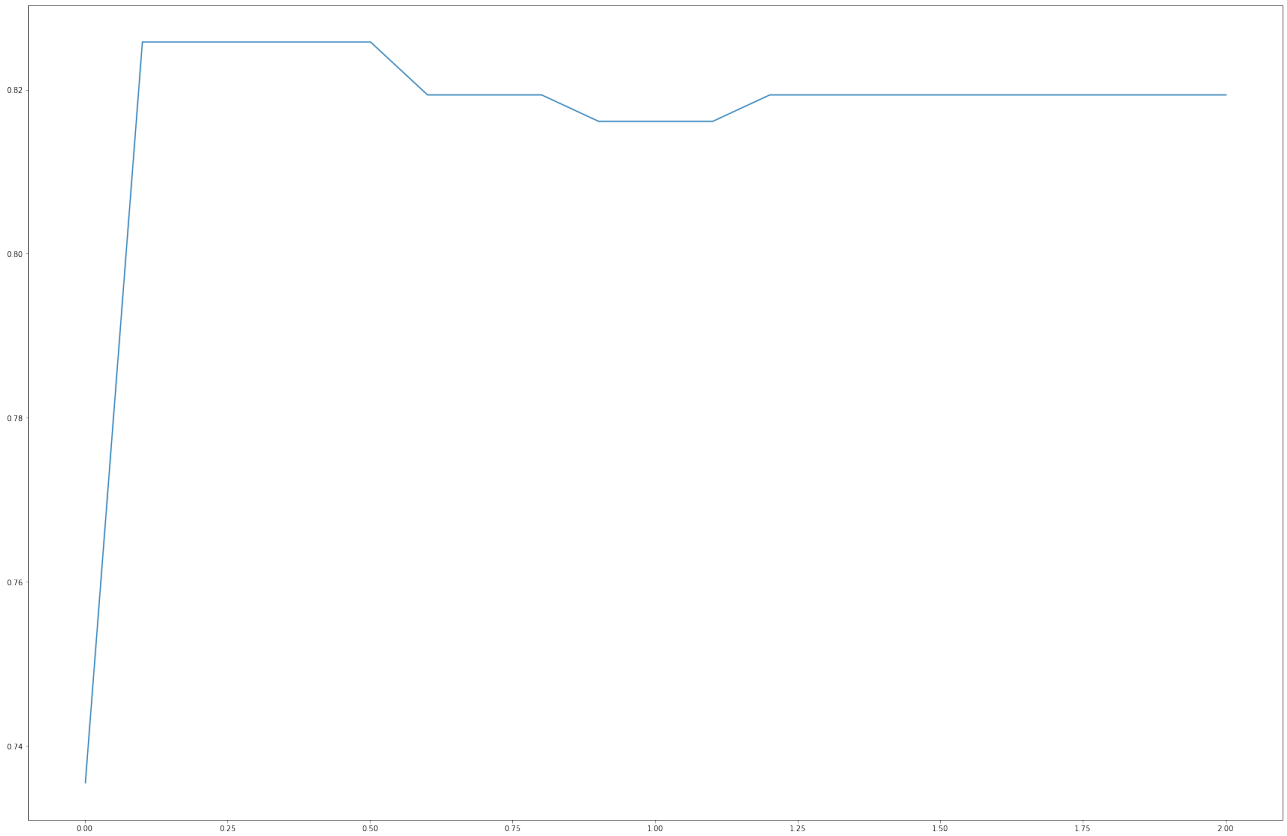
Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

[44]:
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



На тестовом наборе данных картина ровно та же:

[45]:
```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



2.6.2. SVM

Введем список настраиваемых параметров:

[46]:
```
param_range = np.arange(0.1, 1.01, 0.1)
tuned_parameters = [{'gamma': param_range}]
tuned_parameters
```

[46]:
```
[{'gamma': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.
↪])}]
```

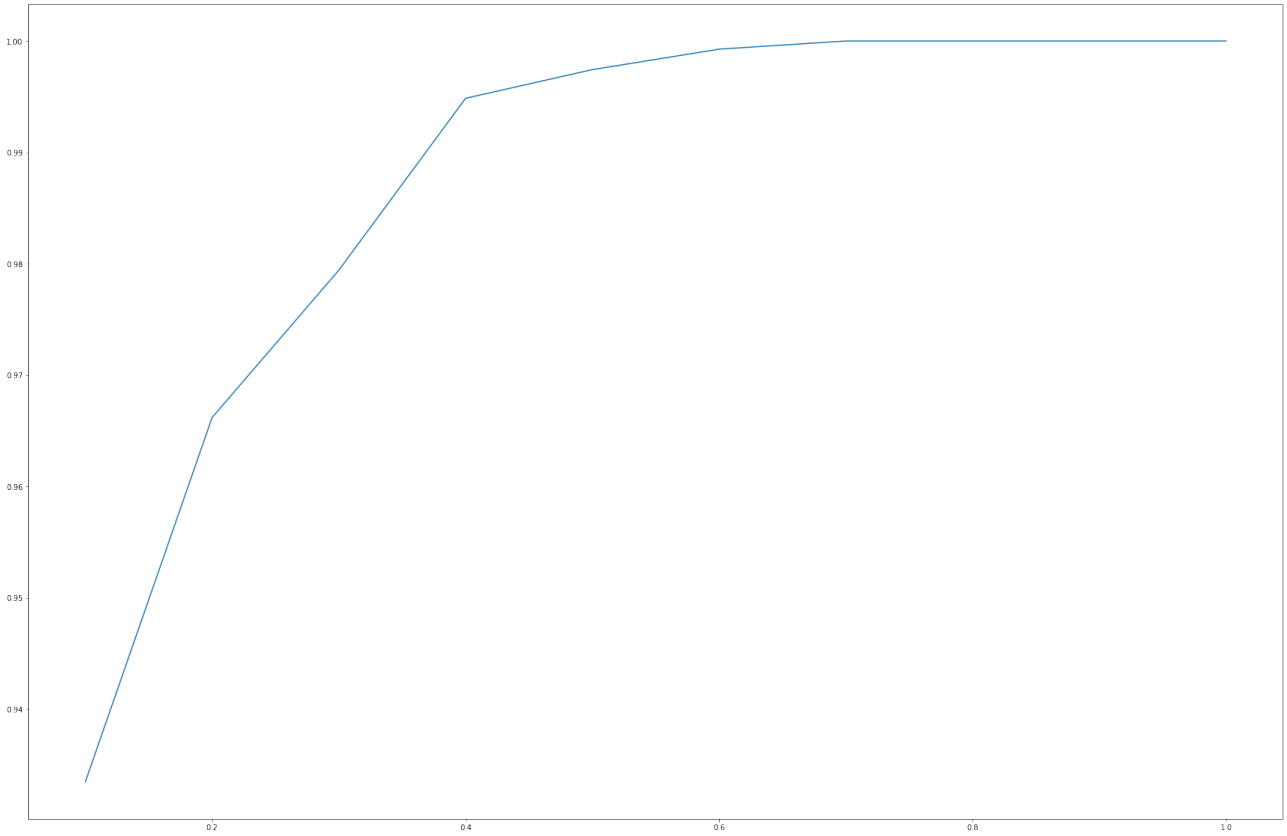
Запустим подбор параметра:

[47]:
```
gs = GridSearchCV(SVC(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10),
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

[47]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None,
↪coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.1,
↪kernel='rbf',
    max_iter=-1, probability=False, random_state=None,
↪shrinking=True,
    tol=0.001, verbose=False)
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

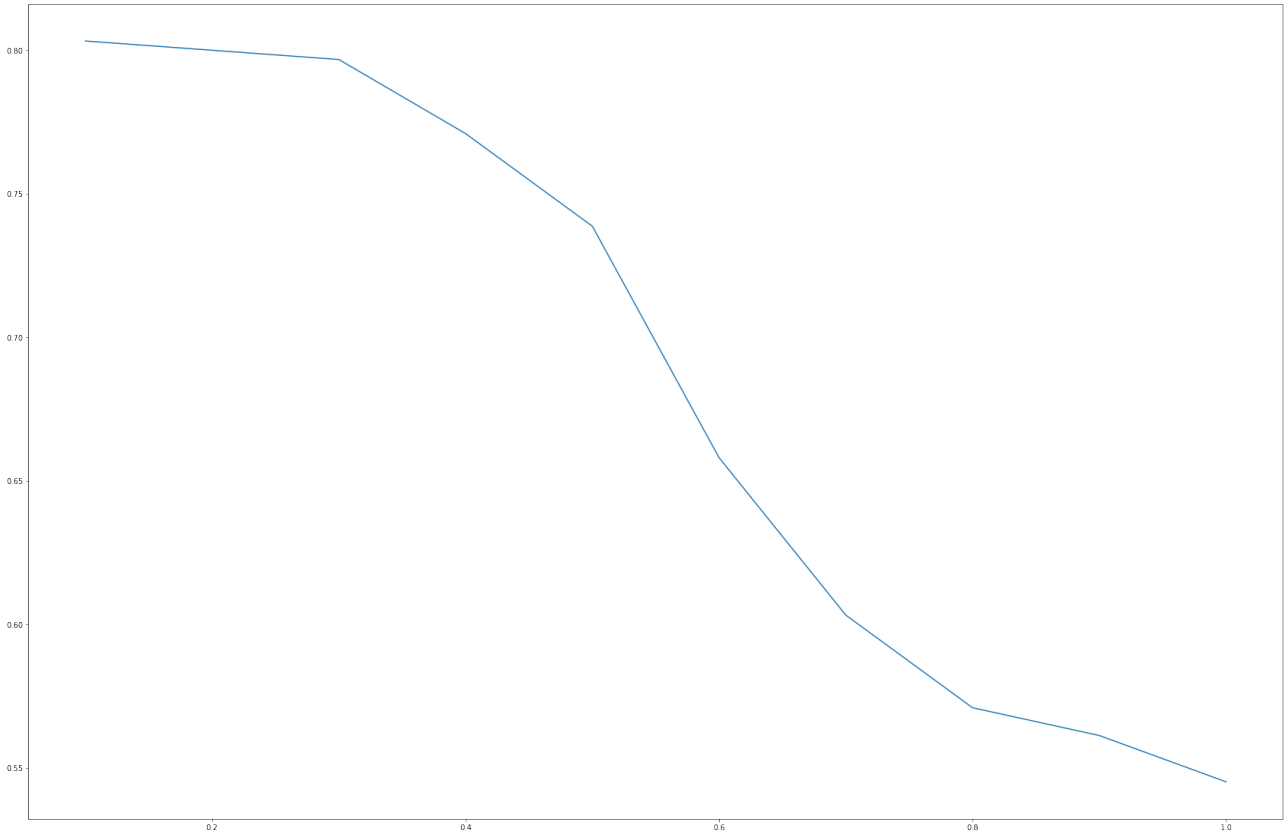
[48]:
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



Видно, что метод NuSVR справляется лучше, но не глобально. При этом также видно, что разработчики библиотеки scikit-learn провели хорошую работу: получившееся оптимальное значение $\nu = 0{,}5$ является стандартным для данного алгоритма [**?**].

На тестовом наборе данных картина ровно та же:

[49]:
```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



Так как параметры подобраны те же, то и обучение модели заново производить не будем.

2.6.3. Дерево решений

Введем список настраиваемых параметров:

[50]:
```
param_range = np.arange(1, 51, 2)
tuned_parameters = [{'max_depth': param_range}]
tuned_parameters
```

[50]:
```
[{'max_depth': 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:

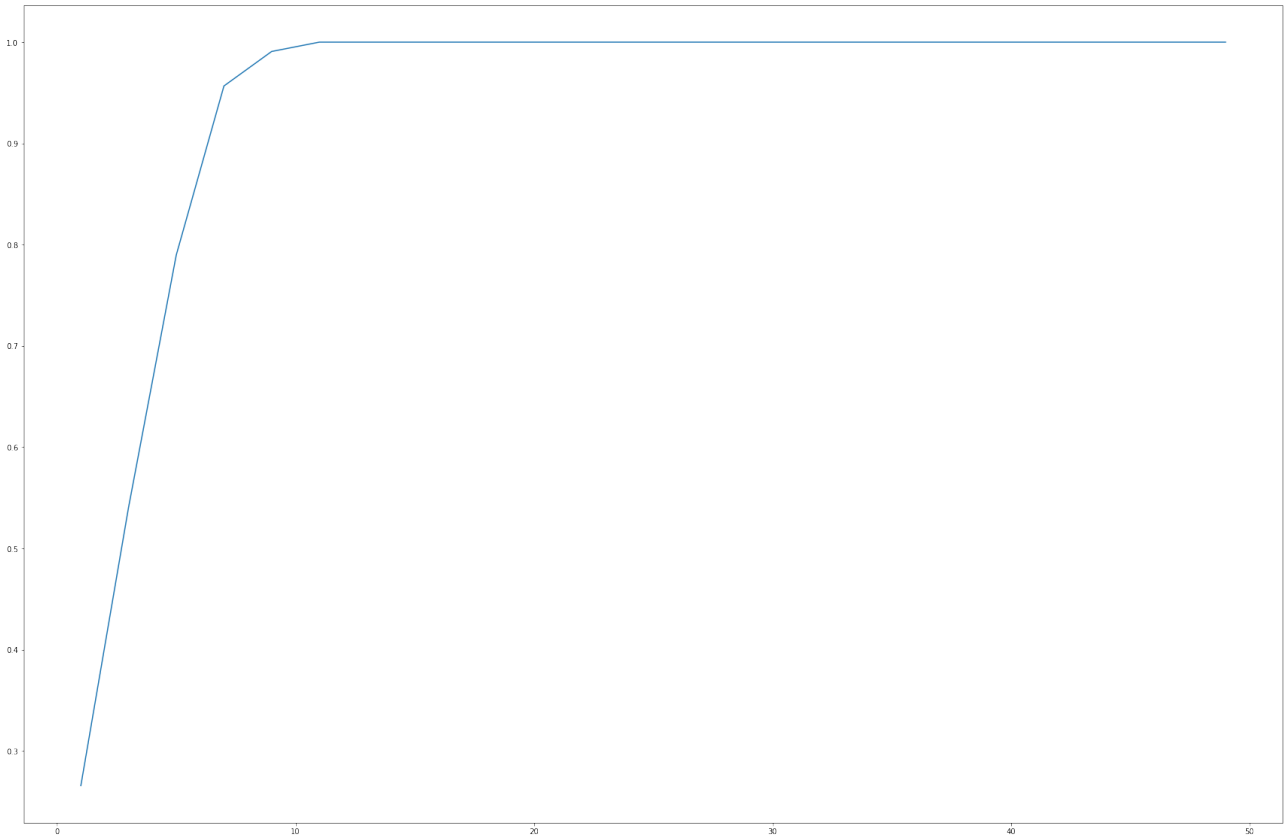
[51]:
```
gs = GridSearchCV(DecisionTreeRegressor(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

[51]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=3,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0,
                      min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
```

```
                       min_weight_fraction_leaf=0.0,
    ↪presort='deprecated',
                       random_state=None, splitter='best')
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

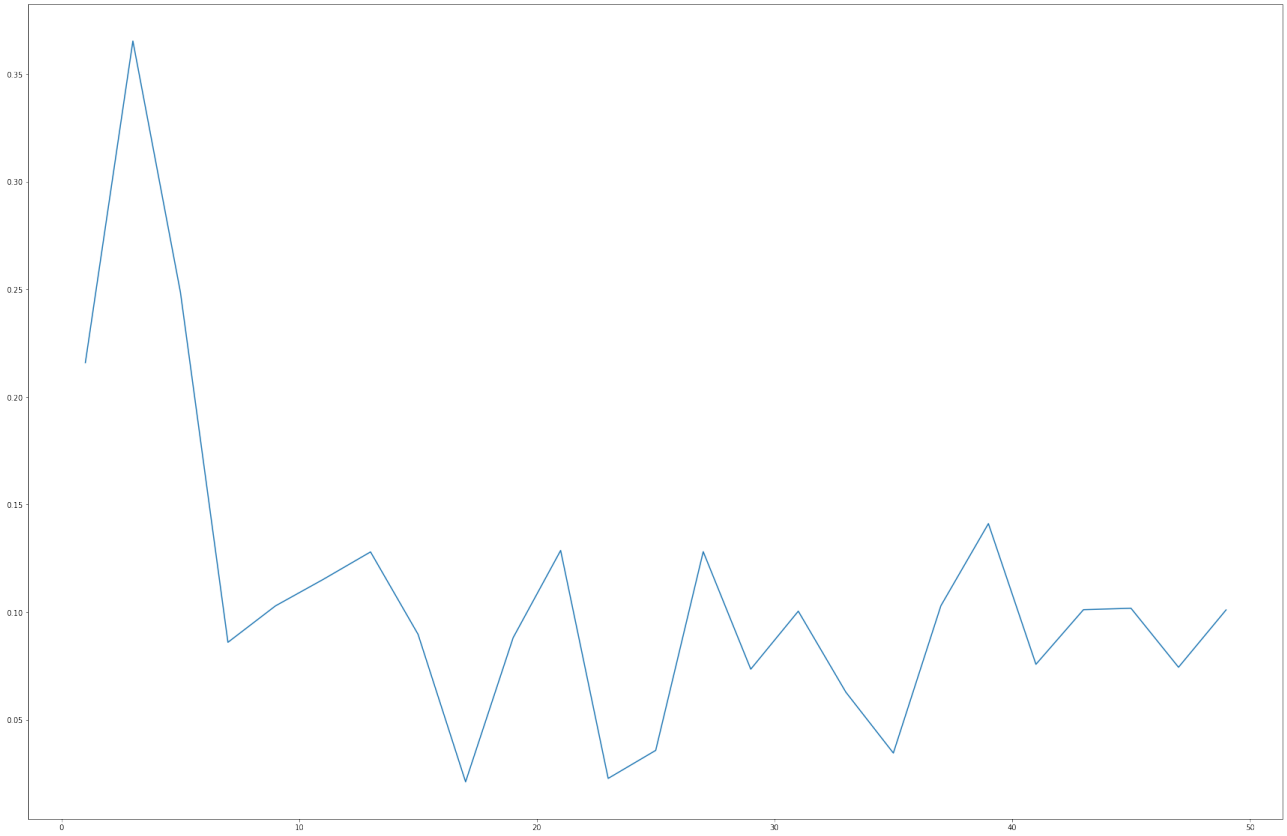
[52]:
```
plt.plot(param_range, gs.cv_results_["mean_train_score"]);
```



Видно, что на тестовой выборке модель легко переобучается.

На тестовом наборе данных картина интереснее:

[53]:
```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



Проведем дополнительное исследование в районе пика.

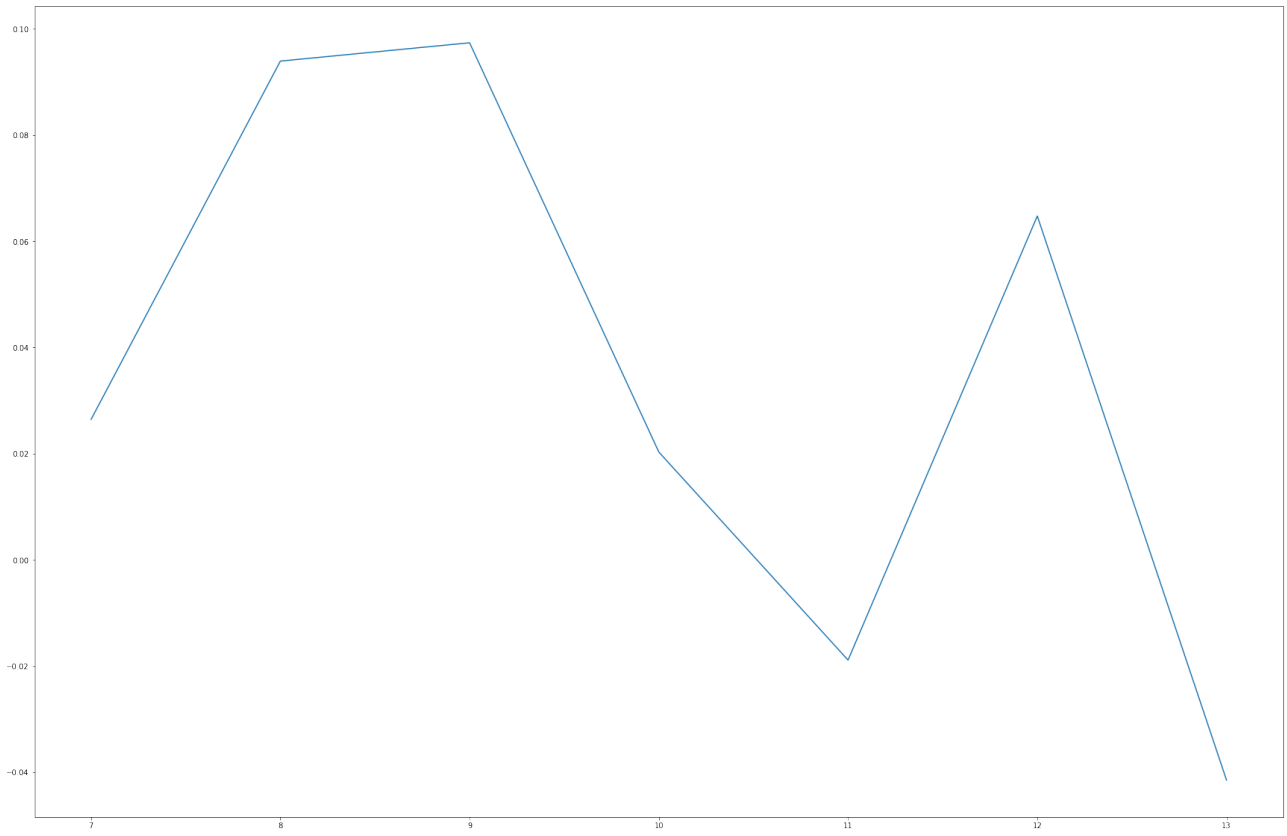
[54]:
```
param_range = np.arange(7, 14, 1)
tuned_parameters = [{'max_depth': param_range}]
tuned_parameters
```

[54]:
```
[{'max_depth': array([ 7,  8,  9, 10, 11, 12, 13])}]
```

[55]:
```
gs = GridSearchCV(DecisionTreeRegressor(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_estimator_
```

[55]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=9,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0,
 ↪min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0,
 ↪presort='deprecated',
                      random_state=None, splitter='best')
```

[56]:
```
plt.plot(param_range, gs.cv_results_["mean_test_score"]);
```



Получили, что глубину дерева необходимо ограничить 10 уровнями. Проверим этот результат.

[57]:
```
reg = gs.best_estimator_
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 0.2894736842105263
median_absolute_error: 0.0
accuracy: 0.7105263157894737
balanced_accuracy: 0.7154121863799283
```

[58]:
```
X_train
```

[58]:
```
array([[-1.1432911 ,  0.68100522, -0.93851463, …,  0.97635214,
         0.26508221, -0.51292188],
       [ 0.29046364,  0.68100522,  1.00257707, …,  0.97635214,
         0.26508221,  1.12302895],
       [ 0.84190778, -1.46841752, -0.93851463, …, -2.27457861,
         2.22410436,  1.12302895],
       …,
       [ 0.40075247,  0.68100522, -0.93851463, …,  0.97635214,
        -0.71442887,  1.12302895],
       [-0.26098049,  0.68100522,  1.97312292, …, -0.64911323,
        -0.71442887, -2.14887271],
       [-0.26098049,  0.68100522, -0.93851463, …,  0.97635214,
```

```
       0.26508221, -0.51292188]])
```

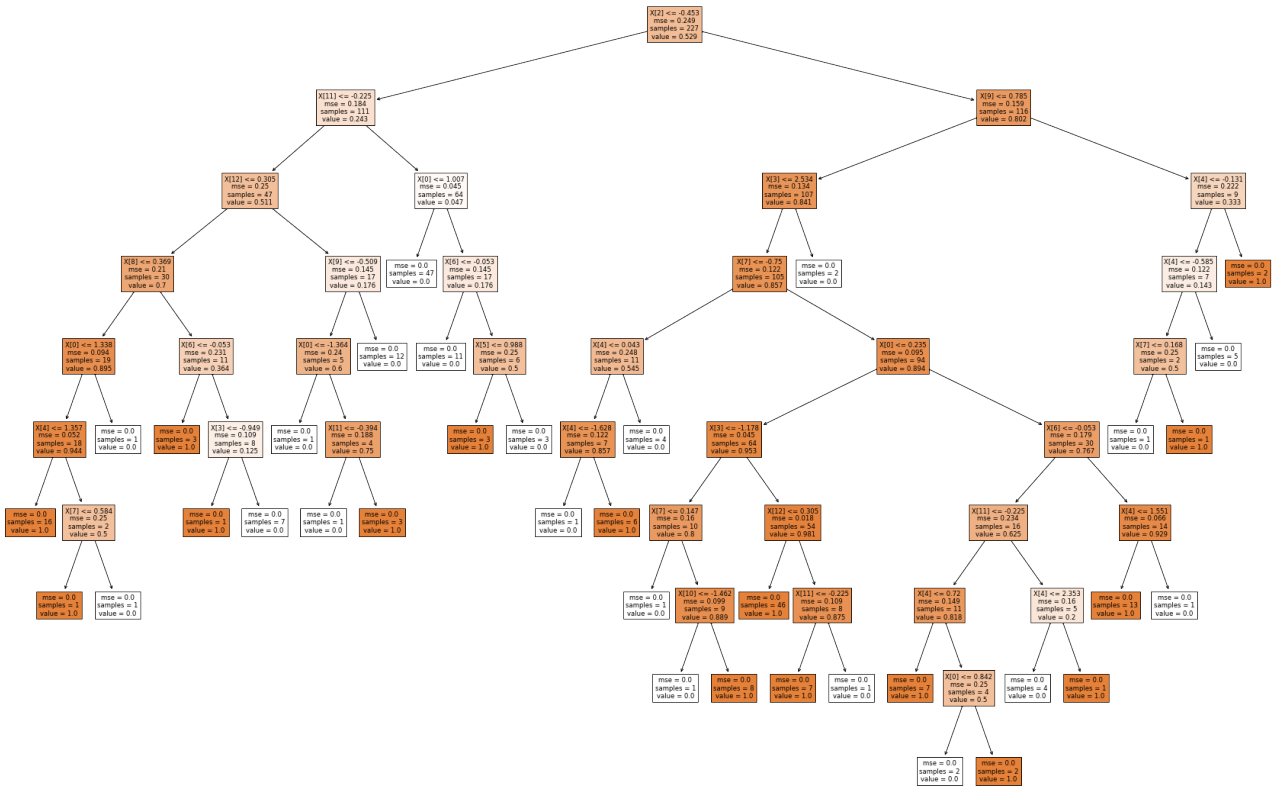
Вновь посмотрим статистику получившегося дерева решений.

[59]:
```
stat_tree(reg)
```

```
Всего узлов: 71
Листовых узлов: 36
Глубина дерева: 9
Минимальная глубина листьев дерева: 3
Средняя глубина листьев дерева: 6.083333333333333
```

В целом получили примерно тот же результат. Коэффициент детерминации оказался немного выше, тогда как абсолютные ошибки также стали немного выше. Видно, что дерево решений достигло своего предела. При этом весьма поразительно, насколько хорошо данный метод решил задачу регрессии. Посмотрим на построенное дерево.

[60]:
```
plot_tree(reg, filled=True);
```



Вывод функции `plot_tree` выглядит весьма странно. Видимо, для настолько больших деревьев решений она не предназначена. Возможно, это со временем будет исправлено, так как эту функциональность только недавно добавили.

3. Вывод

Такое дерево уже можно анализировать. Видно, что сгенерировалось огромное множество различных условий, и, фактически, модель переобучена, но с другой стороны дерево решений и не могло быть построено иначе для задачи регрессии. К тому же на тестовой выборке данное дерево работает также довольно хорошо, так что оно имеет право на существование. Особенно при том, что у нас стоит задача классификации.

[ ]: