МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Систем обработки информации и управления»

ОТЧЕТ

Лабораторная работа №4
по курсу «Методы машинного обучения»

Тема: «Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей.»

ИСПОЛНИТЕЛЬ: _Бабин В.Е.________
ФИО

группа ИУ5-22М ____________________
подпись

"__"_________2020 г.

ПРЕПОДАВАТЕЛЬ: ____________________
ФИО

____
подпись

"__"_________2020 г.

Москва - 2020

Задание:

1)Выберите 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.

lab4

May 4, 2020

[133]:
```
import pandas as pd
import numpy as np
import io
import requests
from sklearn.impute import SimpleImputer


# mushrooms dataset
url = "https://www.wolframcloud.com/obj/d0c0084e-0b60-46db-a4d9-beb33412905e"
s = requests.get(url).content
data = pd.read_csv(io.StringIO(s.decode('utf-8')))
data.head()
```

[133]:
```
  CapShape CapSurface CapColor  …  Population  Habitat      Class
0   convex     smooth    brown  …   scattered    urban  poisonous
1   convex     smooth   yellow  …    numerous  grasses     edible
2     bell     smooth    white  …    numerous  meadows     edible
3   convex      scaly    white  …   scattered    urban  poisonous
4   convex     smooth     gray  …    abundant  grasses     edible

[5 rows x 23 columns]
```

Dataset preparation

[134]:
```
rows, columns = data.shape
print('rows = {}; columns = {}'.format(rows, columns), '\n\n')

# go through each dataset column and check unique values to find empty one␣
↪(like NaN or Missing[])
for col in data.columns:
  print('{}: {}'.format(col, data[col].unique()))
```

```
rows = 8124; columns = 23


CapShape: ['convex' 'bell' 'sunken' 'flat' 'knobbed' 'conical']
CapSurface: ['smooth' 'scaly' 'fibrous' 'grooves']
```

```
CapColor: ['brown' 'yellow' 'white' 'gray' 'red' 'pink' 'buff' 'purple'
'cinnamon'
 'green']
Bruises: [ True False]
Odor: ['pungent' 'almond' 'anise' 'none' 'foul' 'creosote' 'fishy' 'spicy'
 'musty']
GillAttachment: ['free' 'attached']
GillSpacing: ['close' 'crowded']
GillSize: ['narrow' 'broad']
GillColor: ['black' 'brown' 'gray' 'pink' 'white' 'chocolate' 'purple' 'red'
'buff'
 'green' 'yellow' 'orange']
StalkShape: ['enlarging' 'tapering']
StalkRoot: ['equal' 'club' 'bulbous' 'rooted' 'Missing[]']
StalkSurfaceAboveRing: ['smooth' 'fibrous' 'silky' 'scaly']
StalkSurfaceBelowRing: ['smooth' 'fibrous' 'scaly' 'silky']
StalkColorAboveRing: ['white' 'gray' 'pink' 'brown' 'buff' 'red' 'orange'
'cinnamon' 'yellow']
StalkColorBelowRing: ['white' 'pink' 'gray' 'buff' 'brown' 'red' 'yellow'
'orange' 'cinnamon']
VeilType: ['partial']
VeilColor: ['white' 'brown' 'orange' 'yellow']
RingNumber: [1 2 0]
RingType: ['pendant' 'evanescent' 'large' 'flaring' 'none']
SporePrintColor: ['black' 'brown' 'purple' 'chocolate' 'white' 'green' 'orange'
'yellow'
 'buff']
Population: ['scattered' 'numerous' 'abundant' 'several' 'solitary' 'clustered']
Habitat: ['urban' 'grasses' 'meadows' 'woods' 'paths' 'waste' 'leaves']
Class: ['poisonous' 'edible']
```

[135]:

```
"""
on initial viewing it seems that we have single column
with absence of values in rows: this column is StalkRoot and
absence of values is indicated like Missing[]
"""

# Take a look at columns more precisely to ensure that this column is single␣
 ↪with absence of values
for col in data.columns:
    # Missing[] amount
    temp_null_count = data[data[col] == 'Missing[]'].shape[0]
    dt = str(data[col].dtype)
    if temp_null_count>0 and (dt=='object'):
        temp_perc = round((temp_null_count / rows) * 100.0, 2)
        print('Column {}. Data type {}. amount of Missing[] values {}, {}%.'.
 ↪format(col, dt, temp_null_count, temp_perc))
```

```
Column StalkRoot. Data type object. amount of Missing[] values 2480, 30.53%.
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/ops/array_ops.py:253:
FutureWarning: elementwise comparison failed; returning scalar instead, but in
the future will perform elementwise comparison
  res_values = method(rvalues)
```

[136]:
```
data['StalkRoot'].unique()
```

[136]:
```
array(['equal', 'club', 'bulbous', 'rooted', 'Missing[]'], dtype=object)
```

[137]:
```
# impute data with most frequent values
imputation = SimpleImputer(missing_values='Missing[]', strategy='most_frequent')
data_imputed = imputation.fit_transform(data[['StalkRoot']])
np.unique(data_imputed)
```

[137]:
```
array(['bulbous', 'club', 'equal', 'rooted'], dtype=object)
```

[138]:
```
# put imputed data in our dataset
for i in range(rows):
  data['StalkRoot'][i] = data_imputed[i][0]
data['StalkRoot'].unique()
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[138]:
```
array(['equal', 'club', 'bulbous', 'rooted'], dtype=object)
```

[139]:
```
# now we`ll transform categorical columns to columns with values in [0, 1]
object_columns = []

# initially, find amount of unique values for each column
# and add categorical columns to object_columns
for column in data.columns:
    dt = str(data[column].dtype)
    amount_unique = len(pd.unique(data[column]))
    print('{}: {}, {}'.format(column, amount_unique, dt))
    if dt == 'object':
        object_columns.append({'column': column, 'amount': amount_unique})
```

```
CapShape: 6, object
CapSurface: 4, object
CapColor: 10, object
```

```
Bruises: 2, bool
Odor: 9, object
GillAttachment: 2, object
GillSpacing: 2, object
GillSize: 2, object
GillColor: 12, object
StalkShape: 2, object
StalkRoot: 4, object
StalkSurfaceAboveRing: 4, object
StalkSurfaceBelowRing: 4, object
StalkColorAboveRing: 9, object
StalkColorBelowRing: 9, object
VeilType: 1, object
VeilColor: 4, object
RingNumber: 3, int64
RingType: 5, object
SporePrintColor: 9, object
Population: 6, object
Habitat: 7, object
Class: 2, object
```

[0]:
```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
```

[0]:
```
"""
after oneHot encoding for single value we`ll have something like this [0.0, 1.
↪0, 0.0, 0.0, ...]
but we need to have just a number - so this function will normalize it with␣
↪next formula:
norm_val = (N - i) / N, where
N - amount of unique values for our column
i - index of 1.0 in values [0.0, 1.0, 0.0, 0.0, ...] before normalizing
so for example if we`ll have 10 different values - then in normalized view it␣
↪will variating
from 0.1 (if index = 9) to 1.0 (if index = 0)
"""
def from_bytes_to_num(col_in_arr, uniquie_amount):
    normalized_col = []
    for value in col_in_arr:
        normalized_value = (uniquie_amount - np.where(value == 1.0)[0][0]) /␣
↪uniquie_amount
        normalized_col.append(float("{0:.4f}".format(normalized_value))) #␣
↪digits after float point
    return normalized_col


normalized_data = []
```

```
for col in object_columns:
    uniquie_amount = col['amount']
    col_name = col['column']

    #encode with oneHot
    column_after_encoding = ohe.fit_transform(data[[col_name]])

    #fetch it to array
    col_in_arr = column_after_encoding.toarray()

    #normilizing column values
    normalized_col = from_bytes_to_num(col_in_arr, uniquie_amount)

    normalized_data.append({'column': col_name, 'data': normalized_col})
```

[142]:
```
#set normalized values for general dataset
for col in normalized_data:
    col_in_dataFrame = pd.DataFrame(data={col['column']: col['data']})
    data[col['column']] = col_in_dataFrame

data
```

[142]:
```
      CapShape  CapSurface  CapColor  …  Population  Habitat  Class
0       0.6667        0.25       1.0  …      0.5000   0.4286    0.5
1       0.6667        0.25       0.1  …      0.6667   1.0000    1.0
2       1.0000        0.25       0.2  …      0.6667   0.7143    1.0
3       0.6667        0.50       0.2  …      0.5000   0.4286    0.5
4       0.6667        0.25       0.7  …      1.0000   1.0000    1.0
…          …         …        …  …         …        …      …
8119    0.3333        0.25       1.0  …      0.8333   0.8571    1.0
8120    0.6667        0.25       1.0  …      0.3333   0.8571    1.0
8121    0.5000        0.25       1.0  …      0.8333   0.8571    1.0
8122    0.3333        0.50       1.0  …      0.3333   0.8571    0.5
8123    0.6667        0.25       1.0  …      0.8333   0.8571    1.0

[8124 rows x 23 columns]
```

Data splitting

[143]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, train_test_split

dcopy = data.copy()

X = dcopy.drop("Class", axis=1)
y = dcopy["Class"]
```

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

```
   CapShape  CapSurface  CapColor  …  SporePrintColor  Population  Habitat
0    0.6667        0.25       1.0  …           1.0000      0.5000   0.4286
1    0.6667        0.25       0.1  …           0.8889      0.6667   1.0000
2    1.0000        0.25       0.2  …           0.8889      0.6667   0.7143
3    0.6667        0.50       0.2  …           1.0000      0.5000   0.4286
4    0.6667        0.25       0.7  …           0.8889      1.0000   1.0000

[5 rows x 22 columns]

0    0.5
1    1.0
2    1.0
3    0.5
4    1.0
Name: Class, dtype: float64
```

[0]:
```python
columns = X.columns
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

[145]:
```python
# divide train and test selections
X_train, X_test, y_train, y_test = train_test_split(X, y,
                            test_size=0.25, random_state=1)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(6093, 22)
(2031, 22)
(6093,)
(2031,)
```

K

Nearest neighbor model for an arbitrary given hyperparameter K

[0]:
```python
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error, r2_score

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)))
```

[147]:
```
reg_15 = KNeighborsRegressor(n_neighbors=15)
reg_15.fit(X_train, y_train)
```

[147]:
```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=15, p=2,
                    weights='uniform')
```

[148]:
```
test_model(reg_15)
```

```
mean_absolute_error: 0.00218283275890366
median_absolute_error: 0.0
r2_score: 0.9921832263403219
```

-

Using cross validation

[149]:
```
from sklearn.model_selection import KFold, RepeatedKFold, ShuffleSplit

scores = cross_val_score(KNeighborsRegressor(n_neighbors=15), X, y,
                         cv=KFold(n_splits=10), scoring="r2")
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.9973447  1.         1.         1.         1.         0.99432662
 0.88597886 0.98065168 0.9964011  0.95745946]
0.9812162420110482 ± 0.03421807954746192
```

[150]:
```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=15), X, y,
                         cv=RepeatedKFold(n_splits=5, n_repeats=2),
                         scoring="r2")
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.99423453 0.99658153 0.99644782 0.9954995  0.99693324 0.9961005
 0.99626767 0.99451314 0.9945565  0.99599151]
0.9957125930439743 ± 0.0009125396963525019
```

[151]:
```
scores = cross_val_score(KNeighborsRegressor(n_neighbors=15), X, y,
                         cv=ShuffleSplit(n_splits=10), scoring="r2")
print(scores)
print(scores.mean(), "±", scores.std())
```

```
[0.99704793 0.99424454 0.99973745 0.99222284 0.99758438 0.99687133
 0.99314695 0.99492516 0.99365927 0.99584219]
```

```
0.995528203082066 ± 0.002196769232138002
```

K

Selection of hyperparameter K

[152]:
```
from sklearn.model_selection import GridSearchCV

n_range = np.array(range(1, 25, 2))
tuned_parameters = [{'n_neighbors': n_range}]
n_range
```

[152]:
```
array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23])
```

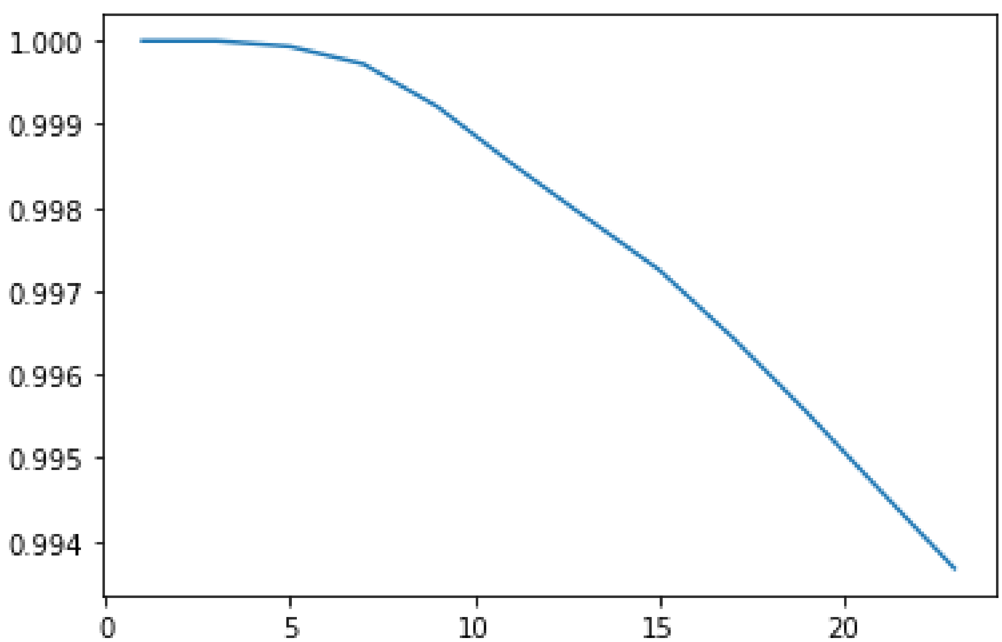
[153]:
```
gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters,
                  cv=ShuffleSplit(n_splits=10), scoring="r2",
                  return_train_score=True, n_jobs=-1)
gs.fit(X, y)
gs.best_params_
```

[153]:
```
{'n_neighbors': 1}
```

[154]:
```
import matplotlib.pyplot as plt
from sklearn.model_selection import learning_curve, validation_curve

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

[154]:
```
[<matplotlib.lines.Line2D at 0x7f3340de8c50>]
```

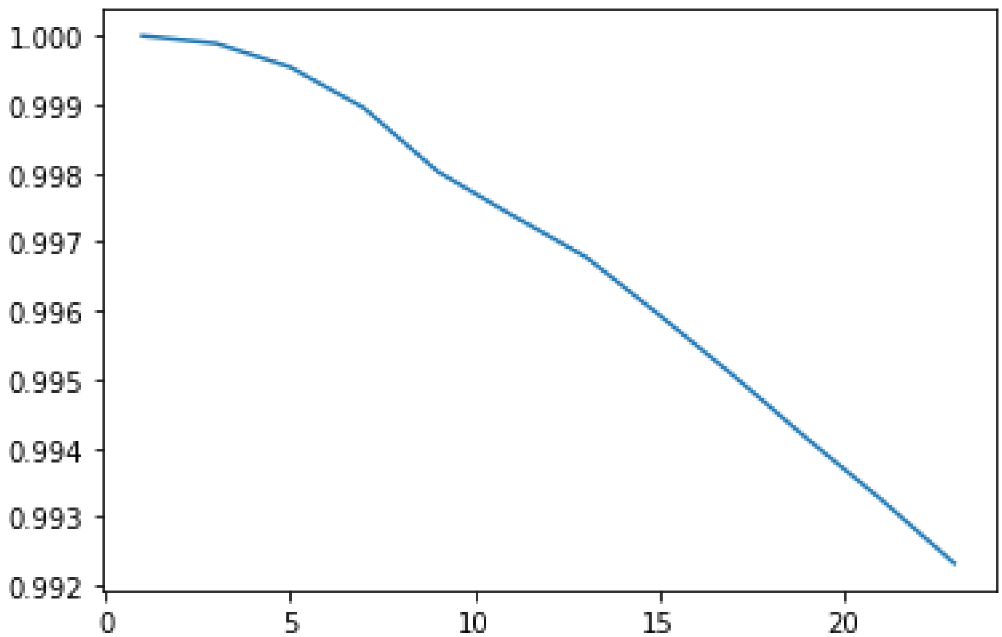
[154]:



[155]:
```
plt.plot(n_range, gs.cv_results_["mean_test_score"])
```

[155]:
```
[<matplotlib.lines.Line2D at 0x7f333ef6e780>]
```

[155]:



[156]:
```
reg = KNeighborsRegressor(**gs.best_params_)
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 0.0
median_absolute_error: 0.0
r2_score: 1.0
```

[0]:
```
# train curve
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(
```

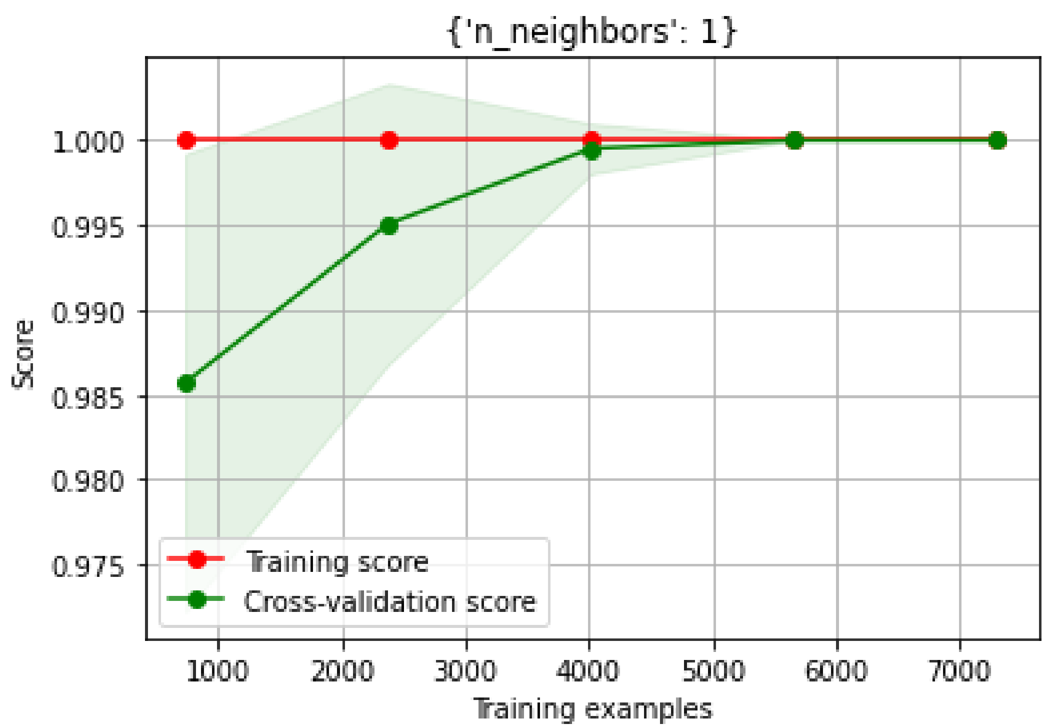
```
        estimator, X, y, cv=cv, n_jobs=-1, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    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
```

[158]:

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

[158]: <module 'matplotlib.pyplot' from '/usr/local/lib/python3.6/dist-packages/matplotlib/pyplot.py'>

[158]:



[0]:

```
# validation curve
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
```

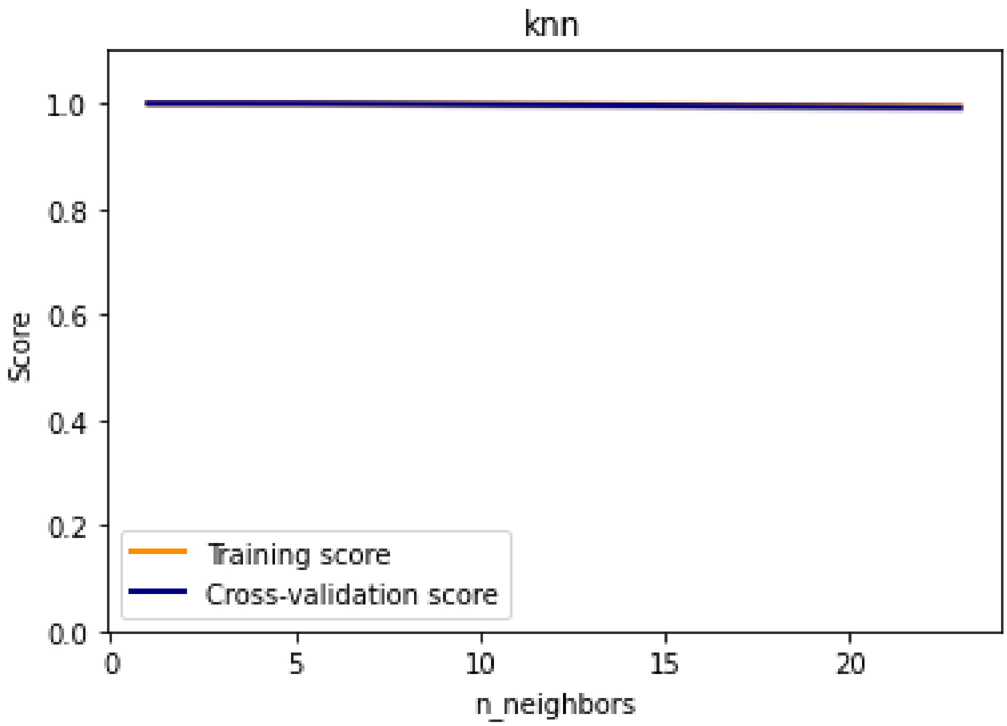
[160]:

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

[160]:

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.6/dist-
packages/matplotlib/pyplot.py'>
```

[160]:


knn
1.0
0.8
0.6
0.4
0.2
0.0
Score
Training score
Cross-validation score
0
5
10
15
20
n_neighbors

[0]: