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

Лабораторная работа №4

по дисциплине

«Методы машинного обучения»

на тему

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

Выполнил:
студент группы ИУ5-24М
Зубаиров В. А.

____

Москва — 2020 г.

[54]:

```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from sklearn.datasets import load_iris, load_boston
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import KFold, RepeatedKFold, LeaveOneOut, LeavePOut,
  ↪ShuffleSplit, StratifiedKFold
from sklearn.model_selection import learning_curve, validation_curve
from sklearn.metrics import mean_absolute_error, mean_squared_error,
  ↪mean_squared_log_error, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

[3]:

```
data = pd.read_csv("heart.csv")
```

[4]:

```
data.info(5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

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

[5]:
```
              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
```

[6]:
```
data.corr()
```

[6]:
```
               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
```

[9]:
```
np.unique(data.target)
```

[9]:
```
array([0, 1])
```

[15]:
```
target = data.iloc[:, -1]
```

[17]:
```
data_data = data.iloc[:, 0:-1]
```

[19]:
```
target.shape
```

[19]:
```
(303,)
```

[20]:
```
data_data.shape
```

[20]:
```
(303, 13)
```

[21]:
```
heart_X_train, heart_X_test, heart_y_train, heart_y_test = train_test_split(
    data_data, target, test_size=0.5, random_state=1)
```

[22]:
```
heart_X_train.shape, heart_y_train.shape
```

```
[22]: ((151, 13), (151,))
```

```
[23]: heart_X_test.shape, heart_y_test.shape
```

```
[23]: ((152, 13), (152,))
```

```
[24]: np.unique(heart_y_train), np.unique(heart_y_test)
```

```
[24]: (array([0, 1]), array([0, 1]))
```

```
[25]: cl1_1 = KNeighborsClassifier(n_neighbors=2)
      cl1_1.fit(heart_X_train, heart_y_train)
      target1_1 = cl1_1.predict(heart_X_test)
      len(target1_1), target1_1
```

```
[25]: (152,
       array([0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0,
              0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
              0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1,
              0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0,
              1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0,
              0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0,
              0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1]))
```

```
[26]: cl1_2 = KNeighborsClassifier(n_neighbors=10)
      cl1_2.fit(heart_X_train, heart_y_train)
      target1_2 = cl1_2.predict(heart_X_test)
      len(target1_2), target1_2
```

```
[26]: (152,
       array([1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1,
              0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
              1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0,
              0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0,
              1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0,
              1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0,
              0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1]))
```

```
[27]: accuracy_score(heart_y_test, target1_1)
```

```
[27]: 0.6052631578947368
```

```
[28]: accuracy_score(heart_y_test, target1_2)
```

```
[28]: 0.625
```

```
[29]: cl1_3 = KNeighborsClassifier(n_neighbors=30)
      cl1_3.fit(heart_X_train, heart_y_train)
      target1_3 = cl1_3.predict(heart_X_test)
      len(target1_3), target1_3
```

```
[29]: (152,
       array([1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
              1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0,
              1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1,
              0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0,
              1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
              1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
              0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1]))
```

```
[30]: accuracy_score(heart_y_test, target1_3)
```

```
[30]: 0.618421052631579
```

```
[61]: kf = KFold(n_splits=5)
      scores = cross_val_score(KNeighborsClassifier(n_neighbors=10),
                               data_data, target, scoring='f1_weighted',
                               cv=kf)
      scores
```

```
[61]: array([0.67391304, 0.67391304, 0.62214834, 0.58823529, 0.63636364])
```

```
[62]: scores = cross_val_score(KNeighborsClassifier(n_neighbors=50),
                               data_data, target,
                               cv=LeaveOneOut())
      scores, np.mean(scores)
```

```
[62]: (array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1.,
              0., 1., 1., 1., 1., 1., 0., 1., 1., 1., 0., 0., 1., 1., 0., 1., 0.,
              0., 1., 1., 1., 0., 0., 0., 1., 1., 0., 1., 1., 1., 1., 0., 1., 1.,
              0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1.,
              1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0.,
              1., 0., 1., 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 0., 1., 1., 0.,
              1., 1., 1., 0., 0., 1., 1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1.,
              1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1.,
              0., 1., 1., 0., 1., 1., 1., 1., 0., 0., 0., 1., 1., 1., 0., 0., 1.,
              0., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0.,
              1., 0., 0., 0., 1., 1., 0., 0., 1., 1., 1., 1., 0., 0., 1., 0., 0.,
              1., 0., 0., 1., 1., 1., 1., 0., 1., 0., 0., 1., 0., 0., 1., 1., 1.,
              1., 0., 1., 0., 0., 0., 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 0.,
              1., 0., 1., 1., 1., 1., 0., 0., 1., 0., 1., 1., 1., 1., 0., 0., 0.,
              0., 0., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 1., 1., 1., 1., 1.,
              1., 1., 1., 0., 0., 0., 0., 1., 0., 1., 0., 1., 0., 1., 1., 1., 1.,
              1., 0., 1., 0., 1., 1., 0., 1., 1., 0., 0., 0., 0., 1., 0., 0., 1.,
              1., 0., 1., 0., 0., 0., 1., 0., 1., 1., 1., 0., 0., 0.]),
       0.6633663366336634)
```

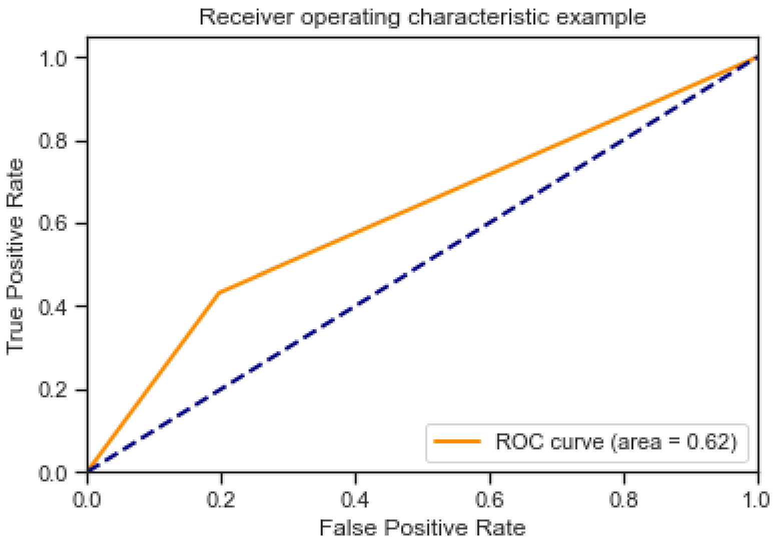
```
[ ]:
```

```
[32]: fpr, tpr, thresholds = roc_curve(heart_y_test, target1_2,
                                       pos_label=1)
      fpr, tpr, thresholds
```

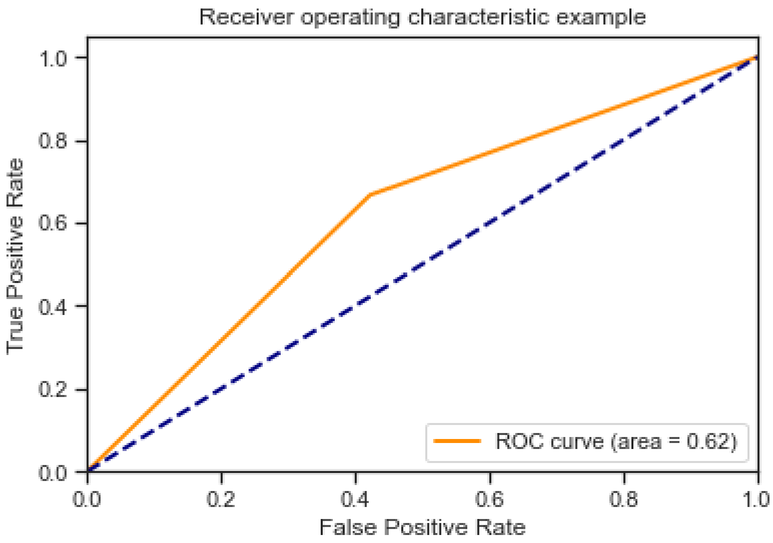
[32]:
```
(array([0.        , 0.42253521, 1.        ]),
 array([0.        , 0.66666667, 1.        ]),
 array([2, 1, 0]))
```

[33]:
```
def draw_roc_curve(y_true, y_score, pos_label, average):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

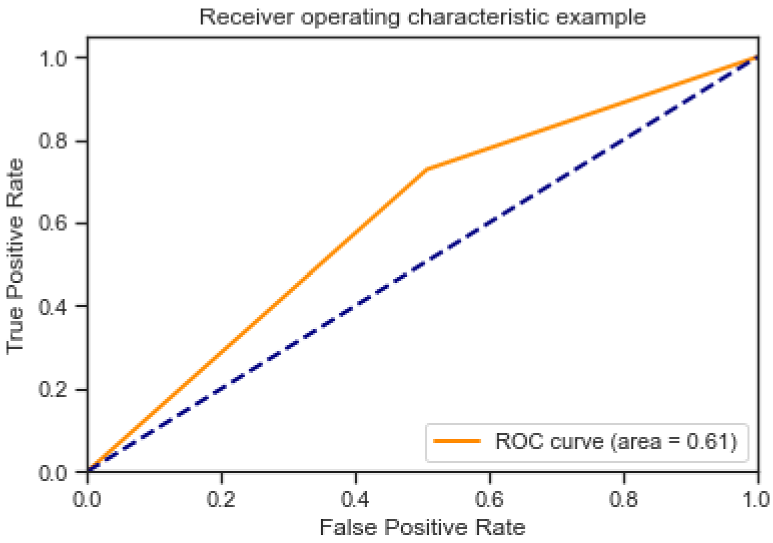
[34]:
```
draw_roc_curve(heart_y_test, target1_1, pos_label=1, average='micro')
```



[35]:
```
draw_roc_curve(heart_y_test, target1_2, pos_label=1, average='micro')
```



```
[36]: draw_roc_curve(heart_y_test, target1_3, pos_label=1, average='micro')
```



```
[56]: scores = cross_val_score(KNeighborsClassifier(n_neighbors=5),
                               data_data, target, cv=3)
```

```
[57]: scores
```

```
[57]: array([0.62376238, 0.6039604 , 0.66336634])
```

```
[58]: np.mean(scores)
```

```
[58]: 0.6303630363036303
```

```
[37]: n_range = np.array(range(5,55,5))
      tuned_parameters = [{'n_neighbors': n_range}]
```

```
tuned_parameters
```

```
[37]: [{'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])}]
```

```
[41]: %%time
      clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='accuracy')
      clf_gs.fit(heart_X_train, heart_y_train)
```

```
CPU times: user 219 ms, sys: 6.41 ms, total: 225 ms
Wall time: 237 ms
```

```
[41]: GridSearchCV(cv=5, error_score=nan,
                   estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                                  metric='minkowski',
                                                  metric_params=None, n_jobs=None,
                                                  n_neighbors=5, p=2,
                                                  weights='uniform'),
                   iid='deprecated', n_jobs=None,
                   param_grid=[{'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40,
      45, 50])}],
                   pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                   scoring='accuracy', verbose=0)
```

```
[42]: clf_gs.best_estimator_
```

```
[42]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                           metric_params=None, n_jobs=None, n_neighbors=35, p=2,
                           weights='uniform')
```

```
[44]: clf_gs.best_score_
```

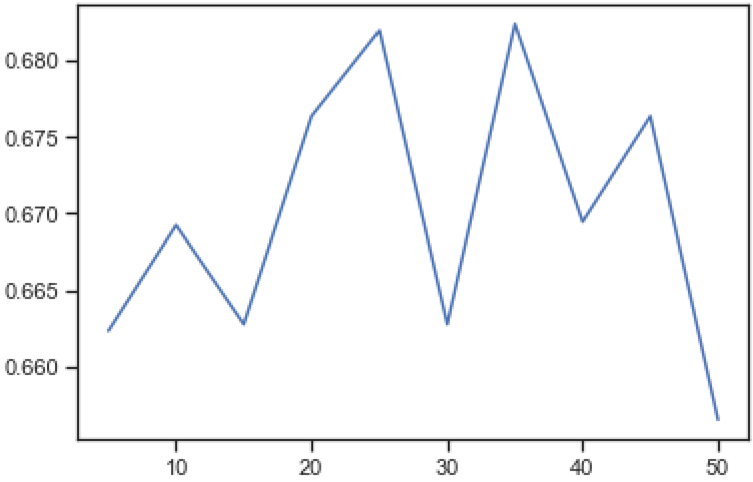
```
[44]: 0.6823655913978494
```

```
[46]: clf_gs.best_params_
```

```
[46]: {'n_neighbors': 35}
```

```
[47]: plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

```
[47]: [<matplotlib.lines.Line2D at 0x12241a5b0>]
```



[48]:

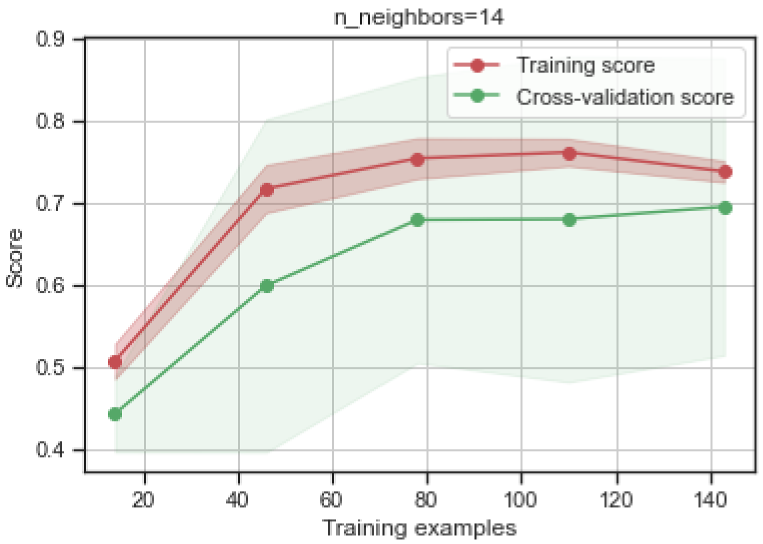
```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    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=0.3,
                     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
```

[50]:

```
plot_learning_curve(KNeighborsClassifier(n_neighbors=14), 'n_neighbors=14',
                    heart_X_train, heart_y_train, cv=20)
```

[50]:

```
<module 'matplotlib.pyplot' from
'/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/matplotlib/pyplot.py'>
```



[51]:

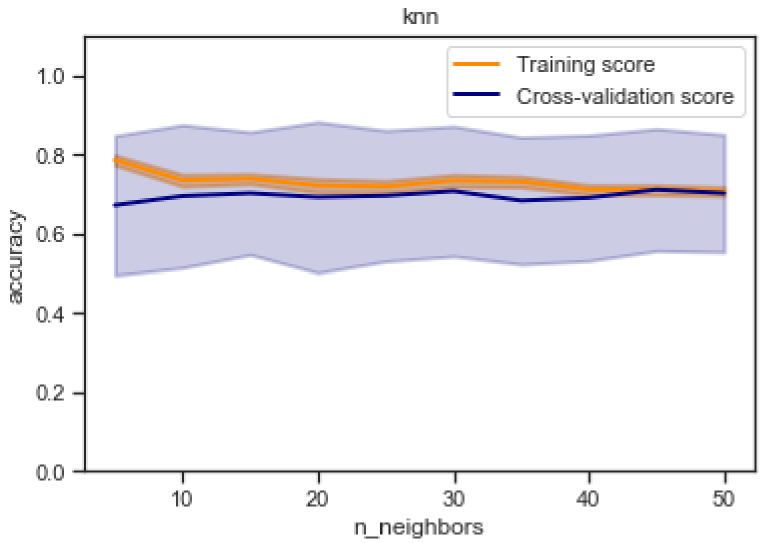
```
def plot_validation_curve(estimator, title, X, y,
                          param_name, param_range, cv,
                          scoring="accuracy"):

    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(str(scoring))
    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=0.4,
                     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
```

[52]:

```
plot_validation_curve(KNeighborsClassifier(), 'knn',
                      heart_X_train, heart_y_train,
                      param_name='n_neighbors', param_range=n_range,
                    cv=20, scoring="accuracy")
```

[52]:

```
<module 'matplotlib.pyplot' from
'/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/matplotlib/pyplot.py'>
```



[65]:

```
n_range = np.array(range(5,55,5))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

[65]:

```
[{'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])}]
```

[67]:

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=LeaveOneOut(),
  ↪scoring='accuracy')
clf_gs.fit(data_data, target)
```

```
CPU times: user 9.65 s, sys: 61.3 ms, total: 9.71 s
Wall time: 9.87 s
```

[67]:

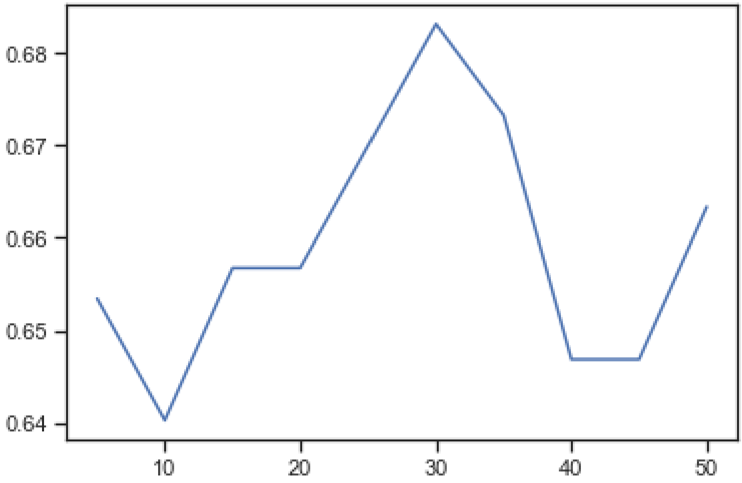
```
GridSearchCV(cv=LeaveOneOut(), error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid=[{'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40,
45, 50])}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

[69]:
```
clf_gs.best_params_
```

[69]:
```
{'n_neighbors': 30}
```

[70]:
```
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

[70]:
```
[<matplotlib.lines.Line2D at 0x122680820>]
```



[ ]:

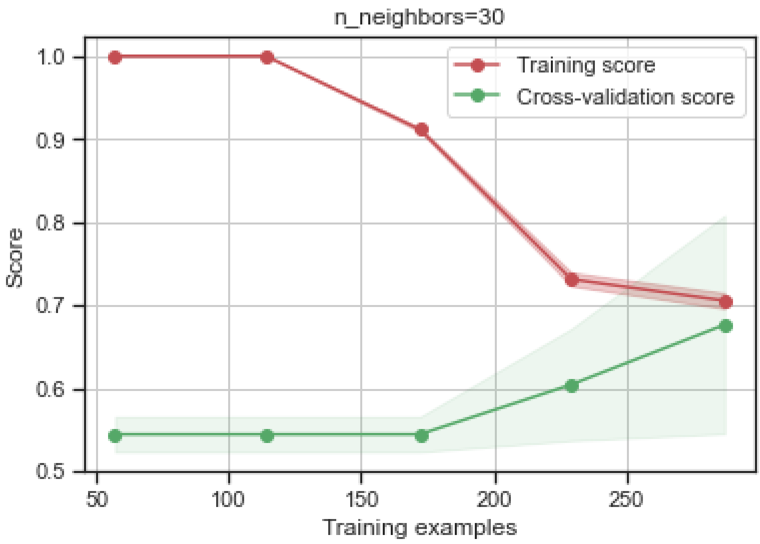
[71]:
```
clf_gs.best_estimator_.fit(heart_X_train, heart_y_train)
target2_0 = clf_gs.best_estimator_.predict(heart_X_train)
target2_1 = clf_gs.best_estimator_.predict(heart_X_test)
```

[72]:
```
accuracy_score(heart_y_train, target2_0), accuracy_score(heart_y_test, target2_1)
```

[72]:
```
(0.7284768211920529, 0.618421052631579)
```

[73]:
```
plot_learning_curve(clf_gs.best_estimator_, 'n_neighbors=30',
                    data_data, target, cv=20, train_sizes=np.linspace(.2, 1.0, 5))
```

[73]:
```
<module 'matplotlib.pyplot' from
'/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/matplotlib/pyplot.py'>
```



[74]:

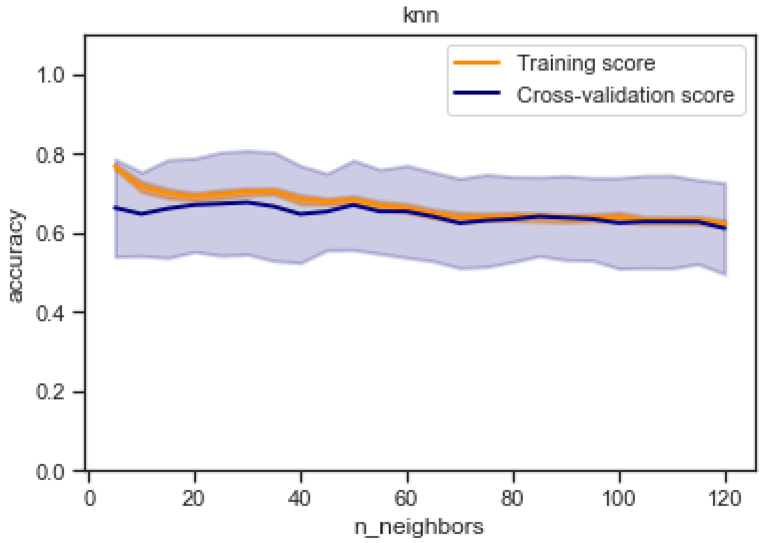
```
n_range2 = np.array(range(5,125,5))
```

[75]:

```
plot_validation_curve(clf_gs.best_estimator_, 'knn',
                      data_data, target,
                      param_name='n_neighbors', param_range=n_range2,
                      cv=20, scoring="accuracy")
```

[75]:

```
<module 'matplotlib.pyplot' from
'/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-
packages/matplotlib/pyplot.py'>
```



[ ]: