**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5.

Курс «Технологии машинного обучения»

Отчет по лабораторной работе №4

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

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-62 |  | преподаватель каф. ИУ5 |
| Король Константин |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2019 г.

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.

**Код программы**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score, recall\_score, f1\_score

from sklearn.metrics import make\_scorer

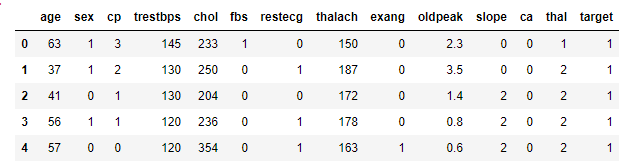
from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import learning\_curve, validation\_curve

import matplotlib.pyplot as plt

data = pd.read\_csv('data/heart.csv', sep=',')

data.head()



data.isnull().sum()

OUT: age 0

sex 0

cp 0

trestbps 0

chol 0

fbs 0

restecg 0

thalach 0

exang 0

oldpeak 0

slope 0

ca 0

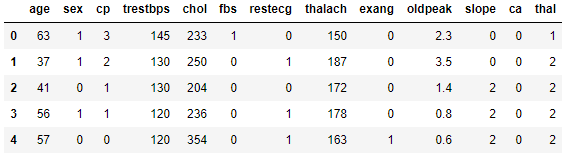
thal 0

target 0

dtype: int64

X = data.loc[:, data.columns != 'target']

X.head()



Y = data['target']

Y.head()

OUT: 0 1

1 1

2 1

3 1

4 1

Name: target, dtype: int64

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, Y, test\_size = 0.2, random\_state = 1)

X\_train.shape, y\_train.shape

OUT: ((242, 13), (242,))

X\_test.shape, y\_test.shape

OUT: ((61, 13), (61,))

knc = KNeighborsClassifier(n\_neighbors=5)

knc.fit(X\_train, y\_train)

predicted = knc.predict(X\_test)

predicted

OUT: array([0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0,

0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0,

1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0], dtype=int64)

Метрики: f1, recall, accuracy  
classification\_report(y\_test, predicted, output\_dict=True)["0"], \

classification\_report(y\_test, predicted, output\_dict=True)["1"]

OUT: ({'precision': 0.5833333333333334,

'recall': 0.4666666666666667,

'f1-score': 0.5185185185185186,

'support': 30},

{'precision': 0.5675675675675675,

'recall': 0.6774193548387096,

'f1-score': 0.6176470588235294,

'support': 31})

# при помощи решетчатого поиска и кросс-валидации найдем оптимальное значение гиперпараметра k

scoring = {

'recall': make\_scorer(recall\_score),

'f1': make\_scorer(f1\_score),

'accuracy': make\_scorer(accuracy\_score)

}

n\_range = np.array(range(1,60,3))

tuned\_parameters = [{'n\_neighbors': n\_range}]

tuned\_parameters

OUT: [{'n\_neighbors': array([ 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49,

52, 55, 58])}]

knc\_gs = GridSearchCV(KNeighborsClassifier(), tuned\_parameters, cv=5, scoring=scoring, refit='f1')

knc\_gs.fit(X\_train, y\_train)

OUT: GridSearchCV(cv=5, error\_score='raise-deprecating',

estimator=KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=None, n\_neighbors=5, p=2,

weights='uniform'),

fit\_params=None, iid='warn', n\_jobs=None,

param\_grid=[{'n\_neighbors': array([ 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49,

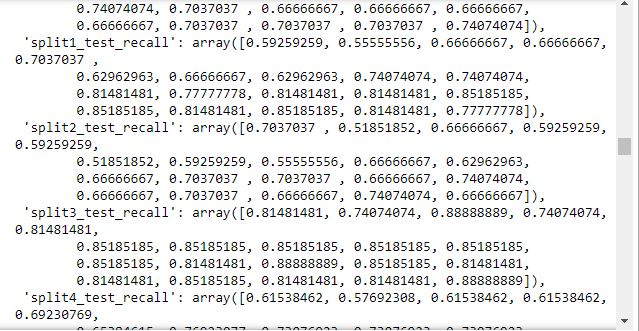
52, 55, 58])}],

pre\_dispatch='2\*n\_jobs', refit='f1', return\_train\_score='warn',

scoring={'recall': make\_scorer(recall\_score), 'f1': make\_scorer(f1\_score), 'accuracy': make\_scorer(accuracy\_score)},

verbose=0)

knc\_gs.cv\_results\_



# Лучшая модель

best\_knc = knc\_gs.best\_estimator\_

best\_knc

OUT: KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=None, n\_neighbors=31, p=2,

weights='uniform')

# Лучшее f1

knc\_gs.best\_score\_

OUT: 0.7032157012880798

# Лучшее k

knc\_gs.best\_params\_

OUT: {'n\_neighbors': 31}

# Проверяем на новом k

best\_knc.fit(X\_train, y\_train)

predicted\_best = knc.predict(X\_test)

predicted\_best

OUT: array([0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0,

0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0,

1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0], dtype=int64)

classification\_report(y\_test, predicted\_best, output\_dict=True)["0"], \

classification\_report(y\_test, predicted\_best, output\_dict=True)["1"]

OUT: ({'precision': 0.5833333333333334,

'recall': 0.4666666666666667,

'f1-score': 0.5185185185185186,

'support': 30},

{'precision': 0.5675675675675675,

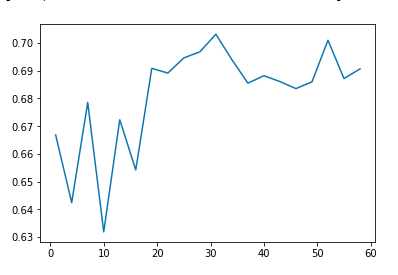
'recall': 0.6774193548387096,

'f1-score': 0.6176470588235294,

'support': 31})

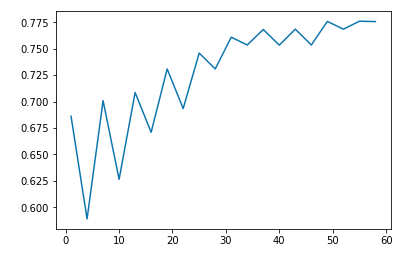
# изменение f1 от k

plt.plot(n\_range, knc\_gs.cv\_results\_['mean\_test\_f1'])



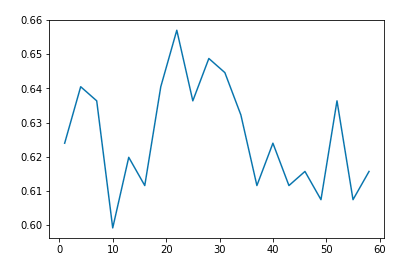
# изменение recall от k

plt.plot(n\_range, knc\_gs.cv\_results\_['mean\_test\_recall'])



# изменение accuracy от k

plt.plot(n\_range, knc\_gs.cv\_results\_['mean\_test\_accuracy'])



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=0.1,

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

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

plot\_validation\_curve(KNeighborsClassifier(), 'knc',

X\_train, y\_train,

param\_name='n\_neighbors', param\_range=n\_range,

cv=20, scoring="f1")



plot\_learning\_curve(KNeighborsClassifier(n\_neighbors=10), 'n\_neighbors=10',

X\_train, y\_train, cv=20)

