Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления (ИУ5)

О т ч е т

по лабораторной работе №3

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

Дисциплина: Технологии машинного обучения

Студент гр. ИУ5-63Б ________________ Терентьев В.О.

(Подпись, дата) (Фамилия И.О.)

Преподаватель ________________ Гапанюк Ю.Е.

(Подпись, дата) (Фамилия И.О.)

Москва, 2021

1. Цель работы

Цель лабораторной работы: изучение способов подготовки выборки и подбора гиперпараметров на примере метода ближайших соседей.

2. Описание задания

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
3. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик.
4. Произведите подбор гиперпараметра K с использованием GridSearchCV и/или RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Желательно использование нескольких стратегий кросс-валидации.
5. Сравните метрики качества исходной и оптимальной моделей.

3. Основная часть

Лабораторная работа №3

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

Импорт библиотек

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Загрузка данных

В качестве набора данных использован набор данных по предсказанию уровня солнечной радиации - https://www.kaggle.com/dronio/SolarEnergy (https://www.kaggle.com/dronio/SolarEnergy)

In [2]:

```
from datetime import timedelta
data = pd.read_csv('SolarPrediction.csv', sep=",")
# Преобразование данных датасета в нужный формат
data['Data'] = pd.to_datetime(data['UNIXTime'], unit='s') + pd.DateOffset(hours=-10)
data['UNIXTime'] = data['Data'].dt.month
data.rename(columns={'UNIXTime': 'Month'}, inplace = True)
data['Data'] = data['Data'].dt.second + data['Data'].dt.minute*60 + data['Data'].dt.hour*60
data.rename(columns={'Data': 'Time(Seconds)'}, inplace = True)
data['TimeSunRise'] = pd.to_datetime(data['TimeSunRise'], format='%H:%M:%S').dt.time
data['TimeSunSet'] = pd.to_datetime(data['TimeSunSet'], format='%H:%M:%S').dt.time
data['TimeSunSet'] = (pd.to_timedelta(data['TimeSunSet'].astype(str)) - pd.to_timedelta(dat
data.rename(columns={'TimeSunSet': 'DayLenght(Seconds)'}, inplace = True)
data['TimeSunRise'] = pd.to_timedelta(data['TimeSunRise'].astype(str)).dt.total_seconds()
data.rename(columns={'TimeSunRise': 'TimeSunRise(Seconds)'}, inplace = True)
# Удаление ненужных столбцов
data.drop(['Time'], inplace=True, axis=1)
data = data.join(pd.get_dummies(data['Month'].astype(str)))
data.drop(['Month'], inplace=True, axis=1)
# Первые 5 строк датасета
data.head()
```

Out[2]:

	Time(Seconds)	Radiation	Temperature	Pressure	Humidity	WindDirection(Degrees)	Speed
0	86126	1.21	48	30.46	59	177.39	5.62
1	85823	1.21	48	30.46	58	176.78	3.37
2	85526	1.23	48	30.46	57	158.75	3.37
3	85221	1.21	48	30.46	60	137.71	3.37
4	84924	1.17	48	30.46	62	104.95	5.62

Масштабирование данных

In [3]:

```
from sklearn.preprocessing import MinMaxScaler
# MinMax масштабирование
sc1 = MinMaxScaler()
for item in ['Time(Seconds)', 'Temperature', 'Pressure', 'Humidity', 'WindDirection(Degrees
    data.loc[:, item] = sc1.fit_transform(data[[item]])
# Первые 5 строк получившегося датасета
data.head()
```

Out[3]:

	Time(Seconds)	Radiation	Temperature	Pressure	Humidity	WindDirection(Degrees)	Speed
0	0.999315	1.21	0.378378	0.72973	0.536842	0.492692	0.13876
1	0.995800	1.21	0.378378	0.72973	0.526316	0.490996	0.08321
2	0.992354	1.23	0.378378	0.72973	0.515789	0.440894	0.08321
3	0.988815	1.21	0.378378	0.72973	0.547368	0.382426	0.08321
4	0.985369	1.17	0.378378	0.72973	0.568421	0.291391	0.13876

Разделение выборки на обучающую и тестовую

С использованием метода train_test_split

In [4]:

```
from sklearn.model_selection import train_test_split
data_train, data_test, data_y_train, data_y_test = train_test_split(data[['Time(Seconds)',
# Обучающая выборка
data_train
```

Out[4]:

	Time(Seconds)	Temperature	Pressure	Humidity	WindDirection(Degrees)	Speed	Tim
19310	0.115068	0.243243	0.567568	0.515789	0.443256	0.055556	
19368	0.915669	0.378378	0.675676	0.915789	0.307286	0.194321	
12448	0.428432	0.783784	0.783784	0.442105	0.164564	0.305432	
18105	0.302864	0.297297	0.621622	0.968421	0.458651	0.194321	
24541	0.932899	0.189189	0.432432	0.747368	0.573084	0.222222	
...	...	...	...	...	...	...	
32511	0.605692	0.297297	0.540541	0.894737	0.269688	0.083210	
5192	0.679047	0.567568	0.459459	0.926316	0.187962	0.166667	
12172	0.407477	0.864865	0.864865	0.084211	0.158395	0.138765	
235	0.174290	0.324324	0.540541	0.389474	0.521230	0.166667	
29733	0.856319	0.189189	0.351351	0.968421	0.486467	0.111111	

24514 rows × 12 columns

In [5]:

```
# Тестовая выборка
data_test
```

Out[5]:

	Time(Seconds)	Temperature	Pressure	Humidity	WindDirection(Degrees)	Speed	Tim
22830	0.891151	0.540541	0.810811	0.157895	0.622381	0.194321	
17281	0.174069	0.189189	0.648649	0.505263	0.304452	0.249877	
6136	0.682284	0.702703	0.567568	0.873684	0.856027	0.166667	
17582	0.121879	0.324324	0.729730	0.968421	0.386456	0.166667	
20747	0.121844	0.297297	0.675676	0.210526	0.468654	0.111111	
...	...	...	...	...	...	...	
15333	0.212568	0.405405	0.621622	0.905263	0.556494	0.083210	
13404	0.052516	0.486486	0.594595	0.978947	0.345968	0.194321	
7928	0.219484	0.351351	0.594595	0.989474	0.440977	0.138765	
7857	0.466630	0.459459	0.756757	0.957895	0.199994	0.333333	
24538	0.943331	0.189189	0.432432	0.747368	0.605791	0.166667	

8172 rows × 12 columns

Первичное обучение модели

In [6]:

```
from sklearn.neighbors import KNeighborsRegressor
```

In [7]:

```
# Модель ближайших соседей с количеством ближайших соседей = 5
KNeighborsRegressorObj = KNeighborsRegressor(n_neighbors=5)
KNeighborsRegressorObj.fit(data_train, data_y_train)
target0_0 = KNeighborsRegressorObj.predict(data_train)
target0_1 = KNeighborsRegressorObj.predict(data_test)
```

Оценка качества модели с помощью подходящих метрик

In [8]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error
# 1) Mean absolute error - средняя абсолютная ошибка
mean_absolute_error(data_y_test, target0_1)
```

Out[8]:

```
41.82268086147822
```

In [9]:

```
# 2) Mean squared error - средняя квадратичная ошибка
mean_squared_error(data_y_test, target0_1)
```

Out[9]:

```
9381.54396910426
```

In [10]:

```
# Root mean squared error - корень из средней квадратичной ошибки
# Значение RMSE сравнимо с MAE
mean_squared_error(data_y_test, target0_1, squared=False)
```

Out[10]:

```
96.85837067132742
```

In [11]:

```
# 3) Median absolute error
median_absolute_error(data_y_test, target0_1)
```

Out[11]:

```
1.1989999999999998
```

In [12]:

```
# 4) Метрика R2 или коэффициент детерминации
r2_score(data_y_test, target0_1)
```

Out[12]:

```
0.9054334392402374
```

Подбор гиперпараметра с использованием GridSearchCV и кросс-валидации

In [13]:

```
from sklearn.model_selection import cross_val_score, KFold, LeaveOneOut, ShuffleSplit
scores1_1 = cross_val_score(KNeighborsRegressor(n_neighbors=5), data[['Time(Seconds)', 'Tem
scores1_1, np.mean(scores1_1)
```

Out[13]:

```
(array([0.70593175, 0.72155213, 0.71646603, 0.77804221, 0.44361018]),
 0.6731204614710141)
```

In [14]:

```
scores1_2 = cross_val_score(KNeighborsRegressor(n_neighbors=5), data[['Time(Seconds)', 'Tem
scores1_2, np.mean(scores1_2)
```

Out[14]:

```
(array([0.91094506, 0.91545215, 0.90976649, 0.92055748, 0.91852173,
        0.91411547, 0.92005514, 0.91961056, 0.91612227, 0.9021582 ]),
 0.9147304564773581)
```

In [15]:

```
from sklearn.model_selection import GridSearchCV
n_range1_2 = np.array(range(1,11,1))
tuned_parameters1_2 = [{'n_neighbors': n_range1_2}]
```

In [16]:

```
%%time
clf_gs1 = GridSearchCV(KNeighborsRegressor(), tuned_parameters1_2, cv=5, scoring='neg_root_
clf_gs1.fit(data[['Time(Seconds)', 'Temperature', 'Pressure', 'Humidity', 'WindDirection(De
```

```
Wall time: 17.5 s
```

Out[16]:

```
GridSearchCV(cv=5, estimator=KNeighborsRegressor(),
             param_grid=[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,
8,  9, 10])}],
             scoring='neg_root_mean_squared_error')
```

In [17]:

```
clf_gs1.best_params_
```

Out[17]:

```
{'n_neighbors': 10}
```

In [18]:

```
clf_gs1.cv_results_
```

Out[18]:

```
{'mean_fit_time': array([0.1267292 , 0.1333797 , 0.14156952, 0.13587351,
0.1322495 ,
        0.13275561, 0.13373795, 0.13126612, 0.12983956, 0.13451738]),
 'std_fit_time': array([0.03396708, 0.0332352 , 0.04009393, 0.03307412, 0.
03826296,
        0.03557609, 0.03670708, 0.03513169, 0.03032385, 0.03337531]),
 'mean_score_time': array([0.18002419, 0.19028754, 0.20389352, 0.20494266,
0.21105952,
        0.21781735, 0.22166562, 0.2250494 , 0.22774138, 0.23812313]),
 'std_score_time': array([0.03707899, 0.04968698, 0.06170624, 0.04659014,
0.04467735,
        0.0462622 , 0.05480592, 0.0521364 , 0.05137638, 0.06010708]),
 'param_n_neighbors': masked_array(data=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              mask=[False, False, False, False, False, False, False, Fals
e,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 1},
  {'n_neighbors': 2},
  {'n_neighbors': 3},
  {'n_neighbors': 4},
  {'n_neighbors': 5},
  {'n_neighbors': 6},
  {'n_neighbors': 7},
  {'n_neighbors': 8},
  {'n_neighbors': 9},
  {'n_neighbors': 10}],
 'split0_test_score': array([-217.02458582, -202.4660575 , -195.57678672,
-191.51930783,
        -186.78869162, -182.3537172 , -179.71831884, -177.72725017,
        -176.22347649, -174.9258395 ]),
 'split1_test_score': array([-206.25651972, -194.70733086, -188.66725567,
-184.85280644,
        -182.40168022, -179.49991621, -177.32912211, -176.33222828,
        -175.47641539, -175.102098  ]),
 'split2_test_score': array([-199.88040273, -180.40547265, -170.40818758,
-166.03145117,
        -161.54130533, -159.21092802, -158.0921955 , -156.47943976,
        -155.29477167, -154.37233707]),
 'split3_test_score': array([-175.54706393, -162.54175935, -155.71632119,
-153.42761938,
        -150.5313467 , -148.89215254, -147.42731487, -146.68962872,
        -146.1447358 , -145.89571643]),
 'split4_test_score': array([-222.80227962, -198.4025695 , -189.74599407,
-185.02092317,
        -181.58209102, -179.79721396, -177.56214694, -175.56506911,
        -173.34374717, -171.51749709]),
 'mean_test_score': array([-204.30217036, -187.70463797, -180.02290904, -1
76.1704216 ,
        -172.56902298, -169.95078559, -168.02581965, -166.55872321,
        -165.2966293 , -164.36269762]),
 'std_test_score': array([16.45807737, 14.6139059 , 14.79485927, 14.208006
67, 14.05265279,
```

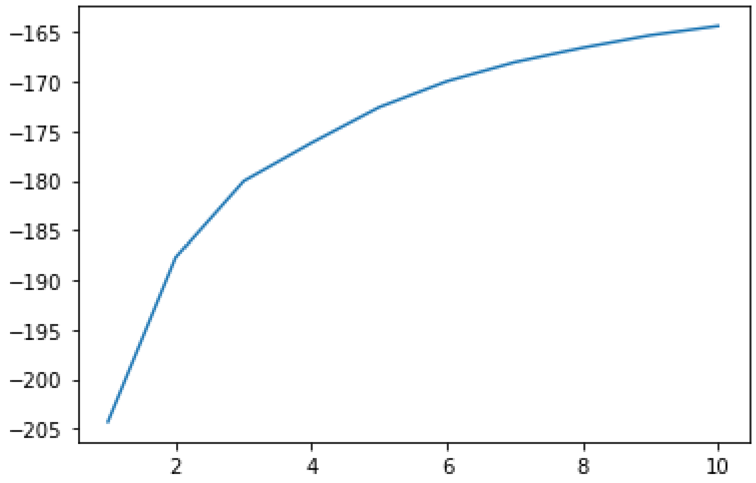
```
13.42223026, 12.93972797, 12.63126702, 12.28505044, 11.9911846 ]),
```

In [19]:

```
plt.plot(n_range1_2, clf_gs1.cv_results_['mean_test_score'])
```

Out[19]:

```
[<matplotlib.lines.Line2D at 0x274d58c0e80>]
```



In [20]:

```
# 5.Обучение модели и оценка качества с учетом подобранных гиперпараметров
clf_gs1.best_estimator_.fit(data_train, data_y_train)
target1_0 = clf_gs1.best_estimator_.predict(data_train)
target1_1 = clf_gs1.best_estimator_.predict(data_test)
```

Сравнение оценок метрик исходной и оптимальной моделей

In [21]:

```
# Новое качество модели
r2_score(data_y_train, target1_0), r2_score(data_y_test, target1_1)
```

Out[21]:

```
(0.9245930167529521, 0.9024037307965285)
```

In [22]:

```
# Качество модели до подбора гиперпараметров
r2_score(data_y_train, target0_0), r2_score(data_y_test, target0_1)
```

Out[22]:

```
(0.9408834310923744, 0.9054334392402374)
```

In [23]:

```
# Новое качество модели
mean_squared_error(data_y_train, target1_0, squared=False), mean_squared_error(data_y_test,
```

Out[23]:

```
(86.83082113323461, 98.39770517513665)
```

In [24]:

```
# Качество модели до подбора гиперпараметров
mean_squared_error(data_y_train, target0_0, squared=False), mean_squared_error(data_y_test,
```

Out[24]:

```
(76.88165695362724, 96.85837067132742)
```

In [25]:

```
n_range2_2 = np.array(range(1,11,1))
tuned_parameters2_2 = [{'n_neighbors': n_range2_2}]
```

In [26]:

```
%%time
clf_gs2 = GridSearchCV(KNeighborsRegressor(), tuned_parameters2_2, cv=ShuffleSplit(), scori
clf_gs2.fit(data[['Time(Seconds)', 'Temperature', 'Pressure', 'Humidity', 'WindDirection(De
```

```
Wall time: 27.3 s
```

Out[26]:

```
GridSearchCV(cv=ShuffleSplit(n_splits=10, random_state=None, test_size=None,
train_size=None),
             estimator=KNeighborsRegressor(),
             param_grid=[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,
8,  9, 10])}],
             scoring='r2')
```

In [27]:

```
clf_gs2.best_params_
```

Out[27]:

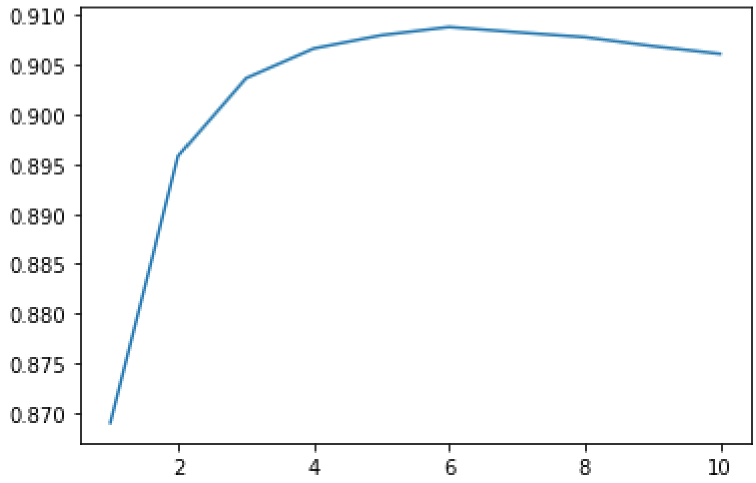
```
{'n_neighbors': 6}
```

In [28]:

```
plt.plot(n_range2_2, clf_gs2.cv_results_['mean_test_score'])
```

Out[28]:

```
[<matplotlib.lines.Line2D at 0x274d59db190>]
```



In [29]:

```
# 5.Обучение модели и оценка качества с учетом подобранных гиперпараметров
clf_gs2.best_estimator_.fit(data_train, data_y_train)
target2_0 = clf_gs2.best_estimator_.predict(data_train)
target2_1 = clf_gs2.best_estimator_.predict(data_test)
```

Сравнение оценок метрик исходной и оптимальной моделей

In [30]:

```
# Новое качество модели
r2_score(data_y_train, target2_0), r2_score(data_y_test, target2_1)
```

Out[30]:

```
(0.9369906171393381, 0.9068410332125085)
```

In [31]:

```
# Качество модели до подбора гиперпараметров
r2_score(data_y_train, target0_0), r2_score(data_y_test, target0_1)
```

Out[31]:

```
(0.9408834310923744, 0.9054334392402374)
```

Построение кривых обучения и валидации

In [32]:

```
from sklearn.model_selection import learning_curve, validation_curve
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    """
    Generate a simple plot of the test and training learning curve.

    Parameters
    ----------
    estimator : object type that implements the "fit" and "predict" methods
        An object of that type which is cloned for each validation.

    title : string
        Title for the chart.

    X : array-like, shape (n_samples, n_features)
        Training vector, where n_samples is the number of samples and
        n_features is the number of features.

    y : array-like, shape (n_samples) or (n_samples, n_features), optional
        Target relative to X for classification or regression;
        None for unsupervised learning.

    ylim : tuple, shape (ymin, ymax), optional
        Defines minimum and maximum yvalues plotted.

    cv : int, cross-validation generator or an iterable, optional
        Determines the cross-validation splitting strategy.
        Possible inputs for cv are:
          - None, to use the default 3-fold cross-validation,
          - integer, to specify the number of folds.
          - :term:`CV splitter`,
          - An iterable yielding (train, test) splits as arrays of indices.

        For integer/None inputs, if ``y`` is binary or multiclass,
        :class:`StratifiedKFold` used. If the estimator is not a classifier
        or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

        Refer :ref:`User Guide <cross_validation>` for the various
        cross-validators that can be used here.

    n_jobs : int or None, optional (default=None)
        Number of jobs to run in parallel.
        ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
        ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
        for more details.

    train_sizes : array-like, shape (n_ticks,), dtype float or int
        Relative or absolute numbers of training examples that will be used to
        generate the learning curve. If the dtype is float, it is regarded as a
        fraction of the maximum size of the training set (that is determined
        by the selected validation method), i.e. it has to be within (0, 1].
        Otherwise it is interpreted as absolute sizes of the training sets.
        Note that for classification the number of samples usually have to
        be big enough to contain at least one sample from each class.
        (default: np.linspace(0.1, 1.0, 5))
    """
    plt.figure()
    plt.title(title)
    if ylim is not None:
```

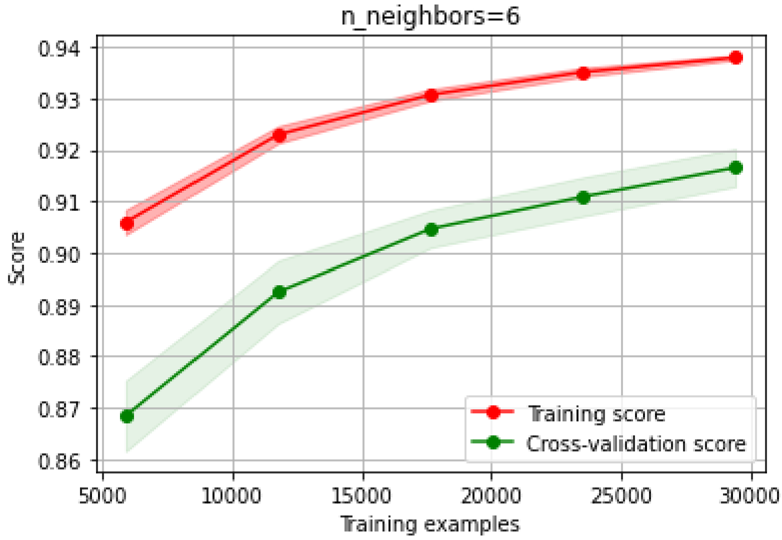
```python
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = 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

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
```

In [33]:

```
# 6.Построение кривых обучения и валидации
plot_learning_curve(clf_gs2.best_estimator_, 'n_neighbors=6',
                    data[['Time(Seconds)', 'Temperature', 'Pressure', 'Humidity', 'WindDire
```

Out[33]:

```
<module 'matplotlib.pyplot' from 'D:\\ProgramData\\Anaconda3\\lib\\site-pack
ages\\matplotlib\\pyplot.py'>
```



In [34]:

```
n_range2 = np.array(range(5,125,5))
plot_validation_curve(clf_gs2.best_estimator_, 'knn',
                      data[['Time(Seconds)', 'Temperature', 'Pressure', 'Humidity', 'WindDi
                      param_name='n_neighbors', param_range=n_range2,
                      cv=ShuffleSplit(), scoring="r2")
```

Out[34]:

```
<module 'matplotlib.pyplot' from 'D:\\ProgramData\\Anaconda3\\lib\\site-pack
ages\\matplotlib\\pyplot.py'>
```