Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Автоматизированные системы обработки информации и управления»

Отчет по лабораторной работе № 4

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

По курсу «Технологии машинного обучения»

ИСПОЛНИТЕЛЬ:

Коционова А. А.
Группа ИУ5-63

_____

"__"_____2019 г.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю. Е.

_____

"__"_____2019 г.

Москва 2019

Цель лабораторной работы.

Изучение сложных способов подготовки выборки и подбора гиперпараметров на примере метода ближайших соседей.

Практическая часть.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import learning_curve, validation_curve, Stratif
iedKFold

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

In [101]:

```
X = pd.read_csv('C:/Users/kotsi/Desktop/x_train.csv',delimiter=';')
Y = pd.read_csv('C:/Users/kotsi/Desktop/y_train.csv',delimiter=';',header=Non
e)
Y.columns=['Exit']
X=X.drop(['numberOfAttemptedLevels','totalBonusScore','totalStarsCount'], axi
s='columns')
```

Характеристики датасета

maxPlayerLevel - максимальный уровень игры, который прошел игрок
numberOfAttemptedLevels - количество уровней, которые попытался пройти игрок
attemptsOnTheHighestLevel - число попыток, сделанных на самом высоком уровне
totalNumOfAttempts - общее число попыток
averageNumOfTurnsPerCompletedLevel - среднее количество ходов, выполненных на успешно пройденных уровнях
doReturnOnLowerLevels - делал ли игрок возвраты к игре на уже пройденных уровнях
numberOfBoostersUsed - количество использованных бустеров
fractionOfUsefullBoosters - количество бустеров, использованных во время успешных попыток (игрок прошел уровнь)
totalScore - общее количество набранных очков
totalBonusScore - общее количество набранных бонусных очков
totalStarsCount - общее количество набранных звезд
numberOfDaysActuallyPlayed - количество дней, когда пользователь играл в игру

In [71]:

```
X.head()
```

Out[71]:

	maxPlayerLevel	attemptsOnTheHighestLevel	totalNumOfAttempts	averageNumOfTurnsPerCompletedLevel	doReturnOnLowerLevels	numberOfBoostersUsed	fractionOfUsefullBoosters	totalScore	numberOfDaysActuallyPlayed
0	39	3	17	24.444444	1	5	0.400000	2650000	2
1	21	19	55	17.045455	1	6	0.333333	5614000	4
2	5	1	6	8.400000	0	1	1.000000	857000	1
3	21	5	6	19.000000	0	1	0.000000	120000	1
4	4	1	5	9.600000	0	1	1.000000	857000	1

In [72]:

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25289 entries, 0 to 25288
Data columns (total 9 columns):
maxPlayerLevel                          25289 non-null int64
attemptsOnTheHighestLevel               25289 non-null int64
totalNumOfAttempts                      25289 non-null int64
averageNumOfTurnsPerCompletedLevel      25289 non-null float64
doReturnOnLowerLevels                   25289 non-null int64
numberOfBoostersUsed                    25289 non-null int64
fractionOfUsefullBoosters               25289 non-null float64
totalScore                              25289 non-null int64
numberOfDaysActuallyPlayed              25289 non-null int64
dtypes: float64(2), int64(7)
memory usage: 1.7 MB
```

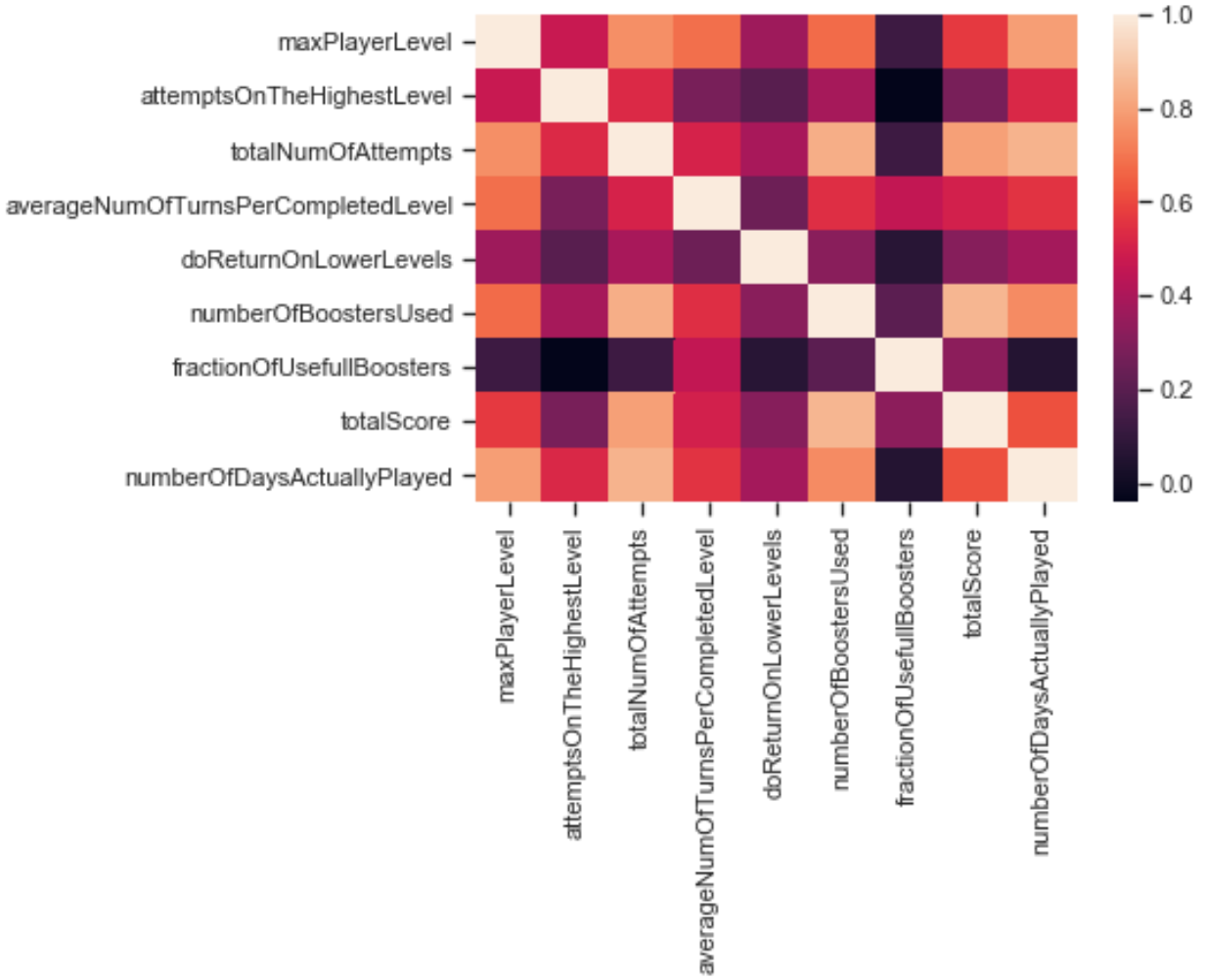
Все параметры числовые, пустых значений нет

In [25]:

```
sns.heatmap(X.corr())
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1f0cde69390>
```



In [26]:

```
X.corr()
```

Out[26]:

	maxPlayerLevel	attemptsOnTheHighestLevel	totalNumOfAttempts	averageNumOfTurnsPerCompletedLevel	doReturnOnLowerLevels	numberOfBoostersUsed	fractionOfUsefullBoosters	totalScore	numberOfDaysActuallyPlayed
maxPlayerLevel	1.000000	0.472142	0.757854	0.683706	0.368297	0.675955	0.126235	0.570234	0.793385
attemptsOnTheHighestLevel	0.472142	1.000000	0.532032	0.277072	0.197035	0.389465	-0.041700	0.277326	0.524109
totalNumOfAttempts	0.757854	0.532032	1.000000	0.509510	0.391969	0.836706	0.128843	0.798051	0.846448

	maxPlayerLevel	attemptsOnTheHighestLevel	totalNumOfAttempts	averageNumOfTurnsPerCompletedLevel	doReturnOnLowerLevels	numberOfBoostersUsed	fractionOfUsefullBoosters	totalScore	numberOfDaysActuallyPlayed
averageNumOfTurnsPerCompletedLevel	0.683706	0.277072	0.509510	1.000000	0.250261	0.543847	0.457168	0.499681	0.555594
doReturnOnLowerLevels	0.368297	0.197035	0.391969	0.250261	1.000000	0.320439	0.071646	0.310218	0.383024
numberOfBoostersUsed	0.675955	0.389465	0.836706	0.543847	0.320439	1.000000	0.203519	0.854263	0.751712
fractionOfUsefullBoosters	0.126235	-0.041700	0.128843	0.457168	0.071646	0.203519	1.000000	0.328287	0.058929
totalScore	0.570234	0.277326	0.798051	0.499681	0.310218	0.854263	0.328287	1.000000	0.617847
numberOfDaysActuallyPlayed	0.793385	0.524109	0.846448	0.555594	0.383024	0.751712	0.058929	0.617847	1.000000

Нормализуем

In [102]:

```
X = (X - X.mean()) / X.std()
X.corr()
```

Out[102]:

	maxPlayerLevel	attemptsOnTheHighestLevel	totalNumOfAttempts	averageNumOfTurnsPerCompletedLevel	doReturnOnLowerLevels	numberOfBoostersUsed	fractionOfUsefullBoosters	totalScore	numberOfDaysActuallyPlayed
maxPlayerLevel	1.000000	0.472142	0.757854	0.683706	0.368297	0.675955	0.126235	0.570234	0.793385
attemptsOnTheHighestLevel	0.472142	1.000000	0.532032	0.277072	0.197035	0.389465	-0.041700	0.277	0.524109

	maxPlayerLevel	attemptsOnTheHighestLevel	totalNumOfAttempts	averageNumOfTurnsPerCompletedLevel	doReturnOnLowerLevels	numberOfBoostersUsed	fractionOfUsefullBoosters	totalScore	numberOfDaysActuallyPlayed
totalNumOfAttempts	0.757854	0.532032	1.000000	0.509510	0.391969	0.836706	0.128843	0.798051	0.846448
averageNumOfTurnsPerCompletedLevel	0.683706	0.277072	0.509510	1.000000	0.250261	0.543847	0.457168	0.499681	0.555594
doReturnOnLowerLevels	0.368297	0.197035	0.391969	0.250261	1.000000	0.320439	0.071646	0.310218	0.383024
numberOfBoostersUsed	0.675955	0.389465	0.836706	0.543847	0.320439	1.000000	0.203519	0.854263	0.751712
fractionOfUsefullBoosters	0.126235	-0.041700	0.128843	0.457168	0.071646	0.203519	1.000000	0.328287	0.058929
totalScore	0.570234	0.277326	0.798051	0.499681	0.310218	0.854263	0.328287	1.000000	0.617847
numberOfDaysActuallyPlayed	0.793385	0.524109	0.846448	0.555594	0.383024	0.751712	0.058929	0.617847	1.000000

In [103]:

```
Y=Y['Exit']
```

In [104]:

```
Y.shape
```

Out[104]:

```
(25289,)
```

In [105]:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, ra
ndom_state = 11)
```

In [106]:

```
y_train.shape
```

Out[106]:

```
(20231,)
```

In [107]:

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

Out[107]:

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

In [108]:

```
y_train_predict = knn.predict(X_train)
y_test_predict = knn.predict(X_test)
```

In [109]:

```
print(accuracy_score(y_test, y_test_predict), precision_score(y_test, y_test_
predict), recall_score(y_test, y_test_predict))
```

```
0.8011071569790431 0.6604057099924868 0.6133984647592463
```

In [110]:

```
'''n_neighbors_array = [150,200,250]
knn = KNeighborsClassifier()
grid = GridSearchCV(estimator=knn, param_grid={'n_neighbors': n_neighbors_arr
ay},cv=3)
grid.fit(X_train, y_train)

best_cv_err = 1 - grid.best_score_
best_n_neighbors = grid.best_estimator_.n_neighbors

print(best_cv_err, best_n_neighbors)'''
```

Out[110]:

```
"n_neighbors_array = [150,200,250]\nknn = KNeighborsClassifier()\ngrid = Grid
SearchCV(estimator=knn, param_grid={'n_neighbors': n_neighbors_array},cv=3)\n
grid.fit(X_train, y_train)\n\nbest_cv_err = 1 - grid.best_score_\nbest_n_neig
hbors = grid.best_estimator_.n_neighbors\n\nprint(best_cv_err, best_n_neighbo
rs)"
```

In [111]:

```
knn = KNeighborsClassifier(n_neighbors=200)
knn.fit(X_train, y_train)

y_train_predict = knn.predict(X_train)
y_test_predict = knn.predict(X_test)
```

In [112]:

```
print(accuracy_score(y_test, y_test_predict), precision_score(y_test, y_test_
predict), recall_score(y_test, y_test_predict))
```

```
0.8238434163701067 0.7185483870967742 0.6217725052337753
```

In [113]:

```
##
```

In [114]:

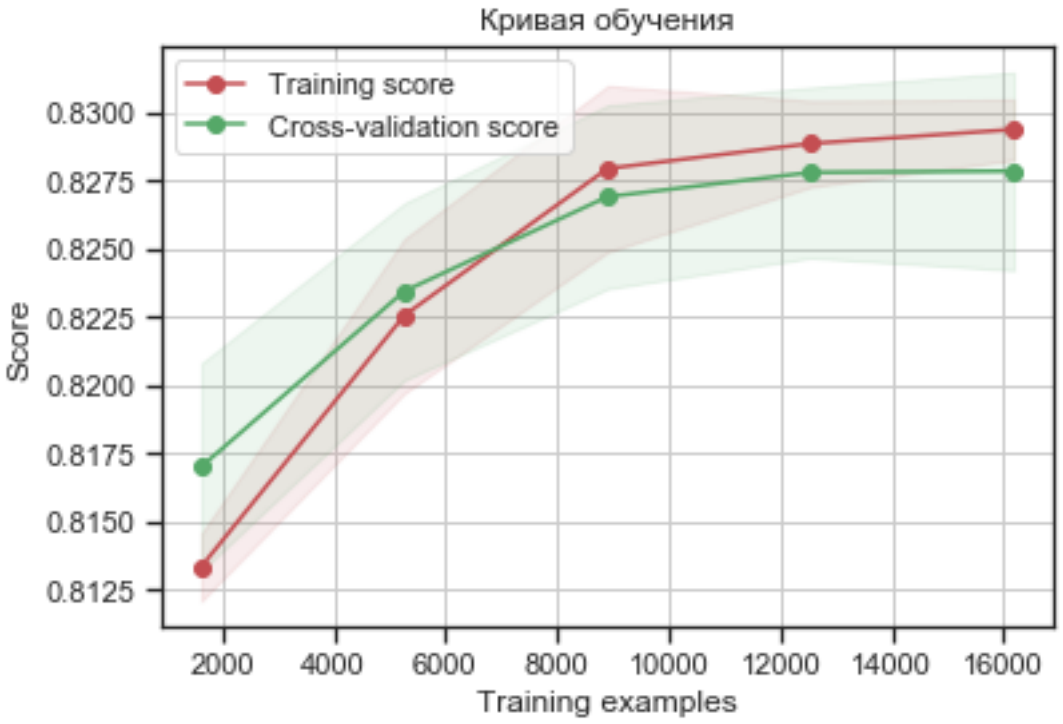
```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title("Кривая обучения")
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = 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
```

In [115]:

```
plot_learning_curve(KNeighborsClassifier(n_neighbors=200), 'n_neighbors=200',
                    X_train, y_train,
                    cv=StratifiedKFold(n_splits=5))
```

Out[115]:

```
<module 'matplotlib.pyplot' from 'C:\\Users\\kotsi\\Anaconda37\\lib\\site-pac
kages\\matplotlib\\pyplot.py'>
```



In [116]:

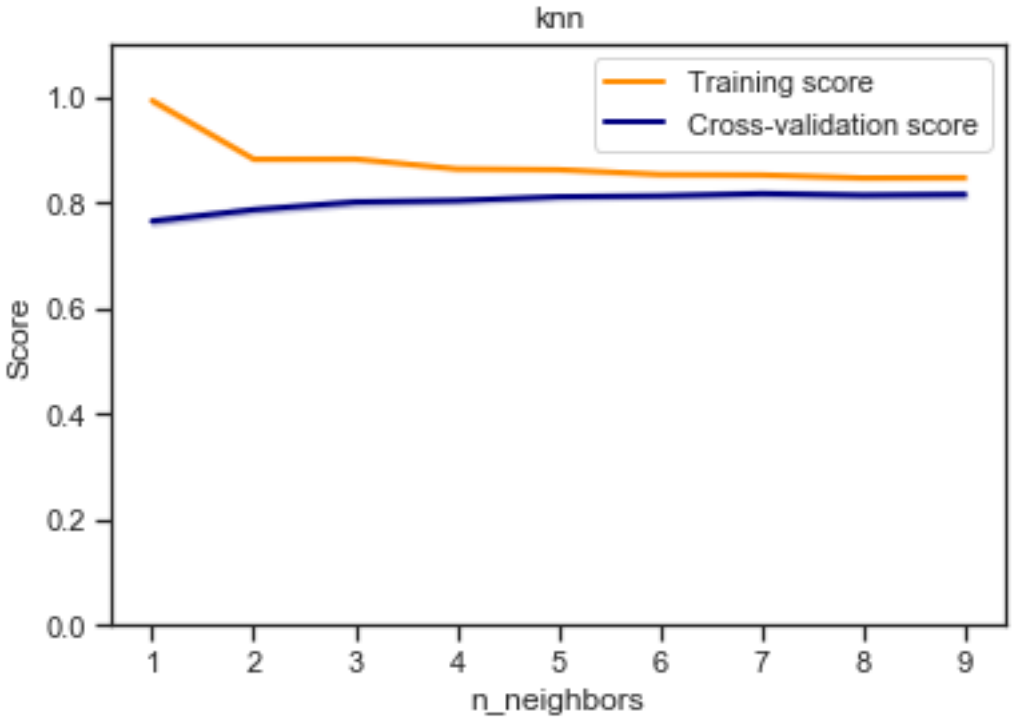
```
def plot_validation_curve(estimator, title, X, y,
                          param_name, param_range, cv,
                          scoring="accuracy"):

    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
```

In [122]:

```
plot_validation_curve(KNeighborsClassifier(n_neighbors=200), 'knn',
                      X_train, y_train,
                      param_name='n_neighbors', param_range=np.array(range(1,
10,1)),
                      cv=StratifiedKFold(n_splits=5), scoring="accuracy")
```

Out[122]:

```
<module 'matplotlib.pyplot' from 'C:\\Users\\kotsi\\Anaconda37\\lib\\site-pac
kages\\matplotlib\\pyplot.py'>
```



In [ ]: