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, StratifiedKFold

**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=**None**)

Y.columns=['Exit']

X=X.drop(['numberOfAttemptedLevels','totalBonusScore','totalStarsCount'], axis='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 |
| **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 | 0.277326 | 0.524109 |
| **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, random\_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\_array},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 = GridSearchCV(estimator=knn, param\_grid={'n\_neighbors': n\_neighbors\_array},cv=3)\ngrid.fit(X\_train, y\_train)\n\nbest\_cv\_err = 1 - grid.best\_score\_\nbest\_n\_neighbors = grid.best\_estimator\_.n\_neighbors\n\nprint(best\_cv\_err, best\_n\_neighbors)"

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]:

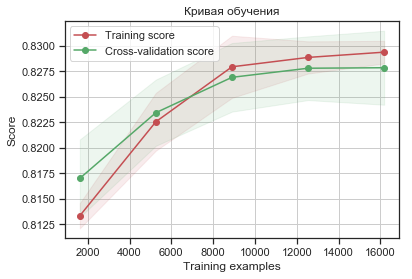
plot\_learning\_curve(KNeighborsClassifier(n\_neighbors=200), 'n\_neighbors=200',

X\_train, y\_train,

cv=StratifiedKFold(n\_splits=5))

Out[115]:

<module 'matplotlib.pyplot' from 'C:\\Users\\kotsi\\Anaconda37\\lib\\site-packages\\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',

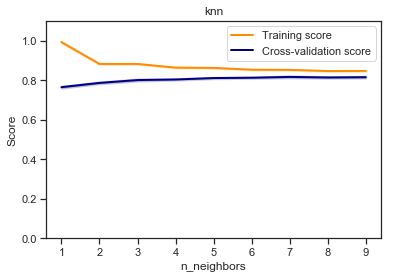
X\_train, y\_train,

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-packages\\matplotlib\\pyplot.py'>



In [ ]: