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**

**по курсу «Технологии машинного обучения»**

**«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на**

**примере метода ближайших соседей.»**

**ИСПОЛНИТЕЛЬ:**

Мехмандаров Мурад

Группа ИУ5-63

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ "\_\_"\_\_\_\_\_\_\_\_\_\_\_2020 г.

**ПРЕПОДАВАТЕЛЬ:**

Гапанюк Ю.Е.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ "\_\_"\_\_\_\_\_\_\_\_\_\_\_2020 г.

Москва 2020

# Цель лабораторной работы

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

# Задание

1. Выбрать набор данных (датасет) для решения задачи классификации или регрессии.
2. С использованием метода train\_test\_split разделить выборку на обучающую и тестовую.
3. Обучить модель k-ближайших соседей для произвольно заданного гиперпараметра K. Оценить качество модели с помощью подходящих для задачи метрик.
4. Построить модель и оценить качество модели с использованием кросс-валидации.
5. Произвести подбор гиперпараметра K с использованием GridSearchCV и кросс-валидации.

# Ход выполнения лабораторной работы

Подключим необходимые библиотеки и загрузим набор данных

[42]:

Out[42]:

**import**

pandas

**as**

pd

**import**

seaborn

**as**

sns

**import**

numpy

**as**

np

**import**

matplotlib

.

pyplot

**as**

plt

**from**

sklearn

.

preprocessing

**import**

StandardScaler

,

LabelEncoder

**from**

sklearn

.

model\_selection

**import**

train\_test\_split

,

cross\_val\_score

,

GridSearchCV

**from**

sklearn

.

neighbors

**import**

KNeighborsClassifier

**from**

sklearn

.

metrics

**import**

accuracy\_score

**%**

matplotlib

inline

*#*

*Устанавливаем*

*тип*

*графиков*

sns

.

set

(

style

**=**

"ticks"

)

*#*

*Для*

*лучшего*

*качествоа*

*графиков*

**from**

IPython

.

display

**import**

set\_matplotlib\_formats

set\_matplotlib\_formats

(

"retina"

)

*#*

*Устанавливаем*

*ширину*

*экрана*

*для*

*отчета*

pd

.

set\_option

(

"display.width"

,

70

)

*#*

*Загружаем*

*данные*

data

**=**

pd

.

read\_csv

(

'games.csv'

)

data

.

head

()

**Name Platform Year\_of\_Release Genre Publisher NA\_Sales EU\_Sales JP\_Sales**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Wii Sports | Wii | 2006.0 | Sports | Nintendo | 41.36 | 28.96 | 3.77 |
| **1** | Super Mario Bros. | NES | 1985.0 | Platform | Nintendo | 29.08 | 3.58 | 6.81 |
| **2** | Mario Kart Wii | Wii | 2008.0 | Racing | Nintendo | 15.68 | 12.76 | 3.79 |
| **3** | Wii Sports Resort | Wii | 2009.0 | Sports | Nintendo | 15.61 | 10.93 | 3.28 |
| **4** | Pokemon Red/Pokemon | GB | 1996.0 | RolePlaying | Nintendo | 11.27 | 8.89 | 10.22 |

Blue

In [43]:

data

.

shape

Out[43]:

(16719, 16)

[44]:

data

.

dtypes

Out[44]:

Name object

Platform object

Year\_of\_Release float64

Genre object

Publisher object

NA\_Sales float64

EU\_Sales float64

JP\_Sales float64

Other\_Sales float64

Global\_Sales float64

Critic\_Score float64

Critic\_Count float64

User\_Score object

User\_Count float64

Developer object Rating object dtype: object In [45]:

data

**=**

data

.

dropna

()

data

.

isna

().

sum

(

)

Out[45]:

Name 0

Platform 0

Year\_of\_Release 0

Genre 0

Publisher 0

NA\_Sales 0

EU\_Sales 0

JP\_Sales 0

Other\_Sales 0

Global\_Sales 0

Critic\_Score 0

Critic\_Count 0

User\_Score 0

User\_Count 0

Developer 0 Rating 0 dtype: int64

[46]:

data

.

isnull

().

sum

(

)

Out[46]:

Name 0

Platform 0

Year\_of\_Release 0

Genre 0

Publisher 0

NA\_Sales 0

EU\_Sales 0

JP\_Sales 0

Other\_Sales 0

Global\_Sales 0

Critic\_Score 0

Critic\_Count 0

User\_Score 0

User\_Count 0

Developer 0 Rating 0 dtype: int64 In [47]:

data

.

shape

Out[47]:

(6825, 16)

Как видим, пустых значений больше нет, значет нет необходимости преобразовывать набор данных Разделим данные на целевой столбец и признаки

In [ ]:

Но сначала сделаем, кодирование категориальных признаков.

In [48]:

le1

**=**

LabelEncoder

()

data

[

'Name'

]

**=**

le1

.

fit\_transform

(

data

[

'Name'

])

;

In [49]:

le2

**=**

LabelEncoder

()

data

[

'Platform'

]

**=**

le2

.

fit\_transform

(

data

[

'Platform'

])

;

In [50]:

le3

**=**

LabelEncoder

()

data

[

'Genre'

]

**=**

le3

.

fit\_transform

(

data

[

'Genre'

])

;

[51]:

le4

**=**

LabelEncoder

()

data

[

'Publisher'

]

**=**

le4

.

fit\_transform

(

data

[

'Publisher'

])

;

In [52]:

le5

**=**

LabelEncoder

()

data

[

'Developer'

]

**=**

le5

.

fit\_transform

(

data

[

'Developer'

])

;

In [53]:

le6

**=**

LabelEncoder

()

data

[

'Rating'

]

**=**

le6

.

fit\_transform

(

data

[

'Rating'

])

;

[54]:

X

**=**

data

.

drop

(

"Name"

,

axis

**=**

1

)

Y

**=**

data

[

"Name"

]

print

(

X

,

"\n"

)

print

(

Y

)

Platform Year\_of\_Release Genre Publisher NA\_Sales \

0 12 2006.0 10 160 41.36

1. 12 2008.0 6 160 15.68
2. 12 2009.0 10 160 15.61
3. 2 2006.0 4 160 11.28
4. 12 2006.0 3 160 13.96 ... ... ... ... ... ...

16667 3 2001.0 0 158 0.01

16677 3 2002.0 2 146 0.01

16696 5 2014.0 0 123 0.00

16700 5 2011.0 8 53 0.01 16706 5 2011.0 11 240 0.00

EU\_Sales JP\_Sales Other\_Sales Global\_Sales Critic\_Score \

0 28.96 3.77 8.45 82.53 76.0

1. 12.76 3.79 3.29 35.52 82.0
2. 10.93 3.28 2.95 32.77 80.0
3. 9.14 6.50 2.88 29.80 89.0
4. 9.18 2.93 2.84 28.92 58.0 ... ... ... ... ... ...

16667 0.00 0.00 0.00 0.01 46.0

16677 0.00 0.00 0.00 0.01 81.0

16696 0.01 0.00 0.00 0.01 80.0

16700 0.00 0.00 0.00 0.01 61.0 16706 0.01 0.00 0.00 0.01 60.0

Critic\_Count User\_Score User\_Count Developer Rating

0 51.0 8 322.0 771 1

1. 73.0 8.3 709.0 771 1
2. 73.0 8 192.0 771 1
3. 65.0 8.5 431.0 771 1
4. 41.0 6.6 129.0 771 1 ... ... ... ... ... ...

16667 4.0 2.4 21.0 411 1

16677 12.0 8.8 9.0 260 4

16696 20.0 7.6 412.0 617 4

16700 12.0 5.8 43.0 99 6

16706 12.0 7.2 13.0 1011 2

[6825 rows x 15 columns]

0 4215

1. 2087
2. 4217
3. 2567
4. 4213 ...

16667 1048

16677 2297

16696 2194

16700 407

16706 3094

Name: Name, Length: 6825, dtype: int64

[55]:

X

.

shape

Out[55]:

(6825, 15) In [56]:

Y

.

shape

Out[56]:

(6825,)

**С использованием метода train\_test\_split разделим выборку на обучающую и тестовую**

In [57]:

X\_train, X\_test, Y\_train, Y\_test **=** train\_test\_split(X, Y, test\_size**=**0.25, random\_sta In [58]:

X\_train: (5118, 15)

X\_test: (1707, 15)

print

(

"X\_train:"

,

X\_train

.

shape

)

print

(

"X\_test:"

,

X\_test

.

shape

)

print

(

"Y\_train:"

,

Y\_train

.

shape

)

print

(

"Y\_test:"

,

Y\_test

.

shape

)

Y\_train: (5118,)

Y\_test: (1707,)

**Обучим модель k-ближайших соседей для произвольно заданного гиперпараметра K**

In [59]:

*# В моделях k-ближайших соседей большое значение k*

*# ведёт к большому смещению и низкой дисперсии (недообучению)*

*# 70 ближайших соседей*

cl1\_1 **=** KNeighborsClassifier(n\_neighbors**=**70) cl1\_1.fit(X\_train, Y\_train) target1\_0 **=** cl1\_1.predict(X\_train) target1\_1 **=** cl1\_1.predict(X\_test) accuracy\_score(Y\_train, target1\_0), accuracy\_score(Y\_test, target1\_1)

Out[59]:

(0.05744431418522861, 0.008201523140011716)

**Построим модель и оценим качество модели с использованием кросс-валидации**

In [77]:

scores **=** cross\_val\_score(KNeighborsClassifier(n\_neighbors**=**2), X, Y, cv**=**3)

[64]:

*#*

*Значение*

*метрики*

*accuracy*

*для*

*3*

*фолдов*

scores

Out[64]:

array([0.09230769, 0.10417582, 0.08483516]) In [65]:

*#*

*Усредненное*

*значение*

*метрики*

*accuracy*

*для*

*3*

*фолдов*

np

.

mean

(

scores

)

Out[65]:

0.09377289377289377

**Произведем подбор гиперпараметра K с использованием GridSearchCV и кросс-валидации**

In

[74]:

*#*

*Список*

*настраиваемых*

*параметров*

n\_range

**=**

np

.

array

(

range

(

1

,

50

,

2

))

tuned\_parameters

**=**

[{

'n\_neighbors'

:

n\_range

}]

n\_range

Out[74]:

array([ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33,

35, 37, 39, 41, 43, 45, 47, 49])

In

[72]:

**%%**

time

clf\_gs

**=**

GridSearchCV

(

KNeighborsClassifier

()

,

tuned\_parameters

,

cv

**=**

5

,

scoring

**=**

'accur

clf\_gs

.

fit

(

X

,

Y

)

clf\_gs

.

best\_params\_

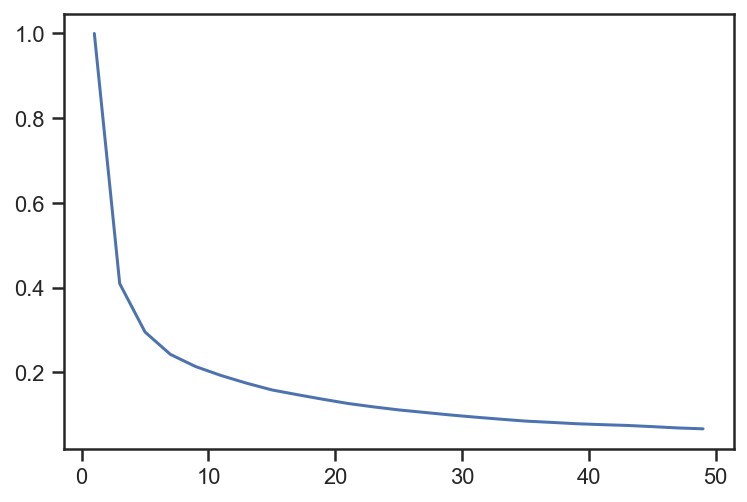
CPU times: user 32.8 s, sys: 386 ms, total: 33.1 s

Wall time: 33.2 s Out[72]:

{'n\_neighbors': 1}

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

[75]:



plt

.

plot

(

n\_range

,

clf\_gs

.

cv\_results\_

[

"mean\_train\_score"

])

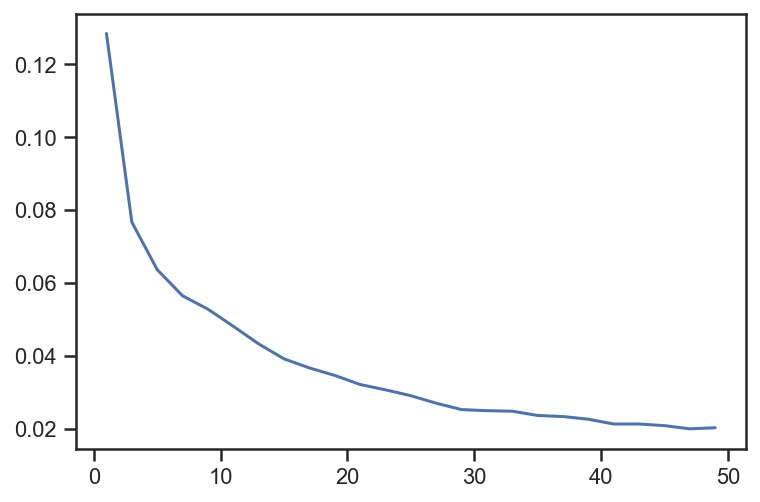
;

Очевидно, что для 𝐾 =1 на тренировочном наборе данных мы находим ровно ту же точку, что и нужно предсказать, и чем больше её соседей мы берём — тем меньше точность.

Посмотрим на тестовом наборе данных

In

[78]:



plt

.

plot

(

n\_range

,

clf\_gs

.

cv\_results\_

[

"mean\_test\_score"

])

;

Проверим получившуюся модель:

[79]:

cl1\_2 **=** KNeighborsClassifier(**\*\***clf\_gs.best\_params\_) cl1\_2.fit(X\_train, Y\_train) target2\_0 **=** cl1\_2.predict(X\_train) target2\_1 **=** cl1\_2.predict(X\_test) accuracy\_score(Y\_train, target2\_0), accuracy\_score(Y\_test, target2\_1) Out[79]:

(1.0, 0.09607498535442296)

Как видим, точность модели улучшилось