Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс   
«Технологии машинного обучения»  
 Отчет по лабораторной работе №3

|  |  |
| --- | --- |
| Выполнил: | Проверил: |
| студент группы ИУ5-63Б | преподаватель каф. ИУ5 |
| Тарновский Д.Р. | Гапанюк Ю.Е. |

Москва, 2022 г.

In [1]:

*#Датасет содержит данные о кредитах на покупку электроники, которые были*

**import** pandas **as** pd

**import** numpy **as** np

**from** sklearn.model\_selection **import** train\_test\_split, GridSearchCV, Rand

**from** sklearn.neighbors **import** KNeighborsClassifier

**from** sklearn.preprocessing **import** MinMaxScaler, StandardScaler

**from** matplotlib **import** pyplot **as** plt

**import** seaborn **as** sns

**from** sklearn.metrics **import** accuracy\_score, precision\_score, recall\_scor

**from** warnings **import** simplefilter

simplefilter('ignore')

In [2]:

*# записываем CSV-файл в объект DataFrame*

data **=** pd**.**read\_csv('credit\_train.csv', encoding**=**'cp1251', sep**=**';')

In [3]:

*# смотрим на первые пять строк*

data**.**head()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Out[3]: | **client\_id** | **gender** | **age** | **marital\_status** | **job\_position** | **credit\_sum** | **credit\_month** | **tariff\_id** |
|  | **0** 1 | M | NaN | NaN | UMN | 59998,00 | 10 | 1.6 |
|  | **1** 2 | F | NaN | MAR | UMN | 10889,00 | 6 | 1.1 |
|  | **2** 3 | M | 32.0 | MAR | SPC | 10728,00 | 12 | 1.1 |
|  | **3** 4 | F | 27.0 | NaN | SPC | 12009,09 | 12 | 1.1 |
|  | **4** 5 | M | 45.0 | NaN | SPC | NaN | 10 | 1.1 |

In [4]:

# Обработка пропусков в данных

<class 'pandas.core.frame.DataFrame'> RangeIndex: 170746 entries, 0 to 170745 Data columns (total 15 columns):

*#проверяем типы данных и заполненность столбцов*

data**.**info()

# Column Non-Null Count Dtype

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 |  | client\_id | 170746 | non-null | int64 |
| 1 |  | gender | 170746 | non-null | object |
| 2 |  | age | 170743 | non-null | float64 |

|  |  |  |  |
| --- | --- | --- | --- |
| 3 marital\_status | 170743 | non-null | object |
| 4 job\_position | 170746 | non-null | object |
| 5 credit\_sum | 170744 | non-null | object |
| 6 credit\_month | 170746 | non-null | int64 |
| 7 tariff\_id | 170746 | non-null | float64 |
| 8 score\_shk | 170739 | non-null | object |
| 9 education | 170741 | non-null | object |
| 10 living\_region | 170554 | non-null | object |
| 11 monthly\_income | 170741 | non-null | float64 |
| 12 credit\_count | 161516 | non-null | float64 |
| 13 overdue\_credit\_count | 161516 | non-null | float64 |
| 14 open\_account\_flg | 170746 | non-null | int64 |

In [5]:

*#удаляем столбец с номером клиента (так как он незначимый)*

*# и с регионом проживания (так как он нуждается в серьезной предобработк*

data**.**drop(['client\_id', 'living\_region'], axis**=**1, inplace**=True**)

dtypes: float64(5), int64(3), object(7) memory usage: 19.5+ MB

In [6]:

*# анализируем столбец marital\_status, смотрим, какое значение в нем явля*

data['marital\_status']**.**describe()

Out[6]:

In [7]:

count 170743

unique 5

top MAR

freq 93954

Name: marital\_status, dtype: object

*# анализируем столбец education, смотрим, какое в нем самое частое значе*

data['education']**.**describe()

Out[7]:

In [8]:

count 170741

unique 5

top SCH

freq 87537

Name: education, dtype: object

*# дозаполняем нечисловые столбцы с пропусками самыми часто встречающимис* data['marital\_status']**.**fillna('MAR', inplace**=True**) data['education']**.**fillna('SCH', inplace**=True**)

In [9]:

*# дозаполняем числовые столбцы с пропусками медианными значениями* data['age']**.**fillna(data['age']**.**median(), inplace**=True**) data['credit\_count']**.**fillna(data['credit\_count']**.**median(), inplace**=True**) data['overdue\_credit\_count']**.**fillna(data['overdue\_credit\_count']**.**median(

In [10]:

*#меняем в столбцах 'credit\_sum', 'score\_shk' запятые на точки и преобр*

**for** i **in** ['credit\_sum', 'score\_shk']:

data[i] **=** data[i]**.**str**.**replace(',', '.')**.**astype('float')

In [11]:

*# дозаполняем ставшие теперь числовыми столбцы 'credit\_sum', 'score\_shk'* data['score\_shk']**.**fillna(data['score\_shk']**.**median(), inplace**=True**) data['monthly\_income']**.**fillna(data['monthly\_income']**.**median(), inplace**=T** data['credit\_sum']**.**fillna(data['credit\_sum']**.**median(), inplace**=True**)

In [12]:

*# смотрим, что получилось*

data**.**info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 170746 entries, 0 to 170745 Data columns (total 13 columns):

# Column Non-Null Count Dtype

In [13]:

category\_cols **=** ['gender', 'job\_position', 'education', 'marital\_status'

* 1. gender 170746 non-null object
  2. age 170746 non-null float64
  3. marital\_status 170746 non-null object
  4. job\_position 170746 non-null object
  5. credit\_sum 170746 non-null float64
  6. credit\_month 170746 non-null int64
  7. tariff\_id 170746 non-null float64
  8. score\_shk 170746 non-null float64
  9. education 170746 non-null object
  10. monthly\_income 170746 non-null float64
  11. credit\_count 170746 non-null float64
  12. overdue\_credit\_count 170746 non-null float64
  13. open\_account\_flg 170746 non-null int64 dtypes: float64(7), int64(2), object(4)

memory usage: 16.9+ MB

# Кодирование категориальных признаков

In [14]:

print("Количество уникальных значений\n")

**for** col **in** category\_cols:

print(f'{col}: {data[col]**.**unique()**.**size}')

Количество уникальных значений

In [15]:

gender: 2

job\_position: 18

education: 5

marital\_status: 5

*# кодируем нечисловые столбцы методом дамми-кодирования*

data **=** pd**.**concat([data,

pd**.**get\_dummies(data['gender'], prefix**=**"gender"), pd**.**get\_dummies(data['job\_position'], prefix**=**"job\_p pd**.**get\_dummies(data['education'], prefix**=**"educatio pd**.**get\_dummies(data['marital\_status'], prefix**=**"mar

axis**=**1)

In [16]:

*#удаляем старые нечисловые столбцы, вместо них уже появились новые число*

data**.**drop(['gender','job\_position','education','marital\_status'], axis**=**1

In [17]:

data**.**head()

Out[17]:

**age credit\_sum credit\_month tariff\_id score\_shk monthly\_income credit\_count over 0** 34.0 59998.00 10 1.6 0.461599 30000.0 1.0

**1** 34.0 10889.00 6 1.1 0.461599 35000.0 2.0

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **2** 32.0 | 10728.00 | 12 | 1.1 | 0.461599 | 35000.0 | 5.0 |
| **3** 27.0 | 12009.09 | 12 | 1.1 | 0.461599 | 35000.0 | 2.0 |
| **4** 45.0 | 21229.00 | 10 | 1.1 | 0.421385 | 35000.0 | 1.0 |

## 5 rows × 39 columns

In [18]:

data\_sample **=** data**.**sample(n**=**20000) y **=** data\_sample['open\_account\_flg']

X **=** data\_sample**.**drop('open\_account\_flg', axis**=**1)

x\_train, x\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=**0.5,

# Разделение выборки на обучающую и тестовую

In [19]:

scaler **=** MinMaxScaler()**.**fit(x\_train)

x\_train **=** pd**.**DataFrame(scaler**.**transform(x\_train), columns**=**x\_train**.**column x\_test **=** pd**.**DataFrame(scaler**.**transform(x\_test), columns**=**x\_train**.**columns) x\_train**.**describe()

# Масштабирование данных

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Out[19]: | **age** | **credit\_sum** | **credit\_month** | **tariff\_id** | **score\_shk** | **monthly\_inco** |
|  | **count** 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000000 | 10000.000 |
|  | **mean** 0.355171 | 0.117709 | 0.243318 | 0.337919 | 0.444368 | 0.058 |
|  | **std** 0.202820 | 0.083551 | 0.109055 | 0.248954 | 0.142031 | 0.041 |
|  | **min** 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000 |
|  | **25%** 0.192308 | 0.060659 | 0.212121 | 0.106383 | 0.342051 | 0.033 |
|  | **50%** 0.307692 | 0.092685 | 0.212121 | 0.319149 | 0.434301 | 0.050 |
|  | **75%** 0.480769 | 0.149042 | 0.272727 | 0.638298 | 0.539093 | 0.075 |
|  | **max** 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000 |

## 8 rows × 38 columns

In [20]:

**from** sklearn.metrics **import** mean\_squared\_error, r2\_score

**def** print\_metrics(y\_test, y\_pred):

print(f"Среднее квадратичное отклонение: {mean\_squared\_error(y\_test, print(f"Коэффициент детерминации: {r2\_score(y\_test, y\_pred)}")

**def** print\_cv\_result(cv\_model, x\_test, y\_test):

print(f'Оптимизация метрики {cv\_model**.**scoring}: {cv\_model**.**best\_score print(f'Лучший параметр: {cv\_model**.**best\_params\_}')

print('Метрики на тестовом наборе') print\_metrics(y\_test, cv\_model**.**predict(x\_test)) print()

# Обучение KNN с произвольным k

In [21]:

base\_k **=** 10

base\_knn **=** KNeighborsClassifier(n\_neighbors**=**base\_k) base\_knn**.**fit(x\_train, y\_train)

y\_pred\_base **=** base\_knn**.**predict(x\_test)

In [22]:

print(f'Test metrics for KNN with k={base\_k}\n') print\_metrics(y\_test, y\_pred\_base)

Test metrics for KNN with k=10

In [23]:

Среднее квадратичное отклонение: 0.42178193417926285 Коэффициент детерминации: -0.22889125410735645

# Кросс-валидация

Результаты кросс-валидации при cv=5

metrics **=** ['accuracy', 'recall', 'f1'] cv\_values **=** [5, 10]

**for** cv **in** cv\_values:

print(f'Результаты кросс-валидации при cv={cv}\n')

**for** metric **in** metrics:

params **=** {'n\_neighbors': range(1, 40)}

knn\_cv **=** RandomizedSearchCV(KNeighborsClassifier(), params, cv**=**c *#knn\_cv = GridSearchCV(KNeighborsClassifier(), params, cv=cv, sc* knn\_cv**.**fit(x\_train, y\_train)

print\_cv\_result(knn\_cv, x\_test, y\_test)

Оптимизация метрики accuracy: 0.821 Лучший параметр: {'n\_neighbors': 32} Метрики на тестовом наборе

Среднее квадратичное отклонение: 0.41809089920733744 Коэффициент детерминации: -0.2074771850363457

Оптимизация метрики recall: 0.0616529757970604 Лучший параметр: {'n\_neighbors': 4}

Метрики на тестовом наборе

Среднее квадратичное отклонение: 0.43231932642434573 Коэффициент детерминации: -0.29106113205545214

Оптимизация метрики f1: 0.12439668198395074 Лучший параметр: {'n\_neighbors': 2}

Метрики на тестовом наборе

Среднее квадратичное отклонение: 0.4393176527297759 Коэффициент детерминации: -0.3331984937758281

Результаты кросс-валидации при cv=10

Оптимизация метрики accuracy: 0.8215 Лучший параметр: {'n\_neighbors': 38} Метрики на тестовом наборе

Среднее квадратичное отклонение: 0.41844951905815353 Коэффициент детерминации: -0.20954951430128244

Оптимизация метрики recall: 0.07733663925679493 Лучший параметр: {'n\_neighbors': 7}

Метрики на тестовом наборе

Среднее квадратичное отклонение: 0.432781700167648 Коэффициент детерминации: -0.29382423774203414

Оптимизация метрики f1: 0.21024813755359556 Лучший параметр: {'n\_neighbors': 3}

Метрики на тестовом наборе

Среднее квадратичное отклонение: 0.46454278597347737 Коэффициент детерминации: -0.4906955179110035

In [24]:

**for** cv **in** cv\_values:

print(f'Результаты кросс-валидации при cv={cv}\n')

**for** metric **in** metrics:

params **=** {'n\_neighbors': range(1, 40)}

*#knn\_cv = RandomizedSearchCV(KNeighborsClassifier(), params, cv=* knn\_cv **=** GridSearchCV(KNeighborsClassifier(), params, cv**=**cv, sco knn\_cv**.**fit(x\_train, y\_train)

print\_cv\_result(knn\_cv, x\_test, y\_test)

Результаты кросс-валидации при cv=5

Оптимизация метрики accuracy: 0.8215999999999999 Лучший параметр: {'n\_neighbors': 38}

Метрики на тестовом наборе

Среднее квадратичное отклонение: 0.41844951905815353 Коэффициент детерминации: -0.20954951430128244

Оптимизация метрики recall: 0.25168854058477325 Лучший параметр: {'n\_neighbors': 1}

Метрики на тестовом наборе

Среднее квадратичное отклонение: 0.5114684741017769 Коэффициент детерминации: -0.8070711190246458

Оптимизация метрики f1: 0.25791015153550567 Лучший параметр: {'n\_neighbors': 1}

Метрики на тестовом наборе

Среднее квадратичное отклонение: 0.5114684741017769 Коэффициент детерминации: -0.8070711190246458

Результаты кросс-валидации при cv=10

Оптимизация метрики accuracy: 0.8216999999999999 Лучший параметр: {'n\_neighbors': 36}

Метрики на тестовом наборе

Среднее квадратичное отклонение: 0.4183300132670378 Коэффициент детерминации: -0.20885873787963694

Оптимизация метрики recall: 0.25674471156863976 Лучший параметр: {'n\_neighbors': 1}

Метрики на тестовом наборе

Среднее квадратичное отклонение: 0.5114684741017769 Коэффициент детерминации: -0.8070711190246458

Оптимизация метрики f1: 0.26291414223680165 Лучший параметр: {'n\_neighbors': 1}

Метрики на тестовом наборе

Среднее квадратичное отклонение: 0.5114684741017769 Коэффициент детерминации: -0.8070711190246458

In [25]:

best\_k **=** 1

y\_pred\_best3 **=** KNeighborsClassifier(n\_neighbors**=**best\_k)**.**fit(x\_train, y\_t

In [29]:

# Сравнение исходной и оптимальной моделей

Исходная модель

print('Исходная модель\n') print\_metrics(y\_test, y\_pred\_base) print(' ') print('\nОптимальная модель\n') print\_metrics(y\_test, y\_pred\_best)

Среднее квадратичное отклонение: 0.42178193417926285 Коэффициент детерминации: -0.22889125410735645

Оптимальная модель

Среднее квадратичное отклонение: 0.5114684741017769 Коэффициент детерминации: -0.8070711190246458