

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:

студент группы ИУ5-63Б
Тарновский Д.Р.

Проверил:

преподаватель каф. ИУ5
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

1) Обработка пропусков в данных

```
In [4]: #проверяем типы данных и заполненность столбцов
data.info()
```

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

```

```

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

```

```

In [5]: #удаляем столбец с номером клиента (так как он незначимый)
# и с регионом проживания (так как он нуждается в серьезной предобработке)
data.drop(['client_id', 'living_region'], axis=1, inplace=True)

```

```

In [6]: # анализируем столбец marital_status, смотрим, какое значение в нем явля
data['marital_status'].describe()

```

```

Out[6]: count      170743
unique         5
top            MAR
freq          93954
Name: marital_status, dtype: object

```

```

In [7]: # анализируем столбец education, смотрим, какое в нем самое частое значе
data['education'].describe()

```

```

Out[7]: count      170741
unique         5
top            SCH
freq          87537
Name: education, dtype: object

```

```

In [8]: # дозаполняем нечисловые столбцы с пропусками самыми часто встречающимися
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(), inplace=True)

```

```

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=True)
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
---  -
0   gender                                170746 non-null object
1   age                                   170746 non-null float64
2   marital_status                        170746 non-null object
3   job_position                          170746 non-null object
4   credit_sum                            170746 non-null float64
5   credit_month                          170746 non-null int64
6   tariff_id                             170746 non-null float64
7   score_shk                             170746 non-null float64
8   education                             170746 non-null object
9   monthly_income                       170746 non-null float64
10  credit_count                          170746 non-null float64
11  overdue_credit_count                  170746 non-null float64
12  open_account_flg                      170746 non-null int64
dtypes: float64(7), int64(2), object(4)
memory usage: 16.9+ MB
```

2) Кодирование категориальных признаков

```
In [13]: category_cols = ['gender', 'job_position', 'education', 'marital_status']
```

```
In [14]: print("Количество уникальных значений\n")
for col in category_cols:
    print(f'{col}: {data[col].unique().size}')
```

Количество уникальных значений

```
gender: 2
job_position: 18
education: 5
marital_status: 5
```

```
In [15]: # 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="education"),
                  pd.get_dummies(data['marital_status'], prefix="marital_status"),
                  axis=1])
```

```
In [16]: #удаляем старые нечисловые столбцы, вместо них уже появились новые числовые
data.drop(['gender', 'job_position', 'education', 'marital_status'], axis=1, inplace=True)
```

```
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

3) Разделение выборки на обучающую и тестовую

```
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,
```

4) Масштабирование данных

```
In [19]: scaler = MinMaxScaler().fit(x_train)
x_train = pd.DataFrame(scaler.transform(x_train), columns=x_train.columns)
x_test = pd.DataFrame(scaler.transform(x_test), columns=x_train.columns)
x_train.describe()
```

```
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

5) Обучение KNN с произвольным k

```
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, y_pred)}")
    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()
```

```
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

Среднее квадратичное отклонение: 0.42178193417926285

Коэффициент детерминации: -0.22889125410735645

6) Кросс-валидация

```
In [23]: 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=cv,
#knn_cv = GridSearchCV(KNeighborsClassifier(), params, cv=cv, sc
        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: 0.821

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
```


7) Сравнение исходной и оптимальной моделей

In [29]:

```
print('Исходная модель\n')
print_metrics(y_test, y_pred_base)
print('_____')
print('\nОптимальная модель\n')
print_metrics(y_test, y_pred_best)
```

Исходная модель

Среднее квадратичное отклонение: 0.42178193417926285

Коэффициент детерминации: -0.22889125410735645

Оптимальная модель

Среднее квадратичное отклонение: 0.5114684741017769

Коэффициент детерминации: -0.8070711190246458