

**Московский государственный технический
университет им. Н. Э. Баумана**

Курс «Технологии машинного обучения»

Отчёт по лабораторной работе №3

Выполнил:
Ювенский Л.А.
группа ИУ5-63Б

Проверил:
Гапанюк Ю.Е.

Дата: 14.03.25

Дата:

Подпись:

Подпись:

Москва, 2025 г.

Цель лабораторной работы: изучение способов подготовки выборки и подбора гиперпараметров на примере метода ближайших соседей.

Задание:

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода `train_test_split` разделите выборку на обучающую и тестовую.
4. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K . Оцените качество модели с помощью подходящих для задачи метрик.
5. Произведите подбор гиперпараметра K с использованием `GridSearchCV` и `RandomizedSearchCV` и кросс-валидации, оцените качество оптимальной модели. Используйте не менее двух стратегий кросс-валидации.
6. Сравните метрики качества исходной и оптимальной моделей.

Ход выполнения:

Практика

Datset: <https://github.com/ongaunje1/credit-score-prediction>

Загрузка и первичный анализ

```
[1] import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

```
[2] df0 = pd.read_csv("/dataset.csv")
df0.info()
df0.head()
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14790 entries, 0 to 14789
Data columns (total 12 columns):
Column Non-Null Count Dtype

0 age 14790 non-null int64
1 annual_income 14790 non-null float64
2 num_bank_acc 14790 non-null int64
3 num_credit_card 14790 non-null int64
4 interest_rate 14790 non-null float64
5 delay_from_due_date 14790 non-null int64
6 outstanding_debt 14790 non-null float64
7 credit_history_age 14790 non-null float64
8 installment_per_month 14790 non-null float64
9 monthly_balance 14790 non-null float64
10 payment_of_min_amount_Yes 14790 non-null bool
11 Predicted Credit Score 14790 non-null int64
dtypes: bool(1), float64(5), int64(6)
memory usage: 1.3 MB

| | age | annual_income | num_bank_acc | num_credit_card | interest_rate | delay_from_due_date | outstanding_debt | credit_history_age | installment_per_month | monthly_balance |
|---|-----|---------------|--------------|-----------------|---------------|---------------------|------------------|--------------------|-----------------------|-----------------|
| 0 | 23 | 19114.12 | 3 | 4 | 3 | 3 | 809.98 | 22.90 | 49.57 | |

Подключено к среде выполнения "Серверный ускоритель Python 3 на базе Google Compute Engine ()".

```
memory usage: 1.3 MB
```

```
[2] age annual_income num_bank_acc num_credit_card interest_rate delay_from_due_date outstanding_debt credit_history_age installment_per_month monthly_balance
```

| | | | | | | | | | | |
|---|----|----------|---|---|---|---|--------|-------|-------|--|
| 0 | 23 | 19114.12 | 3 | 4 | 3 | 3 | 809.98 | 22.90 | 49.57 | |
| 1 | 24 | 19114.12 | 3 | 4 | 3 | 3 | 809.98 | 22.10 | 49.57 | |
| 2 | 28 | 34847.84 | 2 | 4 | 6 | 3 | 605.03 | 27.40 | 18.82 | |
| 3 | 28 | 34847.84 | 2 | 4 | 6 | 3 | 605.03 | 27.50 | 18.82 | |
| 4 | 55 | 30689.89 | 2 | 5 | 4 | 5 | 632.46 | 17.11 | 16.42 | |

- **age** Age of the individual. int64
- **annual_income** Annual income of the individual. float64
- **num_bank_acc** Number of bank accounts owned. int64
- **num_credit_card** Number of credit cards owned. int64
- **interest_rate** Interest rate of credit card. float64
- **delay_from_due_date** Delayed days from payment's due date. int64
- **outstanding_debt** Amount of outstanding debt. float64
- **credit_history_age** Credit history age. float64
- **payment_of_min_amount** Indicates if the minimum amount is paid. bool
- **installment_per_month** Monthly installment amount. float64
- **monthly_balance** Monthly balance float64
- **credit_score** Credit score. int64

Разделение на обучающую и тестовую выборки

```
[4] dfX = df0.drop(columns=["Predicted Credit Score"])
dfY = df0["Predicted Credit Score"]
```

```
[8] print("\n=====X=====\n")
dfX.info()
dfX.head()

print("\n=====Y=====\n")
```

Подключено к среде выполнения "Серверный ускоритель Python 3 на базе Google Compute Engine ()".

Команды + Код + Текст

ОЗУ Диск

✓ [3] from sklearn.model_selection import train_test_split

✓ [9] xTrain, xTest, yTrain, yTest = train_test_split(dfX, dfY, test_size=0.2, random_state=1)

✓ 0 сек. [11] print(xTrain.shape)
print(xTest.shape)
print(yTrain.shape)
print(yTest.shape)

↗ (11832, 11)
(2958, 11)
(11832,)
(2958,)

▼ KNN для произвольно заданного гиперпараметра K

✓ [21] from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

knn = KNeighborsClassifier(n_neighbors = 8, n_jobs=-1)

✓ 0 сек. [25] knn.fit(xTrain, yTrain)

↗ KNeighborsClassifier
KNeighborsClassifier(n_jobs=-1, n_neighbors=8)

▼ Оценка качества модели

✓ 0 сек. [26] yPredict = knn.predict(xTest)

accuracy = accuracy_score(yPredict, yTest)
print("Точность: ", accuracy)

↗ Точность: 0.7491548343475322

Подключено к среде выполнения "Серверный ускоритель Python 3 на базе Google Compute Engine ()".

Подбор гиперпараметра K через GridSearchCV и RandomizedSearchCV

3 сек.

from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import make_scorer, accuracy_score

knn = KNeighborsClassifier()

param_grid = {'n_neighbors': list(range(3, 8))}
grid_search = GridSearchCV(knn, param_grid, cv=3, scoring='accuracy', n_jobs=-1)
grid_search.fit(np.array(dfX), np.array(dfY))

print(f"Лучшие параметры: {grid_search.best_params}")
print(f"Лучшая точность: {grid_search.best_score}")

↗ Лучшие параметры: {'n_neighbors': 6}
Лучшая точность: 0.6778228532792427

✓ 3 сек. [30] from sklearn.model_selection import RandomizedSearchCV
from sklearn.neighbors import KNeighborsClassifier
from scipy.stats import randint

knn = KNeighborsClassifier()

param_dist = {'n_neighbors': randint(3, 8)}
random_search = RandomizedSearchCV(knn, param_distributions=param_dist,
n_iter=2, cv=3, scoring='accuracy',
n_jobs=-1, random_state=42)

random_search.fit(np.array(dfX), np.array(dfY))

print(f"Лучшие параметры: {random_search.best_params}")
print(f"Лучшая точность: {random_search.best_score}")

↗ Лучшие параметры: {'n_neighbors': 6}
Лучшая точность: 0.6778228532792427

Подключено к среде выполнения "Серверный ускоритель Python 3 на базе Google Compute Engine ()".