

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

Курс «Технологии машинного обучения»

Отчёт по лабораторной работе №3

Выполнила:
Беспалова В. А.
группа ИУ5-63Б

Проверил:
Гапанюк Ю.Е.

Дата:
Подпись:

Дата:
Подпись:

Москва, 2025 г.

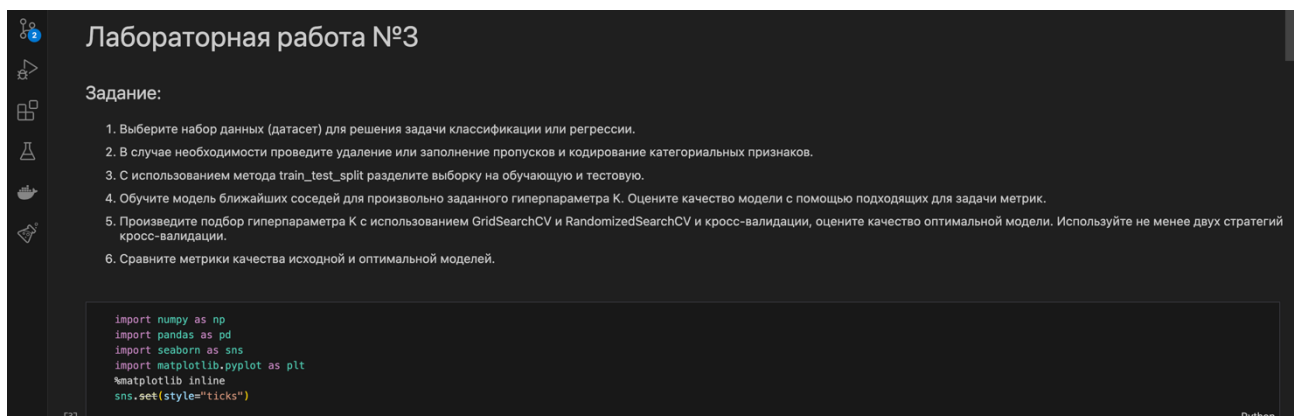
Цель лабораторной работы

Цель: 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.

Код программы и экранные формы



The screenshot shows a Jupyter Notebook interface. The title bar reads 'Лабораторная работа №3'. Below the title, the 'Задание:' (Task:) section lists six steps for the assignment. At the bottom, a code cell contains the following Python imports:

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

The notebook interface includes a sidebar with icons for file operations, search, and other Jupyter features. The bottom status bar shows '(3)' and 'Python'.

The screenshot displays a Jupyter Notebook interface with a dark theme. On the left sidebar, there are icons for file management, search, and other notebook functions. The main area shows a series of text outputs describing a dataset:

- Датасет:** CСылка
- ID: A unique identifier for each individual
- Age: The age of the individual
- Gender: The gender of the individual
- Height: The height of the individual in centimeters
- Weight: The weight of the individual in kilograms
- BMI: The body mass index of the individual, calculated as weight divided by height squared
- Label: The obesity classification of the individual, which can be one of the following:
 - Normal Weight
 - Overweight
 - Obese
 - Underweight

Below the text, a code cell is shown with the following Python code:

```
data = pd.read_csv('obesity_classification.csv')  
data.head()
```

The output of the code is displayed below the code cell, showing the first five rows of the dataset:

	ID	Age	Gender	Height	Weight	BMI	Label
0	1	26	Male	175	80	26.3	Normal Weight
1	2	30	Female	160	60	22.5	Normal Weight
2	3	35	Male	180	90	27.3	Overweight
3	4	40	Female	150	50	20.0	Underweight
4	5	45	Male	190	100	31.2	Obese

	ID	Age	Gender	Height	Weight	BMI	Label
0	1	25	Male	175	80	25.3	Normal Weight
1	2	30	Female	160	60	22.5	Normal Weight
2	3	35	Male	180	90	27.3	Overweight
3	4	40	Female	150	50	20.0	Underweight
4	5	45	Male	190	100	31.2	Obese

```
# Удалим колонку ID
data.drop(columns=['ID'], axis=1, inplace=True)
```

	Age	Gender	Height	Weight	BMI	Label	Gender_encoded	Label_encoded
0	25	Male	175	80	25.3	Normal Weight	1	0
1	30	Female	160	60	22.5	Normal Weight	0	0
2	35	Male	180	90	27.3	Overweight	1	2
3	40	Female	150	50	20.0	Underweight	0	3
4	45	Male	190	100	31.2	Obese	1	1

	Age	Height	Weight	BMI	Gender_encoded	Label_encoded
0	25	175	80	25.3	1	0
1	30	160	60	22.5	0	0
2	35	180	90	27.3	1	2
3	40	150	50	20.0	0	3
4	45	190	100	31.2	1	1

	Age	Height	Weight	BMI	Gender_encoded	Label_encoded
count	108.000000	108.000000	108.000000	108.000000	108.000000	108.000000
mean	46.555556	166.574074	59.409741	20.549074	0.518195	1.767937
std	24.720620	27.787361	28.856233	7.393818	0.501986	1.260848
min	11.000000	120.000000	10.000000	3.000000	0.000000	0.000000
25%	27.000000	140.000000	35.000000	16.700000	0.000000	0.000000
50%	42.500000	175.000000	55.000000	21.200000	1.000000	2.000000
75%	59.250000	190.000000	85.000000	26.100000	1.000000	3.000000
max	112.000000	210.000000	120.000000	37.200000	1.000000	3.000000

Разделение выборки

```
from sklearn.model_selection import train_test_split

X = data.drop('Label_encoded', axis=1)
Y = data['Label_encoded']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=5)
```

```
# Размер обучающей выборки
X_train.shape, Y_train.shape
```

```
... ((86, 5), (86,))
```

```
# Размер тестовой выборки
X_test.shape, Y_test.shape
```

```
... ((22, 5), (22,))
```

```
np.unique(Y_train)
```

```
... array([0, 1, 2, 3])
```

```
np.unique(Y_test)
```

```
... array([0, 1, 2, 3])
```

Вывод: функция `train_test_split` разделила исходную выборку таким образом, чтобы в обучающей и тестовой частях сохранились все классы.

KNN для произвольно заданного гиперпараметра K

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

knn5 = KNeighborsClassifier(n_neighbors = 5)
knn5.fit(X_train, Y_train)

knn10 = KNeighborsClassifier(n_neighbors = 10)
knn10.fit(X_train, Y_train)
```

```
... KNeighborsClassifier
KNeighborsClassifier(n_neighbors=10)
```

Accuracy

```
Y_predict_5 = knn5.predict(X_test)
Y_predict_10 = knn10.predict(X_test)

print("Точность при K=5: ", accuracy_score(Y_test, Y_predict_5))
print("Точность при K=10: ", accuracy_score(Y_test, Y_predict_10))
```

```
... Точность при K=5:  0.6818181818181818
Точность при K=10:  0.7727272727272727
```

Посмотрим accuracy по классам

```
from typing import Dict

def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray) -> Dict[int, float]:
    """
    Вычисление метрики accuracy для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
    Возвращает словарь: ключ - метка класса,
    значение - Accuracy для данного класса
    """
    # Для удобства фильтрации сформируем Pandas DataFrame
    d = {'t': y_true, 'p': y_pred}
    df = pd.DataFrame(data=d)
    # Метки классов
    classes = np.unique(y_true)
    # Результирующий словарь
    res = dict()
    # Перебор меток классов
    for c in classes:
        # отфильтруем данные, которые соответствуют
        # текущей метке класса в истинных значениях
        temp_data_ft = df[df['t']==c]
        # расчет accuracy для заданной метки класса
        temp_acc = accuracy_score(
            temp_data_ft['t'].values,
            temp_data_ft['p'].values)
        # сохранение результата в словарь
        res[c] = temp_acc
    return res

def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики accuracy для каждого класса
    """
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
```

```
[18]
```

```

print('Метка \t Accuracy')
for i in accs:
    print('{} \t {}'.format(i, accs[i]))

```

[18] Python

```

print_accuracy_score_for_classes(Y_test, Y_predict_5)

```

[19] Python

```

... Метка Accuracy
0 0.5555555555555556
1 1.0
2 0.3333333333333333
3 0.8571428571428571

```

```

print_accuracy_score_for_classes(Y_test, Y_predict_10)

```

[20] Python

```

... Метка Accuracy
0 0.6666666666666666
1 1.0
2 0.3333333333333333
3 1.0

```

Вывод: разная ассигасу для разных классов.

Матрица ошибок

Количество верно и ошибочно классифицированных данных, представленное в виде матрицы.

```

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(Y_test, Y_predict_5)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()

```

[] Python

```

cm = confusion_matrix(Y_test, Y_predict_10)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()

```

[32] Python

Подбор гиперпараметра K с помощью GridSearchCV, RandomizedSearchCV и кросс-валидации

Перекрестная проверка — метод оценки аналитической модели и её поведения на независимых данных. При оценке модели имеющиеся в наличии данные разбиваются на k частей. Затем на k-1 частях данных производится обучение модели, а оставшаяся часть данных используется для тестирования. Процедура повторяется k раз; в итоге каждая из k частей данных используется для тестирования. В результате получается оценка эффективности выбранной модели с наиболее равномерным использованием имеющихся данных.

Каждую из k частей принято называть *fold* (фолд). Количество фолдов при кросс-валидации обычно обозначают параметром *cv*. В результате каждого обучения и проверки модели формируются указанные разработчиком метрики качества модели. Например, в результате кросс-валидации для 3 фолдов при решении задачи классификации мы можем получить 3 значения ассигасу, которые были вычислены для каждой комбинации фолдов. Полученные метрики обычно усредняют.

Стратегии кросс-валидации основаны на том, что разработчик задает или количество разбиений (в этом случае автоматически определяются размеры обучающей и тестовой выборки для каждого разбиения) или размер обучающей или тестовой выборки для одного разбиения (в этом случае автоматически определяется количество разбиений).

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, KFold, ShuffleSplit

# GridSearchCV: Полный перебор гиперпараметров.
# RandomizedSearchCV: Выбор гиперпараметров случайным образом.

# KFold: Разбивает данные на K частей (работает в соответствии с определением кросс-валидации).
# ShuffleSplit: Генерируется N случайных перемешиваний данных, в каждом перемешивании заданная доля помещается в тестовую выборку.

knn = KNeighborsClassifier()

# задание сетки гиперпараметров
n_range = np.array(range(2, 30, 2))
param_grid = {'n_neighbors': n_range}

# задание стратегий кросс-валидации
cv_kfold = KFold(n_splits=5, shuffle=True, random_state=5) #shuffle - перемешивание данных
cv_shuffle = ShuffleSplit(n_splits=5, test_size=0.2, random_state=5)

# подбор гиперпараметра K с помощью GridSearchCV и оценка качества с помощью кросс-валидации
grid_search = GridSearchCV(knn, param_grid, scoring='accuracy', cv=cv_kfold)
grid_search.fit(X_train, Y_train)

print("Best parameters (GridSearchCV):", grid_search.best_params_)
print("Best accuracy (GridSearchCV):", grid_search.best_score_)

# подбор гиперпараметра K с помощью RandomizedSearchCV и оценка качества с помощью кросс-валидации
random_search = RandomizedSearchCV(knn, param_grid, scoring='accuracy', cv=cv_shuffle)
random_search.fit(X_train, Y_train)

print("Best parameters (RandomizedSearchCV):", random_search.best_params_)
print("Best accuracy (RandomizedSearchCV):", random_search.best_score_)

[ ]
```

Python

```
... Best parameters (GridSearchCV): {'n_neighbors': np.int64(8)}
Accuracy (GridSearchCV): 0.8241830065359478
Best parameters (RandomizedSearchCV): {'n_neighbors': np.int64(6)}
Accuracy (RandomizedSearchCV): 0.7888888888888889
```