IT Diving | Machine Learning

Сегодня вам предстоит на практике познакомиться с основными задачами машинного обучения. В ходе работы получится поработать с популярными библиотеками `pandas`, `numpy`, `sklearn`.

Для выполнения задания необходимо следовать по этой тетрадке сверху вниз и заполнять недостоющие части кода или отвечать на заданные вопросы.

```
# Импортируем необходимые библиотеки
# Полезно все импорты держать рядом

from os.path import exists

import numpy as np
import pandas as pd
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score,
mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from PIL import Image
from sklearn.cluster import KMeans, SpectralClustering

%matplotlib inline

# Зафиксируем сид для генератора случайных чисел
# Это полезно для воспроизводимости результатов

RANDOM_SEED = 0xC0FFEE
```

Классификация

Для знакомства с задачей классификацией воспользуемся выборкой данных о пациентах с доброкачественными и злокачественными опухолями. Наша задача — научиться их отличать.

Вместе с тетрадкой находится файл `cancer.csv` — это таблица, где каждая строчка соответствует отдельной клетке, а столбцы ее численные характеристики. Подробнее про датасет можно прочитать, например, вот тут.

Начнем с чтения данных с диска, для этого реализуйте функцию `read_cancer_dataset`. Поможет с этим библиотека `pandas` и пара полезных вещей из нее:

1. `DataFrame`
2. `read_csv`

```
def read_cancer_dataset(path_to_csv: str, shuffle: bool = True) ->
pd.DataFrame:
    """Функция для чтения данных с диска, а также их случайного
перемешивания

    Parameters
    ----------
    path_to_csv: Путь к файлу cancer.csv
    shuffle: Если True, то перемешивает данные

    Return
    ------
    dataframe: Данные в формате DataFrame
    """
    dataframe = pd.read_csv(path_to_csv)
    if shuffle:
        dataframe = dataframe.sample(frac=1).reset_index(drop=True)
    return dataframe

# Посмотрим на наши данные:
# Колонка "label" отвечает за тип опухоли
# Колонки 1-30 отвечают за признаки

cancer_dataset = read_cancer_dataset("cancer.csv", shuffle=True)
cancer_dataset.head()
```

```
  label      1      2      3      4        5        6        7
8  \
0     B  12.19  13.29  79.08  455.8  0.10660  0.09509  0.02855
0.028820
1     B  11.60  12.84  74.34  412.6  0.08983  0.07525  0.04196
0.033500
2     B  14.74  25.42  94.70  668.6  0.08275  0.07214  0.04105
0.030270
3     B  13.17  18.22  84.28  537.3  0.07466  0.05994  0.04859
0.028700
4     B  12.58  18.40  79.83  489.0  0.08393  0.04216  0.00186
0.002924

        9  ...     21     22      23     24      25       26       27
\
0  0.1880  ...  13.34  17.81   91.38  545.2  0.1427  0.25850  0.099150

1  0.1620  ...  13.06  17.16   82.96  512.5  0.1431  0.18510  0.192200

2  0.1840  ...  16.51  32.29  107.40  826.4  0.1060  0.13760  0.161100

3  0.1454  ...  14.90  23.89   95.10  687.6  0.1282  0.19650  0.187600
```

```
4  0.1697  ...  13.50  23.08   85.56  564.1  0.1038  0.06624  0.005579


          28      29       30
0  0.081870  0.3469  0.09241
1  0.084490  0.2772  0.08756
2  0.109500  0.2722  0.06956
3  0.104500  0.2235  0.06925
4  0.008772  0.2505  0.06431

[5 rows x 31 columns]
```

Первым делом необходимо подготовить данные к работе, а именно: разбить на тренировочную и тестовую части.

Тренировочная часть используется для обучения моделей, именно по ней ищутся необходимые зависимости в данных.

Тестовая часть используется для оценки качества моделей. Это данные, которые модель не видела, поэтому качество предсказаний по ним позволит оценить ее обобщающие способности.

Крайне важно, чтобы тестовая и тренировочная части описывали одинаковую природу данных. Например, в случае задачи классификации, важно чтобы соотношение классов было приблизительно равно в них. Иначе мы можем неправильно интерпретировать результаты.

Реализуйте функцию `prepare_cancer_dataset`, которая разделяет данные на таргет и признаки, а также выделяет тестовую часть. В этом может помочь `train_test_split` из библиотеки `sklearn`. Не забывайте фиксировать `random_state` или другие аналогичные параметры — это полезная привычка, которая съэкономить вам сотни часов дебага в будущем.

Первым делом необходимо подготовить данные к работе, а именно: разбить на тренировочную и тестовую части.

Тренировочная часть используется для обучения моделей, именно по ней ищутся необходимые зависимости в данных.

Тестовая часть используется для оценки качества моделей. Это данные, которые модель не видела, поэтому качество предсказаний по ним позволит оценить ее обобщающие способности.

Крайне важно, чтобы тестовая и тренировочная части описывали одинаковую природу данных. Например, в случае задачи классификации, важно чтобы соотношение классов было приблизительно равно в них. Иначе мы можем неправильно интерпретировать результаты.

Реализуйте функцию `prepare_cancer_dataset`, которая разделяет данные на таргет и признаки, а также выделяет тестовую часть. В этом может помочь `train_test_split` из библиотеки `sklearn`. Не забывайте фиксировать `random_state` или другие аналогичные

параметры — это полезная привычка, которая съэкономить вам сотни часов дебага в будущем.

```
def prepare_cancer_dataset(
    dataset: pd.DataFrame, label_col_name: str = "label", test_size:
float = 0.1
) -> tuple[np.ndarray, np.ndarray, np.ndarray, np.ndarray]:
    """Функция для выделения таргета и признаков,
    а также разделения на тренировочную и тестовую части.

    Для таргета необходимо привести данные к формату 0/1.
    Сопоставьте 0 доброкачественной опухоли ("B"),
    а 1 злокачественной ("M")

    Parameters
    ----------
    dataset: DataFrame с датасетом
    label_col_name: Название колонки с таргетом
    test_size: доля тестовой выборки относительно всего датасета

    Return
    ------
        4 numpy массива: X_train, X_test, y_train, y_test
        X_train, X_test -- матрицы признаков размером [n_elements; 30]
        y_train, y_test -- массивы из 0 и 1 размером [n elements]
    """
    dataset[label_col_name] = dataset[label_col_name].map({"B": 0,
"M": 1})
    X = dataset.drop(columns=[label_col_name]).values
    y = dataset[label_col_name].values
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=test_size, stratify=y, random_state=50)
    return X_train, X_test, y_train, y_test

# Выполним подготовку данных
X_train, X_test, y_train, y_test =
prepare_cancer_dataset(cancer_dataset)

# Код ниже проверяет правильность подготовки данных
# Если он упал, то надо исправить функцию выше
assert X_train.shape == (512, 30) and y_train.shape == (512,)
assert X_test.shape == (57, 30) and y_test.shape == (57,)

train_ratio = y_train.sum() / len(y_train)
test_ratio = y_test.sum() / len(y_test)
assert train_ratio < 0.5
assert np.abs((test_ratio - train_ratio) / train_ratio) < 0.015
```

Начнем с наивного решения — модель, которая предсказывает наиболее популярный класс. Реализуйте методы `fit` и `predict` у класса ниже.

```
class MostCommonClassification:
    def __init__(self):
        self.predict_class = None

    def fit(self, X: np.ndarray, y: np.ndarray):
        """Функция обучения наивной модели.
        Она получает на вход X и y,
        чтобы иметь схожий интерфейс с другими моделями.

        Функция определяет самый популярный класс и
        сохраняет его в predict_class

        Parameters
        ----------
        X: признаки, не используются
        y: таргет, номера классов, одномерный массив
        """
        self.predict_class = np.bincount(y).argmax()

    def predict(self, X: np.ndarray) -> np.ndarray:
        """Функция для предсказания классов

        Parameters
        ----------
        X: элементы, для которых надо предсказать класс
            матрица размером [n_elements; n_features]

        Return
        ------
            предсказанный класс для каждого элемента
            numpy массив размером [n_elements]
        """
        if self.predict_class is None:
            raise RuntimeError("Call fit before predict")
        # Your code here
        predictions = np.full(X.shape[0],
fill_value=self.predict_class)
        return predictions
```

"Обучим" наивную модель и оценим ее качество.

Для оценки качества воспользуемся двумя популярными метриками:

1. Точность (accuracy) измеряет, как часто модель предсказывает правильные ответы из всех возможных ответов. Она вычисляется как отношение числа правильных предсказаний к общему числу предсказаний. Например, если модель правильно предсказала 80 из 100 объектов, то точность будет равна 0.8 или 80%.

2. F1-score — более сложная метрика, она измеряет сбалансированность модели, учитывая как точность (precision), так и полноту (recall) предсказаний. Точнее говоря, она считает их гармоническое среднее. Использование такой метрики позволяет более точно оценить модели в случае сильной несбалансированности в данных.

Более подробно ознакомиться с метриками классификации можно, например, тут.

```
def print_classification_report(y_test, y_pred):
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    print(f"Accuracy: {accuracy * 100:.2f}%", f"F1-score: {f1 *
100:.2f}%", sep="\n")

model_most_common = MostCommonClassification()
model_most_common.fit(X_train, y_train)
y_pred_most_common = model_most_common.predict(X_test)

print_classification_report(y_test, y_pred_most_common)
```

```
Accuracy: 63.16%
F1-score: 0.00%
```

Во время лекции мы уже успели познакомиться с алгоритмом "K ближайших соседей". Давайте воспользуемся им для решения нашей задачи. Поможет в этом реализация из `sklearn`: `KNeighborsClassifier`

```
model_v1 = KNeighborsClassifier(n_jobs=-1)
model_v1.fit(X_train, y_train)
y_pred_v1 = model_v1.predict(X_test)

print_classification_report(y_test, y_pred_v1)
```

```
Accuracy: 94.74%
F1-score: 93.02%
```

Результат уже стал значительно выше! Если вы все сделали верно, то уже должны получить точность выше 90%.

Однако еще есть куда расти. Один из главных способов поднять качество — это правильно настроить модель.

Ознакомьтесь с документацией алгоритма по ссылке выше и поиграйтесь с параметрами модели. Например, вместо стандартного `n_neighbors=5` можно поставить `n_neighbors=7`. Тогда при предсказании класса модель будет смотреть не на 5 ближайших соседей, а на 7.

Попробуйте получить как можно более высокое качество!

```
model_v2 = KNeighborsClassifier(n_neighbors=5, n_jobs=-1)
model_v2.fit(X_train, y_train)
y_pred_v2 = model_v2.predict(X_test)

print_classification_report(y_test, y_pred_v2)
```

```
Accuracy: 94.74%
F1-score: 93.02%
```

Одна из особенностей алгоритма "K ближайших соседей" — это необходимость вычислять расстояние между векторами признаков. По умолчанию используется обычное евклидово расстояние:

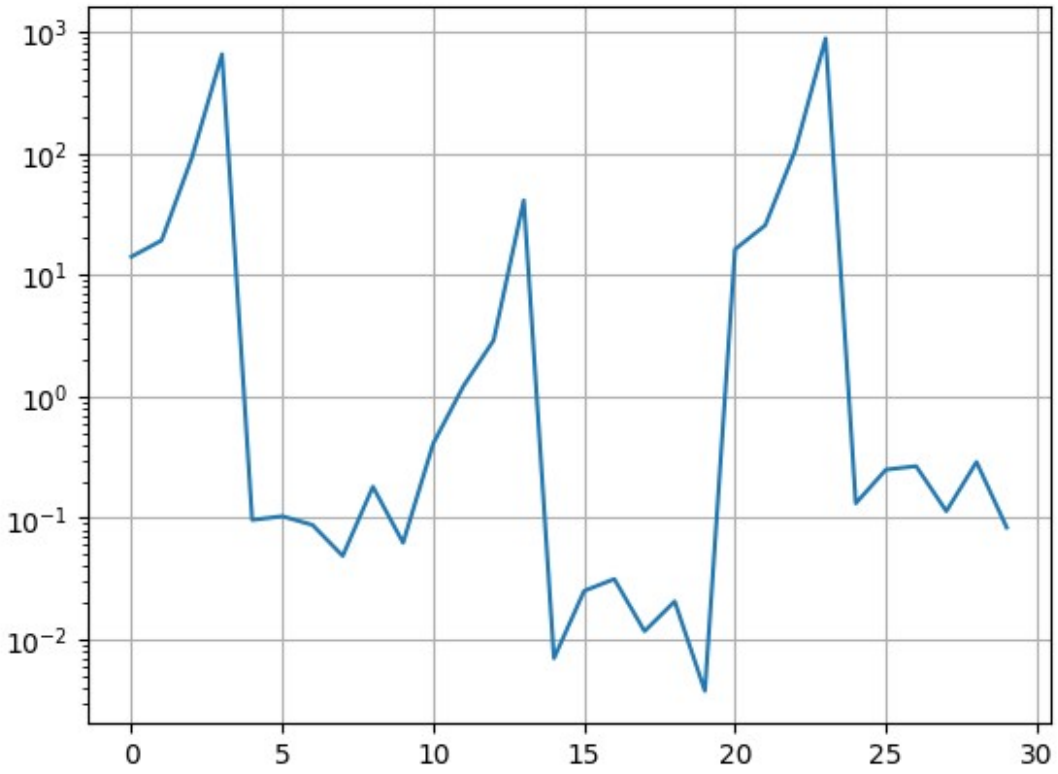
$$\operatorname{dist}(p,q)=\sqrt{\sum_{i=1}^{n}\left(p_i^2-q_i^2\right)}$$

Здесь p и q — это вектора размерности n, то есть массивы, описывающие n признаков.

Из формулы можно заменить, что если значения одного из признаков очень большие, то он будет подавлять вклад признаков с маленькими значениями.

Давайте посмотрим на средние значения каждого признака в нашем датасете.

```
pyplot.grid(visible=True)
pyplot.yscale("log")
pyplot.plot(X_train.mean(axis=0))
pyplot.show()
```



Можно заметить, что некоторые признаки в среднем варьируются возле 1000, тогда как другие меньше 0.01.

Чтобы это исправить можно отмасштабировать признаки, а именно привести каждый признак к среднему 0 и дисперсии 1. Помочь в этом может `StandardScaler` из библиотеки `sklearn`.

Изучите документацию этого алгоритма и реализуйте функцию `scale_features`.

```
def scale_features(
    train_data: np.ndarray, test_data: np.ndarray
) -> tuple[np.ndarray, np.ndarray]:
    """Функция для масштабирования данных
    Переводит каждый столбец данных в новый со средним 0 и дисперсией
1.

    Для подсчета статистики и обучения используется тренировочная
часть.
    Затем масштабирование применяется и к тестовым данным.

    Parameters
        ----------
        train_data: матрица размером [train_size; n_features]
            Тренировочная часть
        test_data: матрица размером [test_size; n_features]
            Тестовая часть
```

```
        Return
        ------
            train_data_scaled, test_data_scaled
                numpy матрицы с отмасштабированными данными
    """
    scaler = StandardScaler()
    scaler.fit(train_data)
    train_data_scaled = scaler.transform(train_data)
    test_data_scaled = scaler.transform(test_data)
    return train_data_scaled, test_data_scaled

X_train_scaled, X_test_scaled = scale_features(X_train, X_test)

mean, std = X_train_scaled.mean(axis=0), X_train_scaled.std(axis=0)
assert np.allclose(mean, 0) and np.allclose(std, 1)
```

Обучите `KNeighborsClassifier` на новых данных, не забудьте подобрать оптимальные гиперпараметры. Возможно достить точности выше 95%!

```
model_v3 = KNeighborsClassifier(n_neighbors=10, n_jobs=-1)
model_v3.fit(X_train_scaled, y_train)
y_pred_v3 = model_v3.predict(X_test_scaled)

print_classification_report(y_test, y_pred_v3)
```

```
Accuracy: 100.00%
F1-score: 100.00%
```

Задачу классификации можно решать множеством разных способов, многие из которых реализованы в библиотеки `sklearn`.

Вы можете ознакомиться со всем списком алгоритмов в библиотеке здесь. Не все они подходят для задачи классификации, ориентируйтесь на слово `Classifier` в названии, а также не стесняйтесь переходить по ссылкам и читать документацию и описание.

Попробуйте применить новые алгоритмы к нашей задаче. Рекомендуем обратить внимание на:

1. `LogisticRegression`
2. `RandomForestClassifier`
3. `SVC`

Для методов на основе линейных преобразований полезно использовать отмасштабированные данные.

Вполне реально получить идеальное качество в 100%!

Регрессия

Для знакомства с задачей регрессии, нам поможет популярный датасет Boston, он прикреплен к заданию в файле `boston.csv`. Это набор данных с информаций о медианной стоимости домов, а также различных характеристик района. Ознакомиться с датасетом можно по ссылке выше, а ниже представленно описание каждого столбца в данных:

```
1. crim      per capita crime rate by town
2. zn        proportion of residential land zoned for lots over 25,000
sq.ft.
3. indus     proportion of non-retail business acres per town
4. chas      Charles River dummy variable (= 1 if tract bounds river;
0 otherwise)
5. nox       nitric oxides concentration (parts per 10 million)
6. rm        average number of rooms per dwelling
7. age       proportion of owner-occupied units built prior to 1940
8. dis       weighted distances to five Boston employment centres
9. rad       index of accessibility to radial highways
10. tax      full-value property-tax rate per \$10,000
11. ptratio  pupil-teacher ratio by town
12. b        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by
town
13. lstat    % lower status of the population
14. medv     median value of owner-occupied homes in \$'s
```

Наша задача — научится предсказывать стоимость дома по критериям района. То есть вместо ограниченного числа значений, модель теперь должна предсказывать любые целые числа.

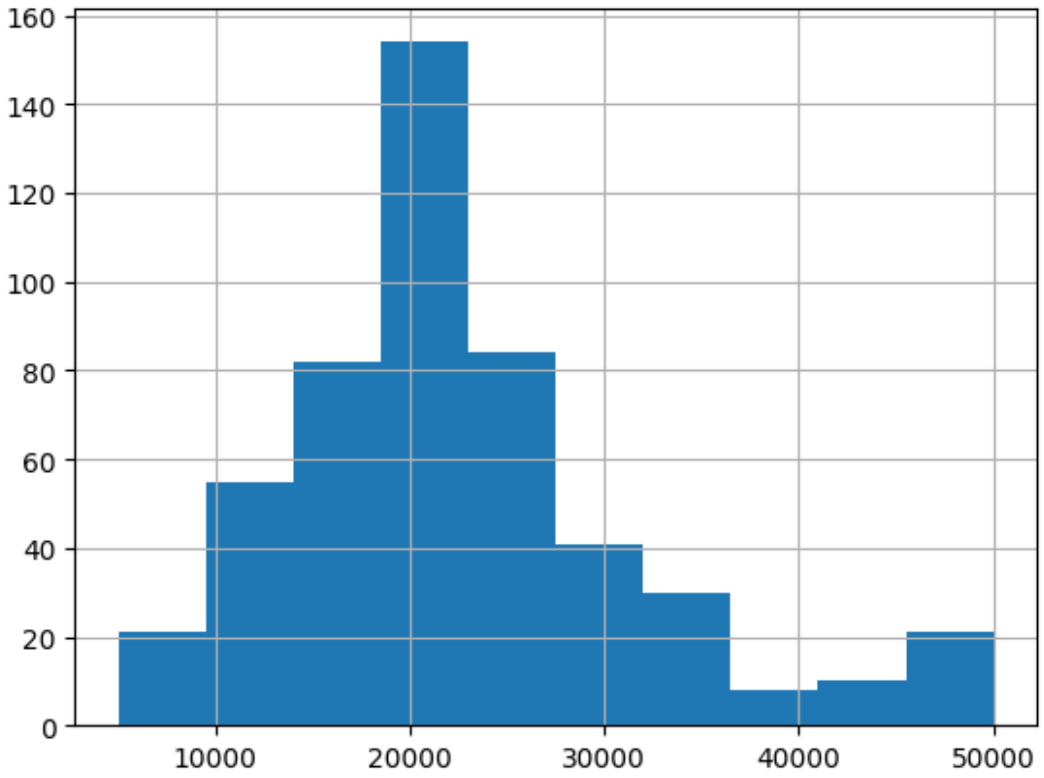
Начнем с функции `read_boston_dataset`, которая считывает датасет с диска. В данных первые 14 строчек не относятся к данным, а описывают колонки, для их пропуска полезно использовать `skiprows` в функции `read_csv`.

```
def read_boston_dataset(path_to_csv: str, shuffle: bool = True) ->
np.ndarray:
    """Функция для чтения данных с диска, а также их случайного
перемешивания

    Parameters
    ----------
    path_to_csv: Путь к файлу boston.csv
    shuffle: Если True, то перемешивает данные

    Return
    ------
    dataframe: Данные в формате DataFrame
    """
    dataframe = pd.read_csv(path_to_csv,skiprows = 14)
    if shuffle:
```

```
        dataframe = dataframe.sample(frac=1).reset_index(drop=True)
        return dataframe


boston_dataset = read_boston_dataset("boston.csv")
boston_dataset.head()
```

```
      crim    zn  indus  chas    nox     rm   age     dis  rad    tax
\
0  0.03502  80.0   4.95     0  0.411  6.861  27.9  5.1167    4  245.0

1  0.16439  22.0   5.86     0  0.431  6.433  49.1  7.8265    7  330.0

2  0.33983  22.0   5.86     0  0.431  6.108  34.9  8.0555    7  330.0

3  0.10008   0.0   2.46     0  0.488  6.563  95.6  2.8470    3  193.0

4  0.15936   0.0   6.91     0  0.448  6.211   6.5  5.7209    3  233.0


   ptratio       b  lstat     medv
0     19.2  396.90   3.33  28500.0
1     19.1  374.71   9.52  24500.0
2     19.1  390.18   9.16  24300.0
3     17.8  396.90   5.68  32500.0
4     17.9  394.46   7.44  24700.0
```

```
# Посмотрим на данные чуть ближе
# Оценим распределение цен в датасете

pyplot.hist(boston_dataset["medv"])
pyplot.grid(visible=True)
pyplot.show()
```



Вопрос: сделайте 2-3 вывода относительно цен.

1. Нормальное распределение с небольшим смещением влево
2. Есть небольшой подъем ближе к значению в 5 * 10^5
3. ...

По аналогии с задачей классификацией, необходимо выделить тренировочную и тестовую выборку. Реализуйте для этого функцию `prepare_boston_dataset`

```
def prepare_boston_dataset(
    dataset: pd.DataFrame, label_col_name: str = "medv", test_size:
float = 0.1
) -> tuple[np.ndarray, ...]:
    """Функция для выделения таргета и признаков,
    а также разделения на тренировочную и тестовую части.

    Parameters
    ----------
    dataset: DataFrame с датасетом
    label_col_name: Название колонки с таргетом
    test_size: доля тестовой выборки относительно всего датасета

    Return
    ------
        4 numpy массива: X_train, X_test, y_train, y_test
        X_train, X_test -- матрицы признаков размером [n_elements; 13]
```

```
        y_train, y_test -- массивы с ценами размером [n elements]
    """
    X = dataset.drop(columns=[label_col_name])
    y = dataset[label_col_name]

    # Разделение на тренировочную и тестовую выборки
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=42)

    # Преобразование в numpy массивы
    X_train = X_train.to_numpy()
    X_test = X_test.to_numpy()
    y_train = y_train.to_numpy()
    y_test = y_test.to_numpy()

    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test =
prepare_boston_dataset(boston_dataset)

# Код ниже проверяет правильность подготовки данных
# Если он упал, то надо исправить функцию выше

assert X_train.shape == (455, 13) and y_train.shape == (455,)
assert X_test.shape == (51, 13) and y_test.shape == (51,)
```

Аналогично задаче классификации, начнем с наивного решения. Для задаче регрессии можно использовать, например, среднее значение по датасету. Однако вы можете предложить и свою оценку на основе анализа графика и вывода выше.

Реализуйте методы `fit` и `predict` у класса ниже.

```
class MeanRegression:
    def __init__(self):
        self.mean_value = None

    def fit(self, X: np.ndarray, y: np.ndarray):
        """Функция обучения наивной модели.
        Она получает на вход X и y,
        чтобы иметь схожий интерфейс с другими моделями.

        Функция определяет среднюю величину таргета
        и сохраняет его в mean_value

        Parameters
        ----------
        X: признаки, не используются
        y: таргет, целые числа, одномерный массив
        """
        # Your code here
```

```
        self.mean_value = np.mean(y)

    def predict(self, X: np.ndarray) -> np.ndarray:
        """Функция для предсказания классов

        Parameters
        ----------
        X: элементы, для которых надо предсказать значение
            матрица размером [n_elements; n_features]

        Return
        ------
            предсказанные значения для каждого элемента
            numpy массив размером [n_elements]
        """
        if self.mean_value is None:
            raise RuntimeError("Call fit before predict")
        # Your code here
        return np.full(X.shape[0], self.mean_value)
```

Обучим "наивную" модель и оценим ее качество.

В задаче регресии также существует большое множество метрик. Ознакомиться с ними можно, например, тут.

В нашем случае мы также будем использовать две метрики:

1. MSE (Mean Squared Error) — среднее квадратичное отклонение, интуитивно понятная метрика, но не всегда хорошо интерпретируется.
2. R2-score — "нормированная" MSE, не имеет границы снизу, 0 в случае предсказания среднего значения и 1 для идеальной работы.

```
def print_regression_report(y_test, y_pred):
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"MSE: {mse:.2f}", f"R2-score: {r2:.2f}", sep="\n")

model_mean = MeanRegression()
model_mean.fit(X_train, y_train)
y_pred_mean = model_most_common.predict(X_test)

print_regression_report(y_test, y_pred_mean)
```

```
MSE: 507854313.73
R2-score: -8.74
```

Получили низкое качество, поэтому перейдем к более серьезным моделям.

Одна из них — это линейная регрессия. Интутивно простая модель, но крайне выразительно и часто применяющаяся в различных вариациях и модификациях. Линейная регрессия предполагает линейную зависимость между признаками и таргетами и описывается следующей формулой:

$$y = w_1 * x_1 + w_2 * x_2 + \ldots + w_n + x_n + b = \sum_{i=1}^{n} w_i x_i + b$$

Здесь y — это таргет, $x_1, \ldots x_n$ — признаки, а $w_1, \ldots, w_n$ и b — параметры модели.

В ходе тренировки модели эти параметры автоматически подбираются под обучающие данные.

Конечно, предполагать линейную зависимость между признаками и таргетом во многих случаях бывает невозможно. Для этого можно воспользоваться специальными ядрами, нелинейными преобразованиями, для обработки данных, созданием полиномиальных признаков или воспользоваться одной из модификацией уже реализованной в `sklearn`.

Обучим `LinearRegression` из `sklearn`

```
model_v1 = LinearRegression(n_jobs=-1)
model_v1.fit(X_train, y_train)
y_pred_v1 = model_v1.predict(X_test)

print_regression_report(y_test, y_pred_v1)
```

```
MSE: 15092187.57
R2-score: 0.71
```

Качество модели значительно лучше.

Давайте проанализируем полученную модель, а именно посмотрим какие веса w получились для каждого признака:

```
feature_names = boston_dataset.columns[:-1]
coefs = model_v1.coef_

for name, cf in sorted(zip(feature_names, coefs), key=lambda x: x[1],
reverse=True):
    print(f"{name}\t{cf}")
```

```
rm    3924.963853678393
chas  2471.842574756806
rad   323.1081101787548
zn    42.48742846649323
indus 29.69209021534287
b     8.669252325819144
age   1.3324460361250665
tax   -12.962464637243919
crim  -121.80752746883128
lstat -537.1537445177411
```

```
ptratio    -989.9323485708026
dis   -1547.506522865456
nox   -18528.43321262043
```

Вопрос: как можно интерпретировать полученный список?

Ответ: Положительные коэффициенты означают, что увеличение значения данного признака приводит к увеличению целевой переменной. Отрицательные коэффициенты указывают на обратное: увеличение значения признака приводит к уменьшению целевой переменной. Чем больше по модулю коэффициент, тем сильнее влияние соответствующего признака на целевую переменную.

Внутри `sklearn` есть множество алгоритмов регрессии, попробуйте применить их для этой задачи.

Например, можно взглянуть на `GradientBoostingRegressor`, довольно мощный алгоритм, но требующий детальной настройки.

В этой задаче можно получить R2-score больше 0.9, удачи!

Кластеризация

Последний блок нашей практики посвящен задаче кластеризации, задаче где отсутствуют таргеты и необходимо уметь группировать данные в осмысленные блоки. Примерами задачи кластеризации может служить разбиение новостей по разным темам или выявление пользователей в соц. сетях с общими интересами.

Мы применим кластеризацию к картинкам, что может быть полезно, если необходимо ее сжать.

Выберите любую картинку, может быть любимый шаблон мема или чья-то фотография. Пример подходящей картинки прикреплен к практике, в файле `image.jpg`.

```
def read_image(image_path: str) -> np.ndarray:
    with Image.open(image_path) as img:
        data = np.array(img)
    return data


def show_image(image: np.ndarray) -> np.ndarray:
    pyplot.axis("off")
    pyplot.tight_layout()
    pyplot.imshow(image)
    pyplot.show()

# Разместите картинку рядом с тетрадкой
# И укажите ее название в переменной ниже

IMAGE_NAME = "image.jpg"
```

```
image = read_image(IMAGE_NAME)
height, width = image.shape[:2]

show_image(image)
```

Картинка в памяти хранится как трехмерный массив `[h; w; 3]`, однако алгоритмы кластеризации требуют от нас двумерный массив `[n_samples; n_features]`. В случае картинок, `n_features` — 3, RGB код цвета каждого пикселя, а `n_samples` общее число пикселей.

Реализуйте функцию `preprocess_image`, которая получает картинку и возвращает нужный двумерный массив.

```
def preprocess_image(image: np.ndarray) -> np.ndarray:
    """Функция для препроцессинга картинки

    Parameters
    ----------
    image: исходная картинка
        массив размером [h; w; 3]

    Return
```

```
    ------
        матрица размером [n_pixels; 3]
    """
    # Your code here

    h, w, _ = image.shape
    return image.reshape(h * w, 3)

X_train = preprocess_image(image)

assert X_train.shape == (height * width, 3)
```

В качестве первого алгоритма возьмем `KMeans`. Его идея близка к алгоритму классификации "K ближайших соседей", считаются попарные расстояния между точками и наиболее близкие объединяются в кластеры

```
k_means = KMeans(n_clusters=5, n_init=1, random_state=RANDOM_SEED)
k_means.fit(X_train)
```

```
KMeans(n_clusters=5, n_init=1, random_state=12648430)
```

Заменим каждый цвет на картинке на средний цвет кластера, куда попал соответствующий кластер.

Для этого реализуйте функцию `replace_to_centroid`, которая принимает полученные индексы кластеров и цвета кластеров и возвращает цвета для каждой точки.

```
def replace_to_centroid(
    predicted_cluster: np.ndarray, centroids: np.ndarray
) -> np.ndarray:
    """Функция для получения центроиды кластера по ее индексу

    Parameters
    ----------
    predicted_cluster: предсказанные кластеры
        массив размером [n_samples]
        каждое значение от 0 до n_clusters
    centroids: центры кластеров
        массив размером [n_clusters; 3]

    Return
    ------
        матрица размером [n_samples; 3]
    """
    # Your code here
    return centroids[predicted_cluster]

predicted_clusters = k_means.predict(X_train)
X_predicted = replace_to_centroid(predicted_clusters,
k_means.cluster_centers_)
```

```
assert X_predicted.shape == X_train.shape
```

Приведем матрицу обратно к формату картинки и посмотрим, что получилось!

```
new_image = X_predicted.reshape(height, width, 3).astype(np.int32)
show_image(new_image)
```

Изучите другие алгоритмы кластеризации, доступные в `sklearn`: алгоритмы.

Для применения к нашей задаче, необходимо выбрать такой, где задается число кластеров, обычно параметр называется `n_clusters`. Помните, что задача кластеризации трудная с вычислительной точки зрения, домашний ПК не всегда может с ней справиться.

Поэкспериментируйте с другими алгоритмами и сравните, как они ведут себя относительно `KMeans` для задачи сжатия изображений.

```

```