Оглавление

[АЛГОРИТМ ДЕЙСТВИЙ 1](#_Toc178266017)

[NumPY 2](#_Toc178266018)

[PANDAS 8](#_Toc178266019)

[MATPLOTLIB 12](#_Toc178266020)

[SEABORN 13](#_Toc178266021)

[SCIKIT-LEARN (SKLEARN) 15](#_Toc178266022)

# АЛГОРИТМ ДЕЙСТВИЙ

1. Предобработка данных
   1. Удаление неинформативных столбцов
   2. Заполнение NaN-значений
   3. Преобразование категориальных признаков
2. Создание новых признаков
3. Анализ целевой переменной и признаков
4. Анализ влияния признаков на целевую переменную
5. Подготовка данных к обучению и валидации

* Разделение данных на train/test
* Масштабирование данных

# NumPY

import numpy as np

Поиск по документации

np.lookfor(‘mean value of array’) # выдаст список функций

?np.ma.mean # подробная информация о функции

np.con\*? # поиск функции по началу

Создание массивов

arr = np.array([1, 2, 3]) # в скобках список или кортеж

Можно указать параметр dtype=’float64’

np.arange([start,] stop[, step,][, dtype]) # (лучше для int) создание массива с равномерно распределенными значениями в заданном интервале, не включает конечную точку. Step п/у = 1

np.linspace (start, stop, num=50) # (лучше для float) создание массива (n=num) с равномерно распределенными значениями между двумя заданными точками, включает конечную точку

np.logspace (0, 3, 12) # последовательность из 12 чисел с постоянным шагом по логарифмической шкале от 100 до 103

np.zeros((n, m), int) # создаёт массив размером n\*m, заполненный нулями; можно указать тип данных (по умолчанию float)

np.zeros\_like(b) # создает нулевой массив такой же(размер и тип), как массив b

np.ones((n, m)) # массив размером n\*m, заполненный единицами

np.empty((n, m))# возвращает пустой (не инициализированный) массив размером n\*m. Он заполнен "мусором"

np.full((n, m), x) # создает массив n x m, заполненный x

np.eye(n) # создаёт единичную матрицу размером n x n

np.diag(arr) # создаёт диагональную матрицу с элементами arr на главной диагонали

Генерация псевдослучайных чисел

np.random.seed(seed) # получение одинаковых случайных чисел при каждом запуске программы

np.random.rand(n, m) # массив чисел в диапазоне [0;1)

np.random.random((n, m)) # то же самое, но принимает кортеж и работает немного быстрее

np.random.randint(low, high, size=(n, m)) # массив целых чисел в диапазоне [low, high)

Количественные характеристики массивов

len(arr) # количество элементов по первой оси

arr.size # возвращает общее количество элементов массива

arr.ndim # количество измерений (осей) массива

arr.shape # возвращает кортеж, указывающий размер массива в каждом измерении

arr.dtype # возвращает тип элементов массива

Обращение к элементам по индексу

arr[1][1] или arr[1, 1]

Изменение формы массива

reshape(arr, (n, m)) # изменение формы массива

arr.shape = (new\_n, new\_m) # изменение формы массива

swapaxes(arr, axis1, axis2) # обмен осями

flat\_arr = np.ravel(arr) # стягивает многомерный массив в одномерный

Добавление элементов в массив

np.append(arr, [4, 5, 6])

Изменение типа данных элементов

arr\_new = arr.astype(np.float64)

Индексация по булевым значениям

selected\_elements = arr[condition]

Пример: arr[arr > 5] # элементы массива, большие пяти

np.any(arr) # проверка, есть ли в массиве хотя бы один True элемент

np.all(arr) #проверка, все ли элементы массива True

Примечание. Вместо arr можно передать condition, а также использовать параметр axis=0/1

Вставка элементов

np.insert(arr, obj, values, axis=None) # вставляет значения values (может быть массивом) по индексу obj в массив arr

np.append(arr, values, axis=None) # добавляет значения в конец существующего массива NumPy

Удаление элементов

np.delete(arr, obj, axis=None) # удаляет элементы из массива по индексу или срезу

**ВАЖНО! Функции insert, append, delete НЕ inplace функции! Они НЕ меняют исходный массив!**

Объединение массивов

* np.concatenate([arr1, arr2]) # можно указать параметр axis=0/1
* np.hstack([arr1, arr2]) # слияние по горизонтали. Эквивалентно np.concatenate() с axis=0.
* np.vstack([arr1, arr2]) # слияние по вертикали
* np.column\_stack([arr1, arr2]) # объединяет 1D массивы как столбцы в 2D массив

Расщепление массива

numpy.split(arr, index\_or\_sections, axis=0) # разделяет массив на несколько подмассивов

Если indices\_or\_sections — целое число N, массив будет разделен на N равных массивов по axis. Если такое разделение невозможно, выдается ошибка

Если indices\_or\_sections представляет собой одномерный массив отсортированных целых чисел, записи указывают, где по axis массив разделен (до указанных индексов). Например, [2, 3] для axis=0 приведет к

* arr[:2]
* arr[2:3]
* arr[3:]

np.array\_split(arr) # похожа на np.split(), но может делить массив на неравные части

np.hsplit(arr), np.vsplit(arr), np.dsplit(arr) # для разделения по горизонтали, вертикали и глубине соответственно

Поиск уникальных элементов

np. unique(arr) # возвращает уникальные значения в массиве в отсортированном порядке

Дополнительные параметры:

return\_index=True # возвращает массив индексов первого вхождения

return\_counts=True # количество вхождений

np.argmax(arr) # возвращает индекс максимального элемента

Операции линейной алгебры

a \* b # поэлементное умножение

a @ b или a.dot(b) # матричное умножение

x, y = numpy.meshgrid(u, v) # используется для создания координатной сетки из одномерных массивов

np.trace(arr) # возвращает сумму элементов вдоль главной диагонали массива

(?) from numpy import linalg as LA

LA.norm(A) # норма матрицы

np.linalg.det(arr) # определитель матрицы

inv\_a = np.linalg.inv(A) # обратная матрица

Система линейных уравнений

b = np.array([[]]) # коэффициенты

b = np.array([]) # вектор

x = LA.solve(a, b)

Унарные операции с массивами

np.sum(arr) # сумма элементов массива

np.cumsum(arr) или arr.cumsum() # кумулятивная сумма элементов: для каждого элемента вычисляется сумма всех предыдущих элементов, включая текущий. Возвращает массив!

np.prod(arr) # произведение элементов массива

np.max(arr) # максимальный элемент массива

np.min(arr) # минимальный элемент массива

np.abs(arr), np.fabs(arr) # взятие модуля, fabs – для float

np.sqrt(arr) # квадратный корень

np.square(arr) # возведение в квадрат

np.exp(arr) # e в степени

np.log(arr), np.log10(arr), np.log2(arr) # логарифмы

np.sign(arr) # возвращает знаки элементов

np.sin(arr), np.cos(arr), np.tan(arr) # тригонометрия

np.isnan(arr) # возвращает булев массив, показывающий, является ли каждый элемент NaN

np.isfinile(arr) # возвращает булев массив, показывающий, является ли каждый элемент конечным числом

np.isinf(arr) # возвращает булев массив, показывающий, является ли каждый элемент бесконечным числом

Округление элементов

* np.ceil(arr) # округление по верхней границе
* np.floor(arr) # округление по нижней границе
* np.rint(arr) # округление до ближайшего целого
* np.modf(arr) # возвращает целую и дробную части каждого элемента в двух отдельных массивах
* np.round(arr, decimals=0(п/у)) # округляет до decimals знаков после запятой

Бинарные функции

np.isin(arr1, arr2) # проверка наличия элементов из arr1 в arr2 (возвращает True/False массив)

np.add(arr1, arr2) # поэлементная сумма массивов

np.subtract(arr1, arr2) # разность

np.multiply(arr1, arr2) # произведение

np.divide(arr1, arr2) # деление

np.floor\_divide(arr1, arr2) # деление и округление вниз

np.power(arr1, arr2) # возводит элементы arr1 в степени arr2

np.maximum(arr1, arr2) # поэлементные максимумы

np.fmax(arr1, arr2) # игнорирует NaN

np.minimum(arr1, arr2) # поэлементные минимумы

np.fmin(arr1, arr2) # игнорирует NaN

np.mod(arr1, arr2) # остаток от деления arr1 на arr2 (или число)

np.copysign(arr1, arr2) # копирует знак значения из arr2 в arr1

np.greater(arr1, arr2) # возвращает булев массив после поэлементного сравнения, если arr1 > arr2

np.greater\_equal(arr1, arr2) # arr1 >= arr2

np.less(arr1, arr2) # arr1 < arr2

np.less\_equal(arr1, arr2) # arr1 <= arr2

np.equal(arr1, arr2) # arr1 == arr2

np.not\_equal(arr1, arr2) # arr1 != arr2

np.where(condition, arr1, arr2) # выполнение условного поэлементного выбора значений из двух массивов

Логические операции

Возвращают булев массив

Пример: print(arr1 > arr2)

np.logical\_and(arr1, arr2) # возвращает булев массив: поэлементно если True и True – будет True

np.logical\_or(arr1, arr2) # логическое или

np.logical\_xor(arr1, arr2) # исключающее или: возвращает True, если значения различны

np.logical\_not(array) # логическое отрицание

Сортировка массива и прочие операции

np.sort(arr) # можно указать axis. **Не изменяет массив arr!**

a.sort() # **Изменяет сам объект!**

np.unique(arr) # возвращает уникальные элементы массива

параметр return\_index=True # вернет массив индексов первого вхождения

параметр return\_counts=True # вернет количество вхождений

np.intersect1d(arr1, arr2) # возвращает отсортированный массив уникальных элементов, **общих** для двух массивов

np.union1d(arr1, arr2) # возвращает отсортированный массив уникальных элементов, **объединённых** из двух массивов

np.in1d(arr1, arr2) # возвращает булев массив той же формы, что arr1, указывающий, содержатся ли элементы из arr1 в arr2

np.setdiff1d(arr1, arr2) # возвращает уникальные элементы, которые есть в arr1, но отсутствуют в arr2

np.setxor1d(arr1, arr2) # возращает уникальные элементы, которые присутствуют в arr1 или arr2, но не в обоих массивах одновременно

# PANDAS

YouTube: What is pandas? (<https://www.youtube.com/playlist?list=PL5-da3qGB5ICCsgW1MxlZ0Hq8LL5U3u9y&si=6faVt2CiaaA73QAQ>)

Documentation: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_table.html>

import pandas as pd

1. Загрузка данных

Создание таблицы из файла

data = pd.read\_csv("/content/titanic.csv")

Загрузка таблицы с компьютера (для Google colab)

from google.colab import files

uploaded = files.upload() # здесь появится кнопка для загрузки файла

data = pd.read\_csv('train.csv', index\_col="PassengerId")

Загрузка таблицы с сайта

data = pd.read\_csv(‘http://bit.ly/chiporders’)

Создание шапки таблицы (названия столбцов)

user\_cols = [‘user\_id’, ‘age’, ‘gender’]

pd.read\_table(‘URL’, names=user\_cols)

Создание колонки с индексами

data = pd.read\_csv("/content/titanic.csv", index\_col='PassengerId') # index\_col задает название колонки в файле, которая будет считаться колонкой индексов

data.set\_index('S.No', inplace=True) # изменяем в уже существующем DataFrame

1. Сохранение данных в файл

submission.to\_csv('submission\_lr.csv', index=None) # когда index=None, индекс DataFrame не будет сохранен в CSV-файле

1. Вывод данных построчно

data.head(n) # вывод первых n строк (если не указано n, по умолчанию – 5)

data.tail(n) # вывод последних n строк

data.sample(n) # выводит рандомные строки в таблице

1. Доступ к элементам по индексам

data.loc[‘row\_name’] # обращение к строке по наименованию

data.iloc[0] # обращение к строке по индексу

data.iloc[[1,7,12]] # вывести 1, 7, 12 строки

data[‘Age’] # столбец Ages

data.Age # то же самое, но не всегда работает

data.loc[1, 'Survived'] # элемент на пересечении 1 строки и столбца Survived

data.loc[(data.Sex == 'male')] # фильтрация по полу

data.loc[(data.Survived == 0), ‘Survived’] = -1 # замена значений в столбце

1. Описание выборки

data.shape # выводит количество строк и столбцов

data.index # выводит индексы строк

data.columns # вывод списка с названиями столбцов

data.dtypes # возвращает типы данных по столбцам

data.info() # сообщает нам, сколько в наших данных строк и столбцов, и для каждого столбца дает его название, тип данных в нем и количество непустых значений в нем

data.describe() # статистические показатели по столбцам

data.values # используется для получения представления данных DataFrame в виде массива NumPy. Возвращает только значения, содержащиеся в DataFrame, и игнорирует метки строк и столбцов

data[‘Age’].value\_counts() # количество уникальных значений в столбце

1. Как переименовать столбец

1 способ

data.rename(columns = {‘Old\_Name’ : ‘New\_Name’, ‘Old\_name\_2’ : ‘New\_Name\_2’}, inplace=True)

2 способ

new\_cols = [‘city’, ‘colors’, ‘shape’, ‘state’, ‘time’]

data.columns = new\_cols

Отредактировать сразу все названия столбцов

data.columns = data.columns.str.replace(' ', ‘\_’) # замена пробела на подчеркивания во всей шапке

При загрузке файла

data = pd.read\_csv(‘URL’, names=new\_cols, header=0)

1. Удаление столбца

data.drop(‘City’, axis=1, **inplace=True**)

или:

processed\_data = data.drop(‘City’, axis=1)

data.drop([‘City’, ‘State’], axis=1, **inplace=True**)

1. Удаление строк

data.drop([0, 1], axis=0, **inplace=True**) # нумерация с нуля, как в iloc

1. Сортировка по столбцам

data[‘Name’].sort\_values() # сортировка по столбцу Name возвращает Series с отсортированным столбцом, но не меняет его в исходной таблице

ascending=False # параметр для сортировки по убыванию

data.sort\_values(‘Name’) # сортирует всю таблицу по столбцу Name и возвращает изменённую таблицу

data.sort\_values([‘Name’, ‘Age’]) # сортировка по нескольким столбцам

1. Соединение двух DataFrame

new\_data = pd.concat([data1, data2], axis=1)

1. Подготовка данных для модели – значения NaN

Алгоритм подготовки данных

* 1. Удаление ненужных признаков
  2. Заполнение пропусков
  3. Перевод категориальных признаков в числовые

data.isna().sum() # количество значений NaN в каждом столбце

data.isna().sum(axis=1) # количество значений NaN в каждой строке

data.dropna(**inplace=True**) # удаление строк с любыми NaN

data.dropna(axis=1) # удаление столбцов с любыми NaN

Заполнение NaN значением медианы

median\_age = data["Age"].median() # вычисление медианы столбца

data["Age"] = data["Age"].fillna(median\_age) # заполнение пропущенных значений медианой

array = df.values # метод .values используется для получения данных из DataFrame или Series в виде numpy-массива. Он возвращает только значения, без индексов и названий столбцов.

1. One-hot encoding

embarked\_columns = pd.get\_dummies(data[‘Embarked’], prefix=‘Embarked’).iloc[:, 1:] # создали новые колонки с префиксами Embarked, заполненные 0 и 1, и удалили первую категорию

data = pd.concat([data, embarked\_columns], axis=1) # слияние новых столбцов с таблицей

data.drop(‘Embarked’, axis=1) # удаление исходного столбца Embarked

2 способ (в одно действие):

data = pd.get\_dummies(data, columns=[‘Sex’, ‘Embarked’], drop\_first=True) # здесь уже и удаляется исходный столбец. Когда drop\_first=True, функция удаляет первый уровень (первую категорию) каждой категориальной переменной при создании фиктивных переменных (например, для пола будет создана только категория «женский»)

3 способ: с помощью scikit-learn

См. далее в scikit-learn

# MATPLOTLIB

import matplotlib.pyplot as plt

или: from matplotlib import pyplot as plt

Пример создания графика

plt.figure(figsize = (10,7)) # размер фигуры (полотна) – необзязательный параметр

plt.plot(x, y) # создание графика

plt.show() # отображение графика

plt.savefig(‘filename.png’) # сохранение графика в виде рисунка

Команды для вызова диаграмм

* plt.scatter() # точечный график
* plt.plot() # линейный
* plt.hist(bins=n) # гистограмма, bins (необяз. параметр) - это интервал
* plt.boxplot([data1, data2]) # «ящик с усами»
* plt.bar(), plt.barh() # столбчатая диаграмма
* plt.pie(data, <labels=data.index>) # круговая диаграмма, параметр autopct='%1.1f%%' – процент с одним знаком после запятой

plt.hist(data[‘Age’]) # диаграмма по колонке датасета

data[‘Age’].hist() # то же самое из pandas

Оформление графика

plt.title(“Мой график”) # название графика

plt.xlabel(“Ось\_y”) # название оси x

plt.ylabel(“Ось\_x”) # название оси y

plt.xlim([-5, 20]), plt.ylim() # ограничение диапазона осей

plt.legend() # добавление легенды

plt.grid() # нанесение сетки

plt.plot(x, y, color=’red’) # замена цвета

plt.style.use(‘ggplot’) # изменение стиля (список стилей: plt.style.available)

Два графика с легендой

plt.plot(x,y, color="red", label='данные y') # label будет виден в легенде

plt.plot(x,z, color="green", label='данные z')

plt.legend(fontsize=15)

# SEABORN

import seaborn as sns

sns.barplot(x="sex", y="survived", data=titanic\_dataset) # гистограмма

sns.scatterplot() # диаграмма рассеяния

sns.lineplot() # линейный график

Обязательные параметры:

* data
* x
* y

Дополнительные параметры:

* color=’red’ # задаёт цвет
* palette='hls' # обеспечивает создание цветовой палитры с равномерно распределенными оттенками

# тепловая карта

sns.heatmap(array, annot=True)

параметр annot=True: добавление аннотаций в ячейки тепловой карты

Взаимное распределение признаков

sns.set() # подключает встроенные темы seaborn

sns.pairplot(data, kind=’scatter’, diag\_kind=’kde’) # взаимные графики на многих признаках

Матрица корреляций

correlation\_matrix = data.corr()

fig, ax = plt.subplots(figsize=(12, 9)) # plt.subplots() создает объект фигуры (fig) и массив осей (ax), на которых будут размещены графики

sns.heatmap(correlation\_matrix, vmax=.8, square=True)

 параметр vmax=.8 устанавливает максимальное значение для цветовой шкалы. Это означает, что любые значения корреляции выше 0.8 будут отображаться одним цветом (в зависимости от выбранной цветовой палитры), что помогает визуально упростить интерпретацию.

 параметр square=True заставляет ячейки тепловой карты быть квадратными, что улучшает визуальное восприятие

Несколько графиков (три круговые диаграммы) в ряд

fig, (ax1, ax2, ax3) = plt.subplots(figsize=(12, 9))

ax1.pie(data.values, labels=data.index, autopct=’%1.1f%%’) # то же самое для ax2 и ax3

ax1.set\_title(“Pclass = 1”) # название графика

# SCIKIT-LEARN (SKLEARN)

Официальная документация: <https://scikit-learn.org/stable/>

!pip uninstall -y scikit-learn # удалим более старую версию библиотеки

!pip install scikit-learn # установим версию поновее

Загрузка готовых датасетов

from sklearn.datasets import fetch\_california\_housing

x, y = fetch\_california\_housing (return\_X\_y=True) # параметр return\_X\_y=True указывает функции вернуть данные в виде отдельных массивов признаков и целевой переменной

Разделение выборки на train/test

X = data.drop(['age'], axis=1).values # удаляем метки из набора признаков + оставляем только значения признаков

y = data['age'].values # берем значения меток в виде numpy-массива

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

Параметры:

1. test\_size # определяет долю данных, которая будет отведена под тестовую выборку
2. (train\_size # альтернативный параметр для задания размера обучающей выборки. Если указан train\_size, test\_size будет автоматически вычислен)
3. random\_state # устанавливает начальное значение генератора случайных чисел для обеспечения воспроизводимости результатов. Если указать одно и то же значение при каждом запуске, результат будет одинаковым.
4. shuffle # указывает, нужно ли перемешивать данные перед разделением. По умолчанию установлено значение True.
5. stratify # позволяет указать массив меток классов (обычно это целевая переменная), по которому будет производиться разбиение. Это гарантирует, что распределение классов в обучающей и тестовой выборках будет таким же, как и в исходном наборе данных.

Обработка данных

1. Стандартизация

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(x\_train)

standardized\_X = scaler.transform(x\_train)

1. Нормализация

from sklearn.preprocessing import Normalizer

scaler = Normalizer().fit(x\_test)

normalized\_X = scaler.transform(x\_test)

1. Бинаризация

from sklearn.preprocessing import Binarizer

binarizer = Binarizer(threshold = 0.0).fit(X) # в данном случае порог 0.0

binary\_X = binarizer.transform(X)

1. One-hot encoding

from sklearn.preprocessing import OneHotEncoder

# Создание экземпляра OneHotEncoder

encoder = OneHotEncoder()

# Применение one-hot encoding к данным

encoded\_data = encoder.fit\_transform(data)

# Получение имен новых столбцов

encoded\_columns = encoder.get\_feature\_names\_out(input\_features=data.columns)

# Создание нового DataFrame с закодированными данными

encoded\_df = pd.DataFrame(encoded\_data, columns=encoded\_columns)

# Объединение с исходным DataFrame

result\_df = pd.concat([data, encoded\_df], axis=1)

1. Label Encoding

from sklearn.preprocessing import LabelEncoder

label\_encoding = LabelEncoder()

data[column] = label\_encoding.fit\_transform(data[column])

Линейная регрессия

from sklearn.linear\_model import LinearRegression # импортирует класс LinearRegression из модуля sklearn.linear\_model

# Шаг 1. создание модели – (объекта) экземпляра класса

lr = LinearRegression(normalize=True)

# Шаг 2. обучение модели

lr.fit(X\_train, y\_train) # метод fit обычно изменяет текущий объект модели

# Шаг 3. Предсказание на тестовых данных

y\_pred = lr.predict(X\_test)

kNN (k-Nearest Neighbors)

# Шаг 1. создание модели

from sklearn.neighbors import KNeighborsRegressor

model = KNeighborsRegressor(n\_neighbors=5) # регрессия

model = KNeighborsClassifier(n\_neighbors=3) # классификация

# Шаг 2. обучение модели

model.fit(X\_train, y\_train) # метод .fit() служит для обучения модели на основе данных. В случае KNN этот метод сохраняет элементы обучающих данных для последующего использования для предсказания ответов на новые элементы

# Шаг 3. Предсказание на тестовых данных

y\_pred = model.predict(X\_test) # метод .predict() делает предсказания на тестовых данных: для каждого тестового элемента находит k ближайших к нему соседей и выдает среднее значение целевой переменной этих k соседей в качестве ответа.

Метрики

В общем случае

from sklearn.metrics import metric\_name # import

metric\_name(<real\_target>, <predicted\_target>) # create\_instance

# Шаг 1. Импорт и получение предсказаний

from sklearn.metrics import mean\_squared\_error

from sklearn.metrics import r2\_score, mean\_absolute\_error

pred\_train = model.predict(X\_train) # предсказание на обучающей выборке

pred\_test = model.predict(X\_test) # предсказание на тестовой выборке

Метрики на обучающей выборке

MSE\_train = mean\_squared\_error(y\_train, pred\_train) # передаём значение целевой переменной и предсказание модели на обучающей выборке

RMSE\_train = np.sqrt(MSE\_train)

R2\_train = r2\_score(y\_train, pred\_train)

MAE\_train = mean\_absolute\_error(y\_train, pred\_train)

Метрики на тестовой выборке

MSE\_test = mean\_squared\_error(y\_test, pred\_test)

RMSE\_test = np.sqrt(MSE\_test)

R2\_test = r2\_score(y\_test, pred\_test)

MAE\_test = mean\_absolute\_error(y\_test, pred\_test)

Дерево решений

from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor(random\_state=44)

model.fit(X\_train, y\_train)

Логистическая регрессия

from sklearn.linear\_model import LogisticRegression # импортирует класс LogisticRegression из модуля sklearn.linear\_model

# Шаг 1. создание модели

logistic\_regression = LogisticRegression(C=1)

# Шаг 2. обучение модели

logistic\_regression.fit(X\_train, y\_train)

# Шаг 3. Предсказание на тестовых данных

y\_pred = logistic\_regression.predict(X\_test)

Кросс-валидация

from sklearn.model\_selection import cross\_validate

cv\_results = cross\_validate(model, X, y, cv=5)

Если нужно указать словарь метрик

from sklearn.model\_selection import cross\_validate

from sklearn.metrics import make\_scorer

scores = cross\_validate(model, X, y, cv=5,

scoring={'r2': make\_scorer(r2\_score),

'mean\_squared\_error': make\_scorer(mean\_squared\_error)},

return\_train\_score=True)

Параметры cross-validate:

 cv: стратегия кросс-валидации, которая определяет, как данные будут разбиты на обучающие и тестовые выборки (количество фолдов для K-Fold).

 scoring: одна или несколько метрик для оценки производительности модели. Можно указать строку с названием метрики (например, 'accuracy', 'f1'), или передать словарь с несколькими метриками.

Печатаем метрики из scores

print('R2 train mean = ', scores['train\_r2'].mean())

print('R2 test mean = ', scores['test\_r2'].mean())

print('MSE train mean = ', scores['train\_mean\_squared\_error'].mean())

print('MSE test mean = ', scores['test\_mean\_squared\_error'].mean())

Оптимизация модели

GridSearch — поиск лучших параметров в фиксированной сетке возможных значений.

from sklearn.model\_selection import GridSearchCV

knn = neighbors.KNeighborsClassifier()

params = {'n\_neighbors': [1, 2, 3], 'metric':['euclidean','cityblock']} # параметры, которые будем перебирать

grid = GridSearchCV(estimator = knn, param\_grid = params)

grid.fit(X\_train, y\_train)

print(grid.best\_score\_) # лучшее значение метрики

print(grid.best\_estimator\_.n\_neighbors) # при каком n\_neighbors был достигнут лучший результат

print(grid.best\_estimator\_.metric) # при каком estimator\_.metric был достигнут лучший результат

Параметры GridSearchCV:

* estimator — модель которую хотим обучать (алгоритм);
* param\_grid —какие параметры хотим подбирать, GridSearchCV на всех параметрах попробует сделать обучение;
* CV — сколько разрезов кросс-валидации мы ходим сделать;
* scoring — выбор метрики ошибки (для разных задач можно выбрать разные функции ошибки).