Московский государственный технический

университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Лабораторная работа №3

«Реализация линейной регрессии на Python»

Выполнил:	Проверил:
Студент ИУ5-34Б	Преподаватель каф. ИУ5
Бурдуковский И.О.	Гапанюк Ю. Е.
Подпись и дата:	Подпись и дата:

Москва, 2023 г.

Задание

Низкоуровневая реализация алгоритма линейной регрессии на Python с применением ООП.

Код программы

```
import pandas as pd
import numpy as np
from abc import ABC

class BaseLoss(ABC):
    def calc_loss(X:np.ndarray, y:np.ndarray, w:np.ndarray) -> float:
        raise NotImplementedError
    def calc_grad(X:np.ndarray, y:np.ndarray, w:np.ndarray) -> np.ndarray:
        raise NotImplementedError


class MSELoss(BaseLoss):
    def calc_loss(self, X: np.ndarray, y: np.ndarray, w: np.ndarray) ->
float:
        Q = ((np.linalg.norm(np.dot(X, w) - y)) ** 2) / len(y)
        return Q

    def calc_grad(self, X: np.ndarray, y: np.ndarray, w: np.ndarray) ->
np.ndarray:
        L = np.dot(X, w) - y
        Xt = np.transpose(X)
        Grad = 2 * np.dot(Xt, L) / len(y)
        return Grad

def gradient_descent(w_init: np.ndarray, X: np.ndarray, y: np.ndarray,
                     loss: BaseLoss, lr: float, n_iterations: int =
100000):
    W = []

    for i in range(n_iterations):
        w_init = w_init - lr*loss.calc_grad(X,y, w_init)
        W.append(w_init)
    return W


class LinearRegression1:
    def __init__(self, loss: BaseLoss, lr: float = 0.1) -> None:
        self.loss = loss
        self.lr = lr
        self.w = None
        self.g = None

    def fit(self, X: np.ndarray, y: np.ndarray) -> 'LinearRegression':
        X = np.asarray(X)
        y = np.asarray(y)
        X = np.hstack([X, np.ones([X.shape[0], 1])])
        shape_X = X.shape

        self.w = np.arange(1, shape_X[-1] + 1)
        self.g = gradient_descent(self.w, X, y, self.loss, lr=self.lr,
n_iterations=100000)
        return self.g[-1]

    def predict(self, X: np.ndarray) -> np.ndarray:
        # Проверяем, что регрессия обучена, то есть, что был вызван fit и в
нём был установлен атрибут self.w
```

```
        assert hasattr(self, "w"), "Linear regression must be fitted first"
        assert hasattr(self, "g"), "Linear regression must be fitted first"

        # добавляем столбец из единиц для константного признака
        X = np.hstack([X, np.ones([X.shape[0], 1])])
        y = np.dot(X, self.g[-1])

        return y

np.random.seed(1337)

n_features = 2
n_objects = 300
batch_size = 10
num_steps = 43
w_true = np.random.normal(size=(n_features,))
X = np.random.uniform(-5, 5, (n_objects, n_features))
X *= (np.arange(n_features) * 2 + 1)[np.newaxis, :]
y = X.dot(w_true) + np.random.normal(0, 1, (n_objects))
w_init = np.random.uniform(-2, 2, (n_features))

linregr = LinearRegression1(MSELoss(), lr=0.01)
linregr.fit(X, y)
xs = np.hstack([X, np.ones([X.shape[0], 1])])
print(MSELoss().calc_loss(xs, linregr.predict(X), linregr.w))
```

Результат

```
534.3924606415659
```