Кудинов Иван Михайлович, 3 курс, 3 группа, ТОиЗИ

Лабораторная работа №1.

Вариант №1.

Регрессия и её виды. Обучение регрессионной модели с помощью градиентного спуска

Цель работы:

Закрепление базовых теоретических основ по принципам работы и обучения алгоритмов машинного обучения с помощью градиентного спуска на примере простейшей линейной регрессии и её различных вариаций, а также получение навыков её программной разработки на примере решения задач из практики.

Условие задания:

Существуют данные о количестве отработанных часов в неделю и сообщаемом уровне счастья (по шкале от 0 до 100) для 20 разных людей. Найдите функциональную зависимость «Уровня счастья» y от «Количества отработанных часов» x. Пусть установлено, что зависимость между ними имеет полиноминальный характер (квадратичная зависимость).

Требования:

1. Визуализируйте имеющиеся данные на графике.
2. Найдите параметры регрессионной модели $\hat{b}$, $\hat{w}_0$ и $\hat{w}_1$ путём минимизации среднеквадратичной ошибки $MSE(\hat{b}, \hat{w}_0, \hat{w}_1)$ с помощью метода градиентного спуска.
3. Визуализируйте совместно график данных и полученную модель регрессии.
4. Визуализируйте график, на котором видно, как менялась ошибка с каждой итерацией градиентного спуска.
5. Имея полученную модель с $\hat{b}$, $\hat{w}_0$ и $\hat{w}_1$ предскажите значение $\hat{y}$ (уровень счастья) при $x = 28$ (количество отработанных часов). Посмотрите насколько близким получилось ваше предсказание с реальным значением y. Реальное значение $y = 94.3743425$ при данном x.

Ход выполнения работы:

Для обучения модели необходимы исходные данные. Для начала импортируем нужные библиотеки для удобной работы с моделью и инициализируем исходные данные для её обучения:

```
import numpy as np
import matplotlib.pyplot as plt


# инициализация исходных данных для обучения
x = np.array([6, 9, 12, 12, 15, 17, 21, 24, 24, 27, 30, 32, 36, 39, 42, 45, 48, 51, 57, 60])


    y = np.array([8.09101123, 26.63756318, 43.14175005, 45.92634805, 55.43083705, 63.80335312, 76.11847963, 84.36672354, 82.90842081, 91.17233418, 94.4116184, 98.1871826, 96.40834265, 99.29222179, 93.72201352, 92.1011422, 82.55116089, 77.59438803, 55.62001874, 41.33021583])
```

Далее визуализируем имеющиеся данные на графике. Для этого используем библиотеку matplotlib.

```
plt.figure(figsize=(10, 10))
plt.plot(x, y, 'ro')
plt.plot(x, y)
```

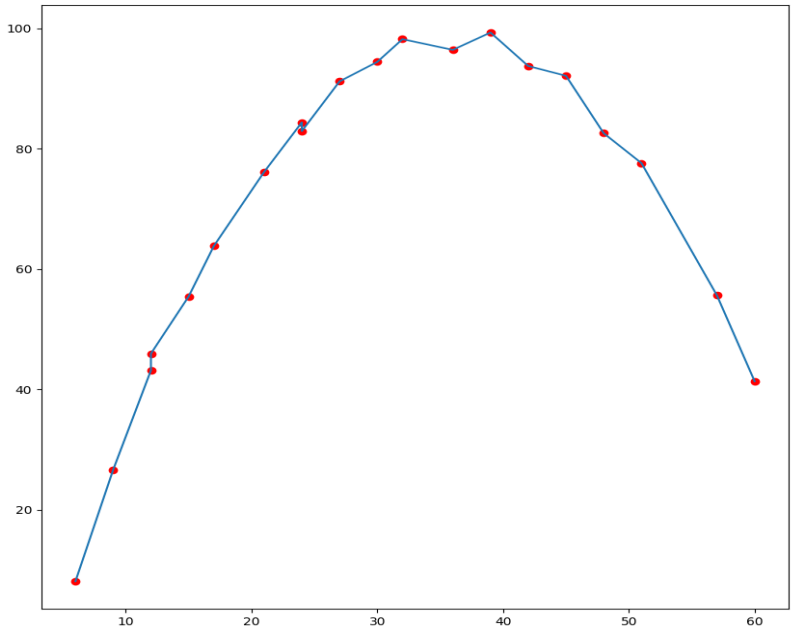

Рисунок 1 – Визуализация исходных данных на графике

Определим функцию:

```
def func(x, b, w0, w1):
    return b + w0 * x + w1 * x ** 2
```

Затем генерируем начальные значения параметров из нормального распределения:

```
# генерация начальных значений параметров из
нормального распределения
b = np.random.uniform(-0.5, 0.5, size=1)
w0 = np.random.uniform(-0.5, 0.5, size=1)
w1 = np.random.uniform(-0.5, 0.5, size=1)


print("b", b)
print("w0", b)
print("w1", b)
```

После чего необходимо настроить процесс обучения модели:

- Скорость обучения модели – условный параметр размера шага, определяет скорость изменения функции.

- Количество эпох – параметр, который определяет количество итераций обучения модели. Чем больше это значение – тем точнее модель.

```
# инициализация параметра скорости обучения
lr = 0.0000001
# количество эпох (количество итераций обновления наших
параметров)
n_epochs = 1000000
# лист для сохранения значений ошибки на каждой
итерации цикла обучения модели
mse_list = []
```

Теперь можно запускать цикл обучения модели:

```
# основной цикл обучения модели
for epoch in range(n_epochs):

    # делаем расчет y на основе сгенерированных
начальных значений параметров
    y_pred = func(x, b, w0, w1)

    # считаем функцию ошибки MSE
    mse = np.mean(((y - y_pred) ** 2))

    # сохраняем ошибку
    mse_list.append(mse)

    # Частная производная по b: -2 * 1/n * (y - y')
    b_grad = -2 * (y - y_pred).mean()  # для
коэффициента b
```

```
# Частная производная по w0: -2 * 1/n * ((y - y') * x)
    w0_grad = -2 * ((y - y_pred) * x).mean()  # для
коэффициента w0

    # Частная производная по w1: -2 * 1/n * ((y - y') *
x^2)
    w1_grad = -2 * ((y - y_pred) * x ** 2).mean()  #
для коэффициента w1


# обновляем параметры, используя коэффициент скорости
обучения
    b = b - lr * b_grad
    w0 = w0 - lr * w0_grad
    w1 = w1 - lr * w1_grad

    # выводим в консоль значение ошибки и параметров
модели на каждой 10000-й итерации цикла обучения
    if epoch % 10000 == 0:
    print('MSE: итерация ', epoch, ': ', mse)
    print('b: ', b, ' w0: ', w0, ' w1: ', w1)
```

На каждой итерации цикла происходит перерасчет значений y при текущих параметрах. Затем считается функция ошибки и результат добавляется в список. Потом считаются частные производные по b, w0, w1 и обновляются параметры, используя коэффициент скорости обучения. Каждую 10000-ю итерацию в консоль выводится значение ошибки.

Вывод на 990 000 итерации:

MSE: итерация 990000 : 54.81283546670089

b: [0.42631031] w0: [5.17367393] w1: [-0.0724204]

Как мы видим, величина ошибки составляет примерно 55.

После завершения цикла получим подобранные параметры. И на их основе рассчитаем линейную регрессию:

```
# выводим в консоль найденные в результате обучения
модели параметры
print('Найденный параметр b: ', b)
print('Найденный параметр w0: ', w0)
print('Найденный параметр w1: ', w1)

# находим линейную регрессию с учётом полученных ранее
параметров
y_pred = func(x, b, w0, w1)
```

Вывод:

Найденный параметр b: [0.42266679]

Найденный параметр w0: [5.17391281]

Найденный параметр w1: [-0.07242366]

Теперь выводим исходные данные и показываем на этом графике полученную линейную регрессию:

```
plt.figure(figsize=(10, 10))
plt.plot(x, y, 'ro')  # выведем наши данные
plt.plot(x, y_pred)  # построим линейную регрессию
plt.show()
```

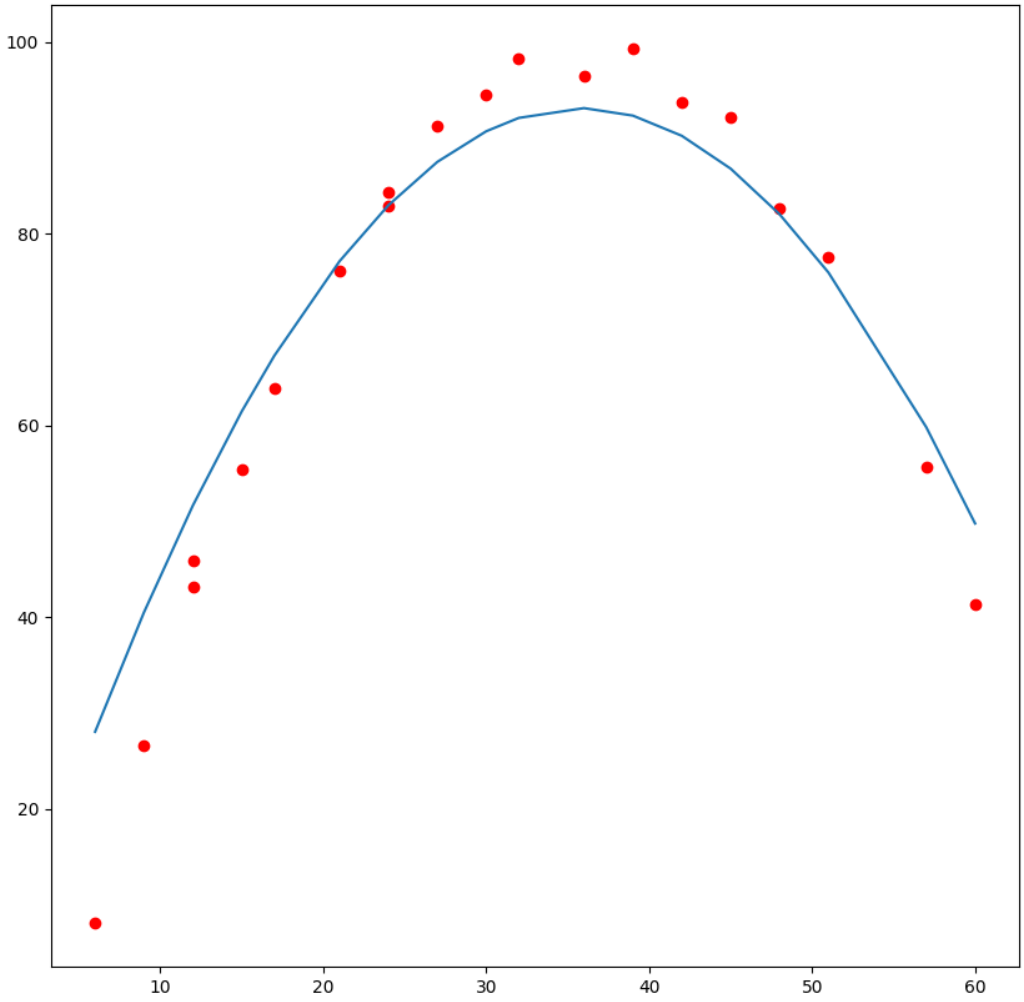

Рисунок 2 – Визуализация графика данных и полученной модели регрессии (при 2 500 000 итераций).

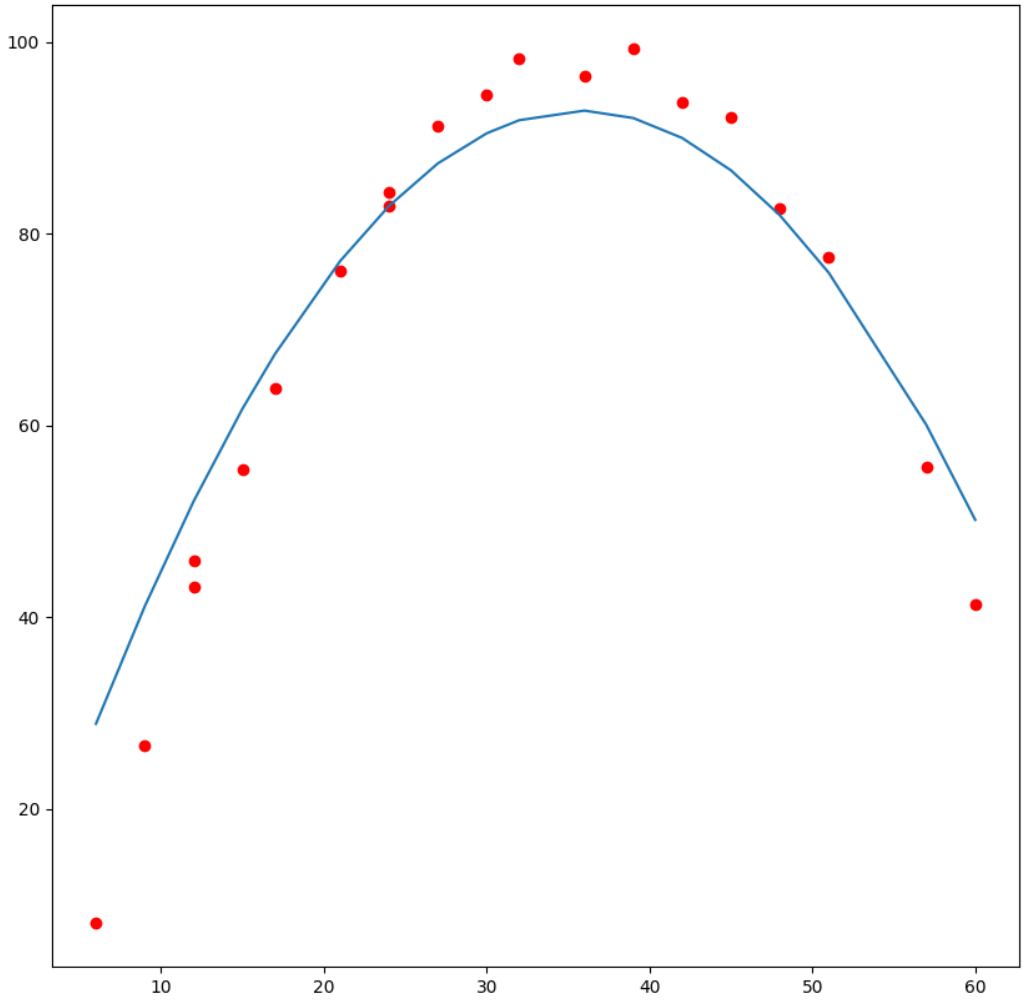

Рисунок 3 – Визуализация графика данных и полученной модели регрессии (при 1 000 000 итераций).

На графиках видно, что модель, обученная на 1 000 000 итерациях менее точная, чем модель, которая обучалась при 2 500 000 итераций.

После строим график изменения ошибки с каждой итерацией цикла обучения модели:

```
# строим график изменения ошибки с каждой итерацией
цикла обучения модели
plt.figure(figsize=(10, 10))
plt.plot(range(n_epochs), mse_list, 'ro')  # ось x -
количество эпох, ось y - значения ошибки
plt.title('Как менялась ошибка с каждой итерацией')
plt.show()
```

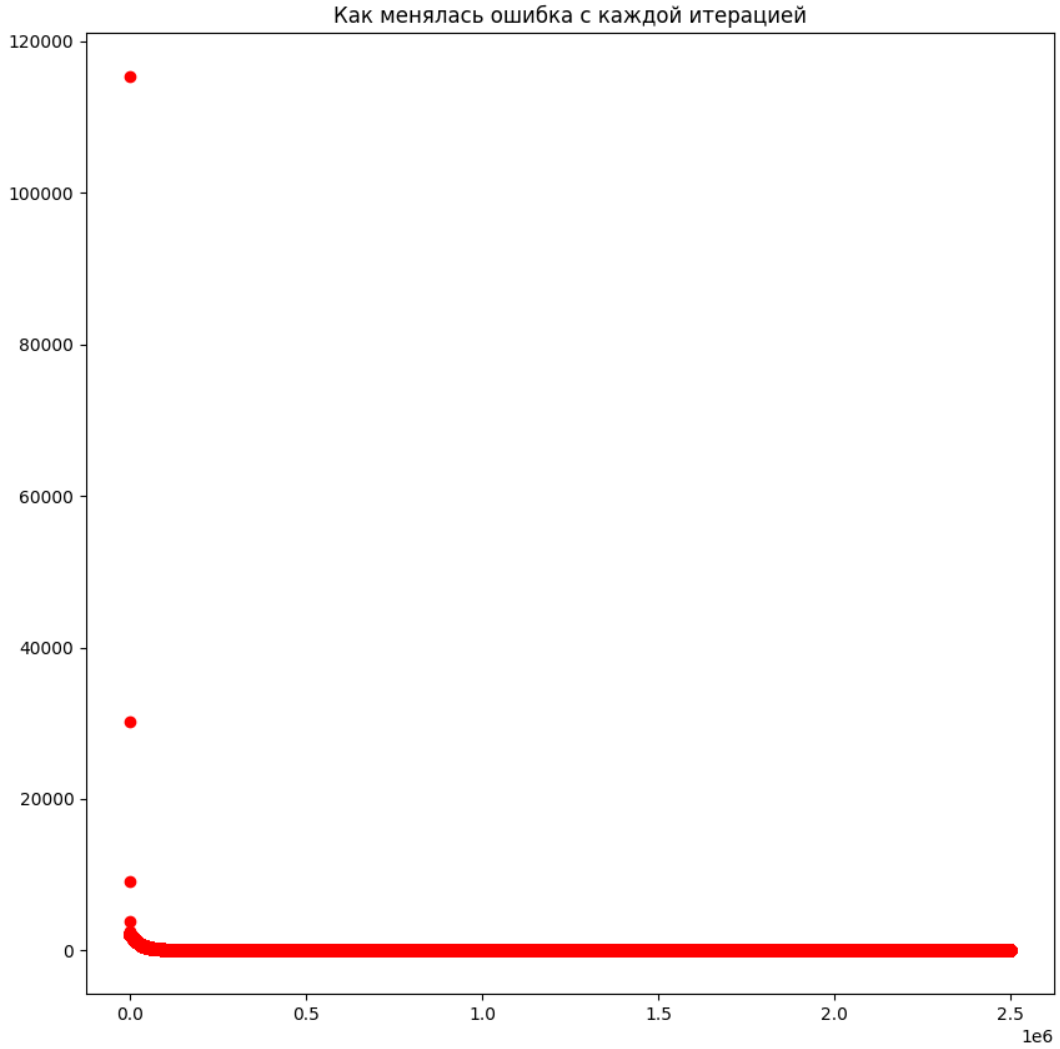

Рисунок 4 – Визуализация графика изменения ошибки с каждой итерацией цикла обучения модели (при 2 500 000 итераций).

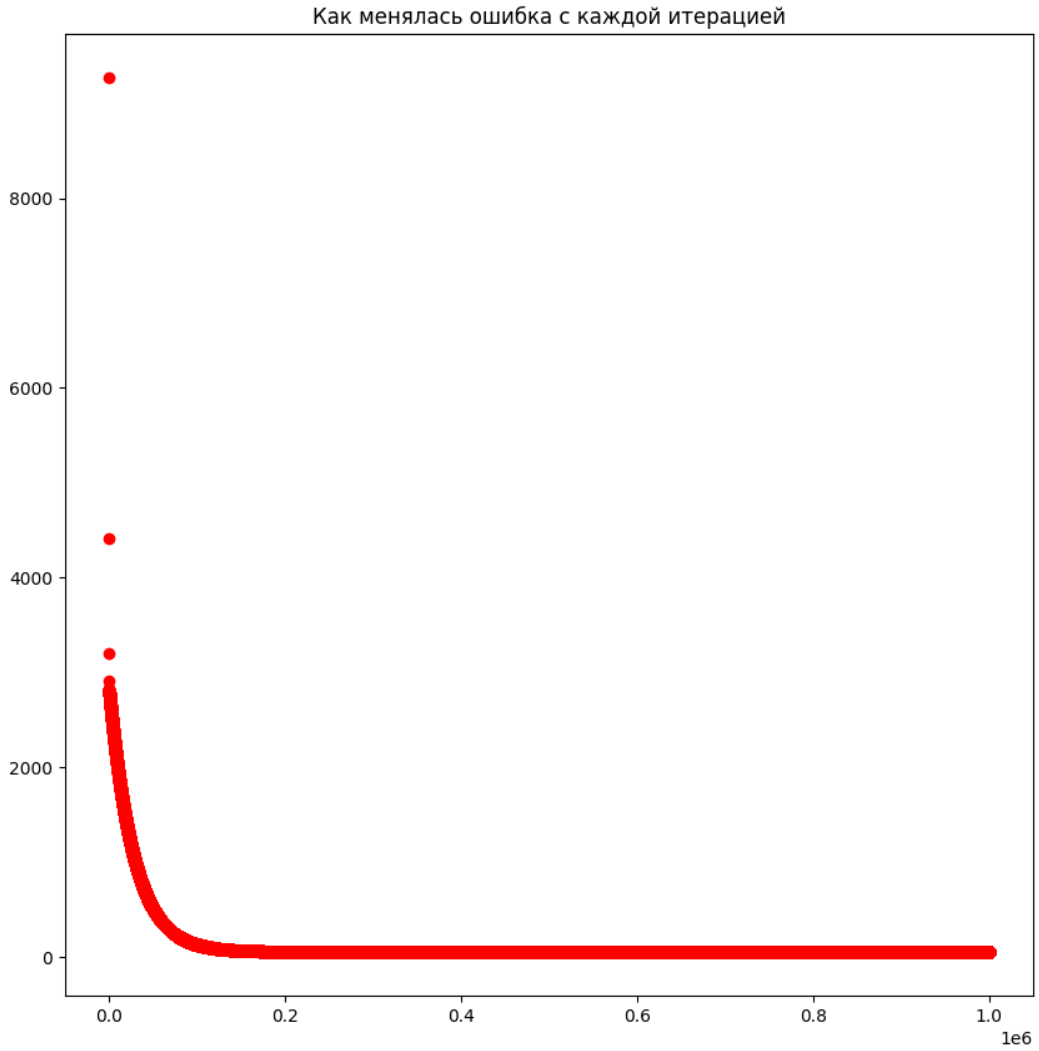

Рисунок 5 – Визуализация графика изменения ошибки с каждой итерацией цикла обучения модели (при 1 000 000 итераций).

Наконец, мы можем предсказать значение y для x = 28 на основе полученной модели:

```
x_for_prediction = 28

predicted_value = func(x_for_prediction, b, w0, w1)

print("Предсказанное значение при x = 28: ",
predicted_value)
```

Вывод:
Предсказанное значение при x = 28: [88.512073]

Выводы:

В результате выполнения работы все цели были достигнуты.

Были подобраны параметры регрессионной модели $\hat{b}$, $\hat{w}_0$ и $\hat{w}_1$ путём минимизации среднеквадратичной ошибки $MSE(\hat{b}, \hat{w}_0, \hat{w}_1)$ с помощью метода градиентного спуска, на их основе рассчитана линейная регрессия. Визуализирован график данных и полученной модели регрессии. На основе полученной модели было предсказано значение y при x = 28. Полученный ответ удовлетворяет реальному значению с учетом всех параметров. Так же мы наглядно убедились в зависимости точности модели и динамики изменения ошибки от количества итераций обучения.