Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Методы машинного обучения

ОТЧЁТ

к лабораторной работе №3

на тему

Градиентный спуск

Студент Маталыга Е.А.

Преподаватель Воронина А.М.

Минск 2025

Задание:

1. Использовать/сгенерировать датасет для задачи регрессии.
2. Написать функцию градиентного спуска (не стохастического).
3. Изучить влияние скорости обучения на результаты.
4. Написать функцию стохастического градиентного спуска.
5. По графикам среднеквадратичной ошибки для обоих методов сделать вывод о разнице скорости сходимости каждого из методов.
6. Реализовать L2-регуляризацию.

Выполнение:

Ссылка на полный код.

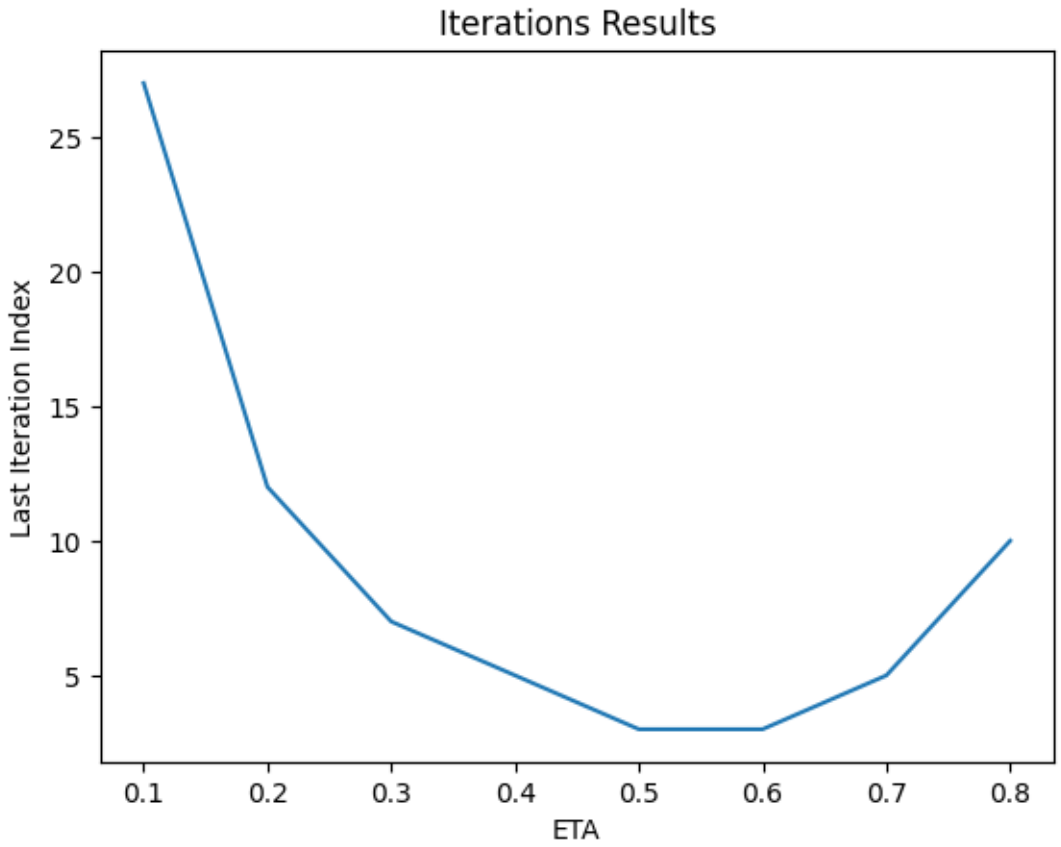
Для исследования работы градиентного спуска датасет был сгенерирован при помощи функции make_regression.

Сама функция градиентного спуска с учётом критериев останова (минимальное расстояние и количество итераций) выглядит следующим образом:

```
def gradient_descent(X, y, w):
  w_dist = 1000
  iter_num = 0
  errors = []
  error = 100
  while w_dist > MIN_W_DIST and iter_num < MAX_ITER and error
> MIN_ERROR:
    y_pred = np.dot(X, w)
    dQ = 2 / y.shape[0] * np.dot(X.T, y_pred - y)
    new_w = w - ETA * dQ
    w_dist = np.linalg.norm(new_w - w, ord=2)
    error = MSError(X, new_w, y)
    errors.append(error)
    iter_num += 1
    w = new_w
  return errors, iter_num
```

Для исследования влияние скорости обучения на точность результата было проведено несколько опытов с разными и скоростями и подсчитано

количество итераций, необходимых алгоритму для получения наилучшего результата. Лучший результат алгоритм показал на скорости 0,5.



Стохастический градиентный спуск, в отличие от стандартного, для обновления весов на итерации формирует предсказания не на всех данных, а на случайных. Поэтому в реализацию алгоритма был добавлен случайный выбор параметров.

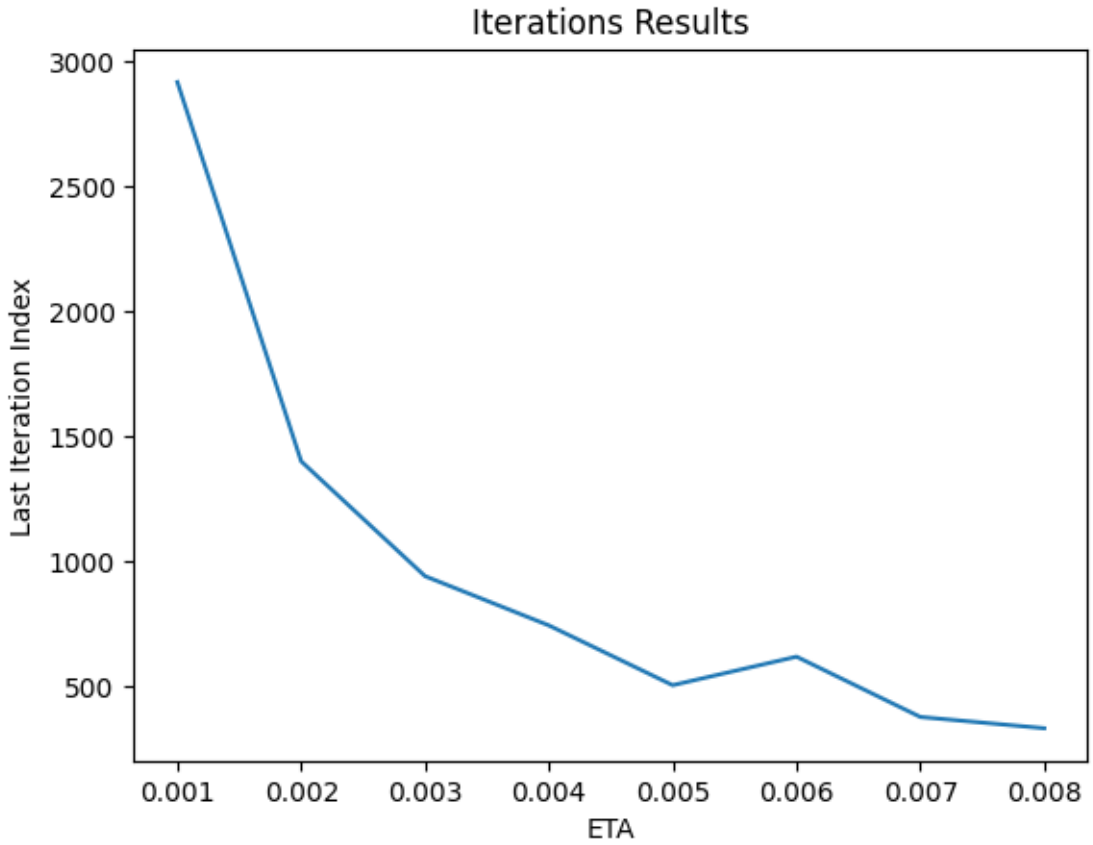
```
    def stochastic_gradient_descent(X, y, w):
        w_dist = 1000
        iter_num = 0
        errors = []
        error = 100
        n_samples = X.shape[0]
         while w_dist > MIN_W_DIST and iter_num < MAX_ITER and
error > MIN_ERROR:
            random_index = np.random.randint(0, n_samples)
            X_i = X[random_index:random_index+1]
            y_i = y[random_index:random_index+1]

            y_pred_i = np.dot(X_i, w)
```

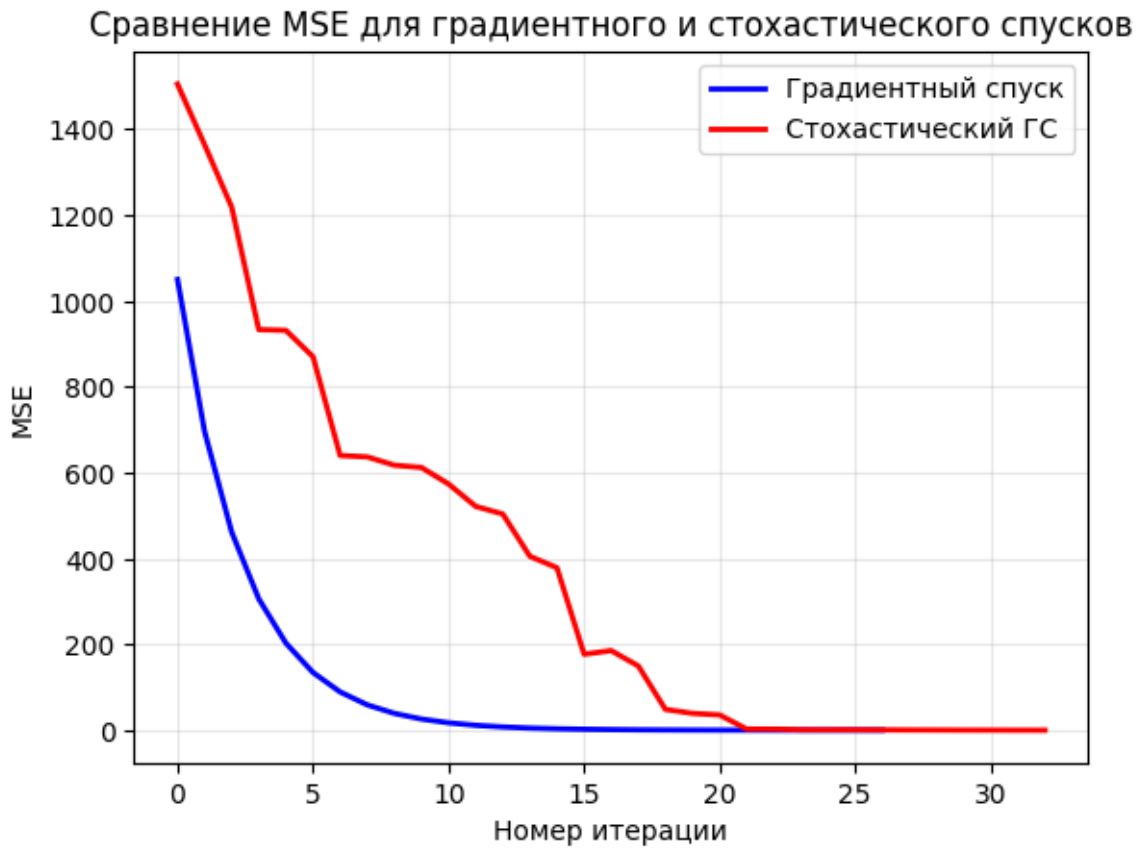
```
            dQ = 2 * np.dot(X_i.T, y_pred_i - y_i)

            new_w = w - ETA * dQ.flatten()
            w_dist = np.linalg.norm(new_w - w, ord=2)
            error = MSError(X, new_w, y)
            errors.append(error)
            iter_num += 1
            w = new_w
        return errors, iter_num
```

Так же, как и для стандартного алгоритма градиентного спуска, было проведено несколько прогонов алгоритма с разными скоростями обучения. Лучшие результаты показал прогон с самой высокой скоростью.



Также для большей наглядности, сравнение работы алгоритмов было проведено на одной скорости 0,1 и посчитано значение среднеквадратической ошибки на каждой итерации.



Стандартный алгоритм градиентного спуска быстрее вышел на минимальную ошибку, но, из-за логики работы алгоритма, на это ему потребовалось больше времени и ресурсов.

L2 регуляризация. L2 регуляризация штрафует большие значения весов, заставляя их приближаться к нулю, но в отличие от L1 регуляризации не зануляет их полностью. Для учёта штрафа на каждом шаге вычисляется его величина и добавляется к итоговому значению градиента:

```
dQ_L2 = 2 * alpha * w
total_gradient = dQ + dQ_L2
```

В ходе обучения с регуляризацией значения весов приближаются к 0.