РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ им. П. Лумумба

Факультет физико-математических и естественных наук

Кафедра теории вероятностей и кибербезопасности

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

Дисциплина: *Компьютерный практикум по моделированию*

Студент: Королёв Иван Андреевич

Группа: НКАбд-04-22

МОСКВА

2024 г.

Цель работы: Научиться работать с библиотеками numpy и pandas.

Выполнение работы

Тема «Вычисления с помощью Numpy»

Задание 1

1. Напишите функцию наподобие gradient_descent_reg_l2, но для применения L1 - регуляризации.

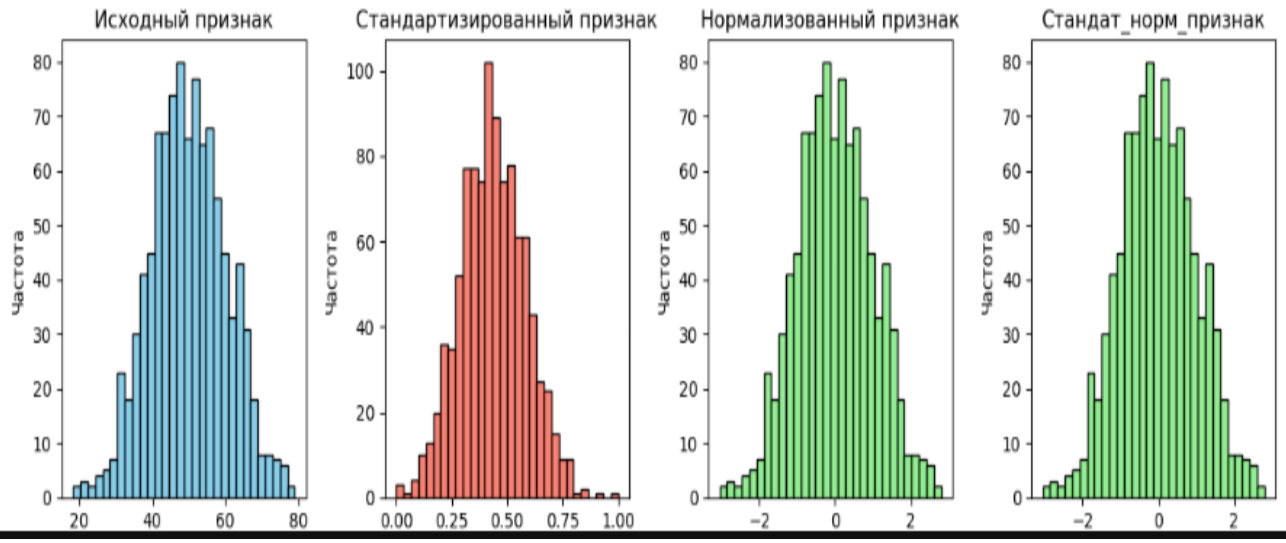
Листинг программы на языке Python:

In [153]:
```
def gradient_descent_reg_l1(X, y, iterations, eta=1e-4, reg=1e-8):
    W = np.random.randn(X.shape[1])
    n = X.shape[0]

    for i in range(0, iterations):
        y_pred = np.dot(X, W)
        err = calc_mse(y, y_pred)

        dQ = 2/n * X.T @ (y_pred - y) # градиент функции ошибки
        dReg = reg * np.sign(W) # градиент L1-регуляризации

        W -= eta * (dQ + dReg)

        if i % (iterations / 10) == 0:
            print(f'Iter: {i}, weights: {W}, error {err}')

    print(f'Final MSE: {calc_mse(y, np.dot(X, W))}')
    return W
```

Задание 2.

Листинг программы на языке Python:

```
# Создаем случайный признак
feature = np.random.normal(loc=50, scale=10, size=1000).reshape(-1, 1)

# Создаем объекты для стандартизации и нормализации
normalized_feature = (feature - np.min(feature)) / (np.max(feature) - np.min(feature)) # # Нормализация
scaler_minmax = (feature - np.mean(feature)) / np.std(feature) # Стандартизация исходного признака
standardized_normalized_feature = (normalized_feature - np.mean(normalized_feature)) / np.std(normalized_feature)
# Стандартизация нормализованного признака

# Строим гистограммы для исходного, стандартизированного и нормализованного признаков
plt.figure(figsize=(12, 4))

plt.subplot(1, 4, 1)
plt.hist(feature, bins=30, color='skyblue', edgecolor='black')
plt.title('Исходный признак')
plt.xlabel('Значение признака')
plt.ylabel('Частота')

plt.subplot(1, 4, 2)
plt.hist(scaler_standard, bins=30, color='salmon', edgecolor='black')
plt.title('Стандартизированный признак')
plt.xlabel('Значение признака')
plt.ylabel('Частота')

plt.subplot(1, 4, 3)
plt.hist(scaler_minmax, bins=30, color='lightgreen', edgecolor='black')
plt.title('Нормализованный признак')
plt.xlabel('Значение признака')
plt.ylabel('Частота')

plt.subplot(1, 4, 4)
plt.hist(standardized_normalized_feature, bins=30, color='lightgreen', edgecolor='black')
plt.title('Стандат_норм_признак')
plt.xlabel('Значение признака')
plt.ylabel('Частота')

plt.tight_layout()
plt.show()
```



Задание 3.

In [115]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

def calc_mse(y, y_pred):
    err = np.mean((y - y_pred)**2)
    return err

def gradient_descent(X, y, iterations, eta=1e-4):
    W = np.random.randn(X.shape[1])
    n = X.shape[0]
    errors = []

    for i in range(0, iterations):
        y_pred = np.dot(X, W)
        err = calc_mse(y, y_pred)
        dQ = 2/n * X.T @ (y_pred - y) # градиент функции ошибки
        W -= (eta * dQ)
        errors.append(err)
        if i % (iterations / 10) == 0:
            print(f'Итерация: {i}, веса: {W}, ошибка {err}')
    print(f'Финальная MSE метода градиентного спуска: {calc_mse(y, np.dot(X, W))}')
    return W, errors

def stochastic_gradient_descent(X, y, iterations, size, eta=1e-4):
    W = np.random.randn(X.shape[1])
    n = X.shape[0]
    errors = []

    for i in range(0, iterations):
        inds = np.random.randint(n, size=size)
        X_tmp = X[inds, ]
        y_tmp = np.array(y)[inds]

        y_pred_tmp = np.dot(X_tmp, W)
        dQ = 2/len(y_tmp) * X_tmp.T @ (y_pred_tmp - y_tmp) # градиент функции ошибки
        W -= (eta * dQ)

        err = calc_mse(y, np.dot(X, W))
        errors.append(err)

        if i % (iterations / 10) == 0:
            print(f'Итерация: {i}, веса: {W}, ошибка {err}')

    print(f'Финальная MSE метода стохастического градиентного спуска: {calc_mse(y, np.dot(X, W))}')
    return W, errors
```

```
    return w, errors

# Генерируем датасет
X, y = make_regression(n_samples=100, n_features=1, noise=10, random_state=42)

# Добавляем столбец с единицами для учёта свободного члена весов
X = np.c_[np.ones(X.shape[0]), X]

# Разбиваем данные на обучающий и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Обучение модели с помощью градиентного спуска
W_gradient, errors_gradient = gradient_descent(X_train, y_train, iterations=1000, eta=0.01)

# Обучение модели с помощью стохастического градиентного спуска
W_stochastic, errors_stochastic = stochastic_gradient_descent(X_train, y_train, iterations=1000, size=10, eta=0.01)

# Построение графика
plt.figure(figsize=(10, 6))
plt.plot(range(len(errors_gradient)), errors_gradient, label='Градиентный спуск')
plt.plot(range(len(errors_stochastic)), errors_stochastic, label='Стохастический градиентный спуск')
plt.title('Среднеквадратичная ошибка во время обучения')
plt.xlabel('Итерация')
plt.ylabel('Среднеквадратичная ошибка')
plt.legend()
plt.grid(True)
plt.show()
```

```
Итерация: 0, веса: [0.85743032 0.97751679], ошибка 1699.3153404644968
Итерация: 100, веса: [-1.60522587 35.98827407], ошибка 130.91623258611497
Итерация: 200, веса: [-0.48066704 42.59152791], ошибка 75.36622602927329
Итерация: 300, веса: [-0.04973149 43.90337728], ошибка 73.0318922621904
Итерация: 400, веса: [ 0.06446061 44.17267458], ошибка 72.92751439123508
Итерация: 500, веса: [ 0.09144541 44.22903179], ошибка 72.92275649081681
Итерация: 600, веса: [ 0.09751782 44.24095502], ошибка 72.9225383784022
Итерация: 700, веса: [ 0.09885257 44.24349275], ошибка 72.9225283632739
Итерация: 800, веса: [ 0.09914249 44.24403465], ошибка 72.92252790318962
Итерация: 900, веса: [ 0.09920507 44.24415057], ошибка 72.92252788205099
Финальная MSE метода градиентного спуска: 72.92252788107973
Итерация: 0, веса: [0.12258137 1.02816795], ошибка 1631.4300156891145
Итерация: 100, веса: [-1.02304439 34.29135786], ошибка 153.95732299446667
Итерация: 200, веса: [-0.41850724 41.97509106], ошибка 77.18415823831361
Итерация: 300, веса: [ 0.1906237  43.84033223], ошибка 73.0764577238189
Итерация: 400, веса: [ 0.20556922 44.39268685], ошибка 72.94817428056035
Итерация: 500, веса: [ 0.24169252 43.95409382], ошибка 73.0236713529647
Итерация: 600, веса: [ 0.17011937 44.28381533], ошибка 72.9281418636219
Итерация: 700, веса: [ 0.20297896 44.26454871], ошибка 72.93309569048482
Итерация: 800, веса: [ 0.24790463 44.49714383], ошибка 72.98834627453846
Итерация: 900, веса: [ 0.2365482  44.49103474], ошибка 72.98350568432197
Финальная MSE метода стохастического градиентного спуска: 72.966631619787
```