Выполнил:

студент группы УВП-412

Рогов К.Д.

**Лабораторная работа № 1**

**Линейная регрессия одной переменной**

**Задание**

Цель работы: разработать систему искусственного интеллекта, прогнозирующую прибыль станции техобслуживания автомобилей (СТО) в зависимости от их количества в населённом пункте.

Имеется обучающий набор данных, в первом столбце которого находится количество автомобилей в населённом пункте, во втором – прибыль СТО.

Для решения задачи необходимо разработать код в следующих файлах:

ex1 – основной файл, который и запускает работу программы обучения системы. Из него вызываются основные функции:

warm\_up\_exercise функция, создающая единичную матрицу размерности nxn двумя

способами: с использованием стандартных функций и без их использования

(размерность n вводится пользователем при запуске основной программы)

plot\_data – функция, визуализирующая обучающие данные.

сompute\_cost. - вычисляет функцию стоимости для одного параметра theta на основе данных из и у. Необходимо реализовать поэлементное и векторное вычисление функции стоимости.

gradient\_descent. - выполняет один шаг в направлении антиградиента для обновления вектора theta. Необходимо реализовать поэлементное и векторное вычисление

work. - программа для работы с обученной системой

**Файл ex1.py**

Этот файл служит основой программы и отвечает за выполнение различных ключевых операций:

* Подготовка данных: Выполняются функции warmup\_exercise\_built\_in и warmup\_exercise\_manual для построения единичных матриц.
* Загрузка данных: Данные загружаются из файла train\_data.txt и разделяются на переменные X (количество автомобилей) и Y (прибыль).
* Добавление столбца единиц: Для учета свободного члена в линейной регрессии к матрице X добавляется столбец единиц.
* Инициализация параметров: Параметры модели (theta) инициализируются нулями, а также задаются значения для числа итераций и скорости обучения.
* Визуализация данных: С помощью функции plot\_data отображаются исходные данные, что позволяет наглядно увидеть зависимость прибыли от количества автомобилей.
* Вычисление стоимости: Рассчитываются значения функции стоимости как векторным (с помощью compute\_cost.compute\_cost\_vector), так и поэлементным (с помощью compute\_cost.compute\_cost\_elements) способами, после чего выводятся полученные результаты.
* Градиентный спуск: Параметры модели оптимизируются с использованием метода градиентного спуска: сначала векторным способом (gradient\_descent.gradient\_descent\_vector), а затем поэлементным способом (gradient\_descent.gradient\_descent\_elements).
* Прогнозирование прибыли: Программа запрашивает у пользователя ввод количества автомобилей и с использованием обеих стратегий предсказывает прибыль на основе рассчитанных параметров.

size = int(input("Введите размерность матрицы:"))  
  
warm\_up\_exercise.warmup\_exercise\_built\_in(size)  
warm\_up\_exercise.warmup\_exercise\_manual(size)  
  
data = np.loadtxt('train\_data.txt', delimiter=',')  
X = data[:, 0]  
Y = data[:, 1]  
m = len(Y)  
  
X = np.column\_stack((np.ones(m), X))  
  
theta = np.zeros(2)  
iterations = 1500  
alpha = 0.01  
  
plot\_data.plot\_data(X[:, 1], Y)  
  
cost\_vector = compute\_cost.compute\_cost\_vector(X, Y, theta)  
print(f'Значение функции стоимости векторным способом: {cost\_vector}')  
  
cost\_elements = compute\_cost.compute\_cost\_elements(X, Y, theta)  
print(f'Значение функции стоимости поэлементным способом: {cost\_elements}')  
  
theta\_vector = gradient\_descent.gradient\_descent\_vector(X, Y, theta, alpha, iterations)  
print(f'Вектор параметров модели векторным способом: {theta\_vector}')  
  
theta\_elements = gradient\_descent.gradient\_descent\_elements(X, Y, theta, alpha, iterations)  
print(f'Вектор параметров модели поэлементным способом: {theta\_elements}')  
  
cars = int(input("Введите количество автомобилей:"))  
profit\_vector = work.predict\_profit(cars, theta\_vector)  
profit\_elements = work.predict\_profit(cars, theta\_elements)  
  
print(f"Прогнозируемая прибыль (векторный способ): {profit\_vector}")  
print(f"Прогнозируемая прибыль (поэлементный способ): {profit\_elements}")

**Функции warm\_up\_exercise\_built\_in.py и warm\_up\_exercise\_manual.py**

Функции создают единичную матрицу размерности nxn двумя способами:

* С использованием встроенной функции: Используется библиотека numpy для создания единичной матрицы.
* Без встроенных функций: Реализуется цикл для создания матрицы вручную.

def warmup\_exercise\_built\_in(n):

matrix = np.eye(n).astype(int)

print(f'Единичная матрица с использованием numpy: \n{matrix}')

def warmup\_exercise\_manual(n):

matrix = [[1 if i == j else 0 for j in range(n)] for i in range(n)]

for row in matrix:

print(row)

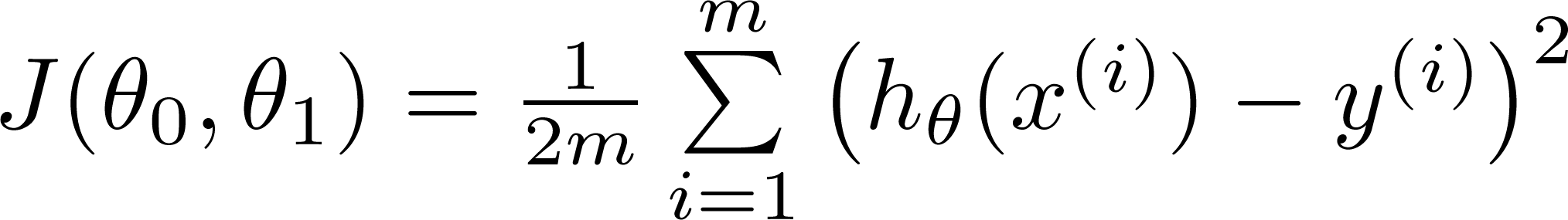
**Функция plot\_data.py**

Функция визуализирует зависимость между количеством автомобилей и прибылью, а затем сохраняет этот график в файл формата PNG.

def plot\_data(x, y):  
 plt.scatter(x, y, marker='x', color='green')  
 plt.xlabel('Количество автомобилей')  
 plt.ylabel('Прибыль')  
 plt.title('График зависимости прибыли от количества автомобилей')  
 plt.savefig("result.png")

**Функции compute\_cost\_vector.py и compute\_cost\_elements.py**

Функция стоимости определяется как среднеквадратичная ошибка между предсказанными и реальными значениями прибыли.



hθ(x) — предсказанное значение, y(i) — реальное значение прибыли, m — количество примеров.

Аргументы:

* x: Матрица входных данных (признаки).
* y: Вектор целевых значений (прибыль).
* theta: Вектор параметров модели (коэффициенты).

def compute\_cost\_vector(x, y, theta):  
 m = len(y)  
 predictions = x.dot(theta)  
 errors = predictions - y  
 cost = (1 / (2 \* m)) \* np.sum(errors \*\* 2)  
 return cost

* Определение размера: m = len(y) — вычисляет количество наблюдений (размер выборки).
* Прогнозы: predictions = x.dot(theta) — предсказывает целевые значения, умножая матрицу x на вектор theta.
* Ошибки: errors = predictions - y — вычисляет разницу между предсказанными и фактическими значениями.
* Стоимость: cost = (1 / (2 \* m)) \* np.sum(errors \*\* 2) — вычисляет среднюю квадратичную ошибку, умноженную на 1/2. Это значение показывает, насколько хорошо модель предсказывает целевые значения.

def compute\_cost\_elements(x, y, theta):  
 m = len(y)  
 total\_cost = 0  
  
 for i in range(m):  
 prediction = x[i].dot(theta)  
 error = prediction - y[i]  
 total\_cost += error \*\* 2  
  
 cost = (1 / (2 \* m)) \* total\_cost  
 return cost

* Определение размера: m = len(y) — вычисляет количество наблюдений.
* Инициализация: total\_cost = 0 — инициализирует переменную для сохранения общей стоимости.
* Цикл по наблюдениям:

Для каждого наблюдения: prediction = x[i].dot(theta) — вычисляет предсказание для i-го наблюдения.

error = prediction - y[i] — определяет ошибку, сравнивая предсказание с фактическим значением.

total\_cost += error \*\* 2 — добавляет квадрат ошибки к общей стоимости.

* Стоимость: cost = (1 / (2 \* m)) \* total\_cost — аналогично первой функции, вычисляет среднюю квадратичную ошибку.

**Функции gradient\_descent\_vector.py и gradient\_descent\_elements.py**

Градиентный спуск обновляет параметры на основе направления антиградиента функции стоимости.



где α – скорость обучения, j – индекс параметра, x (i) j — значения признаков для примера i.

Аргументы:

* x: Матрица входных данных (признаки).
* y: Вектор целевых значений (например, прибыль).
* theta: Вектор параметров модели (коэффициенты) для обновления.
* alpha: Скорость обучения (learning rate).
* iterations: Количество итераций для выполнения градиентного спуска.

def gradient\_descent\_vector(x, y, theta, alpha, iterations):  
 m = len(y)  
  
 for \_ in range(iterations):  
 predictions = x.dot(theta)  
 errors = predictions - y  
 theta -= (alpha / m) \* (x.T.dot(errors))  
 return theta

* Определение размера: m = len(y) — вычисляет количество наблюдений.
* Итеративное обновление параметров: В цикле выполняется обновление параметров theta:

predictions = x.dot(theta) — вычисляет предсказания для всех наблюдений.

errors = predictions - y — вычисляет ошибки между предсказанными и фактическими значениями.

theta -= (alpha / m) \* (x.T.dot(errors)) — обновляет параметры theta с использованием градиента, вычисленного на основе ошибок.

def gradient\_descent\_elements(x, y, theta, alpha, iterations):  
 m = len(y)  
  
 for \_ in range(iterations):  
 predictions = np.zeros(m)  
 errors = np.zeros(m)  
  
 for i in range(m):  
 predictions[i] = x[i].dot(theta)  
 errors[i] = predictions[i] - y[i]  
  
 for j in range(len(theta)):  
 gradient = 0  
 for i in range(m):  
 gradient += errors[i] \* x[i][j]  
 theta[j] -= (alpha / m) \* gradient  
  
 return theta

* Определение размера: m = len(y) — вычисляет количество наблюдений.
* Инициализация: Создаются массивы для предсказаний и ошибок с нулевыми значениями.
* Постепенное обновление параметров:
  + Первый цикл проходит по всем наблюдениям (от 0 до m):

predictions[i] = x[i].dot(theta) — вычисляет предсказание для i-го наблюдения.

errors[i] = predictions[i] - y[i] — определяет ошибку для i-го наблюдения.

* + Второй цикл проходит по каждому параметру (от 0 до количества параметров theta):

gradient инициализируется как 0.

Внутренний цикл проходит по всем наблюдениям, вычисляя градиент, который накапливается в gradient.

theta[j] -= (alpha / m) \* gradient — обновляет значение

параметра theta[j].

**Файл work.py**

функция prediction является простым инструментом для получения прогноза по прибыли, используя линейную регрессию. Она принимает количество автомобилей и коэффициенты модели, и возвращает ожидаемую прибыль.

def prediction(cars, theta):  
 x = np.array([1, cars])  
 return x.dot(theta)

**Результаты работы**

