Машинно обучение

Лабораторно упражнение 4

Линейна регресия и градиентно спускане

Целта на упражнението е да се реализира линейна регресия, в която функцията на цената се минимизира чрез градиентно спускане.

Ще демонстрираме със средствата на Python следните основни етапи от линейна регресия:

1. Зареждане на данни
2. Изчисляване на ф-та на цената (cost function)
3. Градиентно спускане

Ще приложим линейната регресия за предсказване на доходността на хранителен магазин в град с известен брой на населението. Обучителните данни се намират във файла ex1data1.txt. Данните представляват таблица от две колони: население на град (умножено по 10000) и доходност на хранителен магазин в града (умножена по 10000).

Включваме следните пакети:

Pandas – за форматиране на данните и за четене на csv файлове

Numpy – за операции с масиви и матрици

Matplotlib – за графики на функции

In [1]: import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

1. Зареждане на данни от файла ex1.data1.txt

In [2]: data = pd.read_csv('ex1data1.txt', names = ['population', 'profit'])

In [3]: print(data.head())

Резултат:

	population	profit
0	6.1101	17.5920
1	5.5277	9.1302
2	8.5186	13.6620
3	7.0032	11.8540
4	5.8598	6.8233

2. Визуализиране на данните

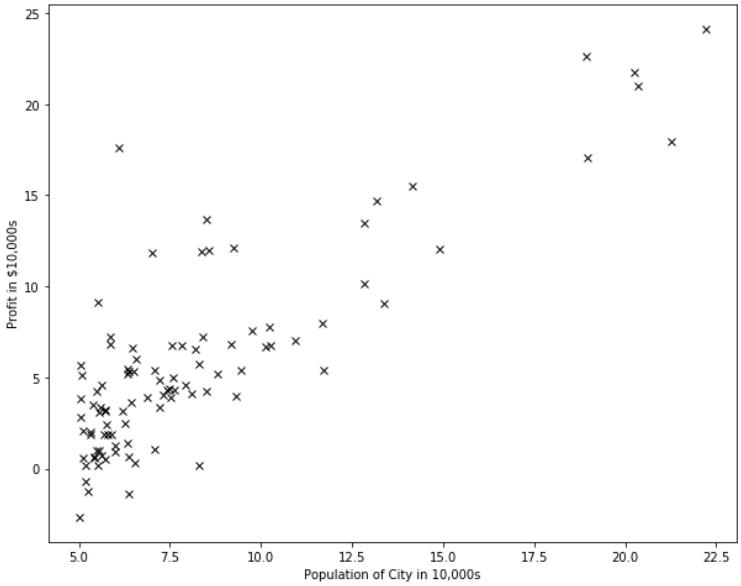
In [4]: *## Разделяме данните население (population) и доходност (profit) в X и y*

```
X_df = pd.DataFrame(data.population)
y_df = pd.DataFrame(data.profit)

## Брой данни за доходността (y)
m = len(y_df)
```

In [5]:
```
plt.figure(figsize=(10,8))
plt.plot(X_df, y_df, 'kx')
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
```

Резултатът е визуализиране на данните, които са обучителни примери в линейната регресия (фиг. 1):



Фиг. 1. Визуализиране на данните за население-доходност

Целта на линейната регресия е да се намери връзката между променливата x и зависимата (изходна) променлива (y). Тази връзка се използва за предсказване на y за нови (непознати) стойности на променливата x.

В разглежданата задача връзката е права линия, която се дефинира с уравнение:

$$y = h_\theta(x) = \theta_0 + \theta_1 x$$

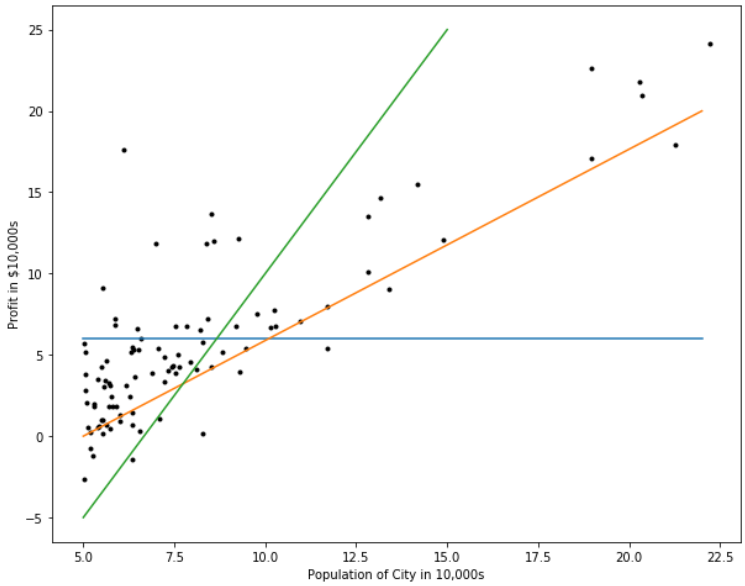
, където:

$h_\theta(x)$ – модел на линейната регресия или **хипотеза**, получена в резултат от обучение

θ_0, θ_1 са параметри на линейната регресия.

Фиг. 2 показва, че са възможни много прави линии, апроксимиращи входните данни. Търсим правата, която минава най-близо до точките, представящи входните данни.

```
In [6]: plt.figure(figsize=(10,8))
plt.plot(X_df, y_df, 'k.')
plt.plot([5, 22], [6,6], '-')
plt.plot([5, 22], [0,20], '-')
plt.plot([5, 15], [-5,25], '-')
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
```



Фиг. 2. Възможни прави линии, апроксимиращи данните

3. Функция на цената

За да намерим най-близката линия до входните данни, трябва да намерим стойности на параметрите θ_0, θ_1, при които предсказаните стойности да са възможно най-близки до действителните стойности за y. С други думи е необходимо разстоянието между хипотезата $h_\theta(x)$ и известната цена от обучителните примери y да бъде минимално.

Дефинираме функция на цената (cost function), която използва т.нар. най-малки квадрати. Това е сума от квадратите на разликите между предсказаните и действителните стойности на y. Функцията изглежда по следния начин:

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

, където

$J(\theta_0, \theta_1)$ – функцията на цената с параметри θ_0, θ_1

m – брой входни данни

$x^{(i)}$ – i-та стойност за x (население) от обучителните данни

$y^{(i)}$ – i-та стойност за y (доходност) от обучителните данни

$h_\theta(x)$ е хипотезата, която търсим. Тя има следният линеен модел:

$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

Търсим стойностите на параметрите θ_0, θ_1 , за които ф-та J има минимална стойност.

За да намерим тези стойности, използваме алгоритъм, който се нар. **градиентно спускане (gradient descent)**. Алгоритъмът се състои от итерации (повторения), като на всяка итерация се изпълняват следните операции:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) {x_j}^{(i)}$$

, където

θ_j – параметрите θ_0, θ_1

α – скорост на обучението (learning rate)

Градиентното спускане е прост алгоритъм, при който променяме параметрите на функцията с много малки стъпки до достигане на минимум на функцията.

Пример за намиране на минимум на функция чрез градиентно спускане

Ще демонстрираме алгоритъма за намиране на минимум за следната проста квадратна функция:

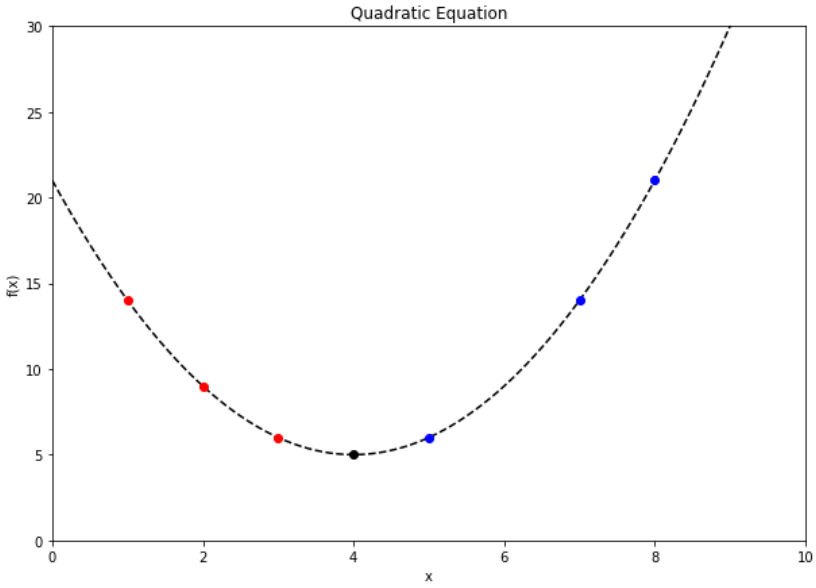
$$y(x) = (x-4)^2 + 5$$

Ще визуализираме алгоритъма чрез изобразяване на графиката на функцията със средствата на Python.

```
In [7]: x_quad = [n/10 for n in range(0, 100)]
        y_quad = [(n-4)**2+5 for n in x_quad]
In [8]: plt.figure(figsize = (10,7))
        plt.plot(x_quad, y_quad, 'k--')
        plt.axis([0,10,0,30])
        plt.plot([1, 2, 3], [14, 9, 6], 'ro')
        plt.plot([5, 7, 8],[6, 14, 21], 'bo')
        plt.plot(4, 5, 'ko')
```

```
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Quadratic Equation')
```

Резултатът изобразява графиката на квадратната функция (фиг. 3).



Фиг. 3. Графика на квадратната функция

Ще определим стойността на *x*, при която функцията *f(x)* има минимална стойност (минимум). Ще приложим градиентно спускане, използвайки графиката на функцията (фиг. 3).

Започваме с начална стойност за *x* и проверяваме графиката на функцията. Наклонът на кривата се нар. **градиент**. Променяме x и проверяваме дали кривата променя посоката си (т.е. дали градиентът променя стойността си). Стойността на x, след която кривата променя наклона си е стойността, съответстваща на минималната стойност y, т.е. това е търсеният минимум на функцията.

Напр., ако започнем с начална стойност x=2, то в точката (2,9) **градиентът е отрицателен (червена точка на фиг. 3)**. При x=3, градиентът в точката (3,6) отново е отрицателен (червена точка на фиг. 3), т.е. посоката на изменение на кривата се запазва. При x=4, y=5. За стойности x>4 кривата променя наклона си (сините точки на фиг. 3), т.е. **градиентът става положителен**. Това означава, че ф-та $y(x) = (x-4)^2 + 5$ има минимум при x=4.

Скоростта на обучението α определя колко бързо се спускаме към минимума на ф-та. При много голяма скорост съществува риск от „прескачане“ на търсения минимум.

Ще реализираме градиентното спускане за задачата с население на град и доходност на хранителен магазин.

В алгоритъма използваме следните променливи:

iterations – брой итерации в градиентното спускане

alpha – скорост на обучението

np.dot се използва за умножение на матрици

In [9]: iterations = 1500

alpha = 0.01

In [10]: ## Добавя колона от 1-ци към вектора X

X_df['intercept'] = 1

Трансформира Numpy масивите за по-лесни операции с матриците

X = np.array(X_df)

y = np.array(y_df).flatten()

theta = np.array([0, 0])

In [11]: def cost_function(X, y, theta):

"""

cost_function(X, y, theta) изчислява функцията на цената за стойности на параметрите theta

"""

брой обучителни примери

m = len(y)

Изчислява цената с въведени параметри

```
        J = np.sum((X.dot(theta)-y)**2)/2/m

        return J
```

In [12]: cost_function(X, y, theta)

Резултат:

32.072733877455676

Това е стойността на цената при $\theta_0 = 0, \theta_1 = 0$.

Условно разделяме алгоритъма на градиентното спускане на следните стъпки:

1. Изчисляваме хипотеза hypothesis[97x1] = x[97x2] * theta[2x1]
2. Изчисляваме разликите loss [97x1] с изваждане елемент по елемент
3. Изчисляваме градиента gradient[2x1] = X' [2x97].loss [97x1]
4. Променяме стойностите на параметрите theta[2x1]
5. Намираме цената с извикване на cost_function()

```
def gradient_descent(X, y, theta, alpha, iterations):
"""
   Ф-та gradient_descent изпълнява градиентно спускане за обучаване на параметрите theta
   theta = gradient_descent(X, y, theta, alpha, iterations) променя стойностите на theta
   iterations – брой итерации
   alpha - скорост на обучение
   """
   cost_history = [0] * iterations

   for iteration in range(iterations):
      hypothesis = X.dot(theta)
      loss = hypothesis-y
      gradient = X.T.dot(loss)/m
```

```
        theta = theta - alpha*gradient
        cost = cost_function(X, y, theta)
        cost_history[iteration] = cost

    return theta, cost_history
```

In [14]: (t, c) = gradient_descent(X,y,theta,alpha, iterations)

In [15]: *## Отпечатваме theta*

print(t)

Резултат:

[1.16636235 -3.63029144]

Това са стойностите на параметрите θ_0, θ_1, при които цената J има минимум.

Ще използваме намерените стойности, за да определим (т.е. да предскажем) доходността на магазин в градове с население съответно 35000 и 70000 жители.

In [16]: *## Prediction*

print(np.array([3.5, 1]).dot(t))

print(np.array([7, 1]).dot(t))

Резултат:

0.45197678677 – в град с население 35000 жители очакваната доходност е 4519 долара

4.53424501294 – в град с население 70000 жители очакваната доходност е 45342 долара

In [17]: *## Изчертаване на най-близката линия*

best_fit_x = np.linspace(0, 25, 20)

best_fit_y = [t[1] + t[0]*xx **for** xx **in** best_fit_x]

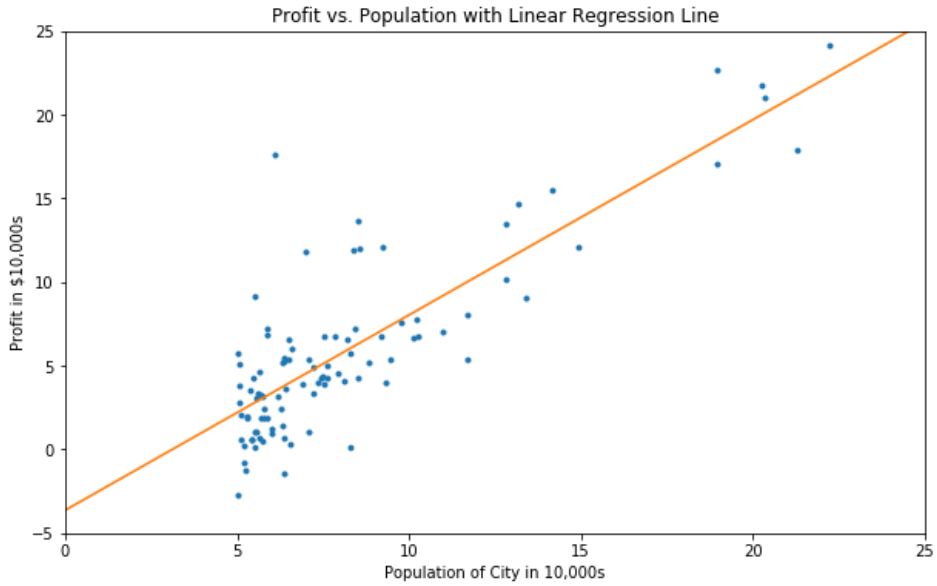
In [18]: plt.figure(figsize=(10,6))

plt.plot(X_df.population, y_df, '.')

plt.plot(best_fit_x, best_fit_y, '-')

```
plt.axis([0,25,-5,25])
plt.xlabel('Population of City in 10,000s')
plt.ylabel('Profit in $10,000s')
plt.title('Profit vs. Population with Linear Regression Line')
```

Резултатът показва линията, преминаваща най-близо до обучителните данни (фиг. 4)



Фиг. 4. Най-близката линия

Задача 1: Начертайте графиката на функцията на цената. Наблюдавайте как се променя стойността на цената с увеличаване на итерациите в градиентното спускане.

Задача 2: Във файла ex1data2.txt са дадени следните данни за къщи:

Площ на къщата (feet^2) * 1000	Цена ($) * 1000
2.104	399.900
1.600	329.900
2.400	369.000
…	

Да се построи модел за предсказване на цената на къща с произволна площ чрез линейна регресия. С намерената хипотеза да се определи цената на къщи с площ 2*1000 и 4.5*1000 кв. фута.