



רגרסיה לינארית במספר משתנים גלעד מרקמן



קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות



דוגמה מחירי דירות

- בסקר שנערך בעיר נס ציונה נבדקו מחירי הדירות לפי שטח הדירה והקומה. התוצאות נתונות בטבלה למטה:
- עליכם למצוא את הנוסחה לחישוב מחיר הדירה בהתאם לשטחה. הפעם אנחנו מתחשבים גם בקומה.
- למעשה אנחנו מבקשים למצוא פונקציה בשני משתנים. המחיר ביחס לשטח ולקומה:

מחיר הדירה במיליוני ₪	קומה	שטח הדירה
2.462	3	90
2.661	2	100
3.177	9	105
2.972	3	110
3.321	7	115
3.226	3	120
3.648	6	130
3.447	1	133
3.848	5	140

$$price(area, floor) = W_1 * area + W_2 * floor + W_3$$

$$W_3 = bias$$



מספר משתנים ונגזרות חלקיות

- האלגוריתם לרגרסיה לינארית עם מספר משתנים זהה לאלגוריתם עם משתנה אחד.
- ההבדל היחיד הוא שאנחנו מבצעים חישוב נגזרת חלקית (גרדיאן) לפי כל משתנה (משקל), ומבצעים אופטימיזציה לכל משקל בהתאם לנגזרת החלקית שחושבה עבורו.
- במקרה שלנו: יש שלושה משתנים שמחושבים להם הנגזרות.

$$price(area, floor) = W_1 * area + W_2 * floor + W_3$$

$$W_3 = bias$$



יישום רגרסיה בשני משתנים

למזלנו pytorch מבצע עבורנו את כל העבודה גם ברגרסיה שני משתנים (ואף יותר משתנים).

• עלינו לעצב את הנתונים בטנסור דו מימדי. כל שורה כוללת את שתי עמודות: שטח הדירה, והקומה (מנורמלים).

```
X1 = torch.from_numpy(X_np.astype(np.float32)) # area
X2 = torch.from_numpy(F_np.astype(np.float32)) # floor
X = torch.cat((X1.reshape(-1,1),X2.reshape(-1,1)),1)
Y = torch.from_numpy(P_np.astype(np.float32)) # price

# Normalize data
max, __ = X.max(dim=0)
min, __ = X.min(dim=0)
X = Normalize_minMax(X, max, min)
Y = Y.view(Y.shape[0],1)

print ("X:", X.shape, X.dtype)
print("Y:", Y.shape, Y.dtype)
print (X)
print (Y)
```

חישוב מקסימום לכל
עמודה בנפרד

```
def Normalize_minMax(X , max, min):
    max = max.expand(X.shape[0],-1)
    min = min.expand(X.shape[0],-1)
    return (X - min) / (max - min)
```

הכפלה של המקסימום
למספר השורות ב X, כדי
שניתן יהיה לבצע את
החישוב לכל שורה

```
X: torch.Size([9, 2]) torch.float32
Y: torch.Size([9, 1]) torch.float32
tensor([[0.0000, 0.2500],
        [0.2000, 0.1250],
        [0.3000, 1.0000],
        [0.4000, 0.2500],
        [0.5000, 0.7500],
        [0.6000, 0.2500],
        [0.8000, 0.6250],
        [0.8600, 0.0000],
        [1.0000, 0.5000]])
tensor([[2.4620],
        [2.6610],
        [3.1770],
        [2.9720],
        [3.3210],
        [3.2260],
        [3.6480],
        [3.4470],
        [3.8480]])
```



מודל למספר משתנים

- בפרמטרים של הפונקציה `nn.Linear` ניתן להגדיר את מספר המשתנים שמתקבלים.
- זהו כל השינוי הנדרש במודל ובאלגוריתם.

```
learning_rate = 0.1
epochs = 200
losses = []

# design model
Model = nn.Linear(2, 1, bias=True)

#construct loss and optimizer
Loss = nn.MSELoss()

# init optimizer
optim = torch.optim.SGD(Model.parameters(), lr=learning_rate)
```



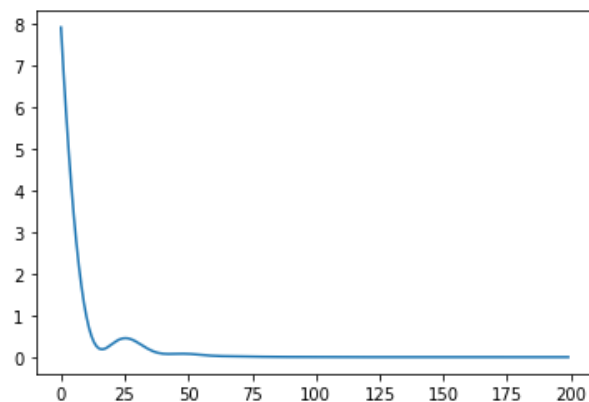
תוצאות ההרצה

- ניתן לראות כי במקרה שבו אנו מתחשבים בכל הנתונים אנו מגיעים לרמת דיוק גבוהה הרבה יותר. במקרה שלנו אפילו הגענו לטעות של 0. בול פגיעה!!!!

```
epoch= 0 loss=7.9028  
epoch= 20 loss=0.3113  
epoch= 40 loss=0.0823  
epoch= 60 loss=0.0311  
epoch= 80 loss=0.0083  
epoch= 100 loss=0.0018  
epoch= 120 loss=0.0004  
epoch= 140 loss=0.0001  
epoch= 160 loss=0.0001  
epoch= 180 loss=0.0000
```

```
] plt.plot(losses)
```

```
[<matplotlib.lines.Line2D at 0x7f9f6684d580>]
```



END: loss=0.0000

