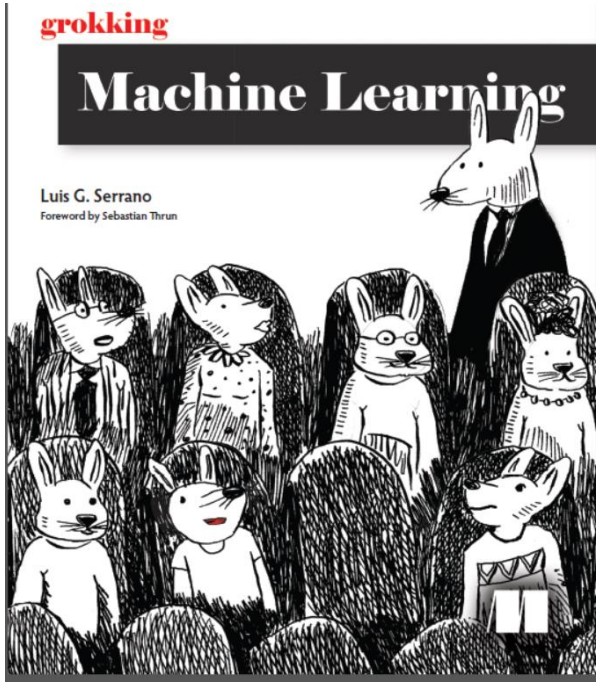
Machine Learning

|Square – Absolute Trick

Prof. Dr. Mostafa Elhosseini

https://youtube.com/@ProfElhosseiniTechVerse



Agenda

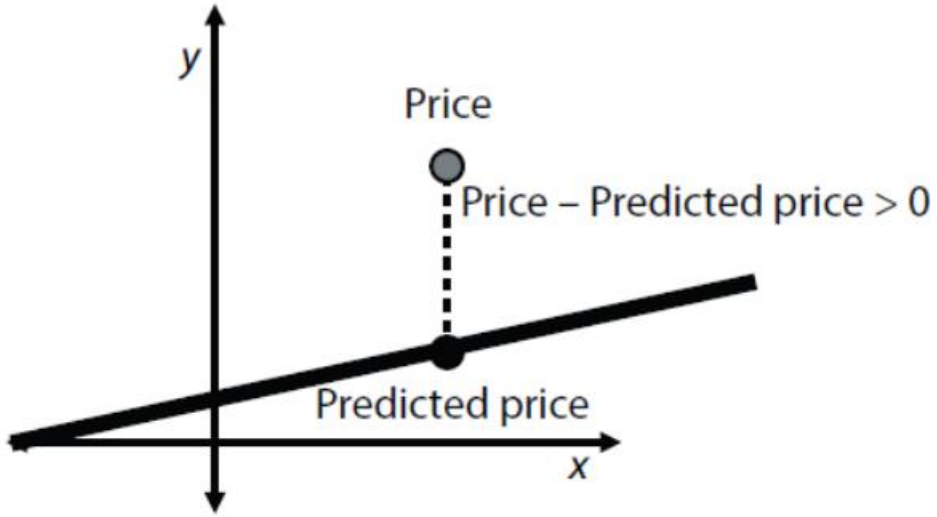
- Square Trick
- Absolute Trick
- Python code

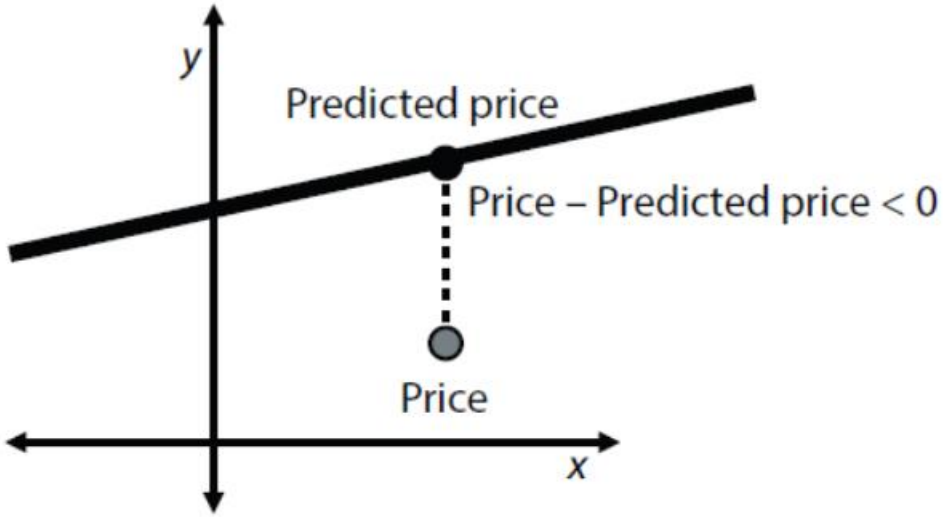
Square trick

- An effective way to move a line closer to a point
- The square trick will bring these four cases down to one by finding values with the correct signs (+ or –) to add to the slope and the y-intercept for the line to always move closer to the point
- In the **simple trick**, when the point is above the line, we add a small amount to the y-intercept. When it is below the line, we subtract a small amount.

Square trick

- If a point is above the line, the value $p - \hat{p}$ (the difference between the Actual price and the predicted price) is positive. If it is below the line, this value is negative





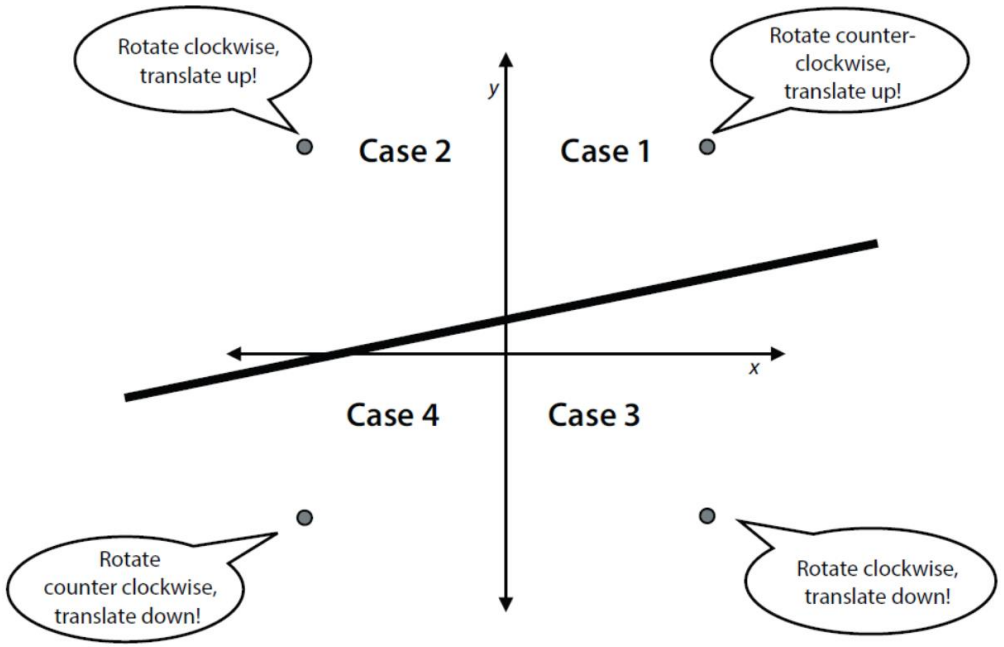
Square trick

- we conclude that if we add the difference $p - \hat{p}$ to the y-intercept, the line will always move toward the point
- However, in machine learning, we always want **to take small steps**. To help us with this, we introduce an important concept in machine learning: the **learning rate**.

- **learning rate:** A very small number that **we pick before training** our model. This number helps us make sure our model changes in very small amounts by training. (η - Greek letter eta).
- Add $\eta(p - \hat{p})$

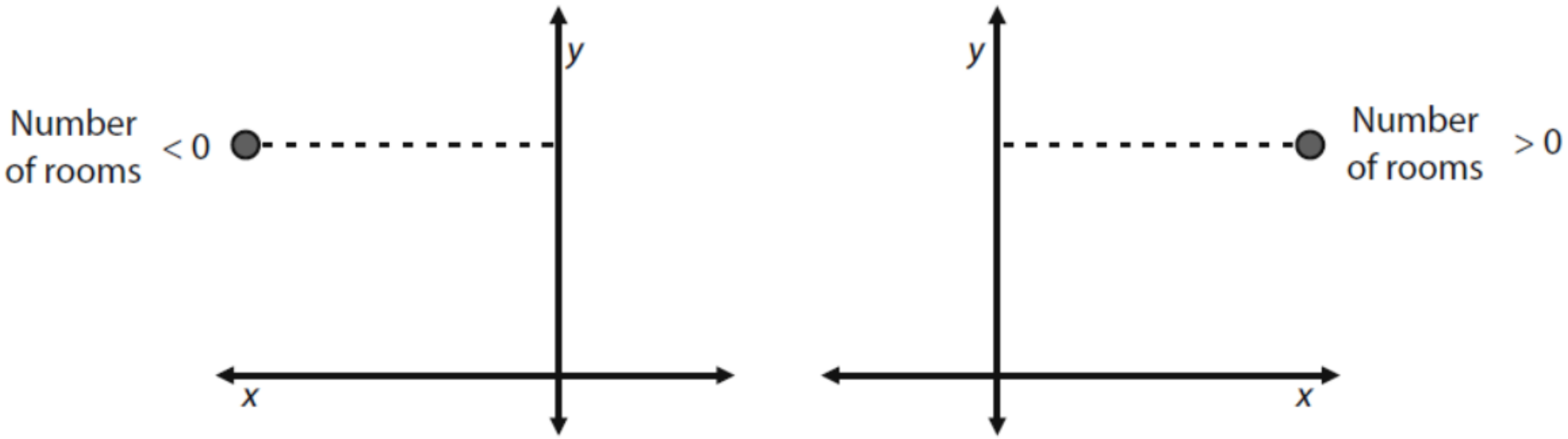
Square trick

- In the **simple trick**, when the point is in **scenario 1 or 4** (above the line and to the right of the vertical axis, or below the line and to the left of the vertical axis), we rotate the line **counterclockwise**. Otherwise (scenario **2 or 3**), we rotate it **clockwise**



Square trick

- If a point (r, p) is to the right of the vertical axis, then r is positive. If the point is to the left of the vertical axis, then r is negative



Square trick

- The value $\boldsymbol{r}(\boldsymbol{p} - \hat{\boldsymbol{p}})$. This value is positive when both r and $\boldsymbol{p} - \hat{\boldsymbol{p}}$ are both positive or both negative. This is precisely what occurs in scenarios **1 and 4**.
- Similarly, $\boldsymbol{r}(\boldsymbol{p} - \hat{\boldsymbol{p}})$ is negative in scenarios **2 and 3**.
- Therefore, due to observation 4, this is the quantity that we need to add to the slope
- Adding $\boldsymbol{\eta r}(\boldsymbol{p} - \hat{\boldsymbol{p}})$ to the slope will always move the line in the direction of the point

Square trick

Inputs:

- A line with slope m, y-intercept b, and equation $\hat{p} = mr + b$
- A point with coordinates (r, p)
- A small positive value η (the learning rate)

Output:

- A line with equation $\hat{p} = m'r + b'$ that is closer to the point

Procedure:

- Add $\eta r(p - \hat{p})$ to the slope m. Obtain $m' = m + \eta r(p - \hat{p})$ (this rotates the line).
- Add $\eta(p - \hat{p})$ to the y-intercept b. Obtain $b' = b + \eta(p - \hat{p})$ (this translates the line).

Return: The line with equation $\hat{p} = m'r + b'$

Square trick - Code

- https://colab.research.google.com/drive/1GvRGWKezxzUFov8mz3X8RYFjW8IglbhA

Square trick - Code

- https://colab.research.google.com/drive/1GvRGWKezxzUFov8mz3X8RYFjW8IglbhA

```
def square_trick(base_price, price_per_room, num_rooms, price, learning_rate):
    predicted_price = base_price + price_per_room*num_rooms
    price_per_room += learning_rate*num_rooms*(price-predicted_price)
    base_price += learning_rate*(price-predicted_price)
    return price_per_room, base_price
```

Absolute trick

- The **square trick** is effective, but another useful trick, which we call the **absolute trick**, is an intermediate between the simple and the square tricks.
- In the **square trick**, we used the two quantities, $(p - \hat{p})$ (price – predicted price) and r (number of rooms), to help us bring the four cases down to one.
- In the **absolute trick**, we use only r to help us bring the four cases down to two

Absolute trick

Inputs:

- A line with slope m, y-intercept b, and equation $\hat{p} = mr + b$
- A point with coordinates (r, p)
- A small positive value η (the learning rate)

Output:

- A line with equation $\hat{p} = m'r + b'$ that is closer to the point

Procedure:

Case 1: If the point is above the line (i.e., if $p > \hat{p}$):

- Add ηr to the slope m. Obtain $m' = m + \eta r$ (this rotates the line counterclockwise if the point is to the right of the y-axis, and clockwise if it is to the left of the y-axis).
- Add η to the y-intercept b. Obtain $b' = b + \eta$ (this translates the line up).

Case 2: If the point is below the line (i.e., if $p < \hat{p}$):

- Subtract ηr from the slope m. Obtain $m' = m - \eta r$ (this rotates the line clockwise if the point is to the right of the y-axis, and counterclockwise if it is to the left of the y-axis).
- Subtract η from the y-intercept b. Obtain $b' = b - \eta$ (this translates the line down).

Return: The line with equation $\hat{p} = m'r + b'$

Linear Regression Algorithm

- Repeating the absolute or square trick many times to move the line closer to the points

Inputs:

- A dataset of houses with number of rooms and prices

Outputs:

- Model weights: price per room and base price

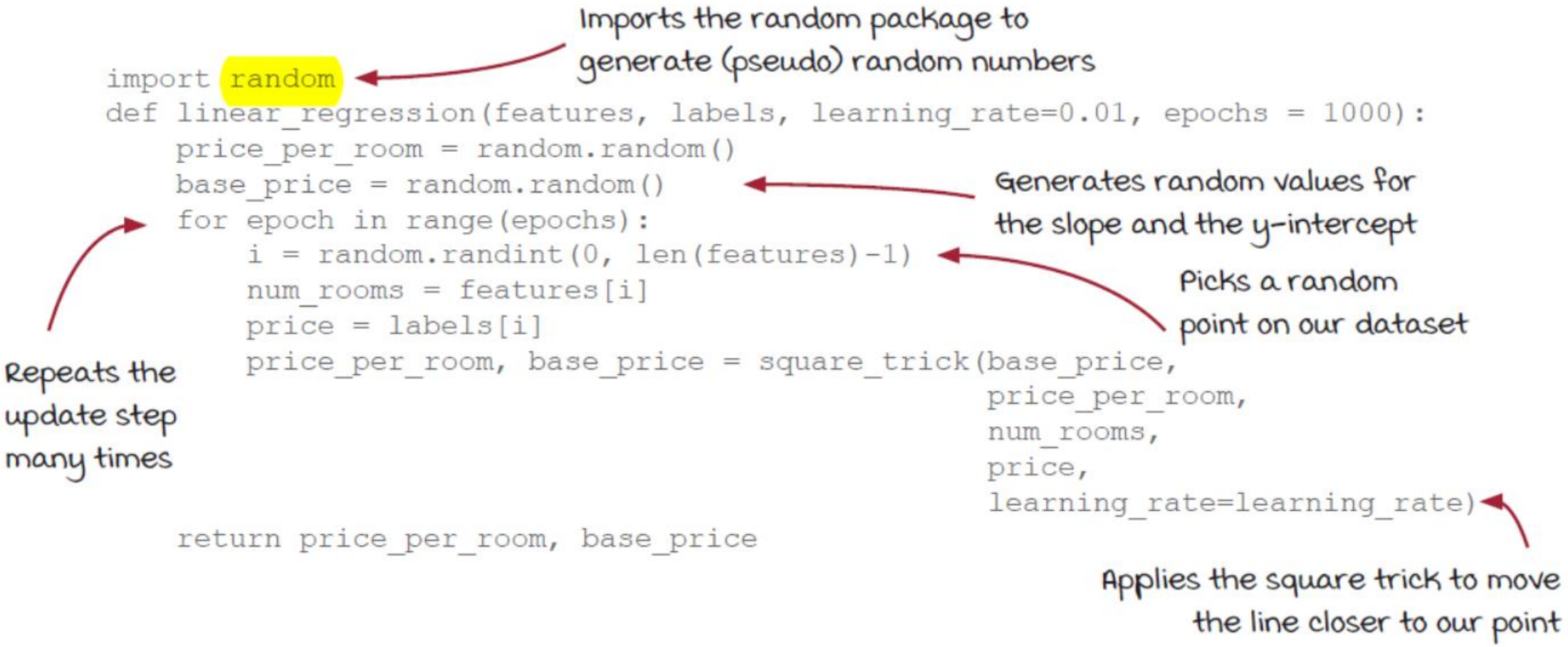
Procedure:

- Start with random values for the slope and y-intercept.
- Repeat many times:
  - Pick a random data point.
  - Update the slope and the y-intercept using the absolute or the square trick.

Linear Regression Algorithm

- Each iteration of the loop is called an epoch, and we set this number at the beginning of our algorithm.
- The simple trick was mostly used for illustration. it doesn’t work very well.
- In real life, we **use the absolute or square trick**, which works a lot better.
- In fact, although both are commonly used, the **square trick is more popular**. Therefore, we’ll use that one for our algorithm, but feel free to use the absolute trick if you prefer.

Linear Regression Algorithm



The general linear regression algorithm

- In real life, we will be working with datasets with many features
- The only difference is that **each of the features** is updated in the **same way that the slope** was updated.
- In the **housing example**, we had **one slope and one y-intercept**. In the **general case**, think of **many slopes** and **still one y-intercept**.

The general linear regression algorithm

- Dataset of m points and n features. Thus, the model has n weights (think of them as the generalization of the slope) and one bias
- The data points are $x^{(1)}, x^{(2)}, \dots, x^{(m)}$.
- Each point is of the form $\boldsymbol{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)})$
- The corresponding labels are $y_1, y_2, \dots, y_m$.
- The weights of the model are $w_1, w_2, \dots, w_n$.
- The bias of the model is b.

The general linear regression algorithm

Inputs:

- A model with equation $\hat{y} = w_1 x_1 + w_2 x_{2+} + \cdots + w_n x_n + b$
- A point with coordinates (x, y)
- A small positive value η (the learning rate)

Output:

- A model with equation $\hat{y} = w_1' x_1 + w_2' x_2 + \cdots + w_n' x_n + b'$ that is closer to the point

Procedure:

- Add $\eta(y - \hat{y})$ to the y-intercept b. Obtain $b' = b + \eta(y - \hat{y})$.
- For $i = 1, 2, \ldots, n$:
  - Add $\eta x(y - \hat{y})$ to the weight w_i. Obtain $w_i' = w_i + \eta r(y - \hat{y})$.

Return: The model with equation $\hat{y} = w_1' x_1 + w_2' x_2 + \cdots + w_n' x_n + b'$

Next Lecture

- ~~Polynomial regression~~