Linear Regression

- Overview
- Training and Optimization
- Model Evaluation
- PyTorch Linear Layer

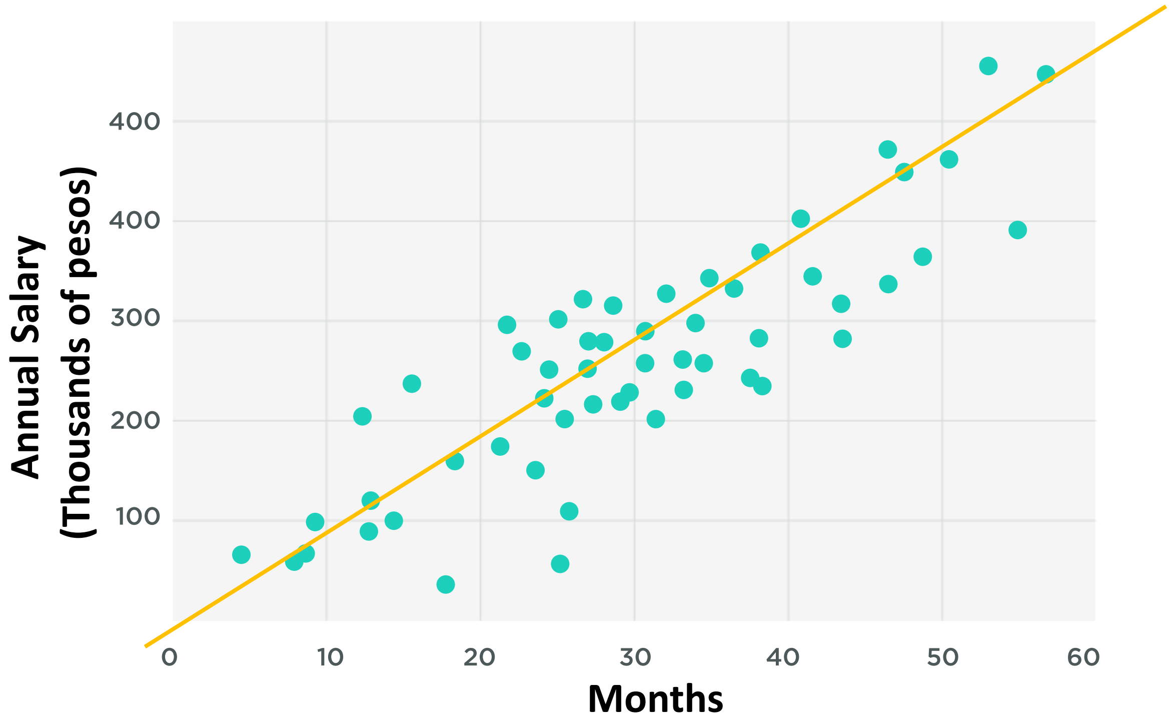
Overview

Linear Regression

Let's consider the following graph and answer the following questions:

- What is observed?
- What would be the salary for an employee with 180 months worked?
- What form would the trend line take?

Relationship between Months Worked and Salary



Line Equation

Linear Regression > Overview

- Linear regression is one of the most widely used techniques in machine learning. Its formula for a single variable is derived from the equation of a line:

$$y = wx+b$$

- Learning involves finding the best parameters (coefficients) for the given data. The best coefficients are those that minimize some measure of error. For linear regression, we use the mean squared error.

Slope

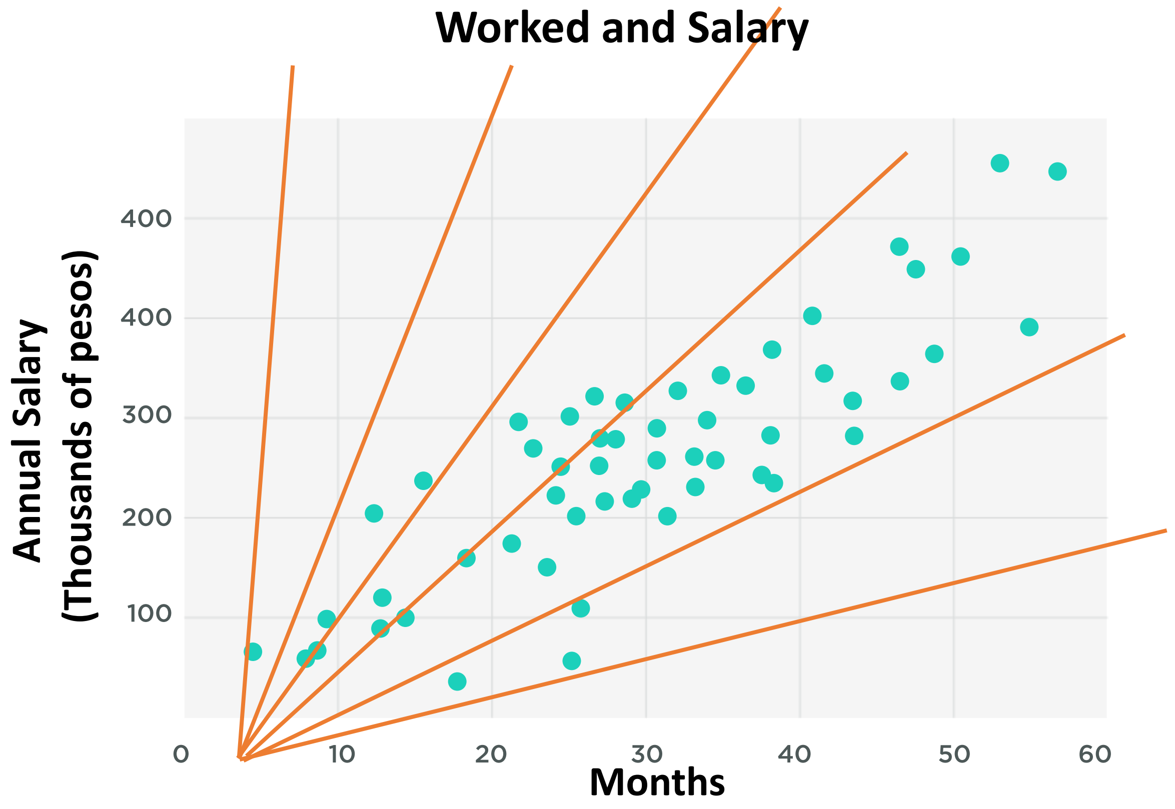
Linear Regression > Overview

Line equation:

$$y = \boldsymbol{w}x + b$$

- The slope (w) indicates how steep our line is. It is also known as the gradient

Relationship between Months Worked and Salary



Bias

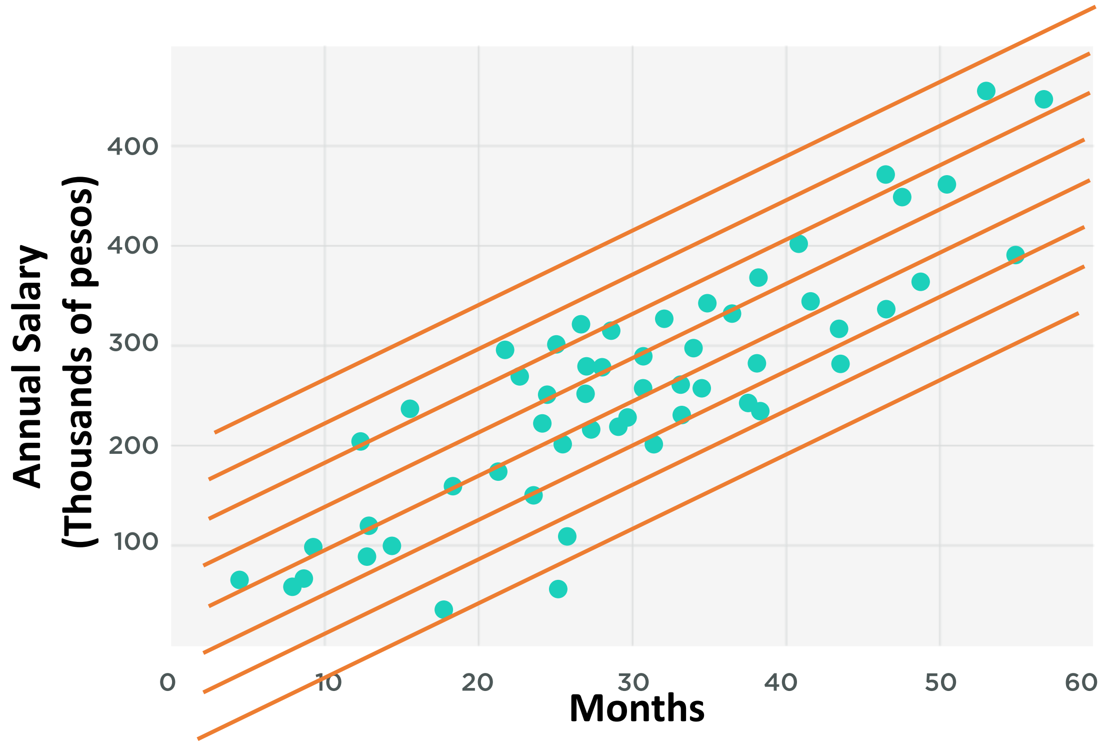
Linear Regression > Overview

Line equation:

$$y = wx + \boldsymbol{b}$$

- The bias (b) indicates how high or low our line is positioned

Relationship between Months Worked and Salary



Slop & Bias in the line equation

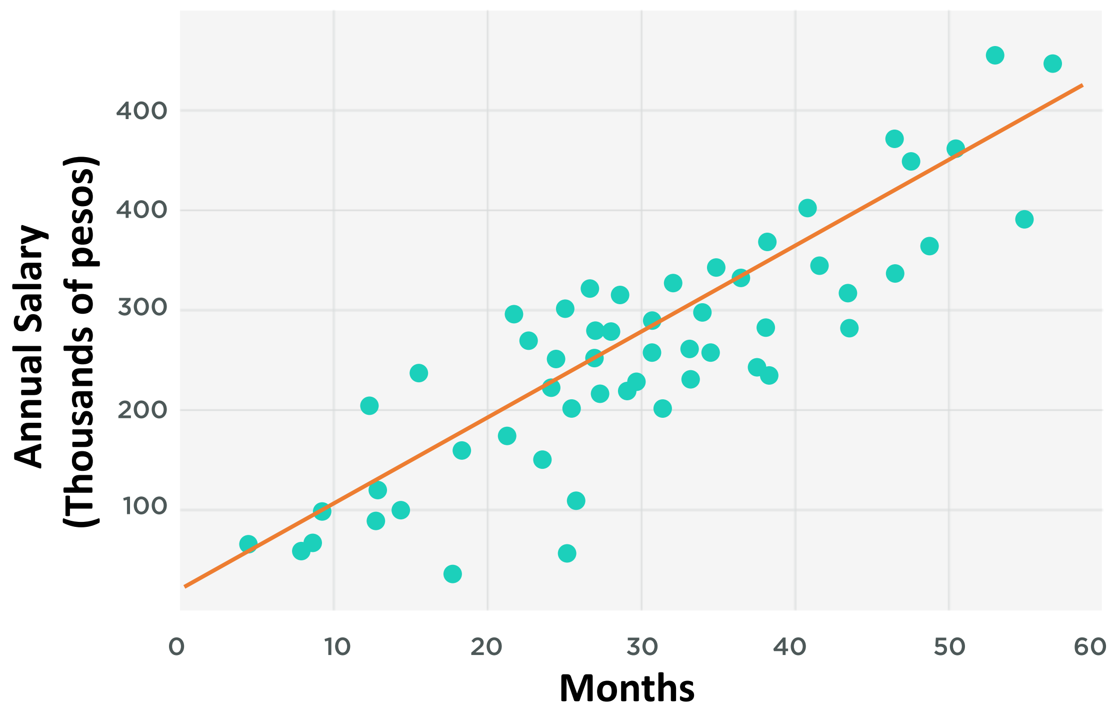
Linear Regression > Overview

- Let's suppose we find the optimal parameters (w, b) for the line that minimizes the data error:

$$y = 0.1350x + 1.3265$$

- We can use this linear regression model to estimate what the results will be for other values of x.

Relationship between Months Worked and Salary



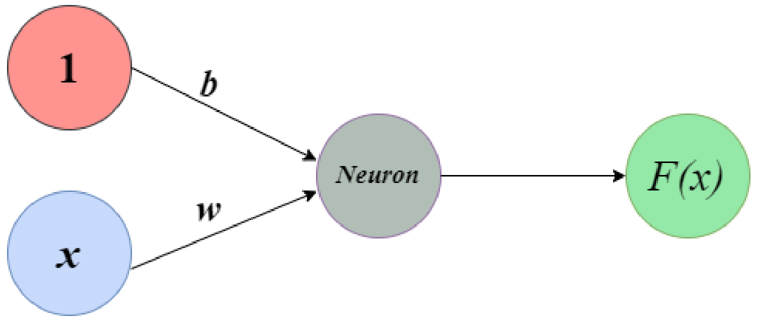
General Equation

Linear Regression > Overview

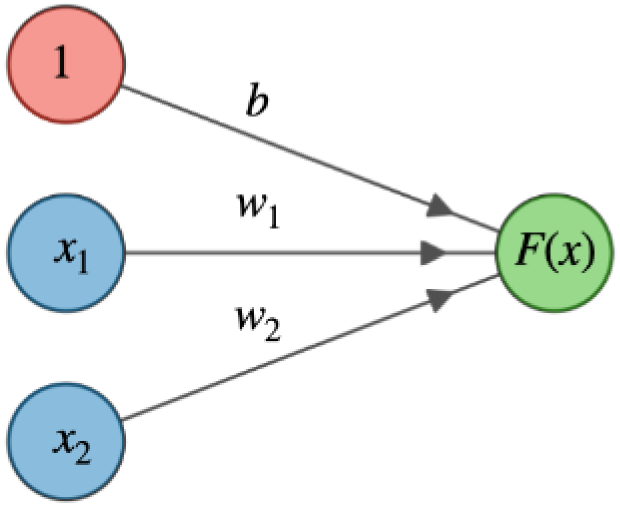
- What does this have to do with neurons?
- Internally, a neuron is a linear function.
- In a generalized way, it can be represented as:

$$F(x) = \sum_{i=1}^{n} w_i + b$$

$$F(x) = wx + b$$



$$F(x) = w_1 x_1 + w_2 x_2 + b$$



Loss Function

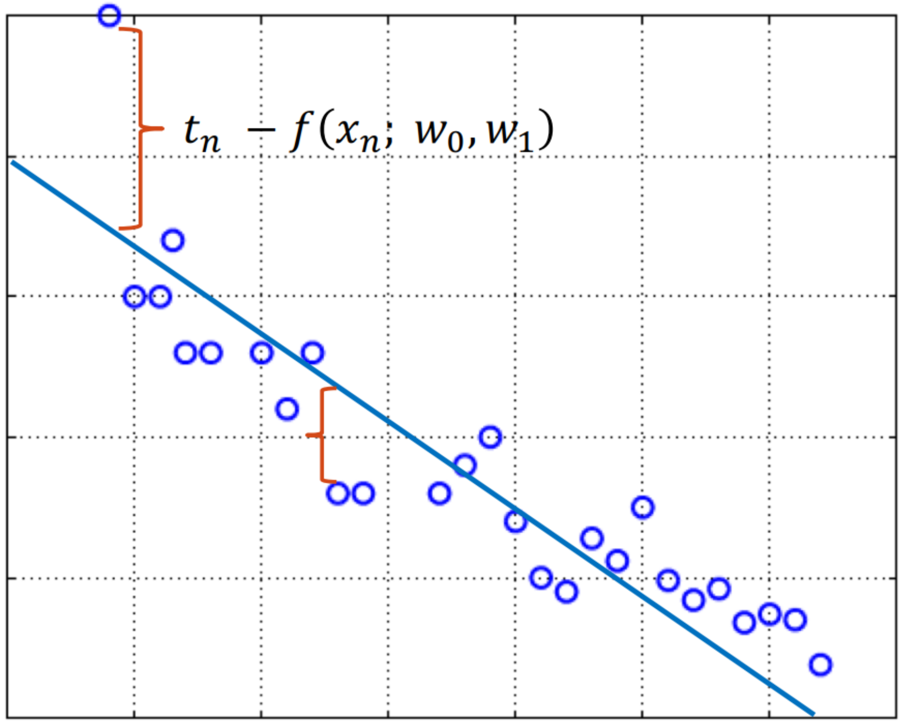
Linear Regression > Overview

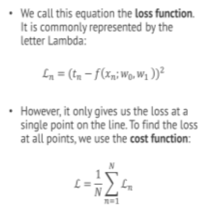
- Since, in the end, the gradient and the bias are parameters, let's make the bias W_0 and the slope W_1.

$$f(x) = w_1 x + \ w_0$$

- Using x_n and t_n to denote the squared distance from the N value to the line, we have:

$$N = (t_n - f(x_n; w_0, w_1\ ))^2$$





- We call this equation the **loss function**. It is commonly represented by the letter Lambda:

$$\mathcal{L}_n = (t_n - f(x_n; w_0, w_1))^2$$

- However, it only gives us the loss at a single point on the line. To find the loss at all points, we use the **cost function**:

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_n$$

Cost Function

Linear Regression > Overview

- We call this equation the **loss function**. It is commonly represented by the letter Lambda:

$$\mathcal{L}_n = (t_n - f(x_n; w_0, w_1))^2$$

- However, it only gives us the loss at a single point on the line. To find the loss at all points, we use the **cost function**:

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_n$$

- We call this equation the **loss function**. It is commonly represented by the letter Lambda:

$$\mathcal{L}_n = (t_n - f(x_n; w_0, w_1))^2$$

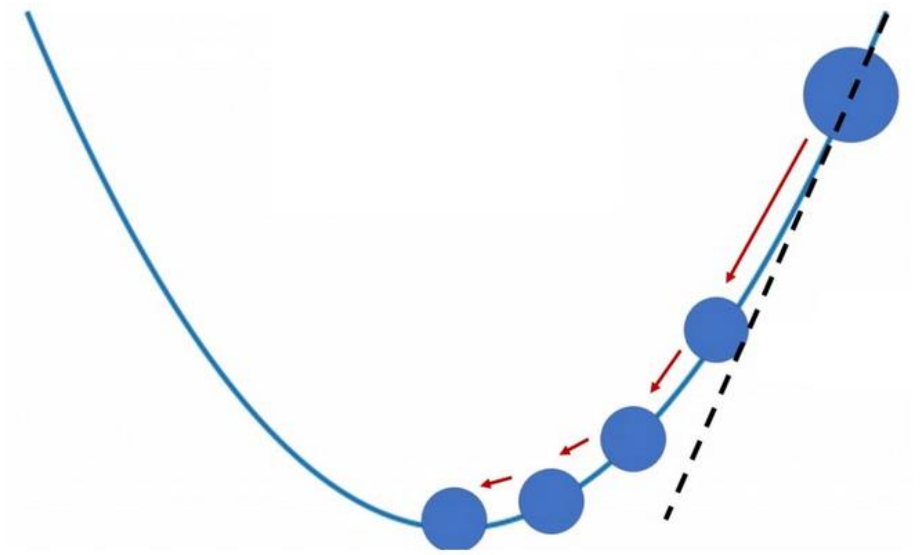
- However, it only gives us the loss at a single point on the line. To find the loss at all points, we use the **cost function**:

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_n$$

Cost Function (cont.)

Linear Regression > Overview

- Since the cost function is a quadratic function, it is possible to find its minimum value when the slope is zero.
- The slope of a function is obtained through its derivative.



Getting Linear Model

Linear Regression > Overview

First, let's find the parameters of the cost function. In the end, we will obtain the linear model.

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_n\left(t_n, f(x_n; w_0, w_1)\right)$$

$$\frac{1}{N}\sum_{n=1}^{N} \left(t_n - f(x_n; w_0, w_1)\right)^2$$

$$\frac{1}{N}\sum_{n=1}^{N} (t_n - w_0 + w_1 x_n)^2$$

Square trinomial

$$(a+b+c)^2 = a^2 + 2ab + 2ac + 2bc + b^2 + c^2$$

$$\boxed{\frac{1}{N}\sum_{n=1}^{N} (w_1^2 x_n^2 + 2w_1 x_n w_0 - 2w_1 x_n t_n + w_0^2 - 2w_0 t_n + t_n^2)}$$

$$\frac{1}{N}\sum_{n=1}^{N} (w_1^2 x_n^2 + 2w_1 x_n (w_0 - t_n) + w_0^2 - 2w_0 t_n + t_n^2)$$

Getting Linear Model (cont.)

Linear Regression > Overview

To find the slope of the cost function

1. Let's partially differentiate the cost function with respect to $\boldsymbol{w1}$.
2. Set it equal to zero to find the global minimum
3. Solve for $w0$ and $w1$

$$\frac{1}{N}\sum_{n=1}^{N}(w_1^2x_n^2 + 2w_1x_nw_0 - 2w_1x_nt_n + \cancel{w_0^2} - \cancel{2w_0t_n} + \cancel{t_n^2})$$

$$\frac{1}{N}\sum_{n=1}^{N}[w_1^2x_n^2 + 2w_1x_nw_0 - 2w_1x_nt_n]$$

Getting Linear Model (cont.)

Linear Regression > Overview

$$w_1^2 \frac{1}{N}\left(\sum_{n=1}^{N} x_n^2\right) + 2w_1 \frac{1}{N}\left(\sum_{n=1}^{N} x_n \left(w_0 - t_n\right)\right)$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = 2w_1 \frac{1}{N}\left(\sum_{n=1}^{N} x_n^2\right) + \frac{2}{N}\left(\sum_{n=1}^{N} x_n \left(w_0 - t_n\right)\right)$$

- Let's partially differentiate the cost function with respect to $\boldsymbol{W_0}$.

$$\frac{1}{N}\sum_{n=1}^{N}(\cancel{w_1^2 x_n^2} + 2w_1 x_n w_0 - \cancel{2w_1 x_n t_n} + w_0^2 - 2w_0 t_n + \cancel{t_n^2})$$

$$\frac{\partial \mathcal{L}}{\partial w_0} = 2w_0 + 2w_1 \frac{1}{N}\left(\sum_{n=1}^{N} x_n\right) - \frac{2}{N}\left(\sum_{n=1}^{N} t_n\right)$$

Getting Linear Model (cont.)

Linear Regression > Overview

- Set it equal to zero to find the global minimum

$$2w_0 + 2w_1 \frac{1}{N}\left(\sum_{n=1}^{N} x_n\right) - \frac{2}{N}\left(\sum_{n=1}^{N} t_n\right) = 0$$

$$2w_0 = \frac{2}{N}\left(\sum_{n=1}^{N} t_n\right) - w_1 \frac{2}{N}\left(\sum_{n=1}^{N} x_n\right)$$

$$w_0 = \frac{1}{N}\left(\sum_{n=1}^{N} t_n\right) - w_1 \frac{1}{N}\left(\sum_{n=1}^{N} x_n\right) \longrightarrow \widehat{w_0} = \bar{t} - w_1\bar{x}$$

Getting Linear Model (cont.)

Linear Regression > Overview

- Solve for $w0$ and $w1$

$$\frac{\partial \mathcal{L}}{\partial w_1} = 2w_1 \frac{1}{N}\left(\sum_{n=1}^{N} x_n^2\right) + \frac{2}{N}\left(\sum_{n=1}^{N} x_n\,(w_0 - t_n)\right)$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = 2w_1 \frac{1}{N}\left(\sum_{n=1}^{N} x_n^2\right) + \frac{2}{N}\left(\sum_{n=1}^{N} x_n\,(\widehat{w_0} - t_n)\right)$$

$$w_1 \frac{2}{N}\left(\sum_{n=1}^{N} x_n^2\right) + \frac{2}{N}\left(\sum_{n=1}^{N} x_n\,(\bar{t} - w_1\bar{x} - t_n)\right)$$

$$w_1 \frac{2}{N}\left(\sum_{n=1}^{N} x_n^2\right) + \bar{t}\frac{2}{N}\left(\sum_{n=1}^{N} x_n\right) - w_1\bar{x}\frac{2}{N}\left(\sum_{n=1}^{N} x_n\right) - \frac{2}{N}\left(\sum_{n=1}^{N} x_n\,t_n\right)$$

Getting Linear Model (cont.)

Linear Regression > Overview

$$\frac{\partial \mathcal{L}}{\partial w_1} = 2w_1\left[\left(\frac{1}{N}\sum_{n=1}^{N} x_n^2\right) - \bar{x}\bar{x}\right] + 2\bar{t}\bar{x} - 2\frac{1}{N}\left(\sum_{n=1}^{N} x_n\, t_n\right)$$

$$\widehat{w_1} = \frac{\frac{1}{N}\left(\sum_{n=1}^{N} x_n\, t_n\right) - \bar{t}\bar{x}}{\left(\frac{1}{N}\sum_{n=1}^{N} x_n^2\right) - \bar{x}\bar{x}}$$

$$\widehat{w_1} = \frac{\overline{xt} - \bar{x}\bar{t}}{\overline{x^2} - (\bar{x})^2}$$

Code Implementation

Linear Regression > Overview

- Let's implement now the linear regression equation getting $w0$ and $w1$ values

$$\widehat{w_0} = \bar{t} - w_1\bar{x}$$

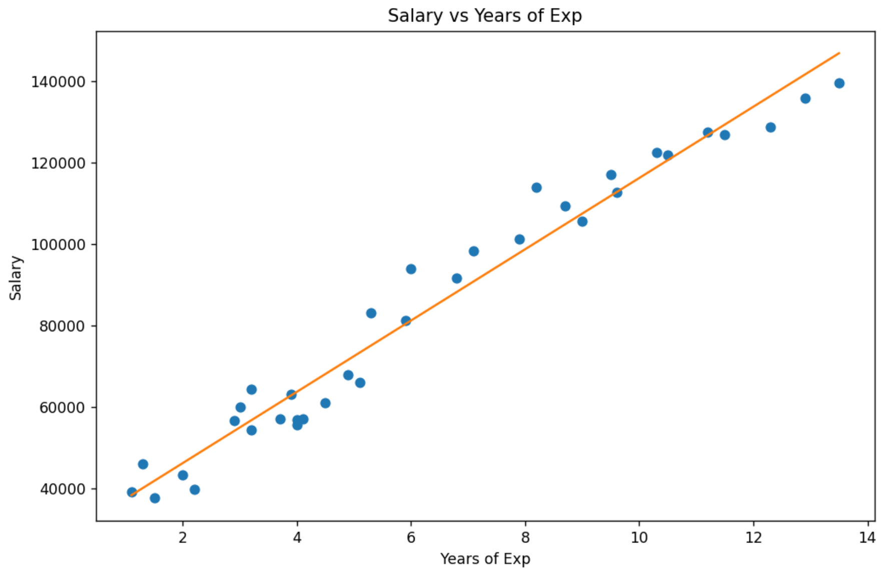
$$\widehat{w_1} = \frac{\overline{xt} - \bar{x}\bar{t}}{\overline{x^2} - (\bar{x})^2}$$

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

plt.rcParams['figure.figsize'] = (12, 8)

data = pd.read_csv('Salary.csv')
print('Columns ', list(data))
x_col = 'YearsExperience'
y_col = 'Salary'
x = data[x_col]
t = data[y_col]
n = len(x)

w1 = ?
w0 = ?

t_hat = w0 + w1 * x

plt.title('Salary vs Years of Exp')
plt.xlabel('Years of Exp')
plt.ylabel('Salary')
plt.plot(x, t, 'o', x, t_hat, '-')
plt.show()
```

Activity

Linear Regression > Overview

- Compute $w0$ and $w1$ in Python to complete the previous code.
- Download the dataset from Canvas



Remember a horizontal line above a letter means the average.

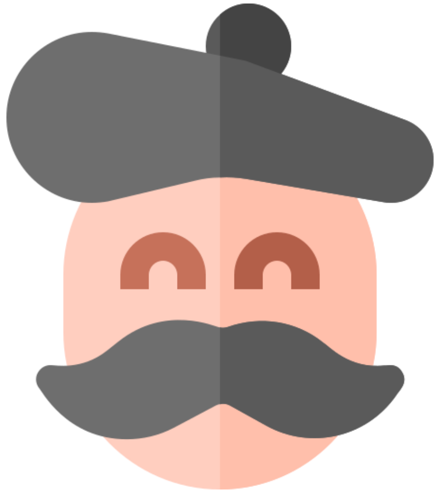
sum(x)/len(x)

TECNOLOGÍA

La Inteligencia Artificial sí tiene impactos ambientales

Mientras por un lado se puede utilizar la IA para fomentar proyectos que ayuden a combatir el cambio climático, su uso y desarrollo también contamina.

vie 09 junio 2023 06:00 AM



EL IMPACTO MEDIOAMBIENTAL DE LA IA: ¿SALVADORA O DESTRUCTORA?

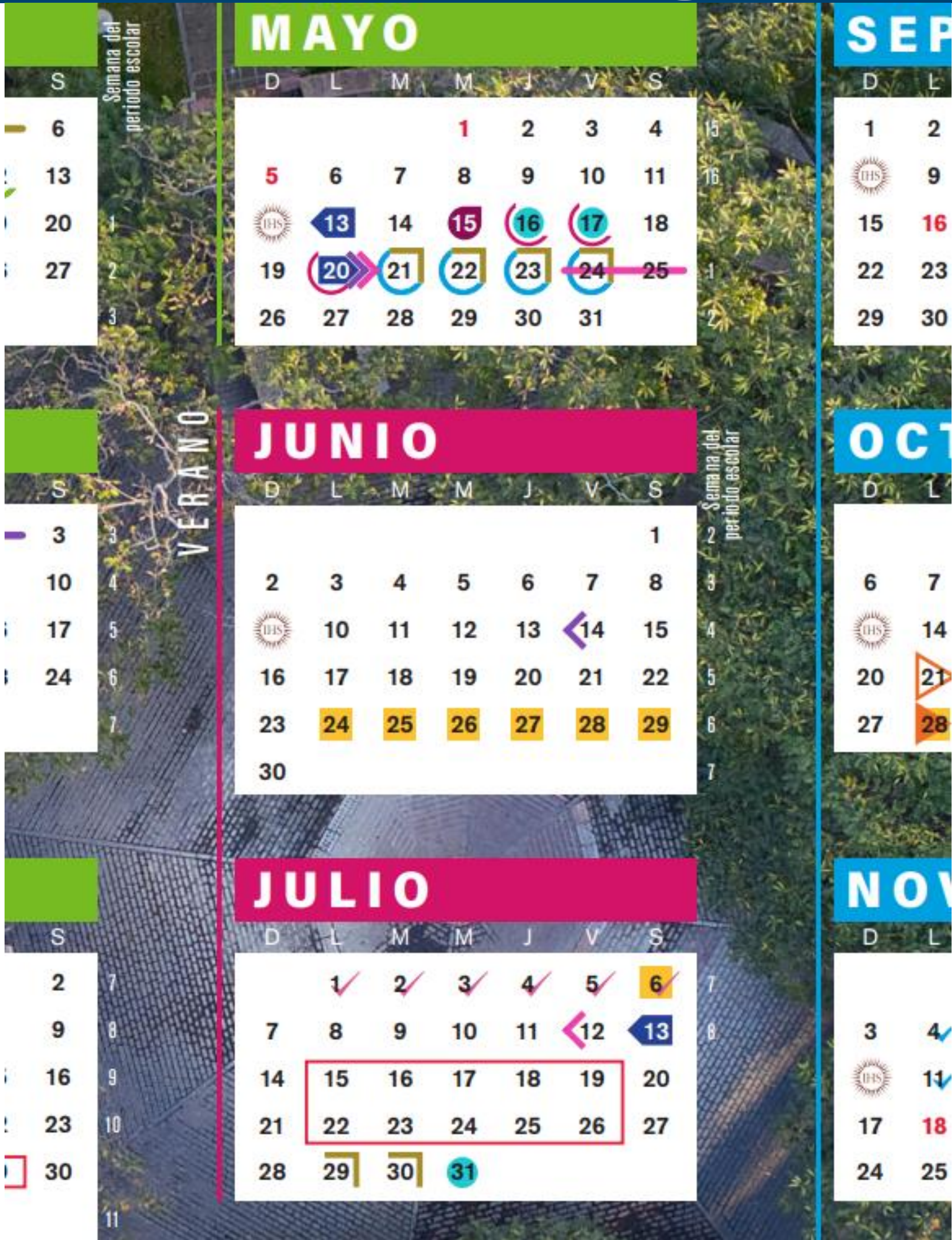
agosto 5, 2022 Algoritmos

La inteligencia artificial a menudo se presenta como la nueva solución para proteger el medio ambiente. De hecho, la IA ya se ha puesto en uso con este propósito; IBM la aplica en su sistema de detección de incendios forestales 'Bee2FireDetection', mientras que Microsoft ha apoyado proyectos como 'Wild me' en el que se utiliza para combatir la extinción de especies.

ITESO, Universidad Jesuita de Guadalajara

Schedule

May 20th	Introduction
May 22th	Python / Libraries
May 27th	Model Evaluation
May 29th	Pytorch
June 3rd	Lineal Regression
June 5th	Model Evaluation
June 10th	Logistic regression
June 12th	PyTorch Implementation



VERANO

Semana del periodo escolar

MAYO

D	L	M	M	J	V	S
			1	2	3	4
5	6	7	8	9	10	11
IHS	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

JUNIO

D	L	M	M	J	V	S
						1
2	3	4	5	6	7	8
IHS	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

JULIO

D	L	M	M	J	V	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Project Checklist

- ☐ **Define the problem**
- ☐ **Collect and explore the data**
- ☐ Pre-process the data
- ☐ Split the data into training and test sets
- ☐ Choose a model and train it
- ☐ Evaluate the model
- ☐ Fine-tune the model
- ☐ Present the results
- ☐ Deploy the model

ETA: June 16th

Define the Problem

1. Clearly state the problem you are trying to solve
2. Identify the goal of your ML model
3. Ensure the goal is specific, such as predicting an event or classifying data accurately

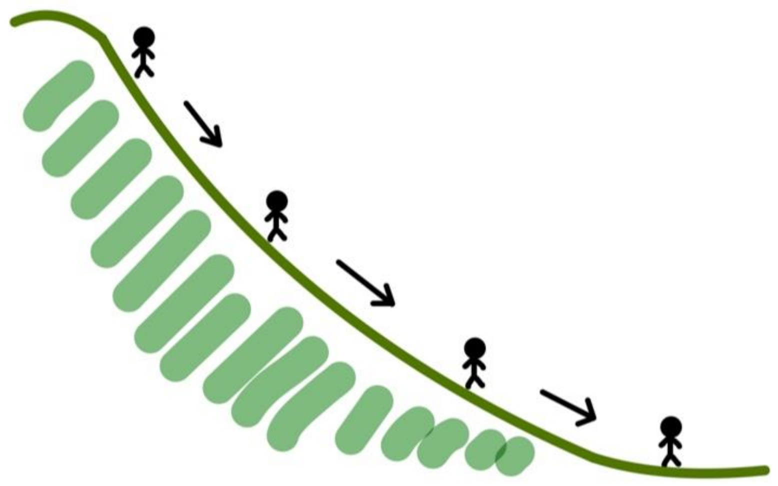
Collect and explore the data

1. Extract data from multiple sources
2. Create the first version of your dataset. E.g. text, images, audio, etc..
3. Define a format for your data. E.g. CSV files, images into folders, etc.
4. Upload to Canvas your dataset as a zip

Gradient Descent

Linear Regression > Training and Optimization

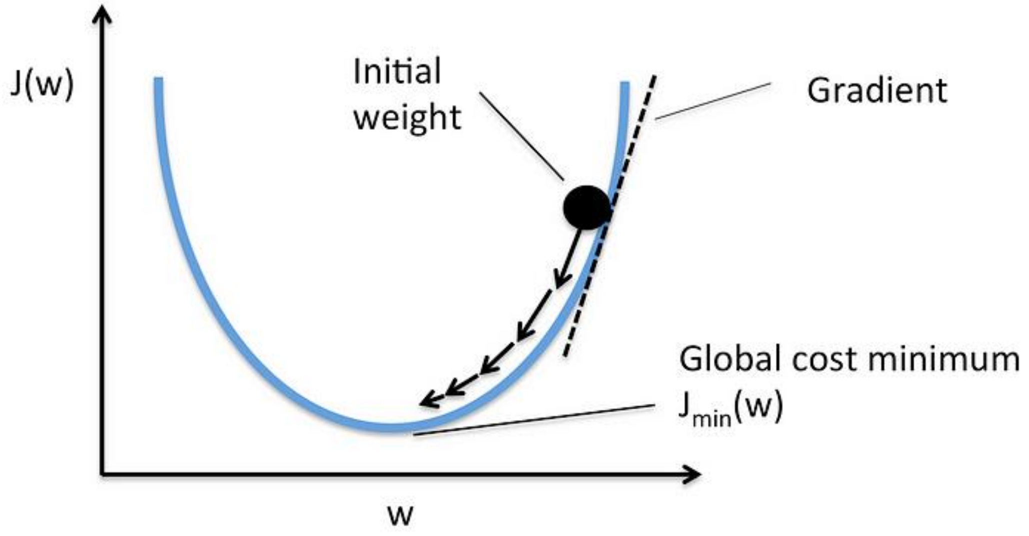
- Gradient descent is one way to find the parameters that **minimize the cost function**.
- Gradient descent an <u>iterative optimization algorithm</u> for finding a local minimum of a differentiable function.
- You are lost in the middle of a mountain in darkness. Pretty much the only way for you to find your way is to <u>keep moving towards</u> somewhere lower than your location. You just need to keep climbing down the mountain until you reach the lowest point.



Gradient Descent (cont.)

Linear Regression > Training and Optimization

- First, initialize the parameters w0 and w1 randomly, and you keep updating them until the minimum cost is returned.
- The minimum cost is achieved when the gradient or the slope of the cost function is equal to 0
- The gradient **always points towards the direction of the greatest increase**, so to find the minimum or descent point, we update the weights in the opposite direction of the gradient.



Pseudocode of the gradient descent algorithm

Linear Regression > Training and Optimization

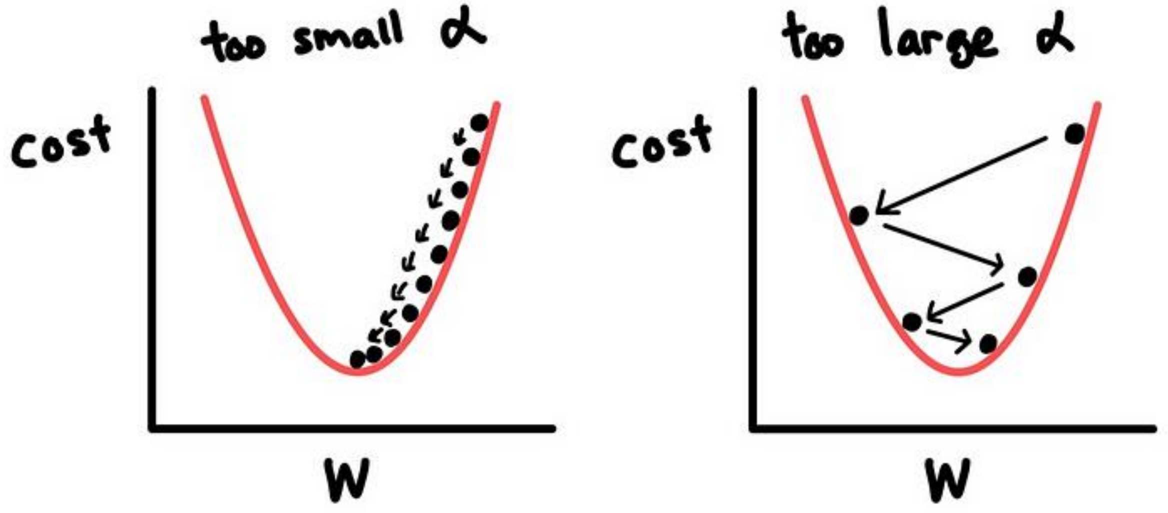
- The equation has w as weights, alpha as the learning rate and the gradient of the cost function j(w,b).
- Even though we use a partial derivative of the cost function J, we call it a derivative or gradient.
- you may find α which we haven't talked about yet. This is variable is called **learning rate**

$$\mathrm{w} = \mathrm{w}\text{-}\alpha * \frac{\partial}{\partial w} J(w, b)$$

Learning Rate

Linear Regression > Training and Optimization

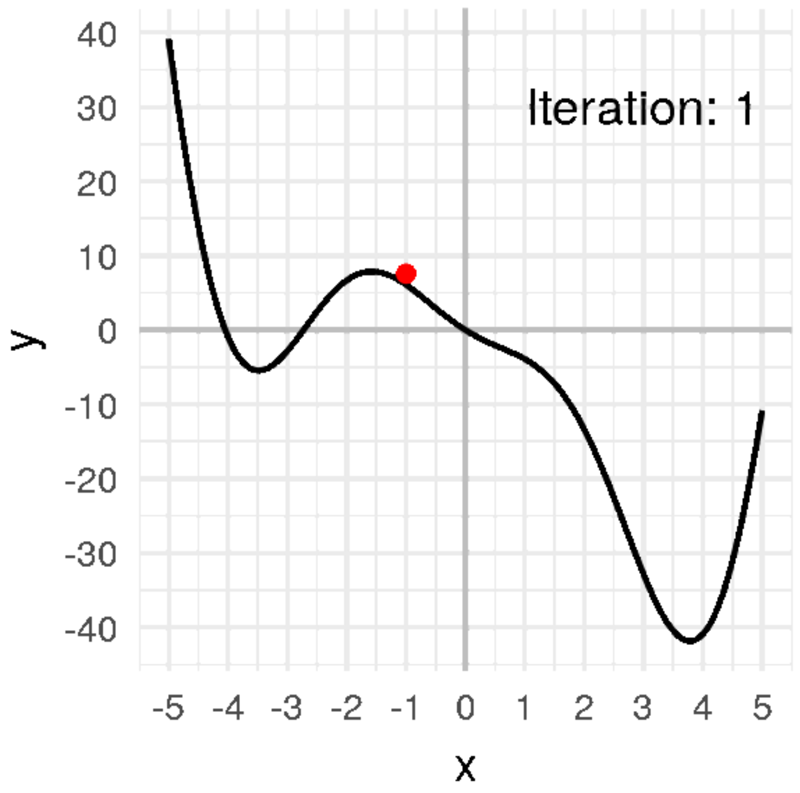
- In machine learning, learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration.
- In other words, the larger the learning rate, the larger the step size.
- Too small learning rate or too large learning rate can return unwanted results.



Learning Rate

Linear Regression > Training and Optimization

- Think of a simple function that has one number as input and one number as an output. The goal will be to start the input point and figure out in which direction to move in order to make the output lower.
- The slope of the function where the input point is can be calculated and the point will move to the right if the slope is negative or move to the left if the slope is positive.
- The aim is to iterate this in order to find the global minimum, notice that when the point approaches the minimum the number of steps increase. This helps the point from overshooting.



Training with numpy

Linear Regression > Training and Optimization

- We are computing the gradient manually using the mean square error (MSE)

```python
import numpy as np

# f = 2 * x (f = w * x)
X = np.array([1, 2, 3, 4], dtype=np.float32)
Y = np.array([2, 4, 6, 8], dtype=np.float32)
w = 0.0

# model prediction
def forward(x):
    return w * x

def loss(y, y_predicted):
    return ((y_predicted - y) ** 2).mean()

# gradient | MSE = 1/N * (w*x - y)**2 | dJ/dw = 1/N 2X (w*x - y)
def gradient(x, y, y_predited):
    return np.dot(2*x, y_predited - y).mean()

print(f'Prediction before training: f(5) = {forward(5):.3f}')

# Training
learning_rate = 0.01
n_iters = 20
for epoch in range(n_iters):
    # prediction = forward pass
    y_pred = forward(X)
    # loss
    l = loss(Y, y_pred)
    # gradients
    dw = gradient(X, Y, y_pred)
    # update weights using the formula. Negative learning rate *
gradient
    w -= learning_rate * dw
    print(f'epoch {epoch + 1}: w = {w:.3f}, loss = {l:.8f}')

print(f'Prediction after training: f(5) = {forward(5):.3f}')
```

Training with autograd

Linear Regression > Training and Optimization

- Changed the input data to use tensors instead of Numpy arrays
- The only parameter that requires autograd is **w**

```
import torch

# f = 2 * x (f = w * x)
X = torch.tensor([1, 2, 3, 4], dtype=torch.float32)
Y = torch.tensor([2, 4, 6, 8], dtype=torch.float32)

w = torch.tensor(0.0, dtype=torch.float32, requires_grad=True)

# model prediction
def forward(x):
    return w * x

def loss(y, y_predicted):
    return ((y_predicted - y) ** 2).mean()

print(f'Prediction before training: f(5) = {forward(5):.3f}')

learning_rate = 0.01
n_iters = 100
for epoch in range(n_iters):
    # prediction = forward pass
    y_pred = forward(X)
    # loss
    l = loss(Y, y_pred)
    # gradients = backward pass
    l.backward() # dl/dw
    # update weights using the formula. Negative learning rate *
gradient
    with torch.no_grad():
        w -= learning_rate * w.grad
    # zero gradients
    w.grad.zero_()

    print(f'epoch {epoch + 1}: w = {w:.3f}, loss = {l:.8f}')
print(f'Prediction after training: f(5) = {forward(5):.3f}')
```

Training with torch

Linear Regression > Training and Optimization

- Using loss and the optimizer from PyTorch

```python
import torch
import torch.nn as nn

# f = 2 * x (f = w * x)
X = torch.tensor([1, 2, 3, 4], dtype=torch.float32)
Y = torch.tensor([2, 4, 6, 8], dtype=torch.float32)
w = torch.tensor(0.0, dtype=torch.float32, requires_grad=True)

# model prediction
def forward(x):
    return w * x

print(f'Prediction before training: f(5) = {forward(5):.3f}')

# Training
learning_rate = 0.01
n_iters = 100
loss = nn.MSELoss()
optimizer = torch.optim.SGD([w], lr=learning_rate)

for epoch in range(n_iters):
    # prediction = forward pass
    y_pred = forward(X)
    # loss
    l = loss(Y, y_pred)
    # gradients = backward pass
    l.backward() # dl/dw
    # update weights using the formula. Negative learning rate *
gradient
    optimizer.step()
    # zero gradients
    optimizer.zero_grad()
    print(f'epoch {epoch + 1}: w = {w:.3f}, loss = {l:.8f}')

print(f'Prediction after training: f(5) = {forward(5):.3f}')
```

Training with torch model

Linear Regression > Training and Optimization

- Creating a torch Linear model
- Numpy is now not required

```python
import torch
import torch.nn as nn

# f = 2 * x (f = w * x)
X = torch.tensor([[1], [2], [3], [4]], dtype=torch.float32)
Y = torch.tensor([[2], [4], [6], [8]], dtype=torch.float32)

n_samples, n_feature = X.shape
print(n_samples, n_feature) # 4 1

# Defining the model
input_size = n_feature
output_size = n_feature
model = nn.Linear(input_size, output_size)
X_test = torch.tensor([5], dtype=torch.float32)
print(f'Prediction before training: f(5) = {model(X_test).item():.3f}')

# Training
learning_rate = 0.01
n_iters = 100
loss = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

for epoch in range(n_iters):
    # prediction = forward pass
    y_pred = model(X)
    # loss
    l = loss(Y, y_pred)
    # gradients = backward pass
    l.backward() # dl/dw
    # update weights using the formula. Negative learning rate * gradient
    optimizer.step()
    # zero gradients
    optimizer.zero_grad()
    [w, b] = model.parameters()
    print(f'epoch {epoch + 1}: w = {w[0][0].item():.3f}, loss = {l:.8f}')

print(f'Prediction after training: f(5) = {model(X_test).item():.3f}')
```

Training with class model

Linear Regression > Training and Optimization

- The LinearRegression class was implemented
- The forward method is always required

```python
import torch
import torch.nn as nn

X = torch.tensor([[1], [2], [3], [4]], dtype=torch.float32)
Y = torch.tensor([[2], [4], [6], [8]], dtype=torch.float32)

n_samples, n_feature = X.shape
input_size = n_feature
output_size = n_feature
print(n_samples, n_feature) # 4 1

class LinearRegression(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LinearRegression, self).__init__()
        # define layers
        self.lin = nn.Linear(input_dim, output_dim)
    def forward(self, x):
        return self.lin(x)

model = LinearRegression(input_size, output_size)
X_test = torch.tensor([5], dtype=torch.float32)
print(f'Prediction before training: f(5) = {model(X_test).item():.3f}')

# Training
learning_rate = 0.01
n_iters = 100
loss = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

for epoch in range(n_iters):
    y_pred = model(X)
    l = loss(Y, y_pred)
    l.backward() # dl/dw
    optimizer.step()
    optimizer.zero_grad()
    [w, b] = model.parameters()
    print(f'epoch {epoch + 1}: w = {w[0][0].item():.3f}, loss = {l:.8f}')

print(f'Prediction after training: f(5) = {model(X_test).item():.3f}')
```

Activity

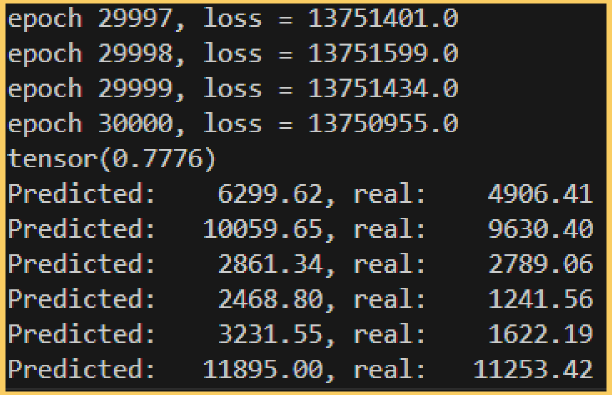
Linear Regression > Training and Optimization

Based on the Medical Cost Personal dataset:

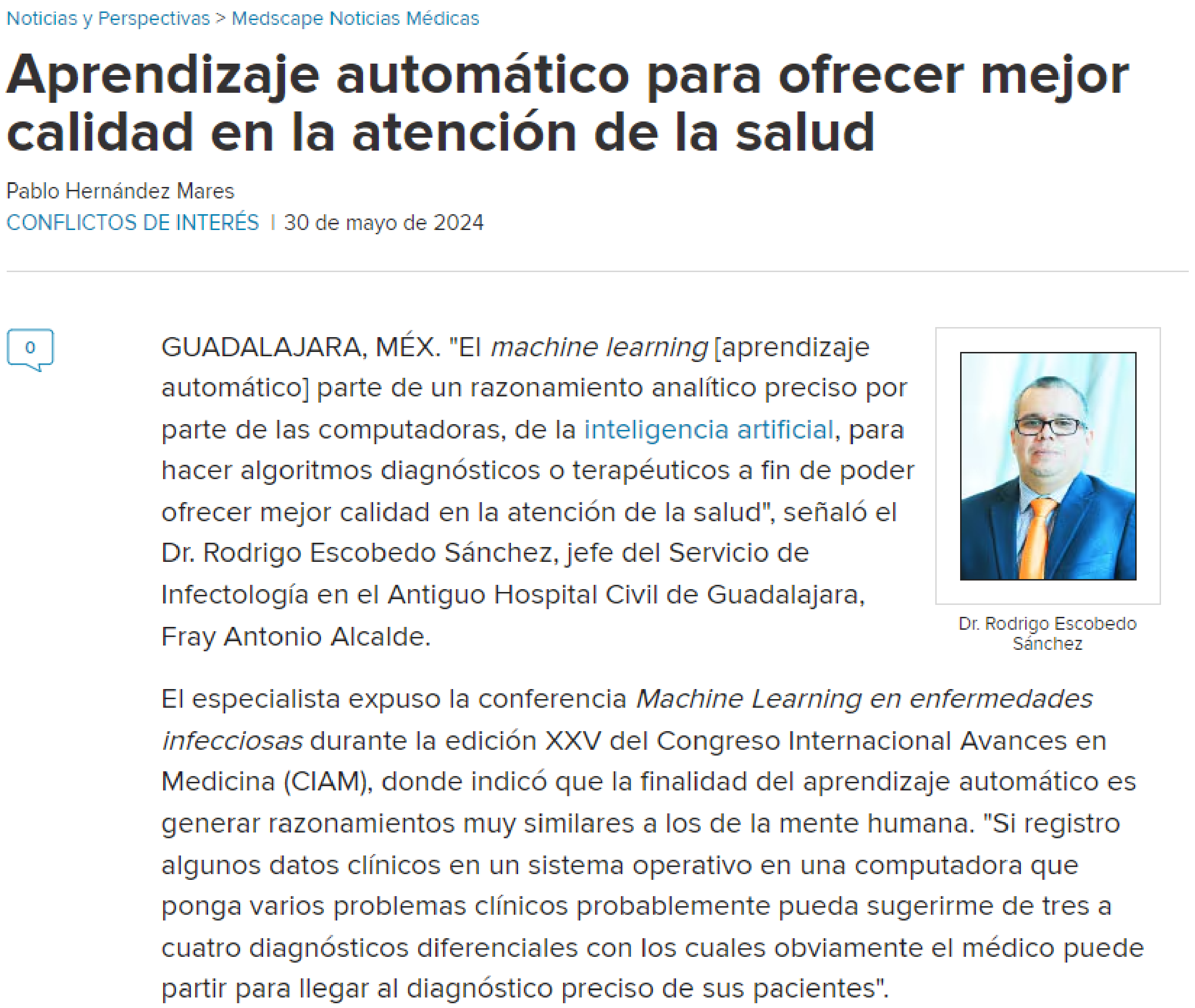
- Write the linear regression model code to predict "charges".
- Expected loss <= 19,000,000

Remember to download your report as a CSV file.

Expected:



```
epoch 29997, loss = 13751401.0
epoch 29998, loss = 13751599.0
epoch 29999, loss = 13751434.0
epoch 30000, loss = 13750955.0
tensor(0.7776)
Predicted:    6299.62, real:    4906.41
Predicted:   10059.65, real:    9630.40
Predicted:    2861.34, real:    2789.06
Predicted:    2468.80, real:    1241.56
Predicted:    3231.55, real:    1622.19
Predicted:   11895.00, real:   11253.42
```



Noticias y Perspectivas > Medscape Noticias Médicas

Aprendizaje automático para ofrecer mejor calidad en la atención de la salud

Pablo Hernández Mares

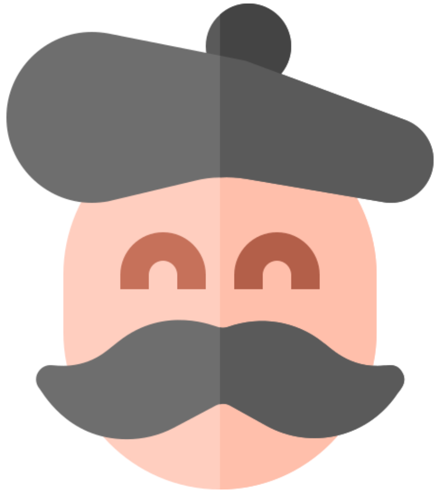
CONFLICTOS DE INTERÉS | 30 de mayo de 2024

0

GUADALAJARA, MÉX. "El *machine learning* [aprendizaje automático] parte de un razonamiento analítico preciso por parte de las computadoras, de la inteligencia artificial, para hacer algoritmos diagnósticos o terapéuticos a fin de poder ofrecer mejor calidad en la atención de la salud", señaló el Dr. Rodrigo Escobedo Sánchez, jefe del Servicio de Infectología en el Antiguo Hospital Civil de Guadalajara, Fray Antonio Alcalde.

Dr. Rodrigo Escobedo Sánchez

El especialista expuso la conferencia *Machine Learning en enfermedades infecciosas* durante la edición XXV del Congreso Internacional Avances en Medicina (CIAM), donde indicó que la finalidad del aprendizaje automático es generar razonamientos muy similares a los de la mente humana. "Si registro algunos datos clínicos en un sistema operativo en una computadora que ponga varios problemas clínicos probablemente pueda sugerirme de tres a cuatro diagnósticos diferenciales con los cuales obviamente el médico puede partir para llegar al diagnóstico preciso de sus pacientes".

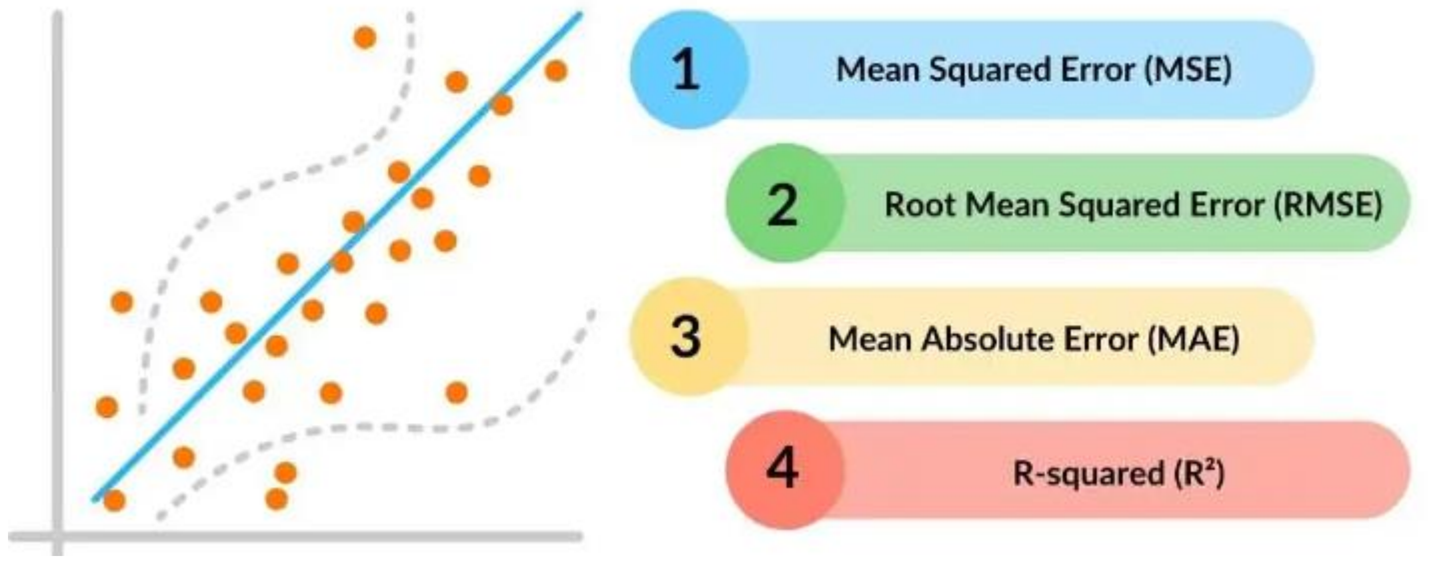


ITESO, Universidad Jesuita de Guadalajara

Model Evaluation

Linear Regression

- The objective of Linear Regression is to find a line that minimizes the prediction error of all the data points.
- The essential step in any machine learning model is to evaluate the accuracy of the model.


1 Mean Squared Error (MSE)
2 Root Mean Squared Error (RMSE)
3 Mean Absolute Error (MAE)
4 R-squared (R²)

Base Code

Linear Regression > Model Evaluation

- Let’s implement now a basic regression code using sklearn

```
import pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load the diabetes dataset
diabetes = load_diabetes()
df = pd.DataFrame(data=diabetes.data, columns=diabetes.feature_names)

# Separate independent and dependent features
X = df.iloc[:,0:-1]
Y = df.iloc[:,-1]

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=42)

# Standardizing the dataset
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

comparison = pd.DataFrame()
comparison['Actual'] = y_test
comparison['Predicted'] = y_pred
print(comparison)
```

Bias

Linear Regression > Model Evaluation

- Residuals are the building blocks of most of the metrics. In simple terms, a residual is a **difference between the actual value and the predicted one**.

residual = actual - prediction

- The simplest error measure would be the sum of residuals, sometimes referred to as **bias**.
- However, as the residuals of opposing signs offset each other, we can obtain a model that generates predictions with a very low bias, while not being accurate at all.

R-Squared (R^2)

Linear Regression > Model Evaluation

R-squared (R^2), also known as the coefficient of determination, represents the proportion of variance explained by a model.

$$R^2 = 1 - \frac{RSS}{TSS} = 1 - \frac{\Sigma_{i=1}^{N}(y_i - \hat{y_i})^2}{\Sigma_{i=1}^{N}(y_i - \bar{y_i})^2}$$

Where:

- RSS is the residual sum of squares, which is the sum of squared residuals. This value captures the prediction error of a model.
- TSS is the total sum of squares. Mean of all the observed actuals.

Interpreting R^2 Score

Linear Regression > Model Evaluation

The formula for calculating the R2 score is as follows:

$$R^2 = 1 - \frac{\text{Sum of Squared Residuals}}{\text{Total Sum of Squares}}$$

- $R^2 = 1$: Perfect fit. The model **explains all the variability** in the target variable.
- $R^2 = 0$: The model does not explain any variability in the target variable.
- $0 < R^2 < 1$: The model explains a proportion of the variability in the target variable.
- A higher R^2 score indicates a better fit,

R^2 Score in Python

Linear Regression > Concepts

- Based on the previous code, let's compute the r2_score from the sklearn metrics Python library
- Using R^2, we can evaluate how much better our model fits the data as compared to the simple mean model.
- We can think of a positive R^2 value in terms of improving the performance of a baseline model — something along the lines of a skill score.

```python
from sklearn.metrics import r2_score
print(r2_score(y_test,y_pred))

from torchmetrics.regression import R2Score
r2 = R2Score()
r2_score = r2(y_pred, y_test)
```

Mean Squared Error (MSE)

Linear Regression > Model Evaluation

- Mean squared error (MSE) is one of the most popular evaluation metrics. As shown in the following formula, MSE is closely related to the residual sum of squares. The difference is that we are now interested in the **average error instead of the total error**.
- As the residuals are squared, MSE puts a significantly heavier penalty on large errors.

$$\text{MSE} = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2$$

MSE in Python

Linear Regression > Concepts

- This code calculates the mean squared error (MSE) between the predicted values **y_pred** and the true values **y_test**.
- It imports the **mean_squared_error** function from the **sklearn.metrics** module and then prints the result of applying this function to **y_test** and **y_pred**.
- The MSE is a measure of the average squared difference between the estimated values and the actual values.

```python
from sklearn.metrics import mean_squared_error
print(mean_squared_error(y_test, y_pred))
```

MSE in PyTorch

Linear Regression > Concepts

- This code defines a Python function **mse_loss** that calculates the mean squared error (MSE) loss between predicted **y_pred** and true **y_true** values.
- It utilizes the **torch.nn.MSELoss**() function from the PyTorch library to compute the MSE loss and then returns the result.

```python
def mse_loss(y_pred, y_true):
    """
    Calculates the mean squared error (MSE) loss between
predicted and true values.

    Args:
    - y_pred: predicted values
    - y_true: true values

    Returns:
    - mse_loss: mean squared error loss
    """
    mse = torch.nn.MSELoss()
    mse_loss = mse(y_pred, y_true)
    return mse_loss
```

Root Mean Squared Error (RMSE)

Linear Regression > Model Evaluation

- Root mean squared error (RMSE) is closely related to MSE, as it is simply the square root of the latter.
- By taking the square we bring the metric back to the scale of the target variable, so it is easier to interpret and understand. However, one fact that is often overlooked is that although RMSE is on the same scale as the target, an RMSE of 10 does not actually mean we are off by 10 units on average

$$\mathrm{RMSE} = \sqrt{\mathrm{MSE}} = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2}$$

RMSE in Python

Linear Regression > Concepts

- This code imports the function **root_mean_squared_error** from the **sklearn.metrics** module and then prints the result of applying this function to the **y_test** and **y_pred** arrays.

```python
from sklearn.metrics import root_mean_squared_error
print(root_mean_squared_error(y_test, y_pred))
```

Mean Absolute Error (MAE)

Linear Regression > Model Evaluation

- The Mean absolute error represents the **average of the absolute difference** between the actual and predicted values in the dataset.
- It measures the average of the residuals in the dataset.
- Similar to MSE and RMSE, MAE is also scale-dependent, so we cannot compare it between different datasets

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=0}^{n-1} |y_i - \hat{y}_i|$$

MAE in Python

Linear Regression > Concepts

- The code imports the **mean_absolute_error** function from **sklearn.metrics** and then prints the mean absolute error between the true values (**y_test**) and the predicted values (**y_pred**).

```python
from sklearn.metrics import mean_absolute_error
print(mean_absolute_error(y_test, y_pred))

import torch.nn as nn
loss = nn.L1Loss()
```

MAE in PyTorch

Linear Regression > Concepts

- The code defines the **MAE function** in PyTorch that calculates and returns the mean absolute error between the predicted values (**y_pred**) and the true values (**y_true**).

```python
import torch

def mae(y_pred, y_true):
    loss = torch.mean(torch.abs(y_pred - y_true))
    return loss
```

When to use each evaluation metric

Linear Regression > Model Evaluation

Metric	Purpose	Range of values	Interpretation
R2R^2R2 (Coefficient of determination)	Evaluate the percentage of variability explained by the model	0 to 1 (can be negative if the model is worse than the mean)	An R^2closer to 1 indicates a better fit of the model. A negative value indicates a very poor model.
MSE (Mean Squared Error)	Evaluate the mean of the squared errors	0 to ∞	A lower value indicates a better fit. It penalizes larger errors more due to the squaring.
RMSE (Root Mean Squared Error)	Evaluate the square root of the mean of the squared errors	0 to ∞	Like MSE but in the same units as the response variable. A lower value indicates a better fit.
MAE (Mean Absolute Error)	Evaluate the mean of the absolute errors	0 to ∞	A lower value indicates a better fit. It is less sensitive to large errors compared to MSE.

When to use each evaluation metric

Linear Regression > Model Evaluation

While choosing the metrics, consider a few of the following questions:

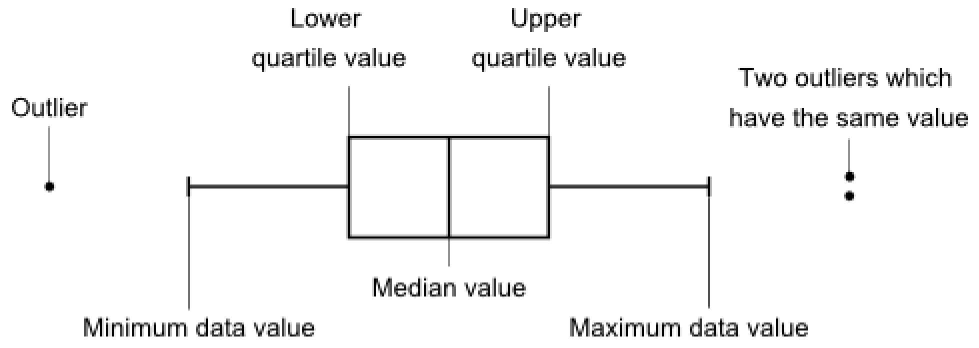
- Do you expect frequent outliers in the dataset? If so, how do you want to account for them?
- Is there a business preference for over forecasting or under forecasting?
- Do you want a scale-dependent or scale-independent metric?

metrics.explained_variance_score
metrics.max_error
metrics.mean_absolute_error
metrics.mean_squared_error
metrics.root_mean_squared_error
metrics.mean_squared_log_error
metrics.root_mean_squared_log_error
metrics.median_absolute_error
metrics.r2_score
metrics.mean_poisson_deviance
metrics.mean_gamma_deviance
metrics.mean_absolute_percentage_error
metrics.d2_absolute_error_score
metrics.d2_pinball_score
metrics.d2_tweedie_score

Boxplot

Linear Regression > Data Analysis

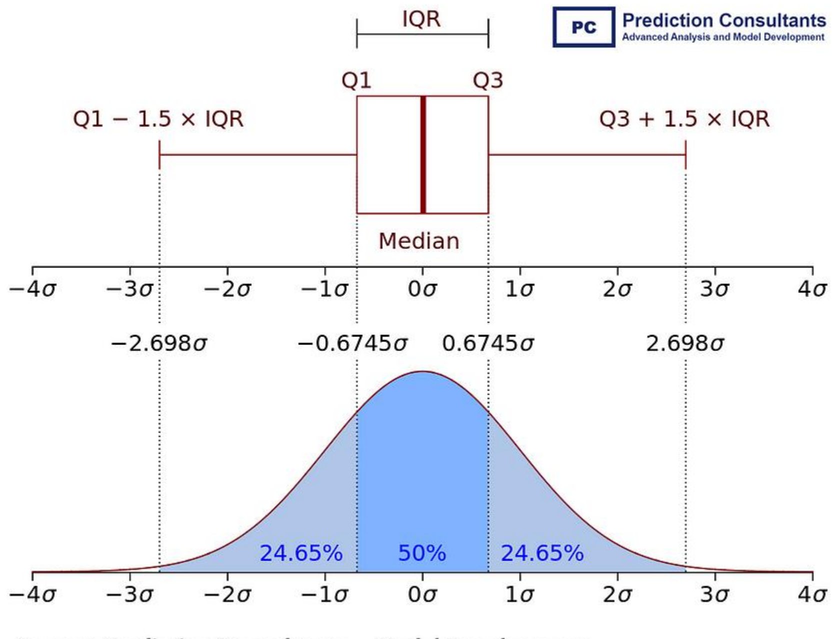
- The line splitting the box in two represents the median value. This shows that 50% of the data lies on the left-hand side of the median value and 50% lies on the right-hand side.
- The left edge of the box represents the lower quartile; it shows the value at which the first 25% of the data falls up to.
- The values at which the horizontal lines stop at are the values of the upper and lower values of the data. The single points on the diagram show the outliers.



Boxplot (cont.)

Linear Regression > Data Analysis

- Boxplot it’s a very good tool for understanding how is my data distributed



Source: Prediction Consultants – Model Development

```python
# Plotting box to show the distribution of charges according to
these features

fig = plt.figure(dpi=220)
plt.rcParams["axes.labelsize"] = 8
fig.subplots_adjust(hspace=0.4, wspace=0.4)
ax = fig.add_subplot(2, 2, 1)
sns.boxplot(data=df, x='smoker', y='charges', ax=ax)
ax = fig.add_subplot(2, 2, 2)
sns.boxplot(data=df, x='region', y='charges', ax=ax)
ax = fig.add_subplot(2, 2, 3)
sns.boxplot(data=df, x='sex', y='charges', ax=ax)
ax = fig.add_subplot(2, 2, 4)
sns.boxplot(data=df, x='children', y='charges', ax=ax)
plt.show()
```

lmplot

Linear Regression > Data Analysis

- It is a combination of Scatter Plot and Regression Line.
- A regression line is simply a single line that best fits the data

```python
# Scatter plots to show the correlation between features and charges
sns.lmplot(data=df, x='age', y='charges', hue='smoker')
plt.show()
sns.lmplot(data=df, x='bmi', y='charges', hue='smoker')
plt.show()
```

Correlation

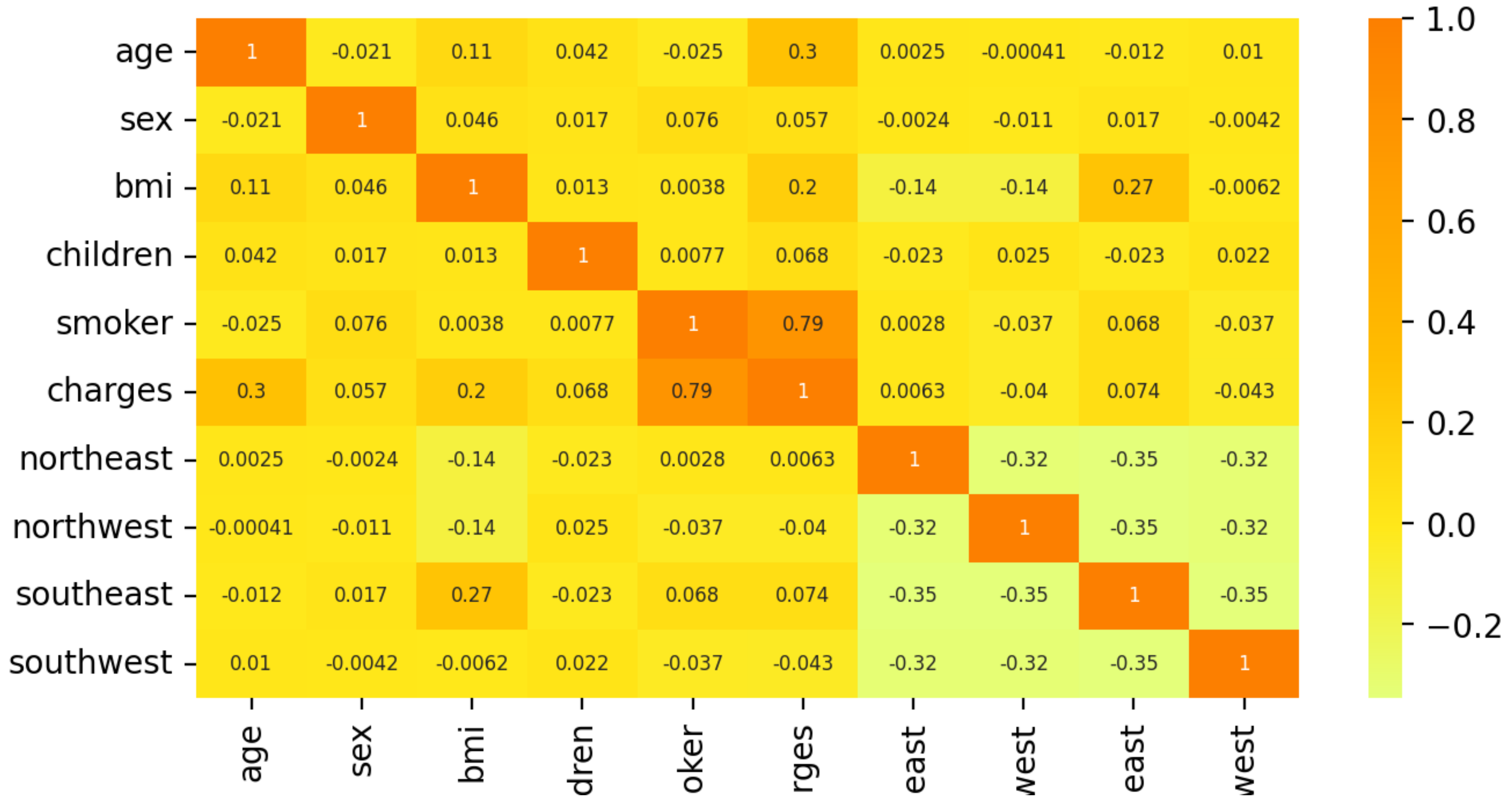
Linear Regression > Data Analysis

- When we have several variables, we may want to see what correlations there are among them.
- We can compute a **correlation matrix** that includes the correlations between the different variables in the dataset.
- When loaded into a Pandas *DataFrame*, we can use the **corr()** method to get the correlation matrix.

```
# Creating a Pearson correlation matrix to visualise correlations
between features

df_corr = df.drop(['log_charges'], axis = 1)
corr = df_corr.corr()
fig, ax = plt.subplots(dpi=200)
sns.heatmap(corr, cmap = 'Wistia', annot= True, ax=ax,
annot_kws={"size": 6})
plt.show()
```

Correlation

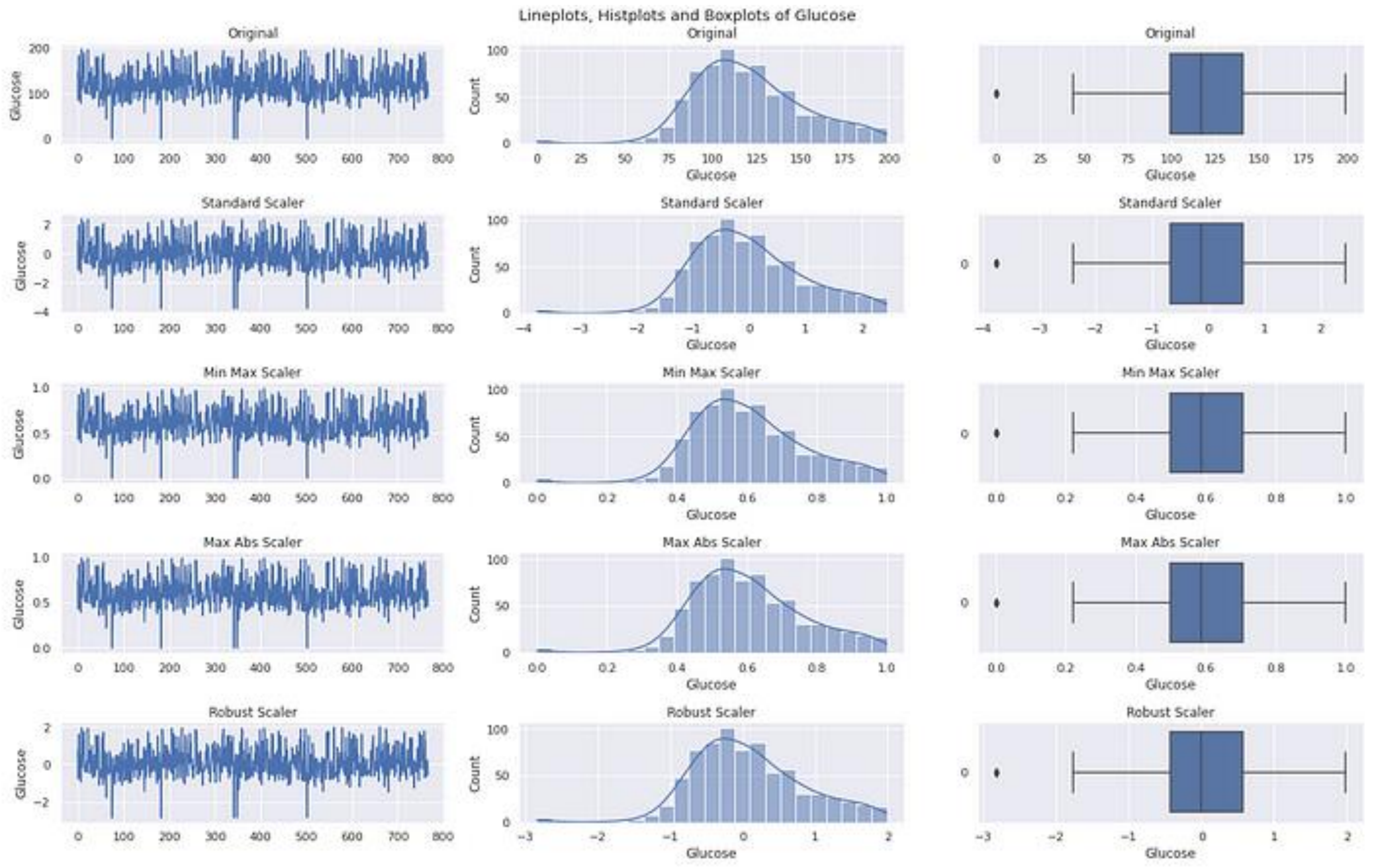
Linear Regression > Data Analysis



	age	sex	bmi	dren	oker	rges	east	west	east	west
age	1	-0.021	0.11	0.042	-0.025	0.3	0.0025	-0.00041	-0.012	0.01
sex	-0.021	1	0.046	0.017	0.076	0.057	-0.0024	-0.011	0.017	-0.0042
bmi	0.11	0.046	1	0.013	0.0038	0.2	-0.14	-0.14	0.27	-0.0062
children	0.042	0.017	0.013	1	0.0077	0.068	-0.023	0.025	-0.023	0.022
smoker	-0.025	0.076	0.0038	0.0077	1	0.79	0.0028	-0.037	0.068	-0.037
charges	0.3	0.057	0.2	0.068	0.79	1	0.0063	-0.04	0.074	-0.043
northeast	0.0025	-0.0024	-0.14	-0.023	0.0028	0.0063	1	-0.32	-0.35	-0.32
northwest	-0.00041	-0.011	-0.14	0.025	-0.037	-0.04	-0.32	1	-0.35	-0.32
southeast	-0.012	0.017	0.27	-0.023	0.068	0.074	-0.35	-0.35	1	-0.35
southwest	0.01	-0.0042	-0.0062	0.022	-0.037	-0.043	-0.32	-0.32	-0.35	1

Scalers

Linear Regression > Data Analysis



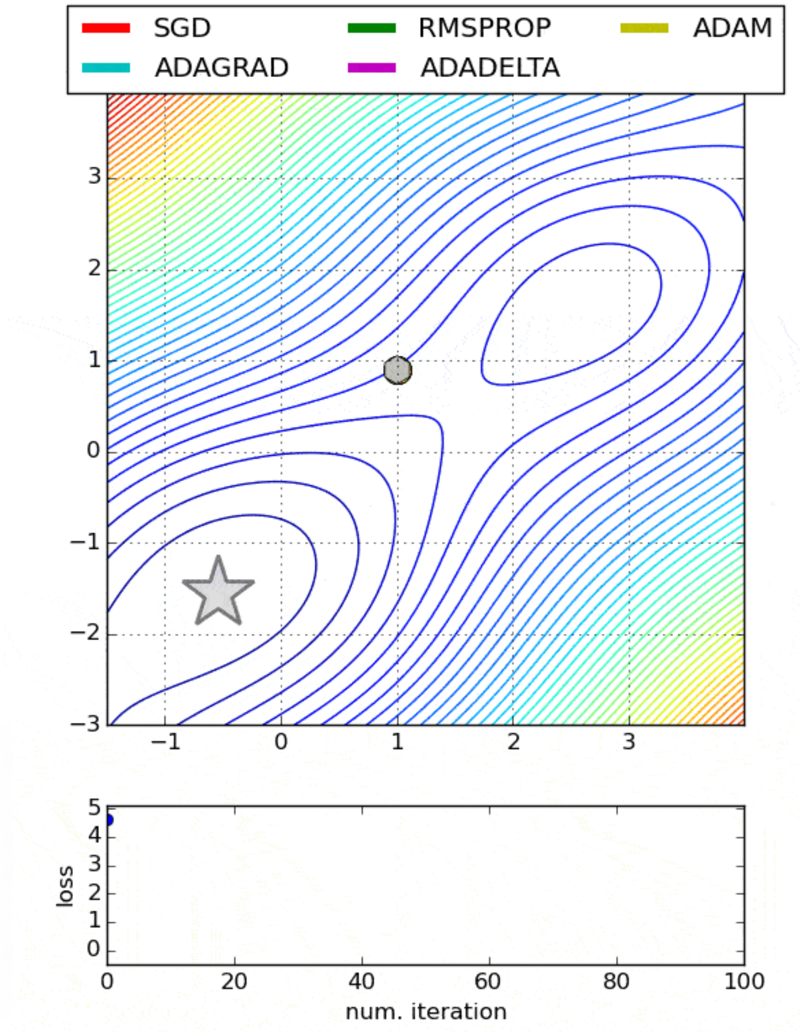
Adagrad (Adaptive Gradient Algorithm)

Linear Regression > Fine Tuning > Adaptive Optimizers

- It adapts the learning rate to the parameters performing small updates for frequently occurring features and large updates for the rarest ones. If the learning rate is too much small, we simply can't update weights and the consequence is that the network doesn't learn anymore.

$$v_t = v_{t-1} + dw_t^2$$

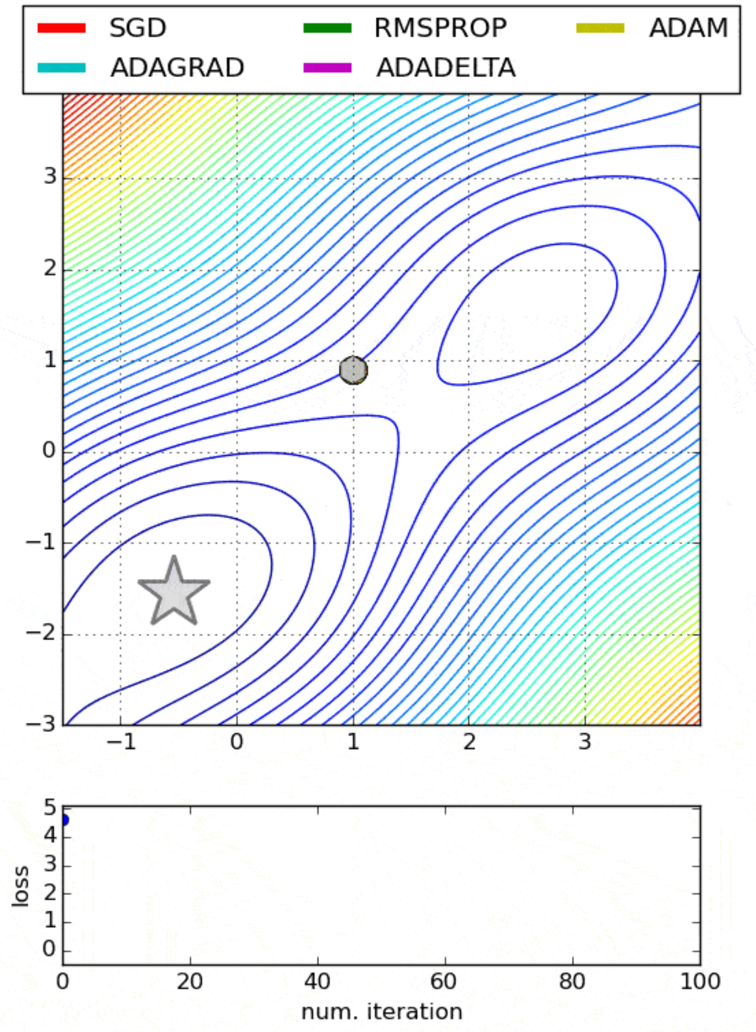
$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{v_t + \varepsilon}} dw_t$$



Adadelta

Linear Regression > Fine Tuning > Adaptive Optimizers

- It improves the previous algorithm by introducing a history window which sets a fixed number of past gradients to take in consideration during the training. In this way, we don't have the problem of the vanishing learning rate.



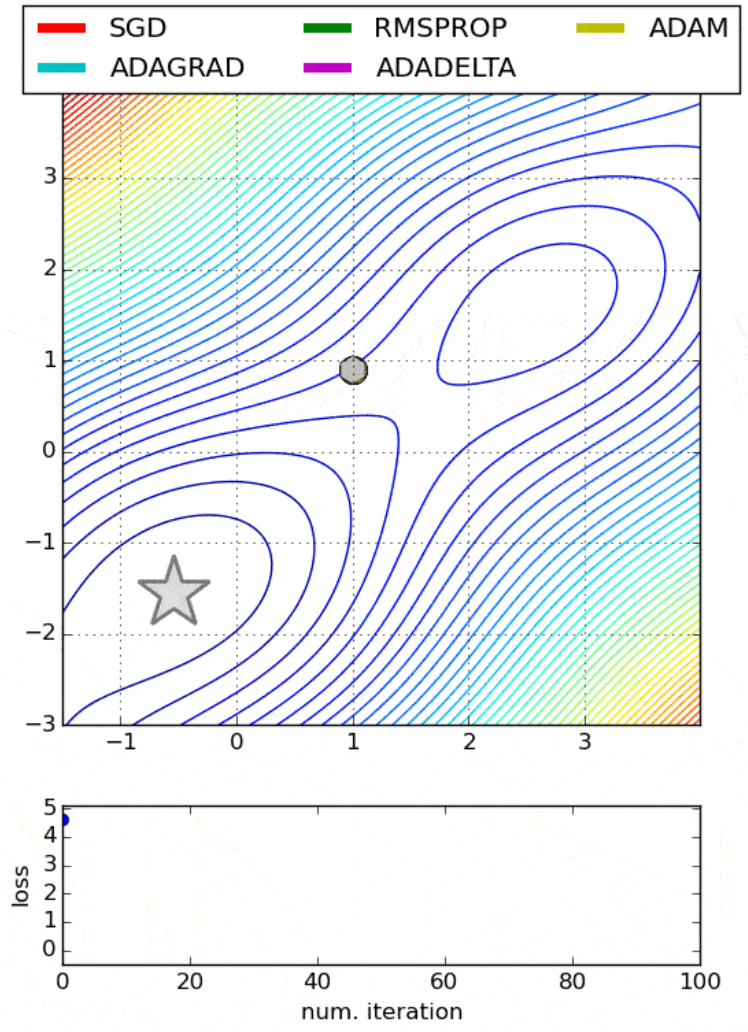
RMSprop

Linear Regression > Fine Tuning > Adaptive Optimizers

- It is very similar to Adadelta.
- The only difference is in the way they manage the past gradients.

$$v_t = \beta v_{t-1} + (1 - \beta) dw_t^2$$

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{v_t + \varepsilon}} dw_t$$



Adam

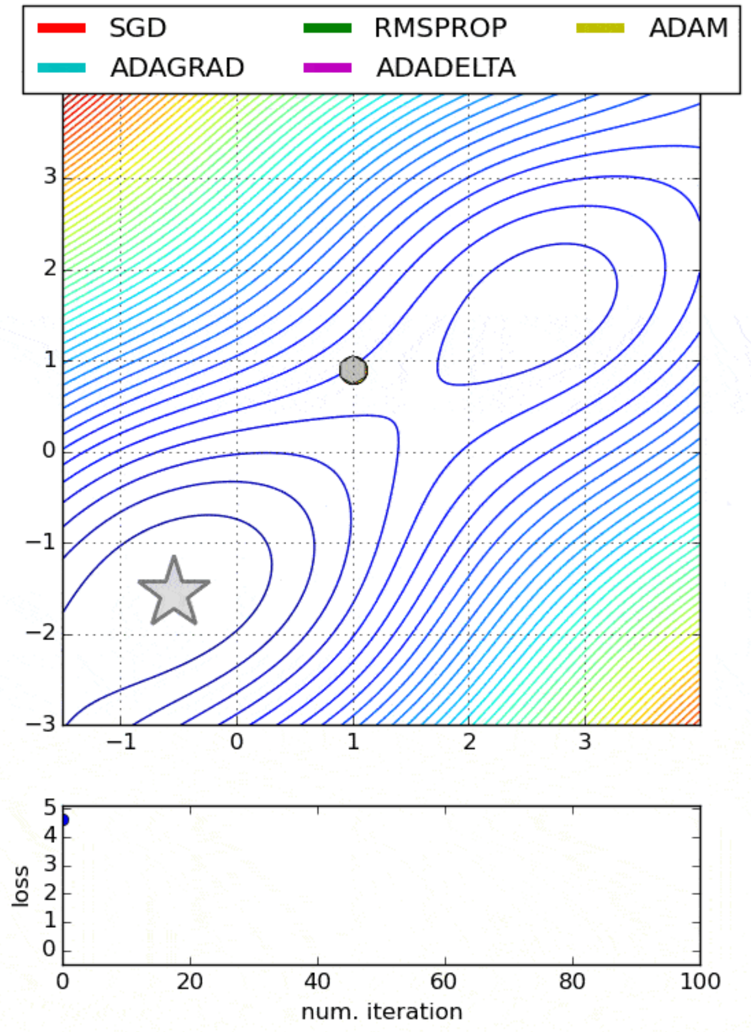
Linear Regression > Fine Tuning > Adaptive Optimizers

- At a high level, Adam combines Momentum and RMSProp algorithms.
- Keeps track of the exponentially moving averages for computed gradients and squared gradients, respectively.

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1) dw_t \xrightarrow{\text{bias}} \hat{v}_t = \frac{v_t}{1 - \beta_1^t}$$

$$s_t = \beta_2 s_{t-1} + (1 - \beta_2) dw_t^2 \xrightarrow[\text{correction}]{} \hat{s}_t = \frac{s_t}{1 - \beta_2^t}$$

$$w_t = w_{t-1} - \frac{\alpha \hat{v}_t}{\sqrt{\hat{s}_t} + \varepsilon} dw_t$$



Activity

Linear Regression > Fine Tuning

Using the sklearn dataset called fetch_california_housing:

1. Analyse the data and plot it
2. Implement a linear regression model
3. Get the R^2.

You can find multiple loss functions from the torch.nn library

Project Checklist

- ☑ Define the problem
- ☑ Collect and explore the data
- ☐ **Pre-process the data**
- ☐ **Split the data into training and test sets**
- ☐ Choose a model and train it
- ☐ Evaluate the model
- ☐ Fine-tune the model
- ☐ Present the results
- ☐ Deploy the model

Pre-process the data

Analyze the data using:

- Correlation matrix for all features
- Correlation of individual features, plot it
- Select features to include in your model
- Get a balanced data
- Select a proper scaler
- Split your data in training and testing dataset