21. 10. 8. 오전 11:35 112-Regression.ipynb - Colaboratory

Regression Data creation

1 

import pandas as pd

2

import os

3

import numpy as np

4

import matplotlib.pyplot as plt 5

import torch

1 

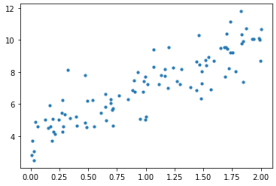
m = 100

2

X = 2 \* torch.rand(m, 1)

3

y = 4 + 3 \* X + torch.randn(m, 1)

1 plt.scatter(X, y, s=10) <matplotlib.collections.PathCollection at 0x7f947ace9590> 

1 

from sklearn.model\_selection import train\_test\_split

2

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

1 

plt.scatter(X\_train, y\_train, s=10)

2

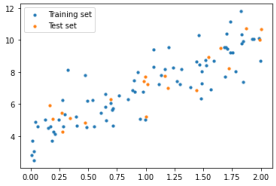
plt.scatter(X\_test, y\_test, s=10)

3

plt.legend(['Training set', 'Test set'])

https://colab.research.google.com/drive/1aXMeQkZreJdJGDUlKNl9HyFOSU5RyFSP#printMode=true 1/13

21. 10. 8. 오전 11:35 112-Regression.ipynb - Colaboratory

<matplotlib.legend.Legend at 0x7f946b656210> Regression model

1 

X\_train\_3 = X\_train[:3] 2

y\_train\_3 = y\_train[:3]

1 X\_train\_3, y\_train\_3 (tensor([[1.7067],

[1.5265],

[0.5538]]), tensor([[9.4494],

[8.4113],

[4.5896]]))

1 plt.scatter(X\_train\_3, y\_train\_3) <matplotlib.collections.PathCollection at 0x7f946b5ee590> 

Hypothesis

H(x) = Wx+b

1 

W = torch.zeros(1, requires\_grad=True) 2

b = torch.zeros(1, requires\_grad=True) 3

hypothesis = X\_train\_3 \* W + b

1 hypothesistensor([[0.],

https://colab.research.google.com/drive/1aXMeQkZreJdJGDUlKNl9HyFOSU5RyFSP#printMode=true 2/13

21. 10. 8. 오전 11:35 112-Regression.ipynb - Colaboratory

[0.],

[0.]], grad\_fn=<AddBackward0>)

Compute loss

cost(W, b) = mean((H(x) - y)^2)

1 cost = torch.mean((hypothesis - y\_train\_3) \*\* 2) 

1 cost tensor(60.3686, grad\_fn=<MeanBackward0>)

Gradient descent

미분으로 계산

1 y\_train\_3 tensor([[9.4494],

[8.4113],

[4.5896]])

1 

## dC/dW

2

sum((2/3) \* ((W \* X\_train\_3 + b) - y\_train\_3) \* X\_train\_3) tensor([-21.0059], grad\_fn=<AddBackward0>)

1 

## dC/db

2

sum((2/3) \* ((W \* X\_train\_3 + b) - y\_train\_3)) tensor([-14.9669], grad\_fn=<AddBackward0>)

torch.optim 라이브러리 활용

1 import torch.optim as optim 

Optimizer 설정 - Stochastic gradient descent 를 활용하여 W와 b를 최적화. learning rate=0.01

1 optimizer = optim.SGD([W, b], lr=0.01)

https://colab.research.google.com/drive/1aXMeQkZreJdJGDUlKNl9HyFOSU5RyFSP#printMode=true 3/13

21. 10. 8. 오전 11:35 112-Regression.ipynb - Colaboratory

최적화 과정 - 3가지가 항상 붙어다님.

1 

hypothesis = X\_train\_3 \* W + b

2

cost = torch.mean((hypothesis - y\_train\_3) \*\* 2)

1 

optimizer.zero\_grad() # 모든 gradient를 0으로 초기화

2

cost.backward(retain\_graph=True) # gradient 계산하여 (parameters).grad를 저장 3

optimizer.step() # step으로 parameter를 개선

gradient 확인

1 W.grad, b.grad (tensor([-21.0059]), tensor([-14.9669]))

1 print(W, b) tensor([0.2101], requires\_grad=True) tensor([0.1497], requires\_grad=True)

1 step이후 확인

1 

hypothesis = X\_train\_3 \* W + b

2

hypothesis

tensor([[0.5082],

[0.4703],

[0.2660]], grad\_fn=<AddBackward0>)

1 

plt.scatter(X\_train\_3, y\_train\_3)

2

plt.plot(X\_train\_3, hypothesis.detach().numpy())

[<matplotlib.lines.Line2D at 0x7f946b4f87d0>] 

Training with Full code

https://colab.research.google.com/drive/1aXMeQkZreJdJGDUlKNl9HyFOSU5RyFSP#printMode=true 4/13

21. 10. 8. 오전 11:35 112-Regression.ipynb - Colaboratory

1 

# Data setup

2

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2) 3

4

# Model initialize

5

W = torch.zeros(1, requires\_grad=True)

6

b = torch.zeros(1, requires\_grad=True)

7

8

# Set optimizer

9

optimizer = optim.SGD([W, b], lr=0.01)

10

11

nb\_epochs = 1000

12

for epoch in range(nb\_epochs + 1):

13

# Calculate H(X)

14

hypothesis = X\_train \* W + b

15

# hypothesis = X\_train\_3 \* W + b

16

17

# Calculate cost

18

cost = torch.mean((hypothesis - y\_train) \*\* 2)

19

# cost = torch.mean((hypothesis - y\_train\_3) \*\* 2)

20

21

# Parameter gradient descent

22

optimizer.zero\_grad()

23

cost.backward()

24

optimizer.step()

25

26

if epoch % 20 == 0:

27

print('Epoch {:4d}/{} W: {:.3f}, b: {:.3f} Cost: {:.6f}'.format( 28

epoch, nb\_epochs, W.item(), b.item(), cost.item()

29

))

Epoch 0/1000 W: 0.160, b: 0.140 Cost: 53.038891

Epoch 20/1000 W: 2.212, b: 1.972 Cost: 9.947540

Epoch 40/1000 W: 3.026, b: 2.758 Cost: 2.665808

Epoch 60/1000 W: 3.333, b: 3.112 Cost: 1.417548

Epoch 80/1000 W: 3.434, b: 3.287 Cost: 1.188125

Epoch 100/1000 W: 3.453, b: 3.386 Cost: 1.132732

Epoch 120/1000 W: 3.439, b: 3.452 Cost: 1.108790

Epoch 140/1000 W: 3.413, b: 3.504 Cost: 1.092014

Epoch 160/1000 W: 3.383, b: 3.547 Cost: 1.078076

Epoch 180/1000 W: 3.354, b: 3.586 Cost: 1.066038

Epoch 200/1000 W: 3.325, b: 3.622 Cost: 1.055560

Epoch 220/1000 W: 3.298, b: 3.655 Cost: 1.046426

Epoch 240/1000 W: 3.272, b: 3.685 Cost: 1.038461

Epoch 260/1000 W: 3.249, b: 3.714 Cost: 1.031516

Epoch 280/1000 W: 3.226, b: 3.740 Cost: 1.025460

Epoch 300/1000 W: 3.205, b: 3.765 Cost: 1.020178

Epoch 320/1000 W: 3.186, b: 3.788 Cost: 1.015573

Epoch 340/1000 W: 3.168, b: 3.810 Cost: 1.011556

Epoch 360/1000 W: 3.151, b: 3.830 Cost: 1.008054

Epoch 380/1000 W: 3.135, b: 3.849 Cost: 1.005000

Epoch 400/1000 W: 3.120, b: 3.867 Cost: 1.002337

Epoch 420/1000 W: 3.106, b: 3.883 Cost: 1.000015

Epoch 440/1000 W: 3.093, b: 3.899 Cost: 0.997989

Epoch 460/1000 W: 3.081, b: 3.913 Cost: 0.996223

Epoch 480/1000 W: 3.070, b: 3.926 Cost: 0.994683

https://colab.research.google.com/drive/1aXMeQkZreJdJGDUlKNl9HyFOSU5RyFSP#printMode=true 5/13

21. 10. 8. 오전 11:35 112-Regression.ipynb - Colaboratory Epoch 500/1000 W: 3.060, b: 3.939 Cost: 0.993340

Epoch 520/1000 W: 3.050, b: 3.951 Cost: 0.992169

Epoch 540/1000 W: 3.041, b: 3.962 Cost: 0.991148

Epoch 560/1000 W: 3.032, b: 3.972 Cost: 0.990258

Epoch 580/1000 W: 3.024, b: 3.981 Cost: 0.989481

Epoch 600/1000 W: 3.017, b: 3.990 Cost: 0.988804

Epoch 620/1000 W: 3.010, b: 3.998 Cost: 0.988213

Epoch 640/1000 W: 3.003, b: 4.006 Cost: 0.987698

Epoch 660/1000 W: 2.997, b: 4.013 Cost: 0.987249

Epoch 680/1000 W: 2.991, b: 4.020 Cost: 0.986858

Epoch 700/1000 W: 2.986, b: 4.027 Cost: 0.986516

Epoch 720/1000 W: 2.981, b: 4.032 Cost: 0.986218

Epoch 740/1000 W: 2.977, b: 4.038 Cost: 0.985959

Epoch 760/1000 W: 2.972, b: 4.043 Cost: 0.985732

Epoch 780/1000 W: 2.968, b: 4.048 Cost: 0.985535

Epoch 800/1000 W: 2.965, b: 4.052 Cost: 0.985362

Epoch 820/1000 W: 2.961, b: 4.057 Cost: 0.985212

Epoch 840/1000 W: 2.958, b: 4.060 Cost: 0.985081

Epoch 860/1000 W: 2.955, b: 4.064 Cost: 0.984967

Epoch 880/1000 W: 2.952, b: 4.067 Cost: 0.984868

Epoch 900/1000 W: 2.949, b: 4.071 Cost: 0.984781

Epoch 920/1000 W: 2.947, b: 4.074 Cost: 0.984705

Epoch 940/1000 W: 2.944, b: 4.076 Cost: 0.984639

Epoch 960/1000 W: 2.942, b: 4.079 Cost: 0.984581

Epoch 980/1000 W: 2.940, b: 4.081 Cost: 0.984531

Epoch 1000/1000 W: 2.938, b: 4.084 Cost: 0.984487

1 hx = (X\_train \* W + b).detach().numpy() 

1 

plt.figure(figsize=[6, 6])

2

plt.scatter(X\_train, y\_train, s=10)

3

plt.scatter(X\_train, hx, s=20, c='r')

<matplotlib.collections.PathCollection at 0x7f946b4d6e10> 

https://colab.research.google.com/drive/1aXMeQkZreJdJGDUlKNl9HyFOSU5RyFSP#printMode=true 6/13

21. 10. 8. 오전 11:35 112-Regression.ipynb - Colaboratory High level implementation with nn.Module

nn.module 을 활용하여 모델 구축

nn.module : 신경망 모듈. 각종 레이어(linear, conv, ...)를 지원하며 output을 return하는 forward(input) 메서드를 포함함

1 

from torch import nn as nn

2

from torch.nn import functional as F

nn.Linear 레이어의 활용

1 

class my\_LinearRegression(nn.Module): 2

def \_\_init\_\_(self):

3

super().\_\_init\_\_()

4

self.linear = nn.Linear(1, 1)

5

6

def forward(self, x):

7

return self.linear(x)

1 model = my\_LinearRegression() 

1 model my\_LinearRegression(

(linear): Linear(in\_features=1, out\_features=1, bias=True) )

1 hypothesis = model(X\_train[:3]) 

1 hypothesis tensor([[ 0.4109],

[-0.4739],

[-0.2649]], grad\_fn=<AddmmBackward>)

1 

hypothesis = model(X\_train)

2

cost = F.mse\_loss(hypothesis, y\_train)

1 cost tensor(55.6750, grad\_fn=<MseLossBackward>)

1 optimizer = optim.SGD(model.parameters(), lr=0.01) 

1 2 3

optimizer.zero\_grad() cost.backward() 

i i ()

https://colab.research.google.com/drive/1aXMeQkZreJdJGDUlKNl9HyFOSU5RyFSP#printMode=true 7/13

21. 10. 8. 오전 11:35 112-Regression.ipynb - Colaboratory

3 optimizer.step()

Training with Full code

1 

# Data setup

2

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2) 3

4

# Model initialize

5

model = my\_LinearRegression()

6

7

# Set optimizer

8

optimizer = optim.SGD(model.parameters(), lr=0.01)

9

10

nb\_epochs = 1000

11

for epoch in range(nb\_epochs + 1):

12

# Calculate H(X)

13

hypothesis = model(X\_train)

14

15

# Calculate cost

16

cost = F.mse\_loss(hypothesis, y\_train)

17

18

# Parameter gradient descent

19

optimizer.zero\_grad()

20

cost.backward()

21

optimizer.step()

22

23

if epoch % 20 == 0:

24

params = list(model.parameters())

25

W = params[0].item()

26

b = params[1].item()

27

print('Epoch {:4d}/{} W: {:.3f}, b: {:.3f} Cost: {:.6f}'.format( 28

epoch, nb\_epochs, W, b, cost.item()

29

))

Epoch 0/1000 W: 0.946, b: 0.348 Cost: 37.520115

Epoch 20/1000 W: 2.637, b: 1.902 Cost: 7.496470

Epoch 40/1000 W: 3.293, b: 2.581 Cost: 2.423627

Epoch 60/1000 W: 3.528, b: 2.899 Cost: 1.537814

Epoch 80/1000 W: 3.592, b: 3.067 Cost: 1.358345

Epoch 100/1000 W: 3.589, b: 3.171 Cost: 1.301265

Epoch 120/1000 W: 3.561, b: 3.246 Cost: 1.268185

Epoch 140/1000 W: 3.523, b: 3.308 Cost: 1.242164

Epoch 160/1000 W: 3.484, b: 3.361 Cost: 1.219972

Epoch 180/1000 W: 3.446, b: 3.410 Cost: 1.200723

Epoch 200/1000 W: 3.410, b: 3.455 Cost: 1.183975

Epoch 220/1000 W: 3.375, b: 3.496 Cost: 1.169392

Epoch 240/1000 W: 3.343, b: 3.535 Cost: 1.156693

Epoch 260/1000 W: 3.313, b: 3.571 Cost: 1.145634

Epoch 280/1000 W: 3.285, b: 3.605 Cost: 1.136003

Epoch 300/1000 W: 3.259, b: 3.636 Cost: 1.127617

Epoch 320/1000 W: 3.234, b: 3.665 Cost: 1.120313

Epoch 340/1000 W: 3.212, b: 3.692 Cost: 1.113953

Epoch 360/1000 W: 3.190, b: 3.718 Cost: 1.108415

https://colab.research.google.com/drive/1aXMeQkZreJdJGDUlKNl9HyFOSU5RyFSP#printMode=true 8/13

21. 10. 8. 오전 11:35 112-Regression.ipynb - Colaboratory

Epoch 380/1000 W: 3.170, b: 3.742 Cost: 1.103592 Epoch 400/1000 W: 3.152, b: 3.764 Cost: 1.099392 Epoch 420/1000 W: 3.135, b: 3.784 Cost: 1.095734 Epoch 440/1000 W: 3.118, b: 3.804 Cost: 1.092549 Epoch 460/1000 W: 3.103, b: 3.822 Cost: 1.089775 Epoch 480/1000 W: 3.089, b: 3.839 Cost: 1.087359 Epoch 500/1000 W: 3.076, b: 3.854 Cost: 1.085256 Epoch 520/1000 W: 3.064, b: 3.869 Cost: 1.083424 Epoch 540/1000 W: 3.053, b: 3.883 Cost: 1.081829 Epoch 560/1000 W: 3.042, b: 3.895 Cost: 1.080440 Epoch 580/1000 W: 3.032, b: 3.907 Cost: 1.079230 Epoch 600/1000 W: 3.023, b: 3.918 Cost: 1.078177 Epoch 620/1000 W: 3.014, b: 3.929 Cost: 1.077259 Epoch 640/1000 W: 3.006, b: 3.938 Cost: 1.076460 Epoch 660/1000 W: 2.998, b: 3.947 Cost: 1.075765 Epoch 680/1000 W: 2.991, b: 3.956 Cost: 1.075159 Epoch 700/1000 W: 2.985, b: 3.964 Cost: 1.074631 Epoch 720/1000 W: 2.979, b: 3.971 Cost: 1.074172 Epoch 740/1000 W: 2.973, b: 3.978 Cost: 1.073772 Epoch 760/1000 W: 2.968, b: 3.984 Cost: 1.073423 Epoch 780/1000 W: 2.963, b: 3.990 Cost: 1.073120 Epoch 800/1000 W: 2.958, b: 3.996 Cost: 1.072855 Epoch 820/1000 W: 2.954, b: 4.001 Cost: 1.072625 Epoch 840/1000 W: 2.950, b: 4.006 Cost: 1.072425 Epoch 860/1000 W: 2.946, b: 4.010 Cost: 1.072251 Epoch 880/1000 W: 2.942, b: 4.015 Cost: 1.072098 Epoch 900/1000 W: 2.939, b: 4.019 Cost: 1.071966 Epoch 920/1000 W: 2.936, b: 4.022 Cost: 1.071851 Epoch 940/1000 W: 2.933, b: 4.026 Cost: 1.071751 Epoch 960/1000 W: 2.930, b: 4.029 Cost: 1.071663 Epoch 980/1000 W: 2.928, b: 4.032 Cost: 1.071587 Epoch 1000/1000 W: 2.926, b: 4.035 Cost: 1.071521

결과 확인

1 hx = (model(X\_train)).detach().numpy() 

1 

plt.figure(figsize=[6, 6])

2

plt.scatter(X\_train, y\_train, s=10) 3

plt.scatter(X\_train, hx, s=20, c='r')

https://colab.research.google.com/drive/1aXMeQkZreJdJGDUlKNl9HyFOSU5RyFSP#printMode=true 9/13

21. 10. 8. 오전 11:35 112-Regression.ipynb - Colaboratory

<matplotlib.collections.PathCollection at 0x7f946b4ad890> Multivariate Linear Regression

1 

m = 100

2

x1 = torch.rand(m, 1)

3

x2 = 2 \* torch.rand(m, 1)

4

x3 = 3 \* torch.rand(m, 1)

5

X = torch.cat((x1, x2, x3), axis=1)

6

y = 4 + 3 \* x1 + 2 \* x2 + 5 \* x3 + torch.randn(m, 1)

1 X.shape, y.shape (torch.Size([100, 3]), torch.Size([100, 1]))

1 

class MultivariateLinearRegressionModel(nn.Module):

2

def \_\_init\_\_(self):

3

super().\_\_init\_\_()

4

self.linear = nn.Linear(3, 1)

5

6

def forward(self, x):

7

return self.linear(x)

1

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2) 2

3

model = MultivariateLinearRegressionModel()

4

5

# Set optimizer

6

optimizer = optim.SGD(model.parameters(), lr=0.01)

7

8

nb\_epochs = 2000

9

for epoch in range(nb\_epochs + 1):

10

# Calculate H(X)

11

hypothesis = model(X\_train)

12

13

# Calculate cost

14

cost = F.mse\_loss(hypothesis, y\_train)

15

16

# Parameter gradient descent

17

optimizer.zero\_grad()

18

cost.backward()

19

optimizer.step()

20

21

if epoch % 20 == 0:

22

li ( l ())

https://colab.research.google.com/drive/1aXMeQkZreJdJGDUlKNl9HyFOSU5RyFSP#printMode=true 10/13

21. 10. 8. 오전 11:35 112-Regression.ipynb - Colaboratory

22

params = list(model.parameters())

23

24

print('Epoch {:4d}/{} {} Cost: {:.6f}'.format(

25

epoch, nb\_epochs, list(model.parameters()), cost.item()

26

))

Epoch 0/2000 [Parameter containing:

tensor([[-0.0884, 0.0534, 0.0660]], requires\_grad=True), Parameter containing: tensor([0.6565], requires\_grad=True)] Cost: 289.553070

Epoch 20/2000 [Parameter containing:

tensor([[1.1863, 2.4604, 4.7005]], requires\_grad=True), Parameter containing: tensor([3.1468], requires\_grad=True)] Cost: 5.294409

Epoch 40/2000 [Parameter containing:

tensor([[1.3738, 2.6122, 5.2532]], requires\_grad=True), Parameter containing: tensor([3.4503], requires\_grad=True)] Cost: 1.672374

Epoch 60/2000 [Parameter containing:

tensor([[1.4377, 2.5288, 5.3444]], requires\_grad=True), Parameter containing: tensor([3.5116], requires\_grad=True)] Cost: 1.539234

Epoch 80/2000 [Parameter containing:

tensor([[1.4851, 2.4322, 5.3775]], requires\_grad=True), Parameter containing: tensor([3.5463], requires\_grad=True)] Cost: 1.468914

Epoch 100/2000 [Parameter containing:

tensor([[1.5282, 2.3454, 5.3987]], requires\_grad=True), Parameter containing: tensor([3.5780], requires\_grad=True)] Cost: 1.414102

Epoch 120/2000 [Parameter containing:

tensor([[1.5685, 2.2693, 5.4139]], requires\_grad=True), Parameter containing: tensor([3.6092], requires\_grad=True)] Cost: 1.370545

Epoch 140/2000 [Parameter containing:

tensor([[1.6062, 2.2025, 5.4247]], requires\_grad=True), Parameter containing: tensor([3.6399], requires\_grad=True)] Cost: 1.335511

Epoch 160/2000 [Parameter containing:

tensor([[1.6416, 2.1438, 5.4320]], requires\_grad=True), Parameter containing: tensor([3.6700], requires\_grad=True)] Cost: 1.307000

Epoch 180/2000 [Parameter containing:

tensor([[1.6749, 2.0921, 5.4363]], requires\_grad=True), Parameter containing: tensor([3.6993], requires\_grad=True)] Cost: 1.283536

Epoch 200/2000 [Parameter containing:

tensor([[1.7061, 2.0464, 5.4383]], requires\_grad=True), Parameter containing: tensor([3.7277], requires\_grad=True)] Cost: 1.264021

Epoch 220/2000 [Parameter containing:

tensor([[1.7354, 2.0059, 5.4383]], requires\_grad=True), Parameter containing: tensor([3.7552], requires\_grad=True)] Cost: 1.247635

Epoch 240/2000 [Parameter containing:

tensor([[1.7629, 1.9699, 5.4369]], requires\_grad=True), Parameter containing: tensor([3.7816], requires\_grad=True)] Cost: 1.233756

Epoch 260/2000 [Parameter containing:

tensor([[1.7888, 1.9377, 5.4343]], requires\_grad=True), Parameter containing: tensor([3.8071], requires\_grad=True)] Cost: 1.221911

Epoch 280/2000 [Parameter containing:

tensor([[1.8130, 1.9090, 5.4308]], requires\_grad=True), Parameter containing: tensor([3.8315], requires\_grad=True)] Cost: 1.211734

Epoch 300/2000 [Parameter containing:

tensor([[1.8358, 1.8832, 5.4266]], requires\_grad=True), Parameter containing: tensor([3.8548], requires\_grad=True)] Cost: 1.202940

Epoch 320/2000 [Parameter containing:

tensor([[1.8573, 1.8600, 5.4218]], requires\_grad=True), Parameter containing: tensor([3.8772], requires\_grad=True)] Cost: 1.195304

Epoch 340/2000 [Parameter containing:

tensor([[1.8774, 1.8391, 5.4167]], requires\_grad=True), Parameter containing: tensor([3.8985], requires\_grad=True)] Cost: 1.188647

https://colab.research.google.com/drive/1aXMeQkZreJdJGDUlKNl9HyFOSU5RyFSP#printMode=true 11/13

21. 10. 8. 오전 11:35 112-Regression.ipynb - Colaboratory

Epoch 360/2000 [Parameter containing:

tensor([[1.8963, 1.8202, 5.4114]], requires\_grad=True), Parameter containing: tensor([3.9188], requires\_grad=True)] Cost: 1.182823

Epoch 380/2000 [Parameter containing:

tensor([[1 9141 1 8030 5 4058]] requires grad=True) Parameter containing: 결과 확인

1 from sklearn.decomposition import PCA 

1 

pca = PCA(n\_components=1)

2

X\_pca = pca.fit\_transform(X\_train)

1 hx = model(X\_train).detach().numpy() 

1 

plt.scatter(X\_pca, y\_train, s=20)

2

plt.scatter(X\_pca, hx, s=20)

<matplotlib.collections.PathCollection at 0x7f946984bbd0> 1

https://colab.research.google.com/drive/1aXMeQkZreJdJGDUlKNl9HyFOSU5RyFSP#printMode=true 12/13

21. 10. 8. 오전 11:35 112-Regression.ipynb - Colaboratory

https://colab.research.google.com/drive/1aXMeQkZreJdJGDUlKNl9HyFOSU5RyFSP#printMode=true 13/13