표지판, 시계이(가) 표시된 사진

자동 생성된 설명

2020년 2학기

GD과제

|  |
| --- |
| 과목: 데이터사이언스개론 |
| 담당교수: 신효섭 |
| 제출일: 2020/10/26 |
| 학과: 기술경영학과 |
| 학번: 201512082 |
| 성명: 백 승 호 |

|  |
| --- |
| [ 목 차 ]  1. Simple linear regression 시각화  2. multiple linear regression 구현  3. 코드 |

1. Simple linear regressioin 시각화

Matplotlib.animation을 이용해서 그래프를 시각화 할 수 있다

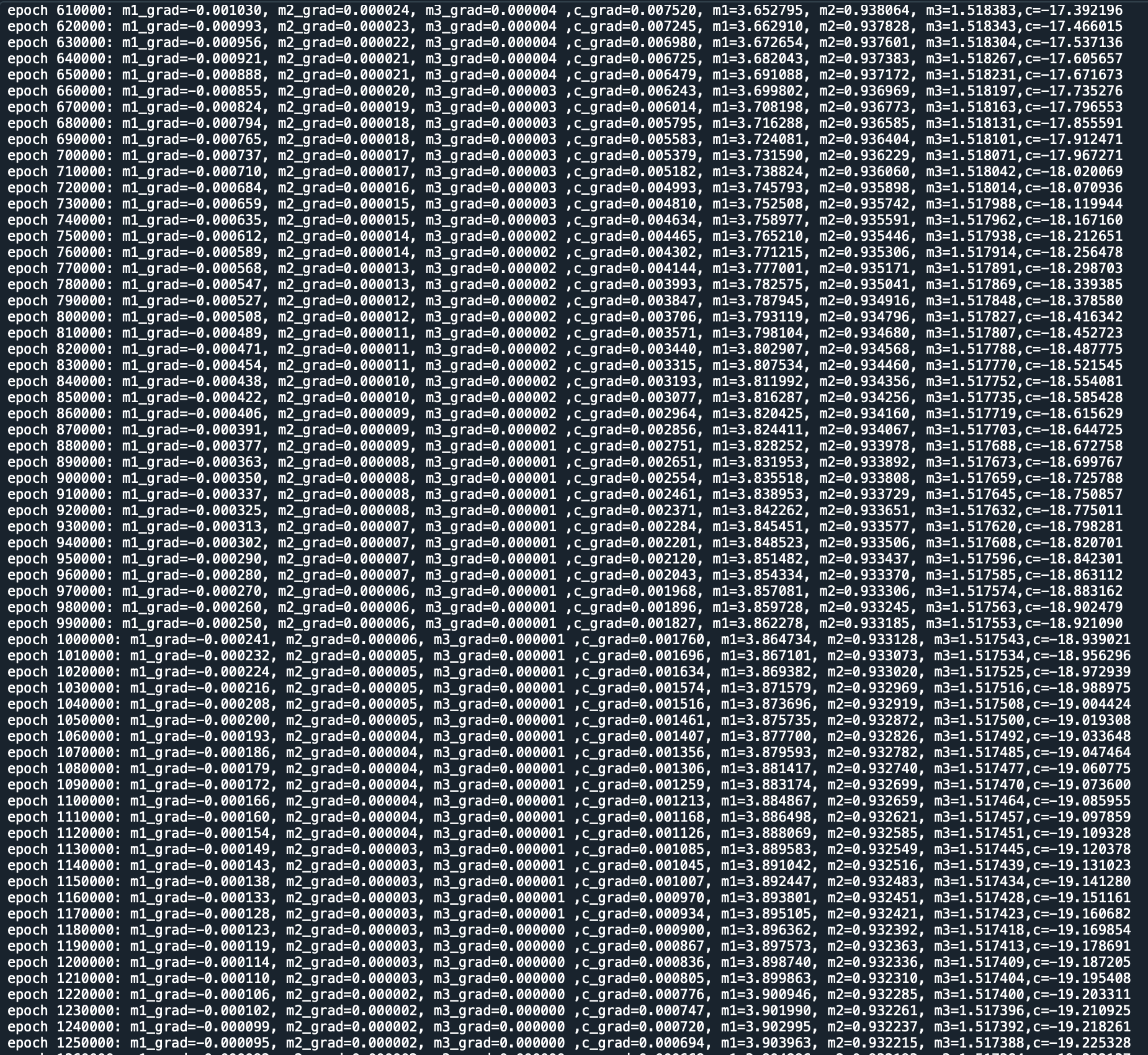
Epoch를 반복할때마다 갱신되는 m과 c를 list에 append하고, list의 계수와 절편값에 따라 직선을 그리는 식을 animate()함수에 담아주면 된다

2. Multiple linear regression 구현

3가지 요소가 있으므로 각각을 x1 x2 x3 로 잡고 그에 해당하는 계수값을 m1 m2 m3로 잡아서 prediction을 m1\*x1 + m2\*x2 + m3\*x3 + c로 계산하면 된다. gradient계산은 simple linear regression때와 동일한 수식을 따른다.

건물, 대형, 앉아있는, 서있는이(가) 표시된 사진

자동 생성된 설명

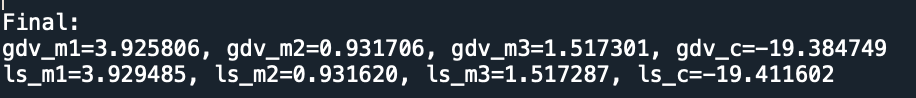


건물, 창문, 앉아있는, 문이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | M1(Attendance) | M2(Homework) | M3(Midterm) | C |
| Least Square | ﻿3.929485 | 0.931620, | 1.517287 | -19.411602 |
| Gradient Descent | ﻿3.925806 | 0.931706 | 1.517301 | -19.384749 |

약 177000 epoch 수행 결과.



3. 코드

(1) 시각화

|  |
| --- |
| ﻿import numpy as np  import matplotlib.pyplot as plt  import matplotlib.animation as animation  import pymysql  def load\_dbscore\_data\_one():  conn = pymysql.connect(host='localhost', user='root', password='password', db='university')  curs = conn.cursor(pymysql.cursors.DictCursor)    sql = "select \* from db\_score"  curs.execute(sql)    data = curs.fetchall()  curs.close()  conn.close()    #X = [ (t['attendance'], t['homework'], t['midterm'] ) for t in data ]  X = [ ( t['midterm'] ) for t in data ]  X = np.array(X)    y = [ (t['score']) for t in data]  y = np.array(y)  return X, y  def gradient\_descent\_vectorized(X, y):  epochs = 100000  min\_grad = 0.0001  learning\_rate = 0.001    m = 0.0  c = 0.0    n = len(y)    c\_grad = 0.0  m\_grad = 0.0  c\_list = []  m\_list = []  for epoch in range(epochs):  y\_pred = m \* X + c  m\_grad = (2\*(y\_pred - y)\*X).sum()/n  c\_grad = (2 \* (y\_pred - y)).sum()/n  if epoch % 100 == 0:  c\_list.append(c)  m\_list.append(m)    m = m - learning\_rate \* m\_grad  c = c - learning\_rate \* c\_grad  if ( epoch % 1000 == 0):  print("epoch %d: m\_grad=%f, c\_grad=%f, m=%f, c=%f" %(epoch, m\_grad, c\_grad, m, c) )    if ( abs(m\_grad) < min\_grad and abs(c\_grad) < min\_grad ):  break  return np.array(m\_list), np.array(c\_list)    def init():  line.set\_data([],[])  return line,  def animate(i):  y\_pred = m[i] \* X + c[i]  line.set\_data(X, y\_pred)  return line,  X, y = load\_dbscore\_data\_one()  m, c = gradient\_descent\_vectorized(X, y)  fig, ax = plt.subplots()    ax.set\_xlim(min(X)-2, max(X)+4)  ax.set\_ylim(min(y)-2, max(y)+4)  ax.set\_xlabel('midterm')  ax.set\_ylabel('score')  line, = ax.plot([], [], color='red')  ax.scatter(X, y)  ani = animation.FuncAnimation(fig, animate,init\_func=init, frames=len(m), interval=10, blit=True)  plt.show() |

(2) Multiple linear regression

|  |
| --- |
| ﻿﻿  import numpy as np  import pymysql  import time  def load\_dbscore\_data\_two():  conn = pymysql.connect(host='localhost', user='root', password='password', db='university')  curs = conn.cursor(pymysql.cursors.DictCursor)    sql = "select \* from db\_score"  curs.execute(sql)    data = curs.fetchall()  curs.close()  conn.close()    X = [ (t['attendance'], t['homework'], t['midterm'] ) for t in data ]  X = np.array(X)  x1 = [ (t['attendance']) for t in data ]  x1 = np.array(x1)  x2 = [ (t['homework']) for t in data ]  x2 = np.array(x2)  x3 = [ (t['midterm']) for t in data ]  x3 = np.array(x3)    y = [ (t['score']) for t in data]  y = np.array(y)  return X,x1,x2,x3, y  def least\_square\_data(X, y):  import statsmodels.api as sm  X\_const = sm.add\_constant(X) # 모델만들고, least square 적용해야함  model = sm.OLS(y, X\_const) # ordinary lesat square(out,in)  ls = model.fit()  print(ls.summary())  # 각 coefficient나옴  ls\_c = ls.params[0] # c  ls\_m1 = ls.params[1] # m1 attendandce  ls\_m2 = ls.params[2] # m2 homework  ls\_m3 = ls.params[3] # m3 midterm  return ls\_c, ls\_m1, ls\_m2, ls\_m3    def m\_gradient\_descent\_vectorized(x1,x2,x3, y):  epochs = 1800000  min\_grad = 0.0001  learning\_rate = 0.001    m1 = 0.0  m2 = 0.0  m3 = 0.0  c = 0.0  n = len(y)  c\_grad = 0.0  m1\_grad = 0.0  m2\_grad = 0.0  m3\_grad = 0.0  for epoch in range(epochs):  y\_pred = m1\*x1 + m2\*x2 + m3\*x3 + c  m1\_grad = (2\*(y\_pred - y)\*x1).sum()/n  m2\_grad = (2\*(y\_pred - y)\*x2).sum()/n  m3\_grad = (2\*(y\_pred - y)\*x3).sum()/n  c\_grad = (2 \* (y\_pred - y)).sum()/n    m1 = m1 - learning\_rate \* m1\_grad  m2 = m2 - learning\_rate \* m2\_grad  m3 = m3 - learning\_rate \* m3\_grad  c = c - learning\_rate \* c\_grad  if ( epoch % 10000 == 0):  print("epoch %d: m1\_grad=%f, m2\_grad=%f, m3\_grad=%f ,c\_grad=%f, m1=%f, m2=%f, m3=%f,c=%f" %(epoch, m1\_grad, m2\_grad, m3\_grad, c\_grad, m1, m2, m3, c) )    if ( abs(m1\_grad) < min\_grad and abs(m2\_grad) < min\_grad and abs(m3\_grad) < min\_grad and abs(c\_grad) < min\_grad ):  break  return m1,m2,m3,c  X, x1,x2,x3, y = load\_dbscore\_data\_two()  ls\_c, ls\_m1, ls\_m2, ls\_m3 = least\_square\_data(X, y)  start\_time = time.time()  m1, m2, m3, c = m\_gradient\_descent\_vectorized(x1,x2,x3, y)  end\_time = time.time()  print("%f seconds" %(end\_time - start\_time))  print("\n\nFinal:")  print("gdv\_m1=%f, gdv\_m2=%f, gdv\_m3=%f, gdv\_c=%f" %(m1,m2,m3,c))  print("ls\_m1=%f, ls\_m2=%f, ls\_m3=%f, ls\_c=%f" %(ls\_m1, ls\_m2, ls\_m3, ls\_c)) |