머신러닝실습 레포트

-2주차-

제출일 : 2022.04.16.

학 번 : 2017142012

이 름 : 김효건

# **0. 목차**

1. **목차**
2. **실습 #1**
   1. 데이터 표현 및 위치 표시
   2. 기본 설정
3. **실습 #2**
   1. 해석해 ( 행렬 사용 )

**3. 실습 #3**

3.0 선형모델의 평균제곱오차

**4. 실습 #4**

* 1. 경사하강법

**5. 실습 #5**

5.1 가우스 함수를 이용한 선형 기저함수 회귀모델의 최적 매개변수

**6. 실습 #6**

6.1 훈련데이터와 선형 기저함수 회귀 모델의 그래프

**7. 실습 #7**

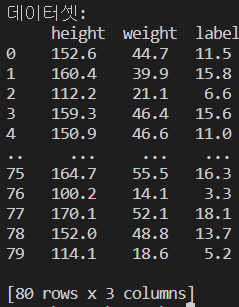
* 1. K값에 대한 평균제곱 오차와 그래프

# **1. 실습 #1**

**1.1 데이터 표현**

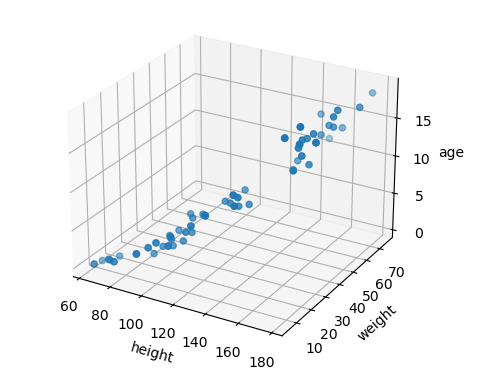
**1개의 특성 age**와 그에 대응하는 **결과 값 tall**이 존재

|  |
| --- |
| current\_path = os.path.dirname(os.path.abspath(\_\_file\_\_))  # 데이터 셋 읽어들이비다.  data = pd.read\_csv(current\_path + "/multiple\_linear\_regression\_data.csv")  print("데이터셋:\n",data)  # 데이터 특성에 따른 값으로 변수에 저장  height = np.array(data["height"])  weight = np.array(data["weight"])  label = np.array(data["label"])  # 실습 1 출력  ax = plt.figure().add\_subplot(111, projection='3d')  ax.scatter(height, weight, label)  ax.set\_xlabel("height"); ax.set\_ylabel("weight"); ax.set\_zlabel("age", rotation=0)  plt.show() |



(그림1. 데이터 셋 구조)

1열은 age, 2열은 tall로 csv파일을 가져옵니다.



(그림2. 데이터 셋 분포도)

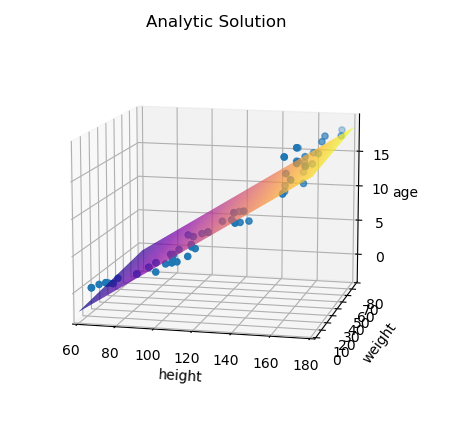
**[그 외 기본 설정]**

|  |
| --- |
| # csv파일 데이터 크기 80 x 3  data\_size = data.shape  # height과 weight을 하나의 데이터 x로 묶고 결과 값인 label을 따로 묶었습니다.  x= np.c\_[height, weight]  y = label |

# **2. 실습 #2**

**2.1 해석해 (행렬 사용)**

|  |
| --- |
| # 기존 데이터에 bias를 추가했습니다.  # bias, height, weight  x\_bias = np.c\_[np.ones(data\_size[0]), x]  #print("bias, height, weight\n",x\_bias)  # 유사역행렬 (pseudoinverse inverse matrix) = Moore-Penrose inverse mastrx  #  square matrix가 아니거나 어떤 특성이 중복되어 행렬 X^T X의 역행렬이 없다면 정규방정식 동작안한다.  x\_bias\_t = x\_bias.T  #행렬 계산을 통한 theta값 : bias, height, weight  theta = np.linalg.pinv(x\_bias\_t.dot(x\_bias)).dot(x\_bias\_t).dot(y)  print("최적 세타 값 : ", theta)  # x축(height)과 y축(weight)의 좌표 구간을 설정하기 위한 최대 최소 설정  # 최소 구간 : 1의 자리에서 내림  # 최대 구간 : 1의 자리에서 올림  # 모든 구간 포함할 수 있게 설정  height\_min = (min(height))  height\_max = (max(height))  height\_lim = [(height\_min//10) \* 10 ,(height\_max // 10 + 1) \* 10]  weight\_min = (min(weight))  weight\_max = (max(weight))  weight\_lim = [(weight\_min//10)\*10, (weight\_max // 10 +1) \* 10]  print("설정한 좌표 범위:\n", height\_lim, weight\_lim)  # 각 자리의 시작부터 끝지점까지 동일한 간격으로 1000개의 데이터 생성  height\_space = np.linspace(height\_lim[0], height\_lim[1], 1000)  weight\_space = np.linspace(weight\_lim[0], weight\_lim[1], 1000)  # 데이터 1차원에서 2차원으로 확장  height\_mesh, weight\_mesh = np.meshgrid(height\_space, weight\_space)  print("height 2d :\n {}\nweight 2d : {}".format(height\_mesh,weight\_mesh))  # 선형모델 생성, 가중치를 사용하여 평면을 만든다.  y\_space = theta[2]\*weight\_mesh + theta[1]\*height\_mesh + theta[0]  print(y\_space)  def origin\_data\_space():      ax = plt.figure().add\_subplot(projection='3d')      ax.scatter(height,weight,label)      ax.axis(height\_lim + weight\_lim)      ax.set\_xlabel('height'); ax.set\_ylabel('weight'); ax.set\_zlabel('age', rotation=0)      ax.plot\_surface(height\_mesh, weight\_mesh, y\_space, cmap="plasma")      plt.title("Analytic Solution")      plt.grid(True)      plt.show()    origin\_data\_space() |



(그림3. 파란색 점 = 원본 데이터, 평면 = 해를 통한 y값)

# **3. 실습 #3**

**3.1 선형모델의 평균제곱오차**

**[수식을 이용한 선형 모델 생성]**

|  |
| --- |
| # y의 예측값을 구하기 위해 비교 대상인 y와 대응하는 동일한 데이터 x사용함  # 따라서 height, weight을 사용함  y\_pred\_eq = theta[2]\*weight + theta[1]\*height + theta[0] # 수식 활용  y\_pred\_matrix = x\_bias @ theta # 행렬 곱 사용  # mse값을 만드는 함수 y의 예측값과 y 원본 값, 데이터의 갯수를 받는다.  def calc\_mse(y\_pred, y\_origin, size):      return sum((y\_pred - y\_origin) \*\* 2) / size  # mse, data\_size는 x의 모양 80 X 3이다. 따라서 data\_size[0]은 80  mse\_eq = calc\_mse(y\_pred\_eq, y, data\_size[0])  mse\_matrix = calc\_mse(y\_pred\_matrix, y, data\_size[0])  print("mse eq값 : ", mse\_eq)  print("mse matrix값 : ", mse\_matrix) |



(그림4. 선형 모델 생성 및 MSE값 구함)

# **4. 실습 #4**

**[초기값]**

|  |
| --- |
| # 경사하강법  # 기본  learning\_rate = 0.000055  n\_iter = 200000  np.random.seed(85) # 이후 모든 출력 값을 동일하게 하기 위해 rand값 고정  gd\_theta = np.random.randn(3,) # 평균 0 표준편차 1인 정규분포따르는 -1~1사이값 가져옴  gd\_theta \*= 3  # -1~1의 값 증폭을 위한 값 (시그마값) |

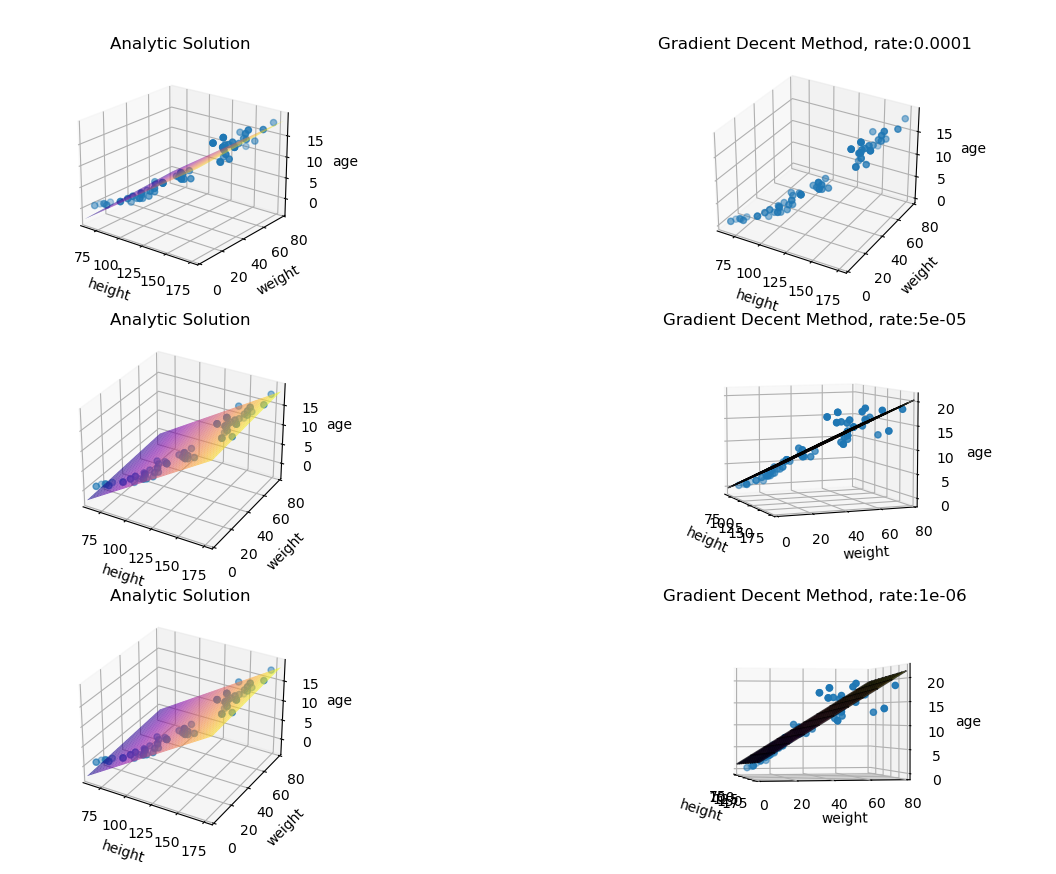
**[경사하강법]**

|  |
| --- |
| # theata의 매개변수 learning rate에 따른 값의 변화를 보기 위한 함수  def y\_space\_graph(ax, y\_pred=None, rate=learning\_rate, epoch=n\_iter, show="origin"):      if ax == None:          return      title = ""      if show == "origin":          title = "Analytic Solution"      elif show=="rate":          title = "Gradient Decent Method, rate:{}".format(rate)      elif show=="epoch":          title = "Gradient Decent Method, epoch:{}".format(epoch)       # Analytic solution인지 Gradient descent인지 구분      # 가장 먼저 기본 데이터를 출력      ax.scatter(height,weight,label)       if show == "origin":          # Analytic          ax.plot\_surface(height\_mesh, weight\_mesh, y\_space,  alpha=0.7, cmap="plasma")      else:          # Gradient Descent          ax.plot\_surface(height\_mesh, weight\_mesh, y\_pred, alpha=0.7,cmap="plasma", rstride=100,cstride=100, edgecolor="black")      ax.axis(height\_lim + weight\_lim)      ax.set\_title(title)      ax.set\_xlabel('height'); ax.set\_ylabel('weight'); ax.set\_zlabel('age', rotation=0)  # 오차 지정  tolerance = 0.000001  # 오차 계산 함수  def isStop(now\_th, before\_th):      # 현재 theta와 befoe theta값에 절대값 사용하여      # 각각의 차이 중 큰 값이 오차 허용 값보다 작을 때 참을 반환      th\_abs = np.abs(now\_th)      before\_th\_abs = np.abs(before\_th)      result = (abs(th\_abs - before\_th\_abs).max() < tolerance)      return result  # iter과 가중치 theta값 고정 후 learning rate에 따른 값 변화  def theta\_graph(rate=learning\_rate, th=gd\_theta, epoch=n\_iter, ax=None, show="origin"):      # 최종적으로 몇번 반복했는지 나타내는 변수      stop\_iter = 0      # 반복 횟수를 고정한 채 진행      for i in range(n\_iter):          before\_th = th          # 행렬을 통한 기울기 계산          gradient = 2/data\_size[0] \* x\_bias.T.dot(x\_bias.dot(th) - y)          th = th - rate \* gradient # 기울기 갱신  # MSE계산을 위한 y 예측값 계산 후 mse 값 계산          y\_pred = x\_bias.dot(th)          mse = calc\_mse(y\_pred, y, data\_size[0])    # 가중치 값 비교를 하여 오차 범위보다 적은지 확인하는 함수 호출          if isStop(th, before\_th):  # 오차 범위보다 적다면 멈추고 해당 횟수 저장              stop\_iter = i+1              break  # 도중에 끊기지 않았다면 마지막 횟수 저장      if stop\_iter == 0:          stop\_iter =  n\_iter    # 그래프로 평면을 출력하기 위해 평면 값들을 활용하여 y값 만듦      y\_pred\_graph = th[2]\*weight\_mesh + th[1]\*height\_mesh + th[0]      y\_space\_graph(ax, y\_pred\_graph, rate)    # 최종 가중치(세타)값을 활용하여 y예측값 만든다.      y\_pred = x\_bias.dot(th)      mse = calc\_mse(y\_pred, y, data\_size[0])      print("[Epoch]:{}, [최종반복 횟수]:{}, [Learning rate]:{} ===> [W초기]:{}, [gradient]:{}, MSE값:{:.6f}%".format(n\_iter, stop\_iter, rate, gd\_theta, th, mse))      return th  # 그래프 출력 함수 기울기 고정일 때  def print\_rate(func):      fig = plt.figure(figsize=(15, 30))      ax = fig.add\_subplot(321, projection='3d')      th = y\_space\_graph(ax, "None", 0)        ax = fig.add\_subplot(322, projection='3d')      th2 = func(rate=0.0001, ax=ax)        ax = fig.add\_subplot(323, projection='3d')      th = y\_space\_graph(ax, "None", 0)        ax = fig.add\_subplot(324, projection='3d')      th4 = func(rate=0.00005, ax=ax)      ax = fig.add\_subplot(325, projection='3d')      th = y\_space\_graph(ax, "None", 0)        ax = fig.add\_subplot(326, projection='3d')      th6 = func(rate=0.000001, ax=ax)      plt.show()        fig = plt.figure(figsize=(20, 20))      ax = fig.add\_subplot(321, projection='3d')      th = y\_space\_graph(ax, "None", 0)        ax = fig.add\_subplot(322, projection='3d')      th2 = func(rate=0.000057, ax=ax)        ax = fig.add\_subplot(323, projection='3d')      th = y\_space\_graph(ax, "None", 0)        ax = fig.add\_subplot(324, projection='3d')      th4 = func(rate=0.000055, ax=ax)      ax = fig.add\_subplot(325, projection='3d')      th = y\_space\_graph(ax, "None", 0)        ax = fig.add\_subplot(326, projection='3d')      th6 = func(rate=0.000053, ax=ax)      plt.show()    print\_rate(theta\_graph)  # learning rate 고정 후 반복 횟수 조정  def print\_iter(func):      fig = plt.figure(figsize=(15, 30))      ax = fig.add\_subplot(321, projection='3d')      th = y\_space\_graph(ax)        ax = fig.add\_subplot(321, projection='3d')      th1 = func(epoch=20000, ax=ax, show="epoch")      ax = fig.add\_subplot(322, projection='3d')      th2 = func(epoch=50000, ax=ax, show="epoch")        ax = fig.add\_subplot(323, projection='3d')      th3 = func(epoch=100000, ax=ax, show="epoch")      ax = fig.add\_subplot(324, projection='3d')      th4 = func(epoch=200000, ax=ax, show="epoch")      ax = fig.add\_subplot(325, projection='3d')      th5 = func(epoch=500000, ax=ax, show="epoch")      ax = fig.add\_subplot(326, projection='3d')      th6 = func(epoch=800000, ax=ax, show="epoch")      plt.show()  print\_iter(theta\_graph) |



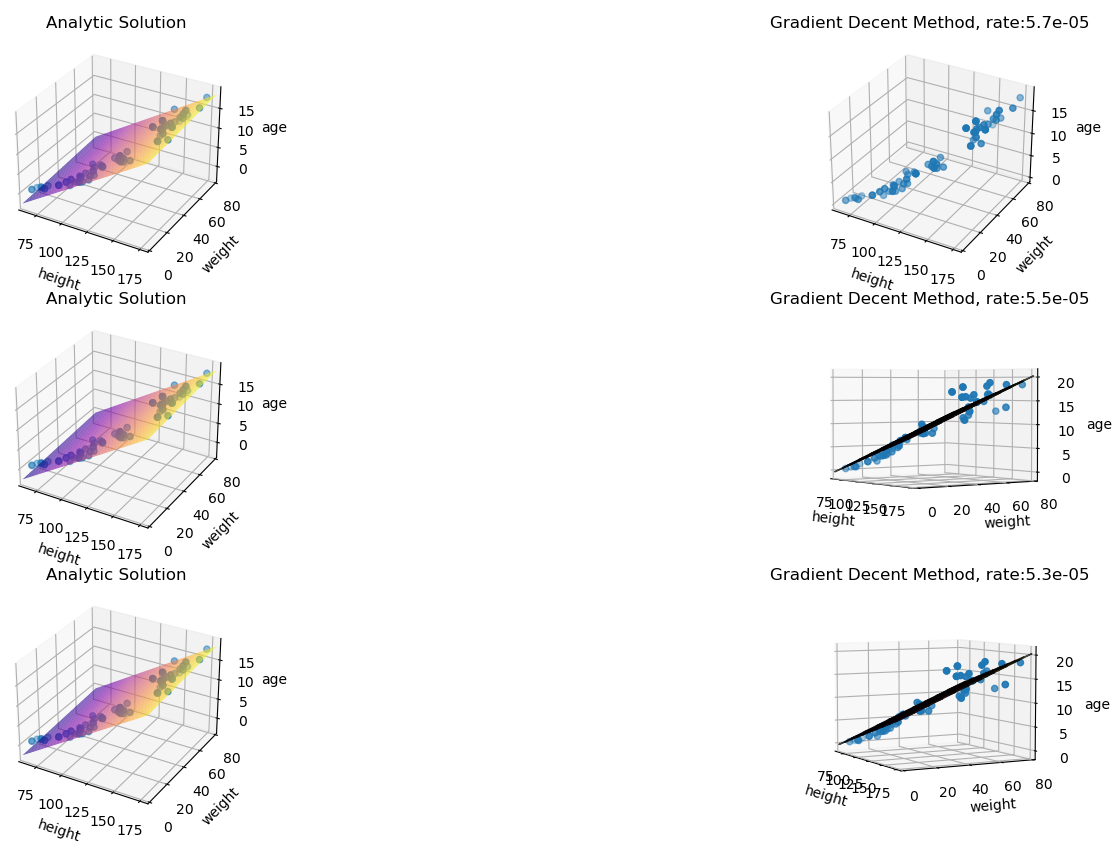


(그림 5. Epock고정, Learning rate값의 변경을 통한 경사하강법 결과)

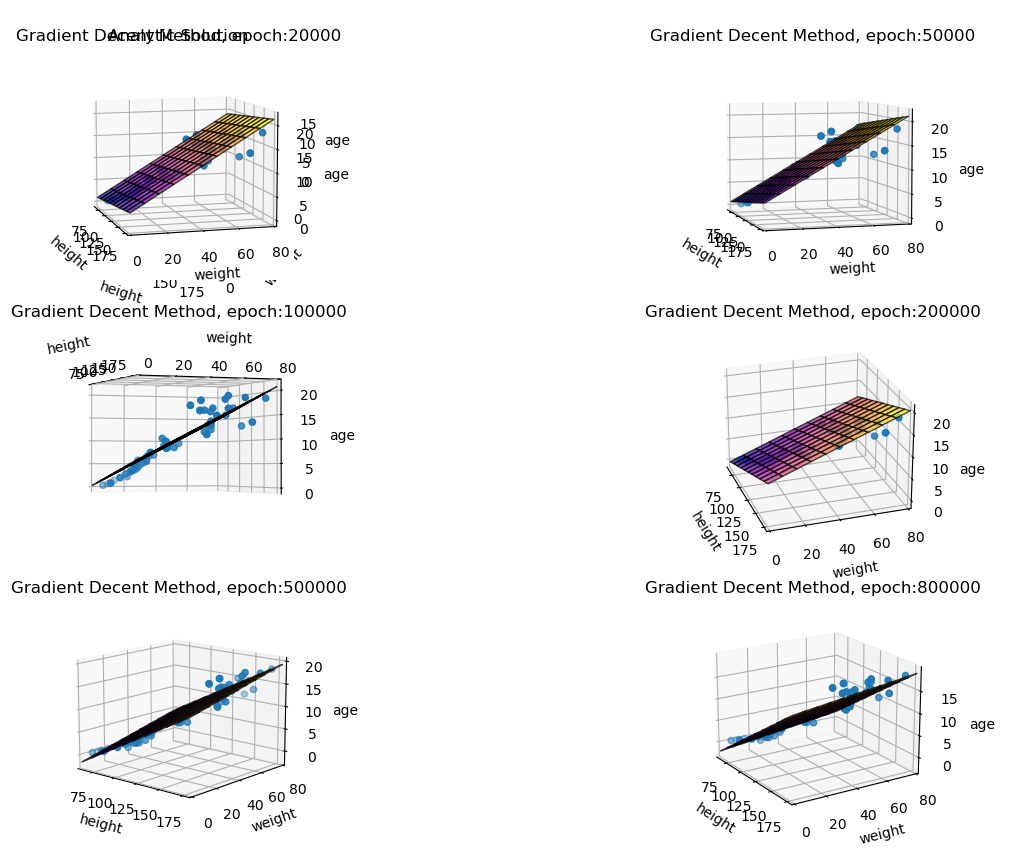


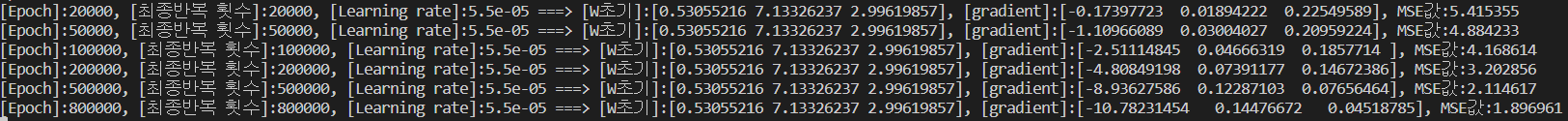
0.0001일때는 발산했으며, 0.00005 에서는 mse값이 3.3, 0.00001에서는 5.8값이 나왔습니다.

따라서 0.00005부근에서 최적의 learning rate값이 있다 생각했습니다.



다시 3번의 반복을 통해 0.000055에서 3.20으로 최적의 learning rate를 발견했습니다.





(그림6. Learning rate고정, Epoch에 따른 경사하강법 결과)

80만번을 반복하여 했을 때 최종적으로 MSE가 1.89가 나오면서 최적의 매개변수를

Learning rate 0.000055와 Epoch=800,000을 구하였고 이때의 가중치(W)는

[bias, height, weight] : -10.7823 0.144766 0.04518785가 놔왔습니다.

# **5. 실습 #5**

**[가우스 함수를 이용한 선형 기저함수 회귀모델의 최적 매개변수]**

|  |
| --- |
| data1 = pd.read\_csv("\linear\_regression\_data01.csv", names=["age", "tall"])  print("지난 데이터:\n",data1)  # 나이를 x1, 결과인 키를 y1로 분리하여 저장  x1 = data1['age']  y1 = data1['tall']  # x1의 최대값과 최소값 저장  x1\_max = max(x1)  x1\_min = min(x1)  x1\_size = x1.shape  # mu값 계산  # K값에 대해 배열을 반환, 각각의 값 미리 계산  def calc\_mu(K):      return [x1\_min + (x1\_max - x1\_min) / (K - 1) \* k for k in range(K)]  # sigma값 계산  # 고정된 값으로 하나의 상수  def calc\_sigma(K):      return (x1\_max - x1\_min) / (K-1)  # 가우스 계산  # x값 자체가 하나의 특성 데이터 전체로 들어옴  # mu는 각 배열의 원소들을 하나씩 넣어준다.  # x는 array, mu자체가 현재 k(새로운 특성)에 해당하는 값중 하나(기존 리스트에서 한개씩 함수로 전달받음)  def calc\_gauss(x, mu, sigma):      return np.exp(-1/2 \* ((x - mu)/sigma)\*\*2)  # 각 특성마다 해당하는 mu값과 함께 가우스를 계산하여 배열에 추가  # result[i]는 각 특성 데이터 값이므로 np.array로 변환하여 T시켜  # 데이터마다 여러개의 특성을 뽑아올 수 있도록 변경  # result[0] = [특성1의 데이터1, 특성1의 데이터2] =>  # [특성1, 특성2, 특성3 ...]  def make\_pi(x, K, mu, sigma):      result = []      for i in range(K):          # 계산한 mu값은 리스트          # mu0, mu1 mu2 ... 값을 하나하나 넣어준다.          result.append(calc\_gauss(x, mu[i], sigma))      return np.array(result).T    # 가우시안 값을 구하기 위해 처음으로 호출하는 함수  def gaussian\_func(K, x\_data):      # u\_k = 평균, sigma = 표준편차      mu = calc\_mu(K) # mu값 구하는 함수 호출      sigma = calc\_sigma(K) # sigma값 구하는 함수 호출        # 기본 데이터값을 변경하지 않기 위해 깊은 복사로 데이터 가져옴      x\_temp = x\_data.copy()      print("[K = {}] : mu={}, sigma={}".format(K,mu, sigma))      # pi 행렬을 구하는 함수      pi = make\_pi(x\_temp,K, mu,sigma)        # y\_hat = bias + pi\_0(x) + pi\_1(x) + pi\_2(x)      pi\_b = np.c\_[np.ones(x1\_size[0]), pi]      return pi\_b  k\_list = [3,5,8,10]  pi\_list = []  # k 갯수 많큼 for문 돌린다.  for i in range(len(k\_list)):      # 구한 pi 행렬을 리스트에 넣어 저장해둔다.      # 각 값은 k\_list의 K 값과 대응 된다.      pi\_list.append(gaussian\_func(k\_list[i], x1))  # 가중치(theta)값 구하는 함수  def calc\_pi\_theta(pi):      # pinv를 사용하여 해석해를 구했습니다.      pi\_t = pi.T      return np.linalg.pinv(pi\_t.dot(pi)).dot(pi\_t).dot(y1)  # theta값  theta\_list = []  for i in range(len(k\_list)):      # 각각의 pi값에 대한 theta(가중치를) 리스트에 저장      theta\_list.append(calc\_pi\_theta(pi\_list[i]))      print("K = {}:[GD,(bias, 0,1,...)]:{}\n".format(k\_list[i], theta\_list[i])) |

텍스트이(가) 표시된 사진

자동 생성된 설명

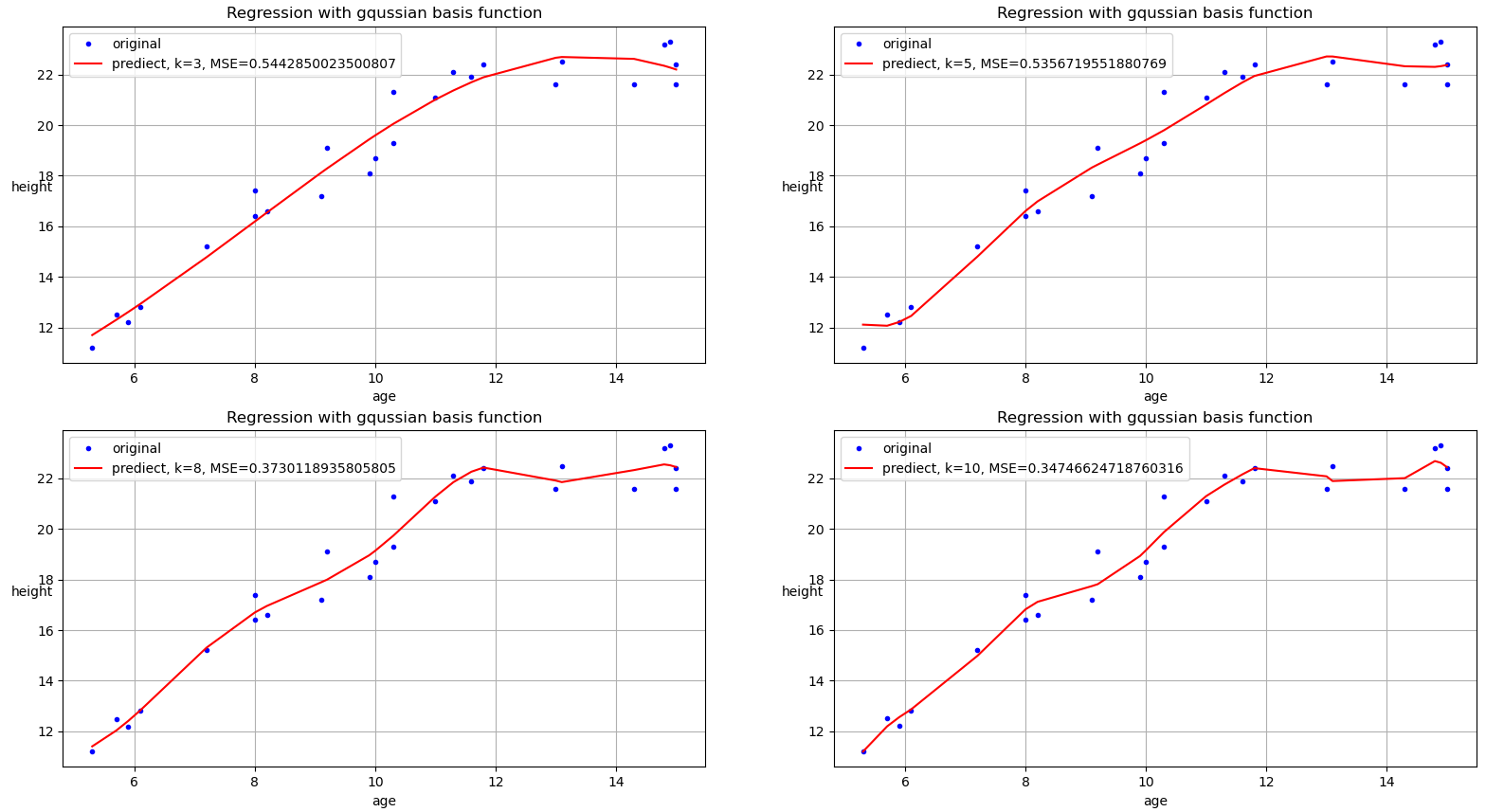
(그림7. K값에 따른 Gradient Discent값)

각각의 Gradient값은 처음이 bias, 2번째부터 0~K-1까지입니다.

# **6. 실습 #6**

**[훈련데이터와 선형 기저함수 회귀 모델의 그래프]**

|  |
| --- |
| # y 예측 값과 mse 값을 구합니다.  y1\_list= []  mse\_pi\_list = []  for i in range(len(k\_list)):      # 위에서 구한 pi값과 theta값을 사용하여 y 예측값을 구하고 리스트에 저장      y1\_list.append(pi\_list[i].dot(theta\_list[i]))      # 저장한 y예측값과 실제 y값을 통해 mse를 구하기 위해      # 위에서 선언한 mse구하는 함수 호출하여 mse값을 저장합니다.      mse\_pi\_list.append(calc\_mse(y1\_list[i], y1, x1\_size[0]))  # 실습 6 출력 함수  def show\_graph6():      # 그래프에서 y예측값 출력은 직선으로 되어있어서 데이터를 정렬할 필요가  # 있습니다.      # 따라서 x와 y예측값을 같이 묶어주기 위해 show\_data 선언      show\_data = np.array(x1)        # 출력할 예측값의 갯수는 k값의 갯수입니다.      # y1\_list는 4 X 25의 형태이므로 한개씩 넣어줬습니다.      for i in range(len(k\_list)):          # 매번 옆에 하나의 y예측값들을 넣습니다.          show\_data = np.c\_[show\_data, y1\_list[i]]      print(show\_data)      # 해당 데이터 집합은 x데이터의 순서대로 정렬해야 하므로      # show\_data[0]을 기준으로 정렬합니다.      show\_data = sorted(show\_data, key = lambda x : x[0])      # 정렬 후 하나의 데이터는 [x값, y1예측값, y2예측값, y3예측값,  # y4예측값]      # 형태이므로 transpose하여 show\_data[0] = [x값들],      #                       show\_data[1~4] = [y예측값들] 이 나오게 한다.      show\_data = np.array(show\_data).T      print(show\_data)      # 그래프 출력      plt.figure(figsize=(20, 20))      for i in range(4):          plt.subplot(2,2,1 + i);          # 원본 데이터          plt.plot(x1, y1, 'b.', label="original")          # y예측 값          plt.plot(show\_data[0], show\_data[i+1], 'r-', label="prediect, k={}, MSE={}".format(k\_list[i],mse\_pi\_list[i]))          plt.xlabel("age")          plt.ylabel("height", rotation=0)          plt.title("Regression with gqussian basis function")          plt.legend(loc="upper left")          plt.grid(True)      plt.show()  show\_graph6() |



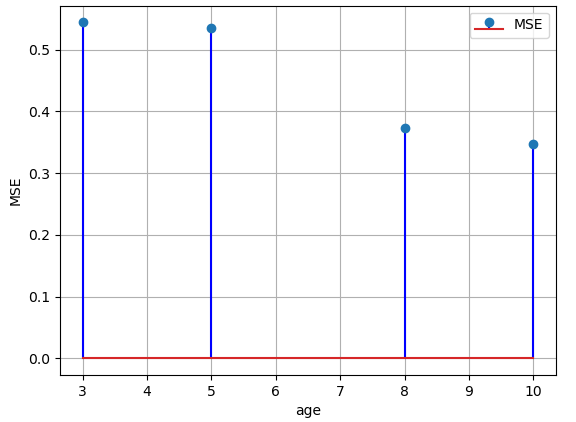
(그림 8. K값에 따른 y 예측값과 원본 데이터 그래프)

k값이 높을수록 더욱 기본 데이터에 근접한 함수를 구할 수 있습니다.

# **7. 실습 #7**

**[평균제곱오차와 그래프]**

|  |  |
| --- | --- |
| # 이미 위에서 한번 구현했습니다.  mse\_pi\_list = []  for i in range(len(k\_list)):      # 저장한 y예측값과 실제 y값을 통해 mse를 구하기 위해      # 위에서 선언한 mse구하는 함수 호출하여 mse값을 저장합니다.      mse\_pi\_list.append(calc\_mse(y1\_list[i], y1, x1\_size[0]))      print("K가 {}일때 MSE : {}".format(i,mse\_pi\_list[i])) | |
| plt.figure()  plt.stem(k\_list,mse\_pi\_list, 'b.', label="MSE")  plt.xlabel("age")  plt.ylabel("MSE")  plt.grid()  plt.legend(loc = "upper right")  plt.show() |



(그림 9. K값에 따른 MSE값)