머신러닝실습 레포트

-3주차-

제출일 : 2022.05.15.

학 번 : 2017142012

이 름 : 김효건

# **0. 목차**

1. **목차**
2. **실습 #1 (8주차)**
   1. 로지스틱 함수 출력
   2. 데이터 출력
   3. Epoch에 따른 경사하강법 결과 값

1.4.1로지스틱 회귀의 정확도와 Decision Boundary 그래프 출력

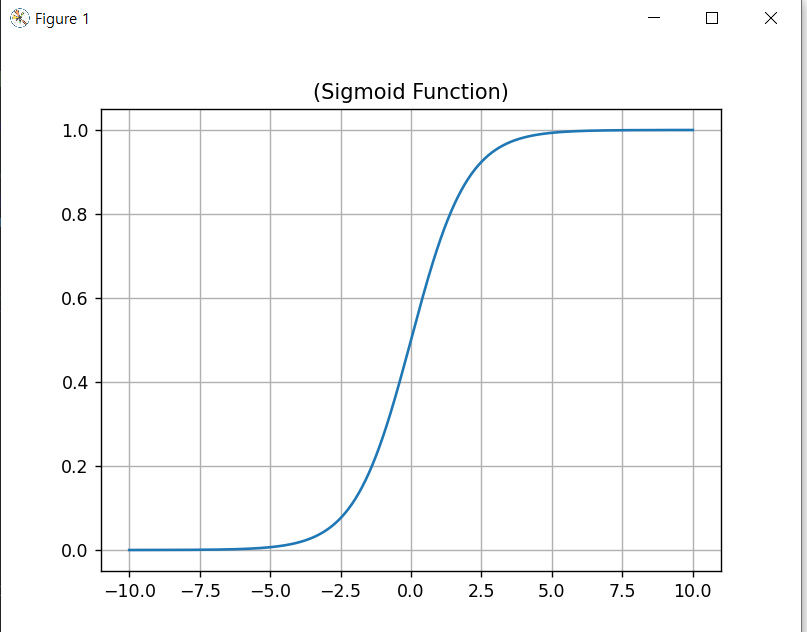
1.4.2 예측하고 싶은 값을 통한 예측 결과 값

1. **실습 #2 (9주차)**
   1. 데이터 요약도 및 분포도
   2. Cross Entropy Loss 를 비용함수로 사용한 경사하강법 결과
      1. 로지스틱 회귀 정확도 및 Decision Boundary 그래프 출력
      2. 예측하고 싶은 값을 통한 예측 결과 값

# **1. 실습 #1-1 (8주차)**

**1.1 로지스틱 함수 출력**

|  |
| --- |
| # 시그모이드 함수 적용, 0~1사이의 값으로 변환  def sigmoid(linear\_equation):      # Args   : linear\_equation (np.array): 데이터 값과 가중치의 선형조합      # Returns:  np.array      return 1 / (1 + np.exp(-linear\_equation))  #  sigmoid function을 그리기 위해 x좌표 설정  x\_lin = np.linspace(-10, 10, 1000)  x\_lin\_sigmoid = sigmoid(x\_lin)  plt.title("(Sigmoid Function)")  plt.plot(x\_lin, x\_lin\_sigmoid)  plt.grid(True)  plt.show() |



(그림1. Sigmoid function)

# **1. 실습 #1-2 (8주차)**

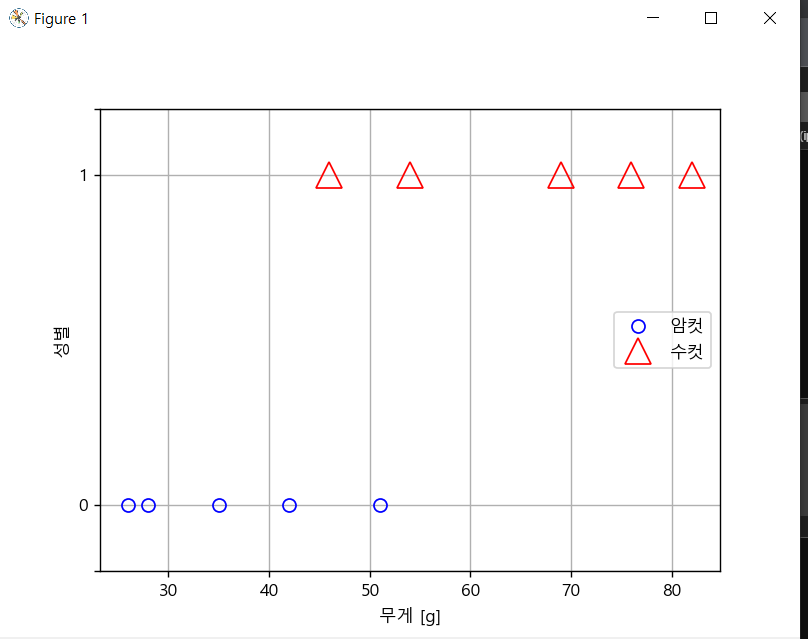
**1.2 데이터 출력**

|  |
| --- |
| # 데이터 셋 읽어들입니다.  dataFrame = pd.read\_csv("binary\_data\_insect.csv")  print("데이터셋:\n",dataFrame)  # Dataframe을 편의상 numpy로 변경  data = dataFrame.to\_numpy()  # 그래프 출력을 위한 여성 남성 분리  female = np.array([x for x in data if x[1] == 0]).T  male = np.array([x for x in data if x[1] == 1]).T  plt.plot(female[0], female[1], 'b.', mfc='none', markersize=15, label="암컷")  plt.plot(male[0], male[1], 'r^', mfc='none', markersize=15, label="수컷")  plt.legend(loc="center right")  plt.yticks([-0.2,0,1, 1.2],[None,0,1,None])  plt.xlabel("무게 [g]")  plt.ylabel("성별")  plt.grid()  plt.show() |

테이블이(가) 표시된 사진

자동 생성된 설명

(그림2. 데이터 셋)

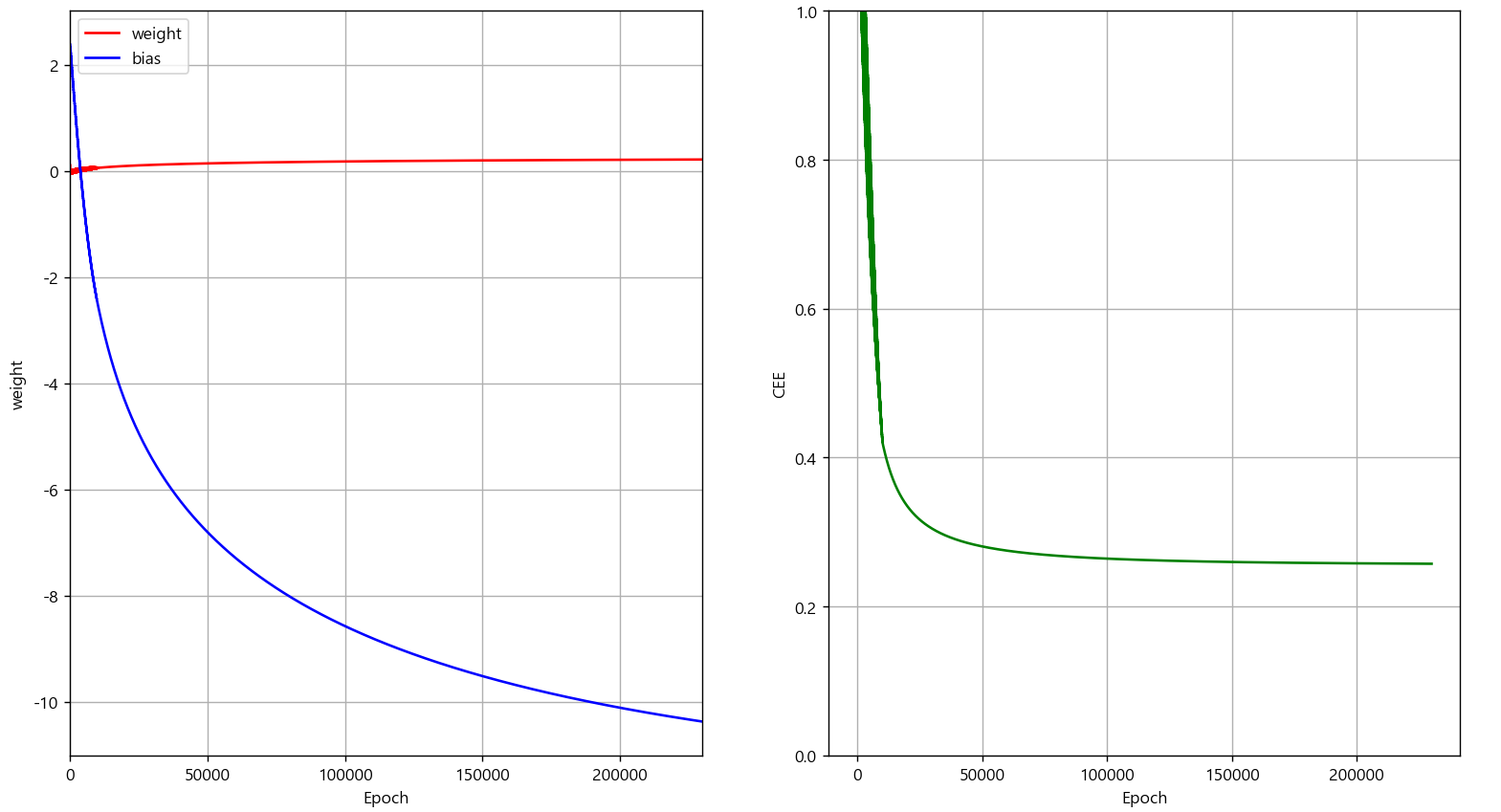


(그림 3. 데이터 셋 분포도)

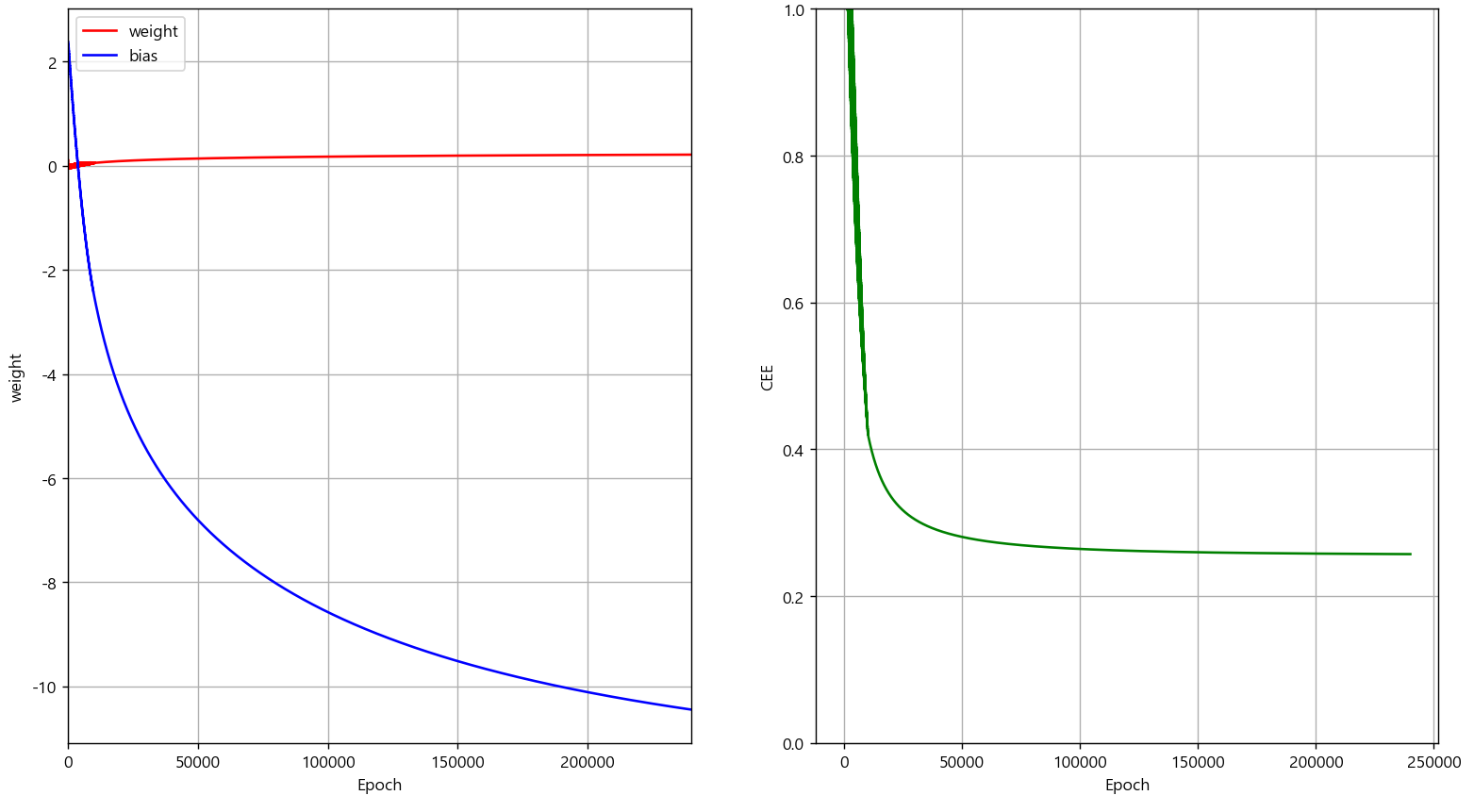
# **1. 실습 #1-3 (8주차)**

**1.3 Epoch에 따른 경사하강법 결과 값**

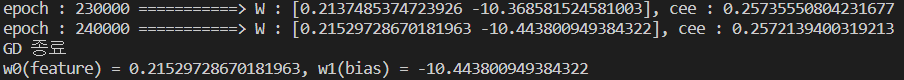
|  |
| --- |
| gender\_N = len(data)        # 데이터 갯수  gender\_M = len(data[0]) - 1 # 특성 갯수 = 레이블 (1) + 특성(1) - 레이블(1)  gender\_M\_b = gender\_M + 1   # bias 추가한 값 , 총 2개  gender\_THRESHOLD = 0.5      # Gedner 경계 값  log\_error = 1e-8            # Log 계산시 Log(0)으로 인한 계산 오류 문제를 위해 사용  # 각 경계값을 기준으로 높은지 아닌지를 결정하여 Bool값이 나온다.  # 이를 int로 변경하여 0과 1로 변경  def probability\_func(pb):      # Args: pb (np.array): 로지스틱 함수를 거쳐 0~1사이로 변경된 값      # Return : list      return list(map(int, pb > gender\_THRESHOLD))  # 손실함수 Cross Entropy Loss 계산  def cross\_entropy\_loss(pb, y):      """      Args:          pb (np.array): 로지스틱 함수를 거쳐 0~1사이로 변경된 값          y (np.array): 데이터 셋의 라벨 값, (실제 값)      Return : float      """      N = len(y)      # Log에 넣는 인자 값이 0이 안되게 만든다.      for i in range(N):          if pb[i] == 1: # np.log(1-pb) 예외처리              pb[i] -= log\_error          elif pb[i] == 0: # np.log(pb) 예외처리, 혹시모를 예외처리              pb[i] += log\_error      cee = sum(y \* np.log(pb) + (1-y)\*np.log(1-pb))      return -cee / N  # 경사하강법  # 초기 값  LEARNING\_RATE = 0.003 # Learning Rate  N\_ITER = 200000       # 반복 횟수  # 특성과 레이블 분리  gender\_x = dataFrame["Weight"]  gender\_y = dataFrame["Gender"]  np.random.seed(85) # 이후 모든 출력 값을 동일하게 하기 위해 rand값 고정  # 평균 0 표준편차 1인 정규분포따르는 -1~1사이값 가져옴  # 2 X 1로 만듬  gd\_theta = np.random.randn(gender\_M\_b,)  # w0 \* x0 + W1 \* 1(bias) 형식으로 만들기 위해 [특성, bias]  gender\_x\_b = np.c\_[gender\_x, np.ones(gender\_N)]  # 그래프 라벨 출력을 위한 변수  PLT\_LABEL = ["weight", "bias", "sepal\_length", "petal\_length", "bias"]  # 그래프 색 : Red, Blue, Black  color\_list = ["#FF0000", "#0000FF", "#000000"]  # theta\_gradient\_descent 함수의 결과 값 인덱싱 매핑  GD\_RESULT\_MAP = {      "weight" : 0,      "cee" : 1,      "epoch" : 2  }  # 오차 허용 범위  tolerance = 0.000000001  # 오차 계산 함수, 허용범위 이내인지 확인 후, [작다:참], [크다:거짓] 반환  def isStop(now\_th, before\_th):      """      Args:          now\_th (np.array)   : 현재 가중치 값          before\_th (np.array): 이전 가중치 값      Returns: Bool      """      th\_abs = np.abs(now\_th)      before\_th\_abs = np.abs(before\_th)      result = (abs(th\_abs - before\_th\_abs).max() < tolerance)      return result  # 경사 하강법 함수  def theta\_gradient\_descent(x\_b, y, M\_b, lr=LEARNING\_RATE, epoch=N\_ITER, th=gd\_theta):      """      Args:          x\_b   (np.array): 특성 데이터 + bias          y     (np.array): 데이터에 따른 라벨값          M\_b   (int): \_description\_          lr    (float, optional): 학습률. Defaults to LEARNING\_RATE          epoch (int, optional): 반복 횟수. Defaults to N\_ITER.          th    (np.array, optional): 초기 가중치 값. Defaults to gd\_theta.      Returns:          List: accuracy(정확도), th(가중치), cee(손실 값), epoch(반복 횟수)      """      N = len(y)      th = th.copy()      stop\_iter = 0      th\_list = []      cee\_list = []      for i in range(epoch):          before\_th = th # 현재 가중치 변경 전에 저장          zn = x\_b.dot(th) # linear equation          probability = sigmoid(zn) # 0~1사이로 변경          cee = cross\_entropy\_loss(probability, y) # Corss Entropy Loss            # w0, w1 ... bias 순으로 경사 하강 시행          for j in range(M\_b):              discent = lr \* 1/ N \*sum((probability-y)\*x\_b.T[j])              th[j] = th[j] - discent            th\_list.append(th.copy())          cee\_list.append(cee)            # 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      if stop\_iter == 0:          stop\_iter =  epoch      epoch\_list  = [i for i in range(stop\_iter)]        # 출력을 위해 [데이터 갯수 X 가중치 갯수(특성의 갯수)] =>      #  [가중치 갯수(특성의 갯수) X 데이터 갯수]      th\_list = np.array(th\_list).T      plt.figure(figsize=(15,15))      plt.subplot(121)      for i in range(len(th\_list)):          plt.plot(epoch\_list, th\_list[i], '-', color=color\_list[i])  plt.plot(epoch\_list, th\_list[i], '-', color=color\_list[i], label=PLT\_LABEL[label\_index+i])      plt.xlabel("Epoch")      plt.ylabel("weight")      plt.xlim(0, epoch)      plt.grid()        plt.subplot(122)      plt.plot(epoch\_list, cee\_list, 'g-')      plt.xlabel("Epoch")      plt.ylabel("CEE")      y\_lim = [0,1] if N==10 else [0,6]      plt.ylim(y\_lim)      plt.grid()      plt.show()      return th, cee\_list[-1], epoch  # epoch에 따른 gd결과 값과 출력  gd\_result1 = theta\_gradient\_descent(gender\_x\_b, gender\_y, gender\_M\_b, epoch=230000)  gd\_result2 = theta\_gradient\_descent(gender\_x\_b, gender\_y, gender\_M\_b, epoch=240000)  print("epoch : {} ===========> W : [{} {}], cee : {}".format(gd\_result1[GD\_RESULT\_MAP["epoch"]], gd\_result1[GD\_RESULT\_MAP["weight"]][0], gd\_result1[GD\_RESULT\_MAP["weight"]][1], gd\_result1[GD\_RESULT\_MAP["cee"]]))  print("epoch : {} ===========> W : [{} {}], cee : {}".format(gd\_result2[GD\_RESULT\_MAP["epoch"]], gd\_result2[GD\_RESULT\_MAP["weight"]][0], gd\_result2[GD\_RESULT\_MAP["weight"]][1], gd\_result2[GD\_RESULT\_MAP["cee"]]))  # 두 결과 값중 높은 결과 값 출력  most\_gd\_result = gd\_result1 \                  if gd\_result1[GD\_RESULT\_MAP["cee"]] < gd\_result2[GD\_RESULT\_MAP["cee"]] else \                   gd\_result2  print("GD 종료\nw0(feature) = {}, w1(bias) ={}".format(most\_gd\_result[GD\_RESULT\_MAP["weight"]][0], most\_gd\_result[GD\_RESULT\_MAP["weight"]][1])) |



(그림4. Epoch = 230,000, 빨간색 = W0, 파란색 = bias, 우측 = Cross Entropy Error)



(그림5. Epoch = 240,000, 빨간색 = W0, 파란색 = bias, 우측 = Cross Entropy Error)

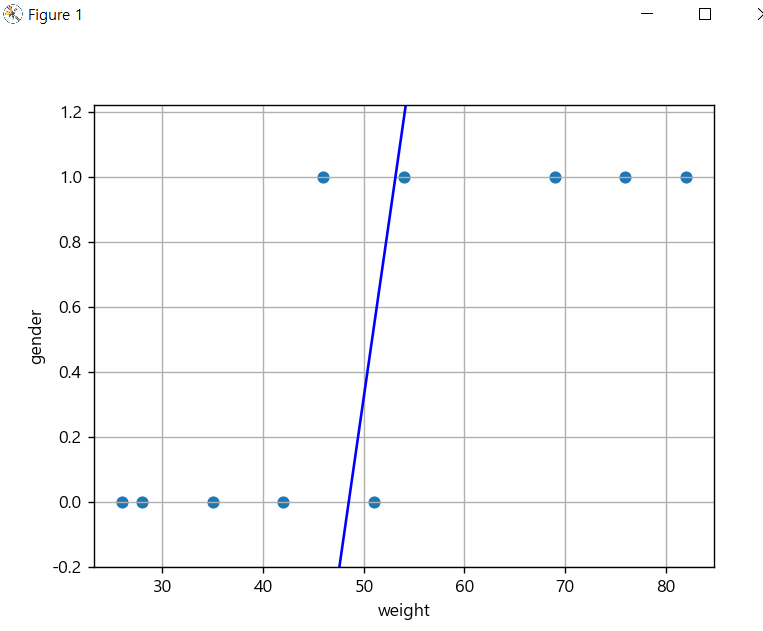


(그림6. Epoch에 따른 결과 값과 최적의 가중치 값)

# **1. 실습 #1-4-1 (8주차)**

**1.4.1 로지스틱 회귀의 정확도와 Decision Boundary 그래프 출력**

|  |
| --- |
| # 정확도 구하는 함수, 예측값과 라벨값이 같은 갯수를 파악하여 전체갯수로 나눠 반환  def find\_accuracy(y\_hat, y):      """      Args:          y\_hat (np.array): Threshold를 통해 구분된 예측 값          y (np.array): 실제 라벨 값      Returns:          flaot      """      count=0      n = len(y)      for i in range(n):          if y[i] == y\_hat[i]:              count += 1      return count / n  # 가장 좋은 가중치 값을 통한 정확도 측정  def best\_th\_accuracy(x\_b, y, th):      """      Args:          x\_b (np.array): 특성 값 + bias          y (np.array): 라벨 값          th (np.array): 최적의 가중치 값      Returns: float      """      zn = x\_b.dot(th) # linear equation      probability = sigmoid(zn) # 0~1사이로 변경      y\_hat = probability\_func(probability) # Corss Entropy Loss      return find\_accuracy(y\_hat, y)  accuracy = best\_th\_accuracy(gender\_x\_b, gender\_y, most\_gd\_result[GD\_RESULT\_MAP["weight"]])  print("훈련결과, 정확도 {}%\nw0(feature) = {}, w1(bias) = {}".format( \                                                               accuracy \* 100,\                                 most\_gd\_result[GD\_RESULT\_MAP["weight"]][0], \                                 most\_gd\_result[GD\_RESULT\_MAP["weight"]][1]))  # 결정 경계 그래프 출력을 위한 X값 설정  x\_lin = np.linspace(min(gender\_x), max(gender\_x),1000)  # weight(x) \* w0 + bias(1) \* w1  dicision\_boundary = most\_gd\_result[GD\_RESULT\_MAP["weight"]][0] \* x\_lin +  most\_gd\_result[GD\_RESULT\_MAP["weight"]][1]  plt.scatter(gender\_x, gender\_y)  plt.plot(x\_lin, dicision\_boundary, 'b-')  plt.ylim(-0.2,1.22)  plt.grid()  plt.show() |



(그림 7. 각 데이터 분포와 Decision Boundary)



(그림 8. 정확도와 최종 최적의 가중치 값)

# **1. 실습 #1-4-2 (8주차)**

**1.4.2 예측하고 싶은 값을 통한 예측 결과 값**

|  |
| --- |
| # 예측된 결과 값에 따라 1이면 남자 0이면 여자를 출력하기 위한 리스트  RESULT\_GENDER = ["female", "male"]  # 예측 함수  def predict(data, weight):      """      Args:          data (List or number(int or float)): 예측을 원하는 값          weight (np.array): 최적화된 가중치      """      # weight = bias + 특성(1) | 2 X 1 matrix      w = np.array(weight)      # data = 데이터 갯수 X 1(feature) martrix      d = np.array([data]).T        # d = 데이터 갯수 X 2(feature + bias) matrix      d\_b = np.c\_[d, np.ones(len(d))]      zn = d\_b @ w # linear equation      probability = sigmoid(zn) # sigmoid      y\_pred = probability\_func(probability) # Threshold = 0.5      for i in range(len(y\_pred)):          print("데이터 {}의 예측 결과 : Probability={},Result={}".format(d[i], probability[i], RESULT\_GENDER[y\_pred[i]]))  predict(20, most\_gd\_result[GD\_RESULT\_MAP["weight"]]) |



(그림 9. 데이터에 대한 예측 값)

# **2. 실습 #2-1 (9주차)**

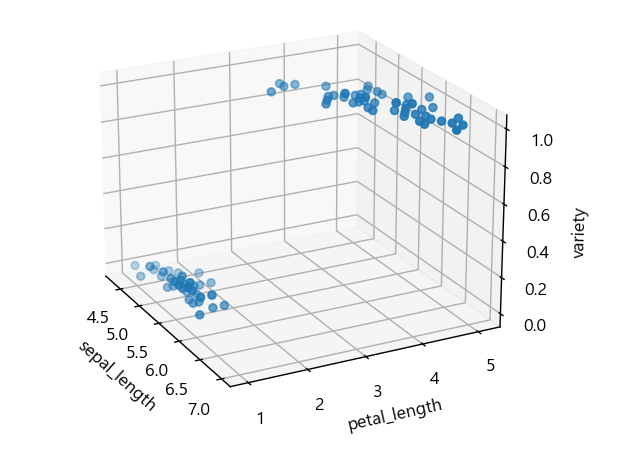
**2.1 데이터 요약도 및 분포도**

|  |
| --- |
| iris\_data = pd.read\_csv("iris.csv")  print(iris\_data)  print(iris\_data.dtypes)  # 특성과 라벨 분리  iris\_df\_feature = iris\_data.iloc[:,:2].to\_numpy() # sepal, petal특성을 가져온다.  iris\_df\_label = iris\_data.iloc[:,-1].to\_numpy()   # variety 라벨 값을 가져온다.  # 각각의 특성을 인덱스로 접근하기 위해 Transpose  # 100 X 3 => 3 X 100  iris\_feature = iris\_df\_feature.T  iris\_label = iris\_df\_label.T  # Object로 되어 있으므로 각 값을 0과 1로 변경하기위해 매핑  LABEL\_NAME = {      "Setosa" : 0,      "Versicolor" : 1  }  # x : Setosa, Versicolor  # LABEL\_NAME["Setosa"] = 0, lABEL\_NAME["Versicolor"] = 1  iris\_label = np.array(list((map(lambda x : LABEL\_NAME[x], iris\_label))))  # 인덱싱 접근의 가독성을 위해 각 특성이름을 매핑  FEATURE\_NAME = {      "sepal\_length" : 0,      "petal\_length" : 1  }  ax = plt.figure().add\_subplot(111, projection='3d')  ax.scatter(iris\_feature[FEATURE\_NAME["sepal\_length"]], iris\_feature[FEATURE\_NAME["petal\_length"]], iris\_label)  ax.set\_xlabel("sepal\_length"); ax.set\_ylabel("petal\_length"); ax.set\_zlabel("variety", rotation=0)  plt.show() |

텍스트이(가) 표시된 사진

자동 생성된 설명

(그림 10. Iris dataset 요약도)

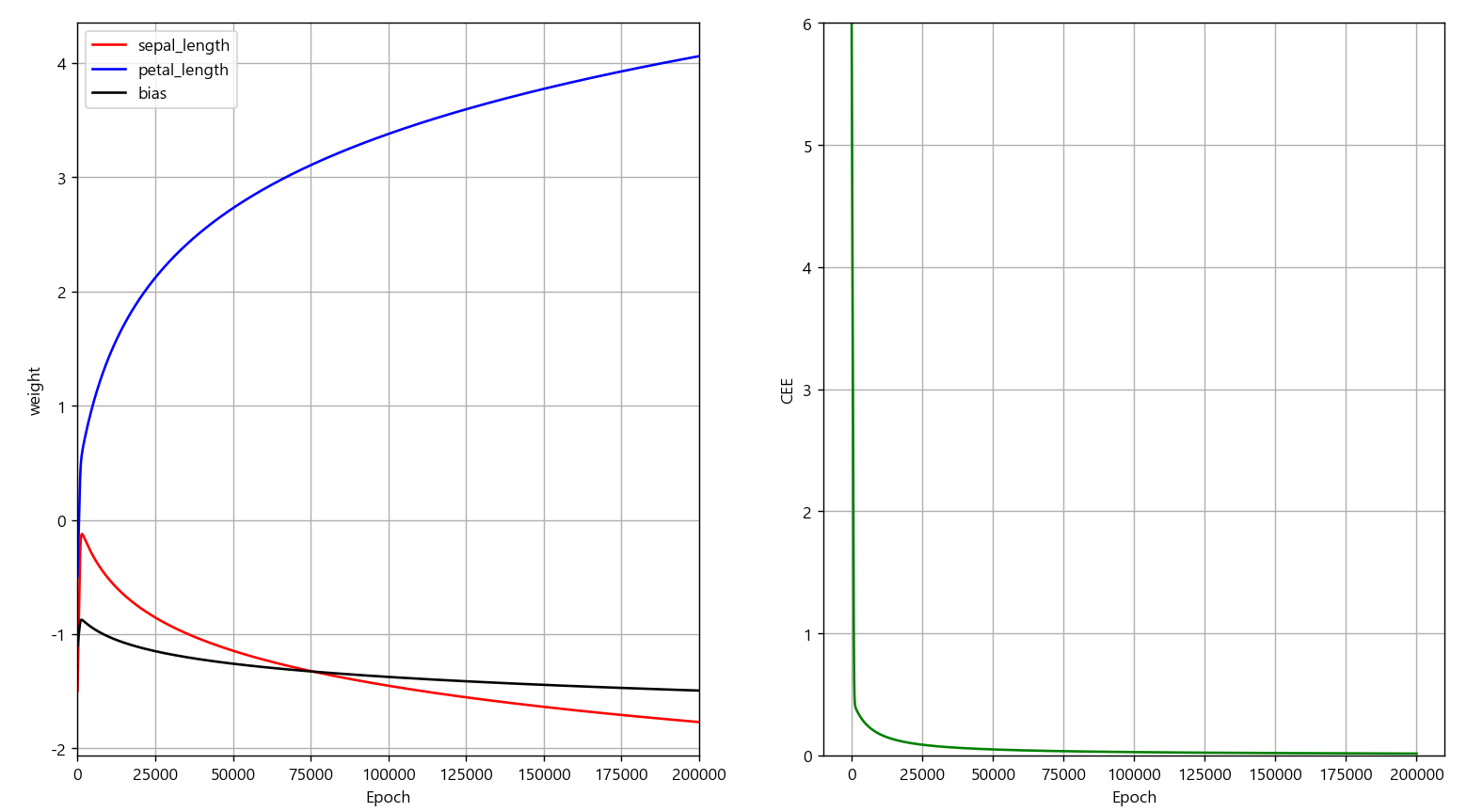


(그림 11. Iris dataset 분포도)

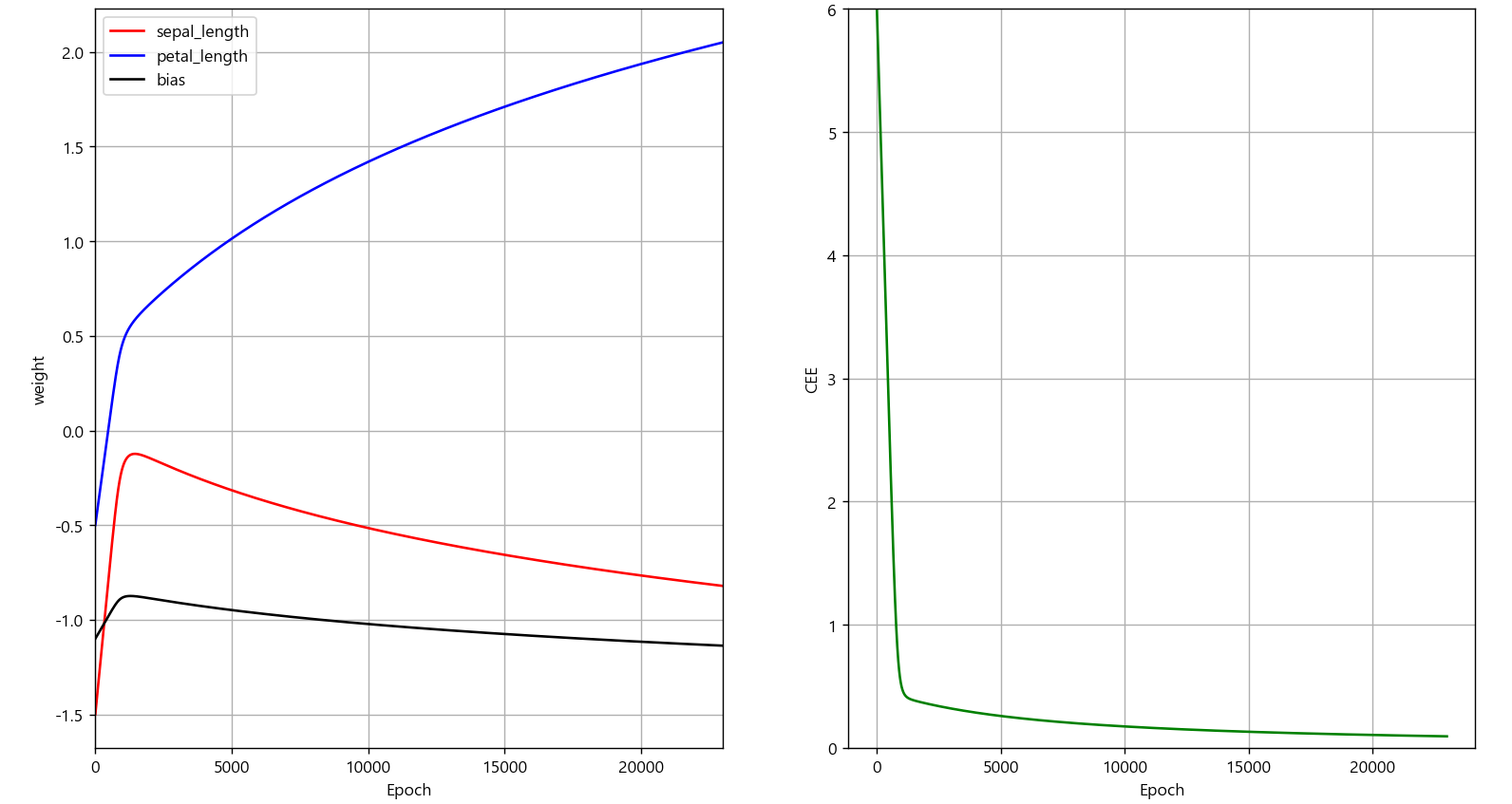
# **2. 실습 #2-2 (9주차)**

**2.2 Cross Entropy Loss를 비용함수로 사용한 경사하강법 결과**

|  |
| --- |
| # 경사하강법  # 초기 값  IRIS\_LEARNING\_RATE = 0.0005  IRIS\_N\_ITER = 20000  iris\_M = len(iris\_feature) # 특성의 갯수  iris\_M\_b = iris\_M+1        # 특성의 갯수 + bias  # -3~3의 임의의 값, (특성+bias)의 갯수만큼 리스트로 만듬  iris\_gd\_theta = np.array([random.uniform(-3,3) for \_ in range(iris\_M\_b)])  iris\_gd\_theta = np.array([-1.5, -0.5, -1.1])  # w0 \* x0 + w1\*x1 +  w2 \* 1(bias)를 만들기 위해 끝에 추가  # [sepal\_length, petal\_length, bias(1)]   [[w0, w1, w2]] <= 이런 형식  iris\_x\_b = np.c\_[iris\_df\_feature, np.ones(len(iris\_df\_label))]  # (특성+bias)의 갯수에 따라 변경되는 로직 작성하였기에  # 동일한 함수로 해당 함수 적용  iris\_result = theta\_gradient\_descent(iris\_x\_b, iris\_label, iris\_M\_b, th=iris\_gd\_theta, lr=IRIS\_LEARNING\_RATE, epoch=IRIS\_N\_ITER) |



(그림 2.1 [Epoch: 200,000] 가중치 값과 Cross Entropy Error 값)



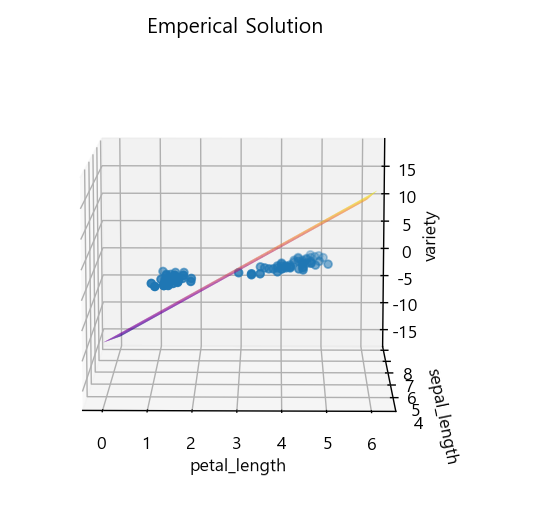
(그림 2.2 [Epoch: 23,000] 가중치 값과 Cross Entropy Error 값)

동일한 초기 가중치 값에 대한 Epoch가 20만번, 2만3천번 했을 때의 결과입니다.

# **2. 실습 #2-3-1 (9주차)**

**2.3.1 로지스틱 회귀 모델의 정확도 및 Decision Boundary 그래프 출력**

|  |
| --- |
| accuracy = best\_th\_accuracy(iris\_x\_b, iris\_label,  iris\_result[GD\_RESULT\_MAP["weight"]])  print("훈련결과, 정확도 {}%\nw0(sepal) = {}, w1(petal) = {}, w2(bias) = {}".format( \                                                           accuracy \* 100,\                                 iris\_result[GD\_RESULT\_MAP["weight"]][0], \                                 iris\_result[GD\_RESULT\_MAP["weight"]][1], \                                iris\_result[GD\_RESULT\_MAP["weight"]][2]))  def origin\_data\_space(weight):      petal\_min = (min(iris\_feature[FEATURE\_NAME["petal\_length"]]))      petal\_max = (max(iris\_feature[FEATURE\_NAME["petal\_length"]]))      petal\_lim = [petal\_min-1 ,petal\_max +1]      sepal\_min = (min(iris\_feature[FEATURE\_NAME["sepal\_length"]]))      sepal\_max = (max(iris\_feature[FEATURE\_NAME["sepal\_length"]]))      sepal\_lim = [sepal\_min-1, sepal\_max +1]        petal\_space = np.linspace(petal\_lim[0], petal\_lim[1], 1000)      sepal\_space = np.linspace(sepal\_lim[0], sepal\_lim[1], 1000)      petal\_mesh, sepal\_mesh = np.meshgrid(petal\_space, sepal\_space)      y\_space = sepal\_mesh \* weight[0] + petal\_mesh \* weight[1] + weight[2]        ax = plt.figure().add\_subplot(projection='3d')      ax.scatter(iris\_feature[FEATURE\_NAME["petal\_length"]], iris\_feature[FEATURE\_NAME["sepal\_length"]], iris\_label)      ax.set\_ylabel("sepal\_length"); ax.set\_xlabel("petal\_length"); ax.set\_zlabel("variety", rotation=0)      ax.plot\_surface(petal\_mesh, sepal\_mesh, y\_space, cmap="plasma")      plt.title("Emperical Solution")      plt.grid(True)      plt.show()    origin\_data\_space(iris\_result[GD\_RESULT\_MAP["weight"]]) |



(그림 3. 모델의 Decisioin boundary 및 분포도)



(그림 4. 정확도 및 최적의 가중치 값)

# **2. 실습 #2-3-2 (9주차)**

**2.3.2 예측하고 싶은 값을 통한 예측 결과 값**

|  |
| --- |
| # 예측된 결과 값에 따라 1이면 남자 0이면 여자를 출력하기 위한 리스트  RESULT\_IRIS = ["Setosa", "Versicolor"]  def predict\_iris(data, weight):      """      Args:          data (List or number(int or float)): 예측을 원하는 값          weight (np.array): 최적화된 가중치      """      # weight = bias + 특성(1) | 3 X 1 matrix      w = np.array(weight)      # data = 데이터 갯수 X 1(feature) martrix      d = np.asarray(data)      # d = 데이터 갯수 X 2(feature + bias) matrix      d\_b = np.c\_[d, np.ones(len(d))]      zn = d\_b @ w # linear equation      probability = sigmoid(zn) # sigmoid      y\_pred = probability\_func(probability) # Threshold = 0.5      for i in range(len(y\_pred)):          print("데이터 {}의 예측 결과 : Result={}".format(d[i], RESULT\_IRIS[y\_pred[i]]))  predict\_iris([[5.7,0.2],[6.4,4]], iris\_result[GD\_RESULT\_MAP["weight"]]) |



(그림15. 분리된 데이터 별 셋)