|  |  |
| --- | --- |
| 교육 제목 |  |
| 교육 일시 | 2021. 10. 26 |
| 교육 장소 | 영우글로벌러닝 |
| **교육 내용** | |
| 오전 | import pandas as pd  import tensorflow as tf  import matplotlib.pyplot as plt  # 데이터 읽어오기  df=pd.read\_csv("./BostonHousing.csv")  print(df.info)  df.head()  x\_data = df.copy()  tf.random.set\_seed(100)  y\_data = x\_data.pop('MEDV')  x\_data.head()  y\_data.head()  x\_data.shape  model = tf.keras.Sequential()  model.add(tf.keras.layers.Dense(64, input\_dim=13, activation='sigmoid' ))  model.add(tf.keras.layers.Dense(128, activation='sigmoid'))  model.add(tf.keras.layers.Dense(256, activation='sigmoid'))  model.add(tf.keras.layers.Dense(1))  model.summary()  optimizer = tf.keras.optimizers.SGD(learning\_rate=0.04)  loss = tf.keras.losses.mean\_squared\_error  metrics = tf.keras.metrics.RootMeanSquaredError()  model.compile(loss=loss, optimizer=optimizer, metrics=[metrics])  result = model.fit(x\_data, y\_data, epochs=100, batch\_size=100)  print(result.history.keys())  loss = result.history['loss']  # loss 그래프  epochs = range(1, len(loss)+1)  plt.subplot(211)  plt.plot(epochs, loss, 'b-', label='Training loss')  plt.title('Training loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  mae = result.history['root\_mean\_squared\_error']  epochs = range(1, len(mae)+1)  # mae그래프  plt.subplot(212)  plt.plot(epochs, mae, 'r-', label='Training mae')  plt.title('Training rmse')  plt.xlabel('Epochs')  plt.ylabel('rmse')  plt.legend()  print(model.evaluate(x\_data, y\_data))  print("\n Test rmse : %.4f" % (model.evaluate(x\_data, y\_data)[1]))  ----------------------------------------------------------------------------  import seaborn as sns  import pandas as pd  import matplotlib.pyplot as plt  data = pd.read\_csv('./test.csv')  print(data.info())  data.head()  x\_data = data.copy()  y\_data = x\_data.pop("D")  # 데이터 분포도 확인하기  sns.pairplot(data[["A","B","C","D"]], diag\_kind='hist')  plt.show()  # data의 min, max, mean, std값 구하기  dataset\_stats = data.describe()  dataset\_stats = dataset\_stats.transpose()  data.min()  data.max()  data.mean()  ## data normalization  def min\_max\_norm(x):  return (x - dataset\_stats['min']) / (dataset\_stats['max'] - dataset\_stats['min'])  def standard\_norm(x):  return (x - dataset\_stats['mean']) / dataset\_stats['std']  min\_max\_norm\_train\_data=min\_max\_norm(data)  standard\_norm\_train\_data=standard\_norm(data)  print("min max : ")  print(min\_max\_norm\_train\_data)  print("standard : ")  print(standard\_norm\_train\_data)  # data 분리  X\_train1, X\_test, Y\_train1, Y\_test=train\_test\_split(standard\_norm\_train\_data, train\_labels, test\_size=0.2, shuffle=False)  X\_train, X\_val, Y\_train, Y\_val=train\_test\_split(X\_train1, Y\_train1, test\_size=0.2, shuffle=False)  print("x train")  print(X\_train)  print("label train")  print(Y\_train)  print("x val")  print(X\_val)  print("label val")  print(Y\_val)  print("x test")  print(X\_test)  print("label test")  print(Y\_test) |
| 오후 | import pandas as pd  import tensorflow as tf  import matplotlib.pyplot as plt  data = pd.read\_csv('./BostonHousing.csv')  x\_data = data.copy()  ori\_y = x\_data.pop('MEDV')  from sklearn.model\_selection import train\_test\_split  x\_train1, x\_test, y\_train1, y\_test = train\_test\_split(x\_data, ori\_y, test\_size=0.3, shuffle=True)  x\_train, x\_valid, y\_train, y\_valid = train\_test\_split(x\_train1, y\_train1, test\_size=0.2, shuffle=True)  input\_layer = tf.keras.layers.Input(shape=(13,))  x = tf.keras.layers.Dense(50, activation='sigmoid')(input\_layer)  x = tf.keras.layers.Dense(100, activation='sigmoid')(x)  x = tf.keras.layers.Dense(300, activation='sigmoid')(x)  out\_layer = tf.keras.layers.Dense(1, activation=None)(x)  model = tf.keras.Model(inputs=[input\_layer], outputs=[out\_layer])  model.summary()  loss = tf.keras.losses.mean\_squared\_error  optimizer = tf.keras.optimizers.SGD(lr=0.0004)  metrics = tf.keras.metrics.RootMeanSquaredError()  model.compile(loss = loss,  optimizer = optimizer,  metrics = [metrics])  result = model.fit(x\_train, y\_train, epochs = 200, batch\_size = 10, validation\_data=(x\_valid, y\_valid))  print(result.history.keys())  loss = result.history['loss']  val\_loss = result.history['val\_loss']  ### loss와 val\_loss를 그래프화  epochs = range(1, len(loss) + 1)  plt.subplot(211) ## 2x1 개의 그래프 중에 1번째  plt.plot(epochs, loss, 'b-', label='Training loss')  plt.plot(epochs, val\_loss, 'r', label='Validation loss')  plt.title('Training and validation loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  ----------------------------------------------------------------------------------  import tensorflow as tf  import pandas as pd  from sklearn.preprocessing import LabelEncoder  import numpy as np  data = pd.read\_csv('./one\_hot\_encording\_test.csv', names = ["length", "width", "height", "volume", "class\_name"])  x\_data = data.copy()  y\_data = x\_data.pop('class\_name')  print("data value : {}".format(x\_data))  print("data Label : {}".format(y\_data))  # case 1) sklearn  # Label Encoder는 독립 변수가 아닌 종속 변수(라벨)에 대해 사용한다.  # 문자열이나 정수로된 라벨 값을 0 ~ K−1 까지의 정수로 변환.  e = LabelEncoder()  e.fit(y\_data)  print("Label Class String : {}".format(e.classes\_))  Y = e.transform(y\_data)  print("Label Class int : {}".format(Y))  y\_encoded = tf.keras.utils.to\_categorical(Y)  print("case 1 One hot label class : {}".format(y\_encoded))  print(np.argmax(y\_encoded, axis=1).reshape(-1,1))  print(y\_encoded.shape)  # case 2) pandas  one\_hot\_label = pd.get\_dummies(y\_data)  print("case2 one\_hot\_label : ", one\_hot\_label)  print(one\_hot\_label.shape)  ------------------------------------------------------------------------------------  import tensorflow as tf  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import LabelEncoder  import pandas as pd  # csv read  data = pd.read\_csv('./iris.csv', names = ["sepal\_length", "sepal\_width", "petal\_length", "petal\_width", "species"])  x\_data = data.copy()  y\_data = x\_data.pop("species")  X = x\_data.copy()  print(y\_data)  # 문자열을 숫자로 변환  e = LabelEncoder()  e.fit(y\_data)  y = e.transform(y\_data)  y\_encoded = tf.keras.utils.to\_categorical(y)  # train과 test를 분리  x\_train1, x\_test, y\_train1, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.1, shuffle=True)  # train set에서 train과 validation 분리  x\_train, x\_valid, y\_train, y\_valid = train\_test\_split(x\_train1, y\_train1, test\_size=0.2, shuffle=True)  print(x\_train.shape)  print(y\_train.shape)  input\_layer = tf.keras.layers.Input(shape=(4,))  x = tf.keras.layers.Dense(16, activation='sigmoid')(input\_layer)  x = tf.keras.layers.Dense(32, activation='sigmoid')(x)  output\_layer = tf.keras.layers.Dense(3, activation='softmax')(x)  model = tf.keras.models.Model(inputs=[input\_layer], outputs = [output\_layer])  model.summary()  loss = tf.keras.losses.categorical\_crossentropy  optimizer = tf.keras.optimizers.SGD(learning\_rate=0.04)  metrics = tf.keras.metrics.categorical\_accuracy  model.compile(loss=loss, optimizer=optimizer, metrics=[metrics])  history = model.fit(x\_train, y\_train, epochs=200, batch\_size=50, validation\_data=(x\_valid, y\_valid))  print(history.history.keys())  loss = history.history['loss']  val\_loss = history.history['val\_loss']  # loss와 val\_loss를 그래프  epochs = range(1, len(loss)+1)  plt.subplot(211)  plt.plot(epochs, loss, 'b-', label='Training loss')  plt.plot(epochs, val\_loss, 'r', label='Validation loss')  plt.title('Training and validation loss')  plt.xlabel('Epochs')  plt.ylabel('Loss')  plt.legend()  acc = history.history['categorical\_accuracy']  val\_acc = history.history['val\_categorical\_accuracy']  plt.subplot(212) ## 2x1 개의 그래프 중에 2번째  plt.plot(epochs, acc, 'b-', label='Training acc')  plt.plot(epochs, val\_acc, 'r', label='Validation acc')  plt.title('Training and validation accuracy')  plt.xlabel('Epochs')  plt.ylabel('Accuracy')  plt.legend()  print("\n Test Accuracy : %.4f" %(model.evaluate(x\_test, y\_test)[1]))  model.save('iris\_multi\_model.h5') |