기계학습 기말고사

2020020490
고려대학교 기계공학과 석사과정 문용환
moonyh1230@kist.re.kr

1. 붓꽃 데이터를 이용한 SVM

sklearn data 패키지의 iris_data를 활용하여 붓꽃을 분류하였다. 붓꽃 데이터는 setosa, versicolor, virginica의 세 가지 붓꽃 종(species)으로 구성된 target data와 꽃받침 길이와 폭, 꽃잎 길이와 폭의 4가지로 구성된 feature data로 이루어져 있다. 이 4종류의 feature data를 SVM의 두가지 kernel(linear, sigmoid)을 활용하여 분류하고, 결과를 비교하였다.

- linear kernel SVM

① cross validation score and accuracy (10 times)

[1. 1. 0.90909091 1. 1. 0.90909091
1. 1. 1. 0.90909091]
0.9727272727272727

② predict accuracy : 0.9737

③ class report

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	13
versicolor	1.00	0.94	0.97	16
virginica	0.90	1.00	0.95	9
accuracy			0.97	38
macro avg	0.97	0.98	0.97	38
weighted avg	0.98	0.97	0.97	38

- sigmoid kernel SVM

① cross validation score and accuracy (10 times)

[0.91666667 1. 0.90909091 1. 0.90909091 0.81818182
1. 0.90909091 0.72727273 0.90909091]
0.9098484848484848

② predict accuracy : 0.8947

③ class report

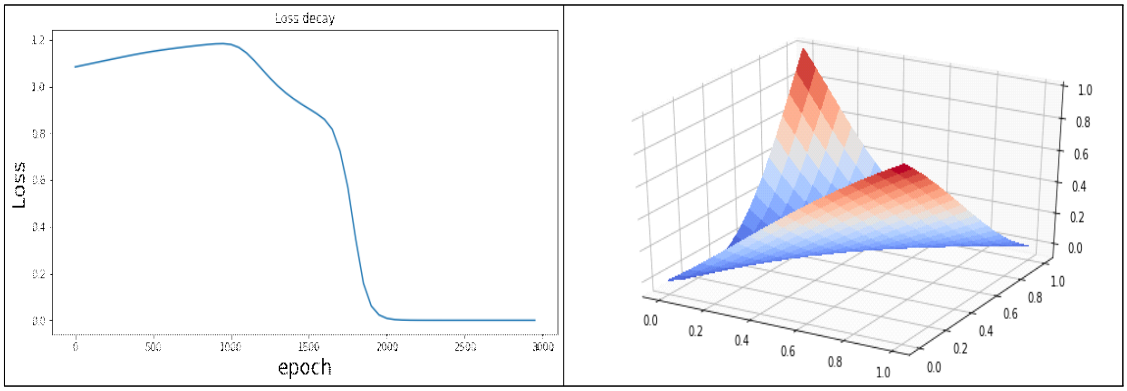
	precision	recall	f1-score	support
0	1.00	1.00	1.00	13
1	0.93	0.81	0.87	16
2	0.73	0.89	0.80	9
accuracy			0.89	38
macro avg	0.89	0.90	0.89	38
weighted avg	0.91	0.89	0.90	38

따라서 붓꽃 데이터를 분류하는데에는 Linear kernel SVM 모델이 더 적절하다고 볼 수 있다.

2. XOR problem

XOR 문제를 각각 sigmoid, linear, ReLu 활성함수를 이용하여 학습을 진행한 뒤 얻은 결과를 비교해 보았다.

- Sigmoid activation function result : hidden unit = 5, train rate = 0.1, 3000 train.



- check trained sigmoid model

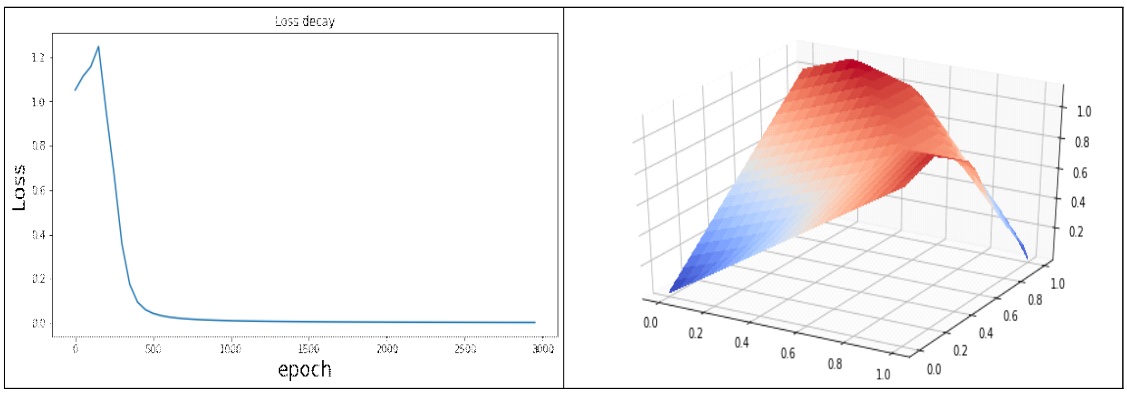
[3.40157921e-09]

[1.]

[1.]

[1.06601957e-09]

- ReLu activation function result : hidden unit = 5, train rate = 0.1, 3000 train.



- check trained ReLu model

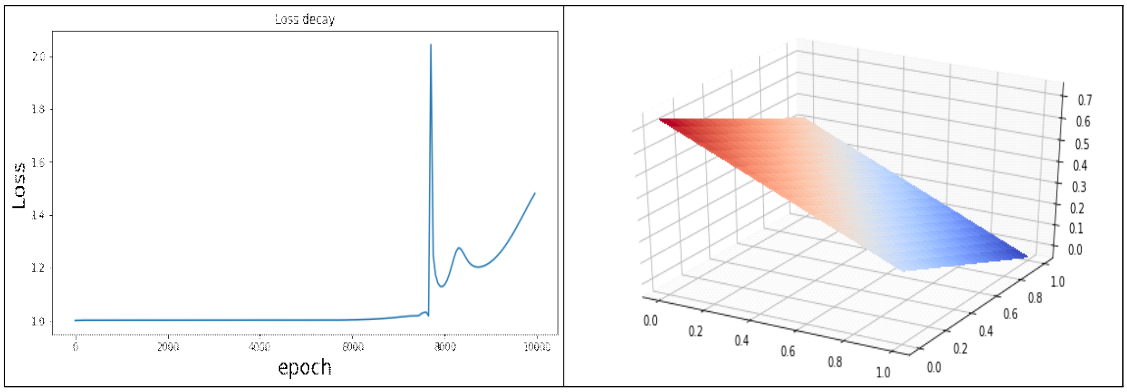
[0.00496469]

[0.99998474]

[0.99998327]

[2.64942907e-07]

- Linear activation function result : hidden unit = 3, train rate = 0.005, 10000 train.



- check trained linear model

[0.74939823]
[0.4167044]
[0.27398851]
[-0.05870532]

- 결과 분석

3가지의 활성함수를 사용하여 XOR 문제를 풀어 보았을 때, 가장 빠른 속도로 수렴하는 함수는 ReLu 활성함수였고, 가장 정밀한 학습 결과를 보여준 함수는 sigmoid 활성함수였다. Linear 활성함수는 다른 함수에 비해 3배 이상의 학습회수에도 불구하고 결국 값에 수렴하지 못했음을 알 수 있었다.

3. ANN

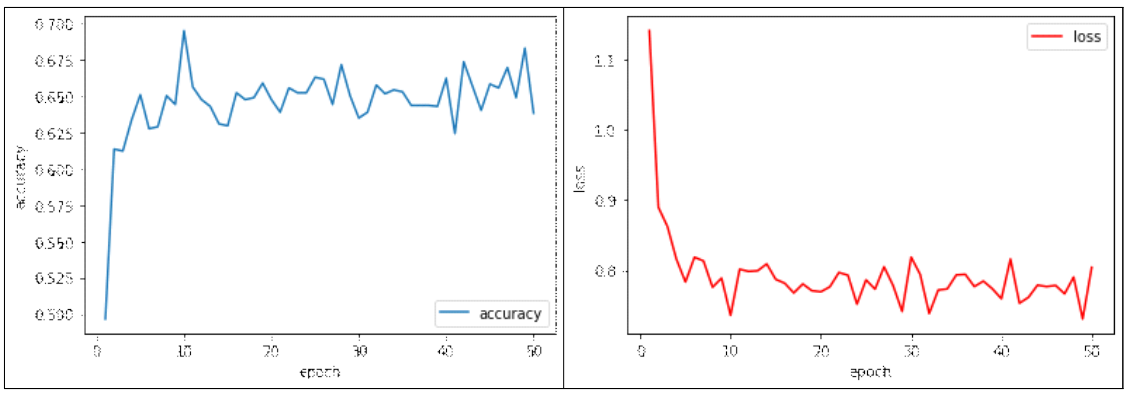
kaggle dataset의 휴대폰 가격 data를 활용하여 ANN 학습을 통해 모델을 수립한 뒤 학습 모델로 휴대폰의 가격을 예측해 보았다.

- 결과 분석

① hidden layer unit = 30, ReLu activation function, epoch = 50

train data acc : 0.8553333282470703

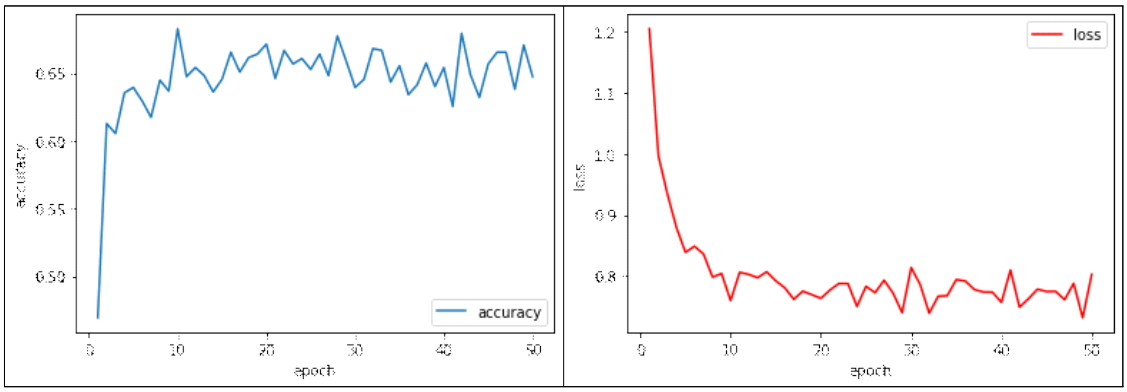
test data acc : 0.8420000076293945



② hidden layer unit = 25, Sigmoid activation function, epoch = 50

train data acc : 0.859333336353302

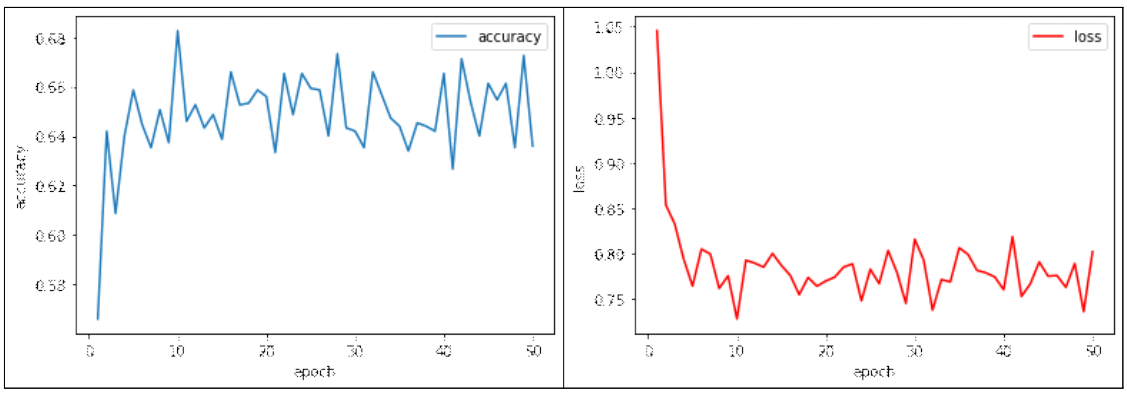
test data acc : 0.8539999723434448



③ hidden layer unit = 25, Sigmoid / ReLu activation function, epoch = 50

train data acc : 0.887333333492279

test data acc : 0.8679999709129333



세 학습 모델로 얻은 결과는 sigmoid activation function을 사용한 경우에서 근소하게 정확도가 높게 나타났고, 두 가지의 activation function을 모두 사용한 결과에서 가장 높게 나타났다. 하지만 두 가지의 activation function을 모두 사용한 모델은 epoch 마다 증감폭이 훨씬 크게 나타난 것을 알 수 있었다. hidden layer unit의 수가 늘어날수록 항상 정확도가 늘어나는 경우는 아니었으며, 적절한 개수의 hidden layer unit을 사용해야 하는 것으로 예상된다.

4. python code

① SVM

```
fromsklearn importdatasets
fromsklearn importsvm
fromsklearn.model_selection importtrain_test_split
fromsklearn.preprocessing importStandardScaler
fromsklearn.model_selection importcross_val_score
fromsklearn.metrics importaccuracy_score
fromsklearn.metrics importconfusion_matrix
fromsklearn.metrics importclassification_report
importpandas aspd
raw_iris = datasets.load_iris()

X = raw_iris.data
y = raw_iris.target
pd.DataFrame(X).head()

X_tn, X_te, y_tn, y_te = train_test_split(X, y, random_state = 1)
std_scale = StandardScaler()
std_scale.fit(X_tn)
X_tn_std = std_scale.transform(X_tn)
X_te_std  = std_scale.transform(X_te)
clf_svm_ln = svm.SVC(kernel = 'linear', random_state = 1)
clf_svm_ln.fit(X_tn_std, y_tn)
clf_svm_sg = svm.SVC(kernel = 'sigmoid', random_state = 1)
clf_svm_sg.fit(X_tn_std, y_tn)

cv_scores = cross_val_score(clf_svm_ln, X_tn_std, y_tn, cv = 10, scoring
= 'accuracy')
print(cv_scores)
print(cv_scores.mean())
print(cv_scores.std())

cv_scores = cross_val_score(clf_svm_sg, X_tn_std, y_tn, cv = 10, scoring
= 'accuracy')
print(cv_scores)
print(cv_scores.mean())
print(cv_scores.std())

pred_svm_ln = clf_svm_lr.predict(X_te_std)
print(pred_svm_ln)

pred_svm_sg = clf_svm_sg.predict(X_te_std)
print(pred_svm_sg)

accuracy_ln = accuracy_score(y_te, pred_svm_ln)
print(accuracy_ln)

accuracy_sg = accuracy_score(y_te, pred_svm_sg)
```

```
print(accuracy_sg)

conf_matrix_ln = confusion_matrix(y_te, pred_svm_ln)
print(conf_matrix_ln)

conf_matrix_sg = confusion_matrix(y_te, pred_svm_sg)
print(conf_matrix_sg)

class_report_ln = classification_report(y_te, pred_svm_ln)
print(class_report_ln)

class_report_sg = classification_report(y_te, pred_svm_sg)
print(class_report_sg)
```

② XOR problem

```
# sigmoid model
importnumpy asnp
importmatplotlib.pyplot asplt
%matplotlib inline
input = [[0,0,1],[0,1,1],[1,0,1],[1,1,1]]
output = [0,1,1,0]
N = np.size(input,0) # number of samples
Ni = np.size(input,1) # dimension of the samples of input
No = 1 # dimension of the sample of output
Nh = 5 # number of hidden units
Ws = 1/4*np.random.rand(Nh,Ni+1)
#print(Ws)
Wo = 1/4*np.random.rand(No,Nh)
#print(Wo)
alpha = 0.1 # Learning rate
t_ = []
loss_ = []
def sigmoid(x):
f = 1/(1+np.exp(-x))
    returnf
def linear(x):
    returnx
def ReLu(x):
    returnnp.where(x < 0, 0, x)
## train the sigmoid model  ==========================================
======================
forepoch in range(0,3000):
loss = 0
    forid_ in range(0,N):
dWs = 0*Ws
dWo = 0*Wo

x = np.append(input[id_],1)

S = np.dot(Ws,x)
```

```
y = np.dot(Wo,sigmoid(S))

d = output[id_]

        forj in range(0,Nh):
            fori in range(0,No):
dWo[i,j] = dWo[i,j] + sigmoid(S[j])*(y[i]-d)

Wo = Wo - alpha*dWo

        fork in range(0,Ni+1):
            forj in range(0,Nh):
                fori in range(0,No):
dWs[j,k] = dWs[j,k] + x[k]*Wo[i,j]*sigmoid(S[j])*(1-sigmoid(S[j]))*(y[i]-
d)

Ws = Ws - alpha*dWs

loss = loss + 1/2*np.linalg.norm(y-d)

    ifnp.mod(epoch,50) == 0:
        print(epoch,"-th epoch trained")

t_ = np.append(t_,epoch)

loss_ = np.append(loss_,loss)

fig = plt.figure(num=0,figsize=[10,5])
plt.plot(t_,loss_,marker="")
plt.title('Loss decay')
plt.xlabel('epoch',FontSize=20)
plt.ylabel('Loss',FontSize=20)
plt.show()

        ## figure out the function shape the model=======================
==================
xn = np.linspace(0,1,20)
yn = np.linspace(0,1,20)
xm, ym = np.meshgrid(xn, yn)
xx = np.reshape(xm,np.size(xm,0)*np.size(xm,1))
yy = np.reshape(ym,np.size(xm,0)*np.size(xm,1))
Z = []
        forid__ in range(0,np.size(xm)):
x = np.append([xx[id__],yy[id__]],[1,1])
S = np.dot(Ws,x)
y_ = np.dot(Wo,sigmoid(S))
Z = np.append(Z,y_)

fig = plt.figure(num=1,figsize=[10,5])
ax = fig.gca(projection='3d')
surf = ax.plot_surface(xm,ym,np.reshape(Z,(np.size(xm,0),np.size(xm,1))),
```

```
cmap='coolwarm',linewidth=0,antialiased=False)
        print("============================================================
==========")
plt.show()


## test the sigmoid trained model ======================================
============================
forid_ in range(0,N):
x = np.append(input[id_],1)

S = np.dot(Ws,x)

y = np.dot(Wo,sigmoid(S))

    print(y)

# ReLu model
importnumpy asnp
importmatplotlib.pyplot asplt
%matplotlib inline
input = [[0,0,1],[0,1,1],[1,0,1],[1,1,1]]
output = [0,1,1,0]
N = np.size(input,0) # number of samples
Ni = np.size(input,1) # dimension of the samples of input
No = 1 # dimension of the sample of output
Nh = 5 # number of hidden units
Ws = 1/4*np.random.rand(Nh,Ni+1)
#print(Ws)
Wo = 1/4*np.random.rand(No,Nh)
#print(Wo)
alpha = 0.1 # Learning rate
t_ = []
loss_ = []
def sigmoid(x):
f = 1/(1+np.exp(-x))
    returnf
def linear(x):
    returnx
def ReLu(x):
    returnnp.where(x < 0, 0, x)
## train the ReLu model  ===============================================
====================
forepoch in range(0,3000):
loss = 0
    forid_ in range(0,N):
dWs = 0*Ws
dWo = 0*Wo

x = np.append(input[id_],1)

S = np.dot(Ws,x)
```

```
y = np.dot(Wo,ReLu(S))

d = output[id_]

        forj in range(0,Nh):
            fori in range(0,No):
dWo[i,j] = dWo[i,j] + ReLu(S[j])*(y[i]-d)

Wo = Wo - alpha*dWo

        fork in range(0,Ni+1):
            forj in range(0,Nh):
                fori in range(0,No):
dWs[j,k] = dWs[j,k] + x[k]*Wo[i,j]*ReLu(S[j])*(1-ReLu(S[j]))*(y[i]-d)

Ws = Ws - alpha*dWs

loss = loss + 1/2*np.linalg.norm(y-d)

    ifnp.mod(epoch,50) == 0:
        print(epoch,"-th epoch trained")

t_ = np.append(t_,epoch)

loss_ = np.append(loss_,loss)

fig = plt.figure(num=0,figsize=[10,5])
plt.plot(t_,loss_,marker="")
plt.title('Loss decay')
plt.xlabel('epoch',FontSize=20)
plt.ylabel('Loss',FontSize=20)
plt.show()

        ## figure out the function shape the model======================
==================
xn = np.linspace(0,1,20)
yn = np.linspace(0,1,20)
xm, ym = np.meshgrid(xn, yn)
xx = np.reshape(xm,np.size(xm,0)*np.size(xm,1))
yy = np.reshape(ym,np.size(xm,0)*np.size(xm,1))
Z = []
        forid__ in range(0,np.size(xm)):
x = np.append([xx[id__],yy[id__]],[1,1])
S = np.dot(Ws,x)
y_ = np.dot(Wo,ReLu(S))
Z = np.append(Z,y_)

fig = plt.figure(num=1,figsize=[10,5])
ax = fig.gca(projection='3d')
surf = ax.plot_surface(xm,ym,np.reshape(Z,(np.size(xm,0),np.size(xm,1))),
cmap='coolwarm',linewidth=0,antialiased=False)
        print("============================================================
```

```
==========")
plt.show()


## test the trained ReLu model ========================================
==========================
forid_ in range(0,N):
x = np.append(input[id_],1)

S = np.dot(Ws,x)

y = np.dot(Wo,ReLu(S))

    print(y)

# Linear model
importnumpy asnp
importmatplotlib.pyplot asplt
%matplotlib inline
input = [[0,0,1],[0,1,1],[1,0,1],[1,1,1]]
output = [0,1,1,0]
N = np.size(input,0) # number of samples
Ni = np.size(input,1) # dimension of the samples of input
No = 1 # dimension of the sample of output
Nh = 3 # number of hidden units
Ws = 1/4*np.random.rand(Nh,Ni+1)
#print(Ws)
Wo = 1/4*np.random.rand(No,Nh)
#print(Wo)
alpha = 0.005 # Learning rate
t_ = []
loss_ = []
def sigmoid(x):
f = 1/(1+np.exp(-x))
    returnf
def linear(x):
    returnx
def ReLu(x):
    returnnp.where(x < 0, 0, x)
## train the linear model  ============================================
======================
forepoch in range(0,10000):
loss = 0
    forid_ in range(0,N):
dWs = 0*Ws
dWo = 0*Wo

x = np.append(input[id_],1)

S = np.dot(Ws,x)

y = np.dot(Wo,linear(S))
```

```
d = output[id_]

        forj in range(0,Nh):
            fori in range(0,No):
dWo[i,j] = dWo[i,j] + linear(S[j])*(y[i]-d)

Wo = Wo - alpha*dWo

        fork in range(0,Ni+1):
            forj in range(0,Nh):
                fori in range(0,No):
dWs[j,k] = dWs[j,k] + x[k]*Wo[i,j]*linear(S[j])*(1-linear(S[j]))*(y[i]-d)


Ws = Ws - alpha*dWs

loss = loss + 1/2*np.linalg.norm(y-d)

    ifnp.mod(epoch,50) == 0:
        print(epoch,"-th epoch trained")

t_ = np.append(t_,epoch)

loss_ = np.append(loss_,loss)

fig = plt.figure(num=0,figsize=[10,5])
plt.plot(t_,loss_,marker="")
plt.title('Loss decay')
plt.xlabel('epoch',FontSize=20)
plt.ylabel('Loss',FontSize=20)
plt.show()

        ## figure out the function shape the model======================
==================
xn = np.linspace(0,1,20)
yn = np.linspace(0,1,20)
xm, ym = np.meshgrid(xn, yn)
xx = np.reshape(xm,np.size(xm,0)*np.size(xm,1))
yy = np.reshape(ym,np.size(xm,0)*np.size(xm,1))
Z = []
        forid__ in range(0,np.size(xm)):
x = np.append([xx[id__],yy[id__]],[1,1])
S = np.dot(Ws,x)
y_ = np.dot(Wo,linear(S))
Z = np.append(Z,y_)

fig = plt.figure(num=1,figsize=[10,5])
ax = fig.gca(projection='3d')
surf = ax.plot_surface(xm,ym,np.reshape(Z,(np.size(xm,0),np.size(xm,1))),
cmap='coolwarm',linewidth=0,antialiased=False)
        print("============================================================
```

```
==========")
plt.show()
## test the trained linear model ========================================
============================
forid_ in range(0,N):
x = np.append(input[id_],1)

S = np.dot(Ws,x)

y = np.dot(Wo,linear(S))

    print(y)
```

③ ANN

```
fromsklearn importdatasets
fromtensorflow.keras.utils importto_categorical
fromsklearn.model_selection importtrain_test_split
importnumpy asnp
importtensorflow astf
importpandas aspd
fromtensorflow.keras.models importSequential
fromtensorflow.keras.layers importDense
fromtensorflow.keras.layers importBatchNormalization
fromtensorflow.keras.layers importActivation
importmatplotlib.pyplot asplt
np.random.seed(0)
tf.random.set_seed(0)
raw_phone = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/train.csv
')
print(raw_phone.head())
X = raw_phone.drop("price_range", axis = 1)
y = raw_phone["price_range"]
# 피쳐 데이터 차원 확인
print(X.shape)
# 타겟 데이터 종류 확인
print(set(y))
# 타겟 데이터 원-핫 인코딩
y_hot = to_categorical(y)
# 트레이닝/테스트 데이터 분할
X_tn, X_te, y_tn, y_te = train_test_split(X, y_hot, random_state = 1)
# 신경망 생성
n_feat = X_tn.shape[1]
n_class = len(set(y))
epo = 50
model = Sequential()
model.add(Dense(25, input_dim = n_feat))
model.add(BatchNormalization())
model.add(Activation('sigmoid'))
model.add(Dense(25))
model.add(Activation('relu'))
model.add(Dense(n_class))
```

```
model.add(Activation('softmax'))
# 신경망 모형 구조 확인
model.summary()
# 모형 컴파일
model.compile(loss='categorical_crossentropy',
optimizer='adam',
metrics=['accuracy'])
# 신경망 학습
hist = model.fit(X_tn, y_tn, epochs=epo, batch_size=5)
# 트레이닝 데이터 평가
print(model.evaluate(X_tn, y_tn)[1])
# 테스트 데이터 평가
print(model.evaluate(X_te, y_te)[1])
epoch = np.arange(1,epo+1)
accuracy = hist.history['accuracy']
loss = hist.history['loss']
# 정확도 학습 그래프
plt.plot(epoch, accuracy, label='accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend()
plt.show()
# 손실 그래프
plt.plot(epoch, loss,'r', label='loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend()
plt.show()
```