# **분류**

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense, Dropout

**import** tensorflow **as** tf

**from** sklearn.ensemble **import** ExtraTreesRegressor

*# (1) 설명변수 데이터 로딩*

sim\_n300\_p1000\_mlinfo **=** pd**.**read\_table('data/sim\_n300\_p1000.mlinfo',sep**=**'\t')

sim\_n300\_p1000\_mldose **=** pd**.**read\_table('data/sim\_n300\_p1000.mldose',sep**=**'\t', header**=None**)

sim\_n150\_p1000\_mlinfo **=** pd**.**read\_table('data/sim\_n150\_p1000.mlinfo',sep**=**'\t')

sim\_n150\_p1000\_mldose **=** pd**.**read\_table('data/sim\_n150\_p1000.mldose',sep**=**'\t', header**=None**)

key\_column **=** ['fid', 'iid']

independent\_column **=** []

dependent\_column **=** ['response\_val\_1', 'response\_val\_2']

**for** idx, column **in** enumerate(sim\_n300\_p1000\_mldose**.**columns) :

**if** idx **>** 1 :

independent\_column**.**append('snp'**+**str(idx**-**1))

*# sim\_n300\_p1000\_mldose\_header.append('snp'+str(idx-1))*

sim\_n300\_p1000\_mldose**.**columns **=** key\_column **+** independent\_column

sim\_n150\_p1000\_mldose**.**columns **=** key\_column **+** independent\_column

snp\_column **=** []

**for** idx **in** sim\_n150\_p1000\_mlinfo**.**index**.**values :

snp\_column**.**append('snp'**+**str(idx **+**1))

sim\_n300\_p1000\_mlinfo['snp\_name'] **=** snp\_column

sim\_n150\_p1000\_mlinfo['snp\_name'] **=** snp\_column

print(sim\_n150\_p1000\_mlinfo)

print(sim\_n300\_p1000\_mldose)

SNP Al1 Al2 Freq1 MAF Quality Rsq snp\_name

0 22:17255527:C:T C T 0.057157 0.942843 0.9 0.95 snp1

1 22:17292258:C:A C A 0.716933 0.283067 0.9 0.95 snp2

2 22:17294997:G:A G A 0.486633 0.513367 0.9 0.95 snp3

3 22:17407755:T:C T C 0.822010 0.177990 0.9 0.95 snp4

4 22:17511898:C:G C G 0.695133 0.304867 0.9 0.95 snp5

.. ... .. .. ... ... ... ... ...

995 22:50922544:T:C T C 0.928957 0.071043 0.9 0.95 snp996

996 22:50935458:A:G A G 0.134260 0.865740 0.9 0.95 snp997

997 22:50986360:T:C T C 0.500530 0.499470 0.9 0.95 snp998

998 22:51068751:C:T C T 0.954100 0.045900 0.9 0.95 snp999

999 22:51101938:A:C A C 0.819027 0.180973 0.9 0.95 snp1000

[1000 rows x 8 columns]

fid iid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 ... \

0 1 1 0.000 1.999 0.002 0.002 0.064 0.997 0.000 0.998 ...

1 2 2 0.014 1.999 1.452 1.007 1.482 1.952 0.067 0.998 ...

2 3 3 0.000 2.000 0.001 1.020 1.980 1.998 1.996 2.000 ...

3 4 4 0.754 1.995 1.001 2.000 1.996 1.999 0.000 1.999 ...

4 5 5 0.001 0.961 1.999 2.000 1.998 1.999 0.000 1.213 ...

.. ... ... ... ... ... ... ... ... ... ... ...

295 296 296 0.001 1.002 1.002 2.000 0.428 0.979 0.000 0.214 ...

296 297 297 0.000 1.003 1.001 2.000 1.987 1.999 0.998 1.000 ...

297 298 298 0.000 1.998 0.001 1.992 1.000 2.000 0.999 1.000 ...

298 299 299 0.153 2.000 0.002 1.339 1.993 1.999 0.000 1.999 ...

299 300 300 0.815 1.017 0.984 1.999 1.989 1.999 1.000 2.000 ...

snp991 snp992 snp993 snp994 snp995 snp996 snp997 snp998 snp999 \

0 1.999 2.000 1.999 2.000 2.000 1.989 0.000 0.267 2.000

1 0.001 0.000 0.000 1.999 2.000 1.014 0.004 0.995 2.000

2 2.000 2.000 1.999 1.010 1.002 1.264 0.000 0.007 1.992

3 0.999 1.000 0.999 1.065 1.027 1.993 0.176 0.025 2.000

4 2.000 2.000 1.999 1.999 2.000 1.996 0.000 0.999 2.000

.. ... ... ... ... ... ... ... ... ...

295 1.000 1.000 0.999 1.989 2.000 1.975 0.993 1.031 1.999

296 1.000 1.000 1.000 2.000 1.999 1.998 0.000 0.006 2.000

297 0.000 0.000 0.000 2.000 2.000 2.000 0.997 1.008 2.000

298 1.000 1.000 1.000 2.000 2.000 2.000 0.000 1.999 2.000

299 0.999 0.999 0.999 2.000 1.960 1.002 0.015 0.542 2.000

snp1000

0 1.999

1 1.999

2 1.996

3 2.000

4 1.003

.. ...

295 1.001

296 2.000

297 2.000

298 1.237

299 0.997

[300 rows x 1002 columns]

*#(2) 반응변수 데이터 로딩*

pheno\_n300\_binary\_phe **=** pd**.**read\_table('data/pheno\_n300\_binary.phe',sep**=**'\t', header**=None**)

pheno\_n300\_binary\_phe**.**columns **=** key\_column **+** dependent\_column

pheno\_n150\_binary\_phe **=** pd**.**read\_table('data/pheno\_n150\_binary.phe',sep**=**'\t', header**=None**)

pheno\_n150\_binary\_phe**.**columns **=** key\_column **+** dependent\_column

print(pheno\_n300\_binary\_phe)

print(pheno\_n150\_binary\_phe)

fid iid response\_val\_1 response\_val\_2

0 1 1 0 0

1 2 2 0 0

2 3 3 1 0

3 4 4 0 1

4 5 5 0 1

.. ... ... ... ...

295 296 296 0 1

296 297 297 0 0

297 298 298 1 0

298 299 299 0 1

299 300 300 0 0

[300 rows x 4 columns]

fid iid response\_val\_1 response\_val\_2

0 1 1 0 0

1 2 2 0 0

2 3 3 0 0

3 4 4 1 0

4 5 5 1 1

.. ... ... ... ...

145 146 146 1 0

146 147 147 1 0

147 148 148 0 0

148 149 149 0 0

149 150 150 0 0

[150 rows x 4 columns]

*#(3) 데이터 병합 및 훈련,테스트 데이터 구별*

*# label로 데이터 변환하여 진행 해보기도 함.*

*# from sklearn.preprocessing import LabelEncoder*

train\_data **=** pd**.**merge(sim\_n300\_p1000\_mldose, pheno\_n300\_binary\_phe, how**=**'inner', on **=**['fid', 'iid'])

test\_data **=** pd**.**merge(sim\_n150\_p1000\_mldose, pheno\_n150\_binary\_phe, how**=**'inner', on **=**['fid', 'iid'])

train\_x **=** train\_data[independent\_column]

test\_x **=** test\_data[independent\_column]

train\_y **=** train\_data[dependent\_column]

test\_y **=** test\_data[dependent\_column]

*# train\_data.to\_csv('train.csv')*

*# test\_data.to\_csv('test.csv')*

*# train\_y = tf.keras.utils.to\_categorical(train\_data[independent\_column], num\_classes=2)*

*# test\_y = tf.keras.utils.to\_categorical(train\_data[independent\_column], num\_classes=2)*

*# print(train\_y)*

*## (4) keras를 통한 NN Network 구성*

*# 반복 실험을 통해 hyper parameter는 최적의 결과로 도출 되는 layer를 이용*

model **=** Sequential()

model**.**add(Dense(2, input\_dim**=**len(train\_x**.**columns), activation**=**'relu'))

*# model.add(Dropout(0.5))*

model**.**add(Dense(5, activation**=**'relu'))

model**.**add(Dense(2, activation**=**'softmax'))

*# categorical\_crossentropy, sparse\_categorical\_crossentropy, binary\_crossentropy*

model**.**compile(loss**=**'categorical\_crossentropy', optimizer**=**'adam', metrics**=**['accuracy', tf**.**keras**.**metrics**.**Precision(name**=**'precision'), tf**.**keras**.**metrics**.**Recall(name**=**'recall')])

model**.**summary()

callbacks **=** [tf**.**keras**.**callbacks**.**EarlyStopping(monitor**=**'val\_loss', patience**=**20)]

result **=** model**.**fit(train\_x, train\_y, validation\_split**=**0.25, epochs**=**100, batch\_size**=**10, callbacks**=**callbacks)

Model: "sequential\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense\_3 (Dense) (None, 2) 2002

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_4 (Dense) (None, 5) 15

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_5 (Dense) (None, 2) 12

=================================================================

Total params: 2,029

Trainable params: 2,029

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Epoch 1/100

23/23 [==============================] - 1s 22ms/step - loss: 0.2937 - accuracy: 0.6355 - precision: 0.1943 - recall: 0.4761 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 2/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2836 - accuracy: 0.8823 - precision: 0.2195 - recall: 0.5376 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 3/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2878 - accuracy: 0.8292 - precision: 0.1874 - recall: 0.4573 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 4/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2842 - accuracy: 0.8793 - precision: 0.2267 - recall: 0.5503 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 5/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2465 - accuracy: 0.8901 - precision: 0.1959 - recall: 0.5564 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 6/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2853 - accuracy: 0.8506 - precision: 0.2068 - recall: 0.5048 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 7/100

23/23 [==============================] - 0s 7ms/step - loss: 0.2272 - accuracy: 0.9072 - precision: 0.1984 - recall: 0.6334 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 8/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2750 - accuracy: 0.8515 - precision: 0.2102 - recall: 0.5285 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 9/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2718 - accuracy: 0.8611 - precision: 0.1969 - recall: 0.5010 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 10/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2729 - accuracy: 0.8572 - precision: 0.1813 - recall: 0.4644 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 11/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2644 - accuracy: 0.8913 - precision: 0.2310 - recall: 0.6126 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 12/100

23/23 [==============================] - 0s 7ms/step - loss: 0.2583 - accuracy: 0.8650 - precision: 0.1802 - recall: 0.4865 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 13/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2533 - accuracy: 0.8860 - precision: 0.2059 - recall: 0.5668 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 14/100

23/23 [==============================] - 0s 6ms/step - loss: 0.3120 - accuracy: 0.8280 - precision: 0.2157 - recall: 0.4793 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 15/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2484 - accuracy: 0.8524 - precision: 0.1648 - recall: 0.4546 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 16/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2768 - accuracy: 0.8819 - precision: 0.2130 - recall: 0.5305 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 17/100

23/23 [==============================] - 0s 6ms/step - loss: 0.3256 - accuracy: 0.8364 - precision: 0.2267 - recall: 0.4834 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 18/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2461 - accuracy: 0.8931 - precision: 0.2006 - recall: 0.5660 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 19/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2311 - accuracy: 0.8525 - precision: 0.1460 - recall: 0.4187 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 20/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2785 - accuracy: 0.8519 - precision: 0.2093 - recall: 0.5206 - val\_loss: 0.2866 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

Epoch 21/100

23/23 [==============================] - 0s 6ms/step - loss: 0.2705 - accuracy: 0.8848 - precision: 0.2219 - recall: 0.5650 - val\_loss: 0.2867 - val\_accuracy: 0.8000 - val\_precision: 0.1867 - val\_recall: 0.4516

**def** vis(history, name) :

plt**.**title(f"{name**.**upper()}")

plt**.**xlabel('epochs')

plt**.**ylabel(f"{name**.**lower()}")

value **=** history**.**history**.**get(name)

val\_value **=** history**.**history**.**get(f"val\_{name}",**None**)

epochs **=** range(1, len(value)**+**1)

plt**.**plot(epochs, value, 'b-', label**=**f'training {name}')

**if** val\_value **is** **not** **None** :

plt**.**plot(epochs, val\_value, 'r:', label**=**f'validation {name}')

plt**.**legend(loc**=**'center', bbox\_to\_anchor**=**(0.05, 1.2) , fontsize**=**10 , ncol**=**1)

**def** plot\_history(history) :

key\_value **=** list(set([i**.**split("val\_")[**-**1] **for** i **in** list(history**.**history**.**keys())]))

plt**.**figure(figsize**=**(12, 4))

**for** idx , key **in** enumerate(key\_value) :

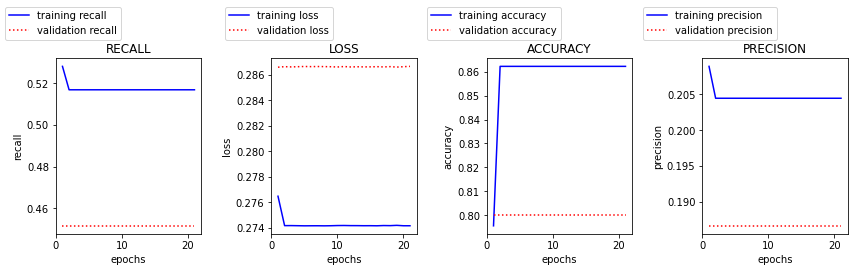
plt**.**subplot(1, len(key\_value), idx**+**1)

vis(history, key)

plt**.**tight\_layout()

plt**.**show()

plot\_history(result)



*#(5) test set으로 검증 시 정확도는 75~85% 사이로 나옴.*

score **=** model**.**evaluate(test\_x, test\_y, batch\_size**=**10)

print('loss and metrics and precision and recall :', score)

plot\_history(result)

*# prediction = model.predict(test\_x)*

*# print(prediction)*

*# print(test\_y.values)*

15/15 [==============================] - 0s 3ms/step - loss: 0.2541 - accuracy: 0.8733 - precision: 0.1933 - recall: 0.5273

loss and metrics and precision and recall : [0.25407883524894714, 0.8733333349227905, 0.19333332777023315, 0.5272727012634277]

# **회귀**

**import** pandas **as** pd

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** keras.models **import** Sequential

**from** keras.layers **import** Dense, Dropout, Flatten

**import** tensorflow **as** tf

**from** sklearn.ensemble **import** ExtraTreesRegressor

*# (1) 설명변수 데이터 로딩*

sim\_n300\_p1000\_mlinfo **=** pd**.**read\_table('data/sim\_n300\_p1000.mlinfo',sep**=**'\t')

sim\_n300\_p1000\_mldose **=** pd**.**read\_table('data/sim\_n300\_p1000.mldose',sep**=**'\t', header**=None**)

sim\_n150\_p1000\_mlinfo **=** pd**.**read\_table('data/sim\_n150\_p1000.mlinfo',sep**=**'\t')

sim\_n150\_p1000\_mldose **=** pd**.**read\_table('data/sim\_n150\_p1000.mldose',sep**=**'\t', header**=None**)

key\_column **=** ['fid', 'iid']

independent\_column **=** []

dependent\_column **=** ['response\_val\_1', 'response\_val\_2']

**for** idx, column **in** enumerate(sim\_n300\_p1000\_mldose**.**columns) :

**if** idx **>** 1 :

independent\_column**.**append('snp'**+**str(idx**-**1))

*# sim\_n300\_p1000\_mldose\_header.append('snp'+str(idx-1))*

sim\_n300\_p1000\_mldose**.**columns **=** key\_column **+** independent\_column

sim\_n150\_p1000\_mldose**.**columns **=** key\_column **+** independent\_column

snp\_column **=** []

**for** idx **in** sim\_n150\_p1000\_mlinfo**.**index**.**values :

snp\_column**.**append('snp'**+**str(idx **+**1))

sim\_n300\_p1000\_mlinfo['snp\_name'] **=** snp\_column

sim\_n150\_p1000\_mlinfo['snp\_name'] **=** snp\_column

print(sim\_n150\_p1000\_mlinfo)

print(sim\_n300\_p1000\_mldose)

SNP Al1 Al2 Freq1 MAF Quality Rsq snp\_name

0 22:17255527:C:T C T 0.057157 0.942843 0.9 0.95 snp1

1 22:17292258:C:A C A 0.716933 0.283067 0.9 0.95 snp2

2 22:17294997:G:A G A 0.486633 0.513367 0.9 0.95 snp3

3 22:17407755:T:C T C 0.822010 0.177990 0.9 0.95 snp4

4 22:17511898:C:G C G 0.695133 0.304867 0.9 0.95 snp5

.. ... .. .. ... ... ... ... ...

995 22:50922544:T:C T C 0.928957 0.071043 0.9 0.95 snp996

996 22:50935458:A:G A G 0.134260 0.865740 0.9 0.95 snp997

997 22:50986360:T:C T C 0.500530 0.499470 0.9 0.95 snp998

998 22:51068751:C:T C T 0.954100 0.045900 0.9 0.95 snp999

999 22:51101938:A:C A C 0.819027 0.180973 0.9 0.95 snp1000

[1000 rows x 8 columns]

fid iid snp1 snp2 snp3 snp4 snp5 snp6 snp7 snp8 ... \

0 1 1 0.000 1.999 0.002 0.002 0.064 0.997 0.000 0.998 ...

1 2 2 0.014 1.999 1.452 1.007 1.482 1.952 0.067 0.998 ...

2 3 3 0.000 2.000 0.001 1.020 1.980 1.998 1.996 2.000 ...

3 4 4 0.754 1.995 1.001 2.000 1.996 1.999 0.000 1.999 ...

4 5 5 0.001 0.961 1.999 2.000 1.998 1.999 0.000 1.213 ...

.. ... ... ... ... ... ... ... ... ... ... ...

295 296 296 0.001 1.002 1.002 2.000 0.428 0.979 0.000 0.214 ...

296 297 297 0.000 1.003 1.001 2.000 1.987 1.999 0.998 1.000 ...

297 298 298 0.000 1.998 0.001 1.992 1.000 2.000 0.999 1.000 ...

298 299 299 0.153 2.000 0.002 1.339 1.993 1.999 0.000 1.999 ...

299 300 300 0.815 1.017 0.984 1.999 1.989 1.999 1.000 2.000 ...

snp991 snp992 snp993 snp994 snp995 snp996 snp997 snp998 snp999 \

0 1.999 2.000 1.999 2.000 2.000 1.989 0.000 0.267 2.000

1 0.001 0.000 0.000 1.999 2.000 1.014 0.004 0.995 2.000

2 2.000 2.000 1.999 1.010 1.002 1.264 0.000 0.007 1.992

3 0.999 1.000 0.999 1.065 1.027 1.993 0.176 0.025 2.000

4 2.000 2.000 1.999 1.999 2.000 1.996 0.000 0.999 2.000

.. ... ... ... ... ... ... ... ... ...

295 1.000 1.000 0.999 1.989 2.000 1.975 0.993 1.031 1.999

296 1.000 1.000 1.000 2.000 1.999 1.998 0.000 0.006 2.000

297 0.000 0.000 0.000 2.000 2.000 2.000 0.997 1.008 2.000

298 1.000 1.000 1.000 2.000 2.000 2.000 0.000 1.999 2.000

299 0.999 0.999 0.999 2.000 1.960 1.002 0.015 0.542 2.000

snp1000

0 1.999

1 1.999

2 1.996

3 2.000

4 1.003

.. ...

295 1.001

296 2.000

297 2.000

298 1.237

299 0.997

[300 rows x 1002 columns]

*# (2) 설명변수 중 Freq1, MAF 분포 특성 파악*

*# 유전자에 대한 도메인이 있으면 특정 이상/이하 값으로 SNP 선별*

*# 정규 분포를 따른다면 이상치 제거 정도는 가능할 것으로 생각해서 그려 봄.*

plt**.**hist(sim\_n300\_p1000\_mlinfo['Freq1'])

plt**.**show()

plt**.**hist(sim\_n300\_p1000\_mlinfo['MAF'])

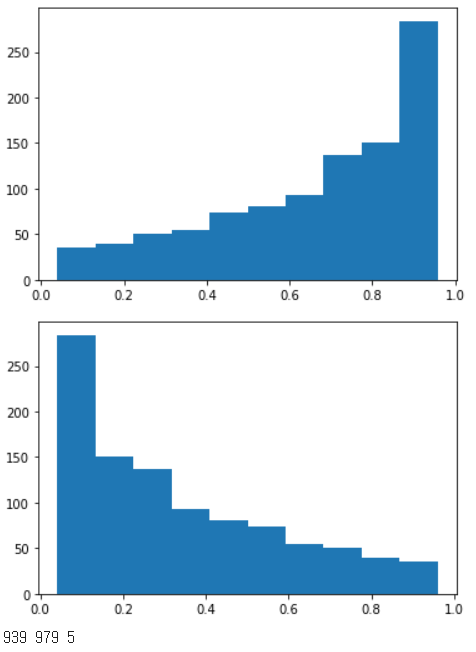
plt**.**show()

feature\_name **=** sim\_n300\_p1000\_mlinfo[sim\_n300\_p1000\_mlinfo['Freq1'] **>** 0.2]['snp\_name']

feature\_name1 **=** sim\_n300\_p1000\_mlinfo[sim\_n300\_p1000\_mlinfo['MAF'] **<** 0.9]['snp\_name']

compare **=** [i **for** i, j **in** zip(feature\_name, feature\_name1) **if** i **==** j]

print(len(feature\_name), len(feature\_name1), len(compare))



*#(3) 반응변수에 데이터 로딩*

pheno\_n300\_conti\_phe **=** pd**.**read\_table('data/pheno\_n300\_conti.phe',sep**=**'\t', header**=None**)

pheno\_n300\_conti\_phe**.**columns **=** key\_column **+** dependent\_column

pheno\_n150\_conti\_phe **=** pd**.**read\_table('data/pheno\_n150\_conti.phe',sep**=**'\t', header**=None**)

pheno\_n150\_conti\_phe**.**columns **=** key\_column **+** dependent\_column

print(pheno\_n300\_conti\_phe)

print(pheno\_n150\_conti\_phe)

fid iid response\_val\_1 response\_val\_2

0 1 1 -2.902394 -0.293452

1 2 2 -0.458173 -1.216158

2 3 3 1.054707 0.522783

3 4 4 0.323021 1.046023

4 5 5 0.139497 2.440253

.. ... ... ... ...

295 296 296 -0.217907 1.167729

296 297 297 -0.458625 -0.737111

297 298 298 1.217275 0.585818

298 299 299 0.637296 0.957087

299 300 300 -0.920817 -0.389378

[300 rows x 4 columns]

fid iid response\_val\_1 response\_val\_2

0 1 1 0.028128 0.719465

1 2 2 0.358462 -1.020880

2 3 3 0.603442 0.840852

3 4 4 1.653470 -1.606827

4 5 5 1.027787 1.210293

.. ... ... ... ...

145 146 146 2.009901 0.347298

146 147 147 1.695812 -0.947804

147 148 148 -0.977098 -0.281299

148 149 149 -0.331803 0.304020

149 150 150 -0.243834 -2.386709

[150 rows x 4 columns]

*#(4) 데이터 병합 및 훈련, 테스트 데이터 구별*

train\_data **=** pd**.**merge(sim\_n300\_p1000\_mldose, pheno\_n300\_conti\_phe, how**=**'inner', on **=**['fid', 'iid'])

test\_data **=** pd**.**merge(sim\_n150\_p1000\_mldose, pheno\_n150\_conti\_phe, how**=**'inner', on **=**['fid', 'iid'])

*# train\_data.to\_csv('train.csv')*

*# test\_data.to\_csv('test.csv')*

train\_x **=** train\_data[independent\_column]

train\_y **=** train\_data[dependent\_column]

test\_x **=** test\_data[independent\_column]

test\_y **=** test\_data[dependent\_column]

*#(5) 선형회귀로 선형관계를 확인 -> 비선형 관계*

*# import statsmodels.api as sm*

*# model = sm.OLS(train\_y['response\_val\_1'], train\_x)*

*# result = model.fit()*

*# result.summary()*

*#(6) 트리 모델에서 feature 중요도를 통한 feature selection을 진행하려 했으나 영향력이 적어* *모든 feature 대상으로 회귀 모델을 구성하려 함.*

etc\_model **=** ExtraTreesRegressor()

etc\_model**.**fit(train\_x, train\_y)

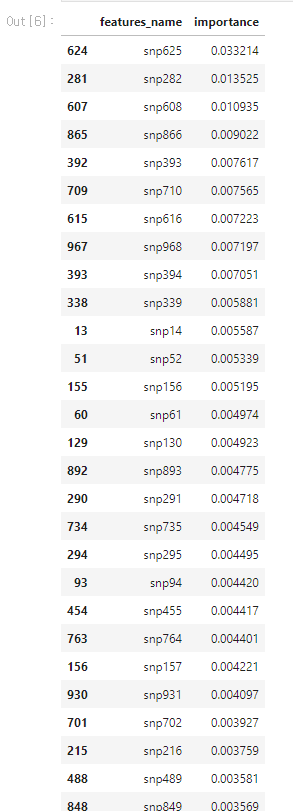
feature\_list **=** pd**.**concat([pd**.**Series(train\_x**.**columns), pd**.**Series(etc\_model**.**feature\_importances\_)], axis**=**1)

feature\_list**.**columns **=** ['features\_name', 'importance']

feature\_list**.**sort\_values("importance", ascending **=False**)[:50]

*# train\_x = train\_data[feature\_list]*

*# test\_x = test\_data[feature\_list]*

(생략)

*#(7) keras를 통한 NN Network 구성*

*# 반복 실험을 통해 hyper parameter는 최적의 결과로 도출 되는 layer를 이용*

model **=** Sequential()

model**.**add(Dense(100, input\_dim**=**len(train\_x**.**columns), activation**=**'relu'))

model**.**add(Dense(50, activation**=**'relu'))

model**.**add(Dense(25, activation**=**'relu'))

model**.**add(Dense(2))

model**.**compile(loss**=**'mean\_squared\_error', optimizer**=**'adam', metrics**=**['accuracy'])

model**.**summary()

callbacks **=** [tf**.**keras**.**callbacks**.**EarlyStopping(monitor**=**'val\_loss', patience**=**30)]

result **=** model**.**fit(train\_x, train\_y, validation\_split**=**0.25, epochs**=**100, batch\_size**=**10, callbacks**=**callbacks)

Model: "sequential\_13"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

dense\_48 (Dense) (None, 100) 100100

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_49 (Dense) (None, 50) 5050

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_50 (Dense) (None, 25) 1275

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_51 (Dense) (None, 2) 52

=================================================================

Total params: 106,477

Trainable params: 106,477

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Epoch 1/100

23/23 [==============================] - 1s 11ms/step - loss: 2.0502 - accuracy: 0.5389 - val\_loss: 0.9058 - val\_accuracy: 0.5733

Epoch 2/100

23/23 [==============================] - 0s 5ms/step - loss: 0.9970 - accuracy: 0.5612 - val\_loss: 0.9365 - val\_accuracy: 0.5733

Epoch 3/100

23/23 [==============================] - 0s 7ms/step - loss: 0.9027 - accuracy: 0.5973 - val\_loss: 0.9935 - val\_accuracy: 0.5200

Epoch 4/100

23/23 [==============================] - 0s 8ms/step - loss: 0.9482 - accuracy: 0.5467 - val\_loss: 0.9466 - val\_accuracy: 0.5200

Epoch 5/100

23/23 [==============================] - 0s 7ms/step - loss: 0.8019 - accuracy: 0.5929 - val\_loss: 0.9360 - val\_accuracy: 0.4933

Epoch 6/100

23/23 [==============================] - 0s 7ms/step - loss: 0.7997 - accuracy: 0.6505 - val\_loss: 0.8917 - val\_accuracy: 0.5867

Epoch 7/100

23/23 [==============================] - 0s 8ms/step - loss: 0.7410 - accuracy: 0.7658 - val\_loss: 0.9100 - val\_accuracy: 0.6000

Epoch 8/100

23/23 [==============================] - 0s 7ms/step - loss: 0.7473 - accuracy: 0.6957 - val\_loss: 1.0034 - val\_accuracy: 0.5600

Epoch 9/100

23/23 [==============================] - 0s 7ms/step - loss: 0.7486 - accuracy: 0.6012 - val\_loss: 1.0128 - val\_accuracy: 0.5733

Epoch 10/100

23/23 [==============================] - 0s 6ms/step - loss: 0.6342 - accuracy: 0.6707 - val\_loss: 0.9522 - val\_accuracy: 0.5600

Epoch 11/100

23/23 [==============================] - 0s 5ms/step - loss: 0.5506 - accuracy: 0.8509 - val\_loss: 1.0872 - val\_accuracy: 0.5867

Epoch 12/100

23/23 [==============================] - 0s 5ms/step - loss: 0.5072 - accuracy: 0.7433 - val\_loss: 0.9098 - val\_accuracy: 0.5600

Epoch 13/100

23/23 [==============================] - 0s 5ms/step - loss: 0.4617 - accuracy: 0.8066 - val\_loss: 0.9847 - val\_accuracy: 0.5867

Epoch 14/100

23/23 [==============================] - 0s 5ms/step - loss: 0.4279 - accuracy: 0.8317 - val\_loss: 1.0309 - val\_accuracy: 0.5733

Epoch 15/100

23/23 [==============================] - 0s 5ms/step - loss: 0.3841 - accuracy: 0.8143 - val\_loss: 0.9218 - val\_accuracy: 0.6133

Epoch 16/100

23/23 [==============================] - 0s 5ms/step - loss: 0.2854 - accuracy: 0.8794 - val\_loss: 0.9330 - val\_accuracy: 0.5733

Epoch 17/100

23/23 [==============================] - 0s 5ms/step - loss: 0.2913 - accuracy: 0.8339 - val\_loss: 0.9316 - val\_accuracy: 0.6000

Epoch 18/100

23/23 [==============================] - 0s 5ms/step - loss: 0.2757 - accuracy: 0.8348 - val\_loss: 0.9744 - val\_accuracy: 0.6133

Epoch 19/100

23/23 [==============================] - 0s 5ms/step - loss: 0.2040 - accuracy: 0.9030 - val\_loss: 1.0480 - val\_accuracy: 0.6533

Epoch 20/100

23/23 [==============================] - 0s 5ms/step - loss: 0.1830 - accuracy: 0.8641 - val\_loss: 0.9836 - val\_accuracy: 0.6400

Epoch 21/100

23/23 [==============================] - 0s 5ms/step - loss: 0.1860 - accuracy: 0.8691 - val\_loss: 1.0632 - val\_accuracy: 0.6267

Epoch 22/100

23/23 [==============================] - 0s 5ms/step - loss: 0.1971 - accuracy: 0.8886 - val\_loss: 0.9991 - val\_accuracy: 0.6533

Epoch 23/100

23/23 [==============================] - 0s 5ms/step - loss: 0.1120 - accuracy: 0.9265 - val\_loss: 1.2318 - val\_accuracy: 0.5867

Epoch 24/100

23/23 [==============================] - 0s 5ms/step - loss: 0.1455 - accuracy: 0.9251 - val\_loss: 1.0247 - val\_accuracy: 0.6533

Epoch 25/100

23/23 [==============================] - 0s 5ms/step - loss: 0.0885 - accuracy: 0.9316 - val\_loss: 1.0673 - val\_accuracy: 0.6267

Epoch 26/100

23/23 [==============================] - 0s 6ms/step - loss: 0.0959 - accuracy: 0.8667 - val\_loss: 1.0958 - val\_accuracy: 0.6533

Epoch 27/100

23/23 [==============================] - 0s 5ms/step - loss: 0.0650 - accuracy: 0.9371 - val\_loss: 1.1061 - val\_accuracy: 0.6400

Epoch 28/100

23/23 [==============================] - 0s 5ms/step - loss: 0.0725 - accuracy: 0.9416 - val\_loss: 1.1685 - val\_accuracy: 0.5867

Epoch 29/100

23/23 [==============================] - 0s 5ms/step - loss: 0.0634 - accuracy: 0.9340 - val\_loss: 1.0942 - val\_accuracy: 0.6667

Epoch 30/100

23/23 [==============================] - 0s 5ms/step - loss: 0.0681 - accuracy: 0.9628 - val\_loss: 1.1234 - val\_accuracy: 0.6400

Epoch 31/100

23/23 [==============================] - 0s 5ms/step - loss: 0.0409 - accuracy: 0.9490 - val\_loss: 1.1173 - val\_accuracy: 0.6400

Epoch 32/100

23/23 [==============================] - 0s 5ms/step - loss: 0.0365 - accuracy: 0.9230 - val\_loss: 1.1840 - val\_accuracy: 0.5867

Epoch 33/100

23/23 [==============================] - 0s 6ms/step - loss: 0.0435 - accuracy: 0.9277 - val\_loss: 1.1037 - val\_accuracy: 0.6667

Epoch 34/100

23/23 [==============================] - 0s 5ms/step - loss: 0.0180 - accuracy: 0.9655 - val\_loss: 1.1774 - val\_accuracy: 0.6400

Epoch 35/100

23/23 [==============================] - 0s 5ms/step - loss: 0.0261 - accuracy: 0.9452 - val\_loss: 1.1828 - val\_accuracy: 0.6533

Epoch 36/100

23/23 [==============================] - 0s 5ms/step - loss: 0.0254 - accuracy: 0.9429 - val\_loss: 1.1001 - val\_accuracy: 0.6400

*# (8) 가시화*

**def** vis(history, name) :

plt**.**title(f"{name**.**upper()}")

plt**.**xlabel('epochs')

plt**.**ylabel(f"{name**.**lower()}")

value **=** history**.**history**.**get(name)

val\_value **=** history**.**history**.**get(f"val\_{name}",**None**)

epochs **=** range(1, len(value)**+**1)

plt**.**plot(epochs, value, 'b-', label**=**f'training {name}')

**if** val\_value **is** **not** **None** :

plt**.**plot(epochs, val\_value, 'r:', label**=**f'validation {name}')

plt**.**legend(loc**=**'upper center', bbox\_to\_anchor**=**(0.05, 1.2) , fontsize**=**10 , ncol**=**1)

**def** plot\_history(history) :

key\_value **=** list(set([i**.**split("val\_")[**-**1] **for** i **in** list(history**.**history**.**keys())]))

plt**.**figure(figsize**=**(12, 4))

**for** idx , key **in** enumerate(key\_value) :

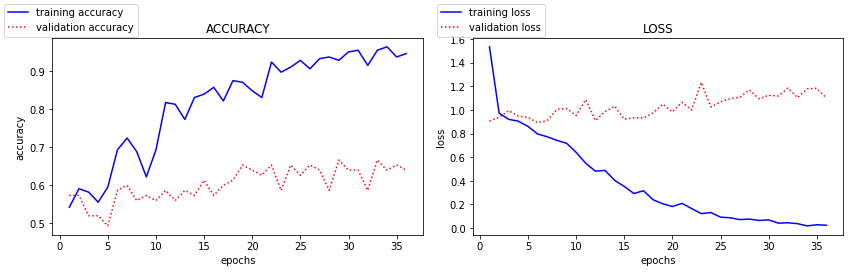
plt**.**subplot(1, len(key\_value), idx**+**1)

vis(history, key)

plt**.**tight\_layout()

plt**.**show()

plot\_history(result)



*#(9) test set으로 검증 시 정확도는 50~60% 사이로 나옴.*

score **=** model**.**evaluate(test\_x, test\_y, batch\_size**=**10)

print('loss and metrics:', score)

15/15 [==============================] - 0s 2ms/step - loss: 1.1838 - accuracy: 0.5933

loss and metrics: [1.1838032007217407, 0.5933333039283752]

*#(10) test set 시각화*

prediction **=** model**.**predict(test\_x)**.**flatten()

plt**.**plot(test\_y**.**values**.**flatten())

plt**.**plot(prediction)

plt**.**xlabel('True Values')

plt**.**ylabel('Predictions')

plt**.**show()

