# MoA Prediction

한현수

## 1. Problem

In this competition, you will have access to a unique dataset that combines gene expression and cell viability data. The data is based on a new technology that measures simultaneously (within the same samples) human cells' responses to drugs in a pool of 100 different cell types (thus solving the problem of identifying ex-ante, which cell types are better suited for a given drug). In addition, you will have access to MoA annotations for more than 5,000 drugs in this dataset.

As is customary, the dataset has been split into testing and training subsets. Hence, your task is to use the training dataset to develop an algorithm that automatically labels each case in the test set as one or more MoA classes. Note that since drugs can have multiple MoA annotations, the task is formally a multi-label classification problem.

### 1) Data Fields

- g-feature: 유전자 발현 데이터
- c-feature: 세포 생존 가능성(능력) 데이터
- cp_type: 화합물로 처리된 표본 (cp_type 이 ctl_vehicle 일 때 MoAs 는 항상 0 (어떻게 처리할지, 행을 삭제할지 생각해야함))
- cp_time: 치료기간 (24 시간, 48 시간, 72 시간)
- cp_dose: 복용량

## 2. Kaggle 필사

Import packge

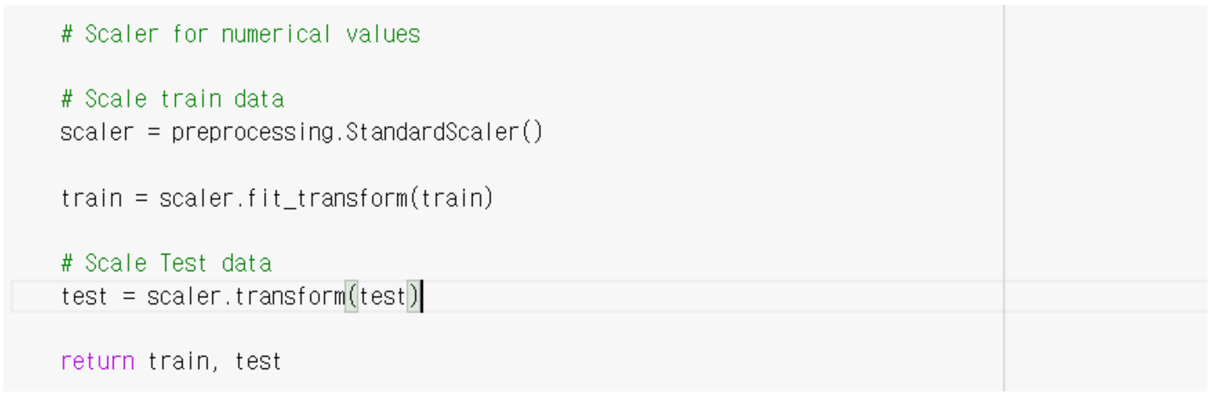
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix
from sklearn.feature_selection import VarianceThreshold
from sklearn.decomposition import PCA
from tensorflow.keras import layers,regularizers,Sequential,Model,backend,callbacks,optimizers,metrics,losses
import tensorflow as tf
import sys
import json
from iterstrat.ml_stratifiers import MultilabelStratifiedKFold
```

Data preprocessing

```
non_ctl_idx = train_features.loc[train_features['cp_type']!='ctl_vehicle'].index.to_list()
train_features = train_features.drop(['sig_id','cp_type','cp_dose','cp_time'],axis=1)
train_targets = train_targets.drop('sig_id',axis=1)
labels_train = train_targets.values

train_features = train_features.iloc[non_ctl_idx]
labels_train = labels_train[non_ctl_idx]

test_features = test_features.drop(['sig_id','cp_dose','cp_time'],axis=1)

json_file_path = '/content/main_predictors.json'

with open(json_file_path, 'r') as j:
    predictors = json.loads(j.read())
    predictors = predictors['start_predictors']
```

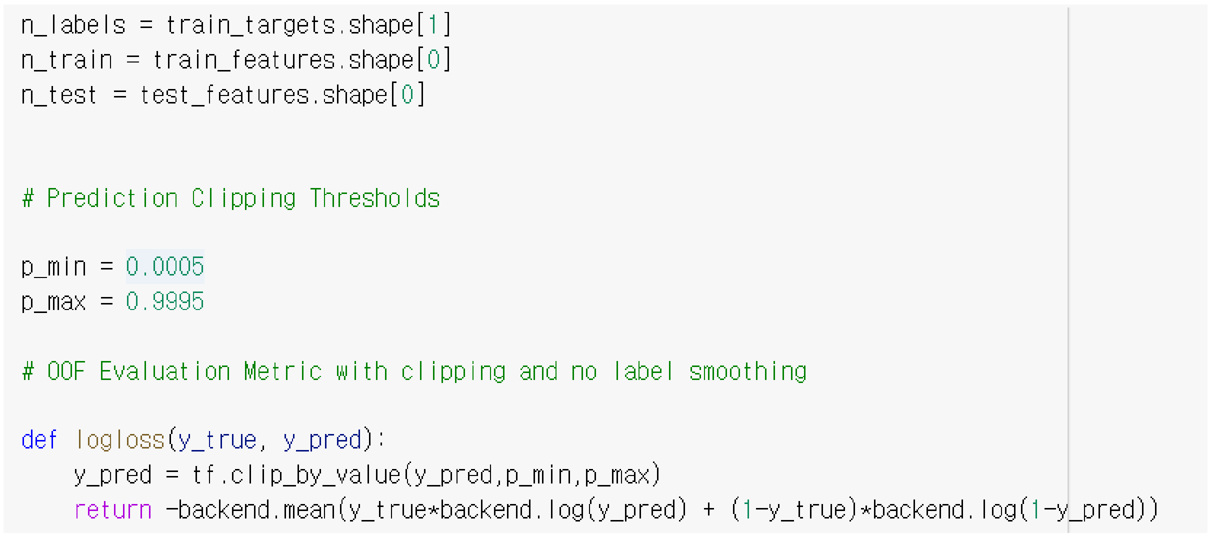
여기선 cp_type 이 ctl_vehicle 인 경우를 제거하고 cp_type, cp_dose ,cp_time 을 제거한다.

```
cs = train_features.columns.str.contains('c-')
gs = train_features.columns.str.contains('g-')

def preprocessor(train,test):

    # PCA

    n_gs = 2 # No of PCA comps to include
    n_cs = 100 # No of PCA comps to include

    pca_cs = PCA(n_components = n_cs)
    pca_gs = PCA(n_components = n_gs)

    train_pca_gs = pca_gs.fit_transform(train[:,gs])
    train_pca_cs = pca_cs.fit_transform(train[:,cs])
    test_pca_gs = pca_gs.transform(test[:,gs])
    test_pca_cs = pca_cs.transform(test[:,cs])

    # c-mean, g-mean

    train_c_mean = train[:,cs].mean(axis=1)
    test_c_mean = test[:,cs].mean(axis=1)
    train_g_mean = train[:,gs].mean(axis=1)
    test_g_mean = test[:,gs].mean(axis=1)

    # Append Features

    train = np.concatenate((train,train_pca_gs,train_pca_cs,train_c_mean[:,np.newaxis]
                            ,train_g_mean[:,np.newaxis]),axis=1)
    test = np.concatenate((test,test_pca_gs,test_pca_cs,test_c_mean[:,np.newaxis],
                           test_g_mean[:,np.newaxis]),axis=1)
```



```
    # Scaler for numerical values

    # Scale train data
    scaler = preprocessing.StandardScaler()

    train = scaler.fit_transform(train)

    # Scale Test data
    test = scaler.transform(test)

    return train, test
```

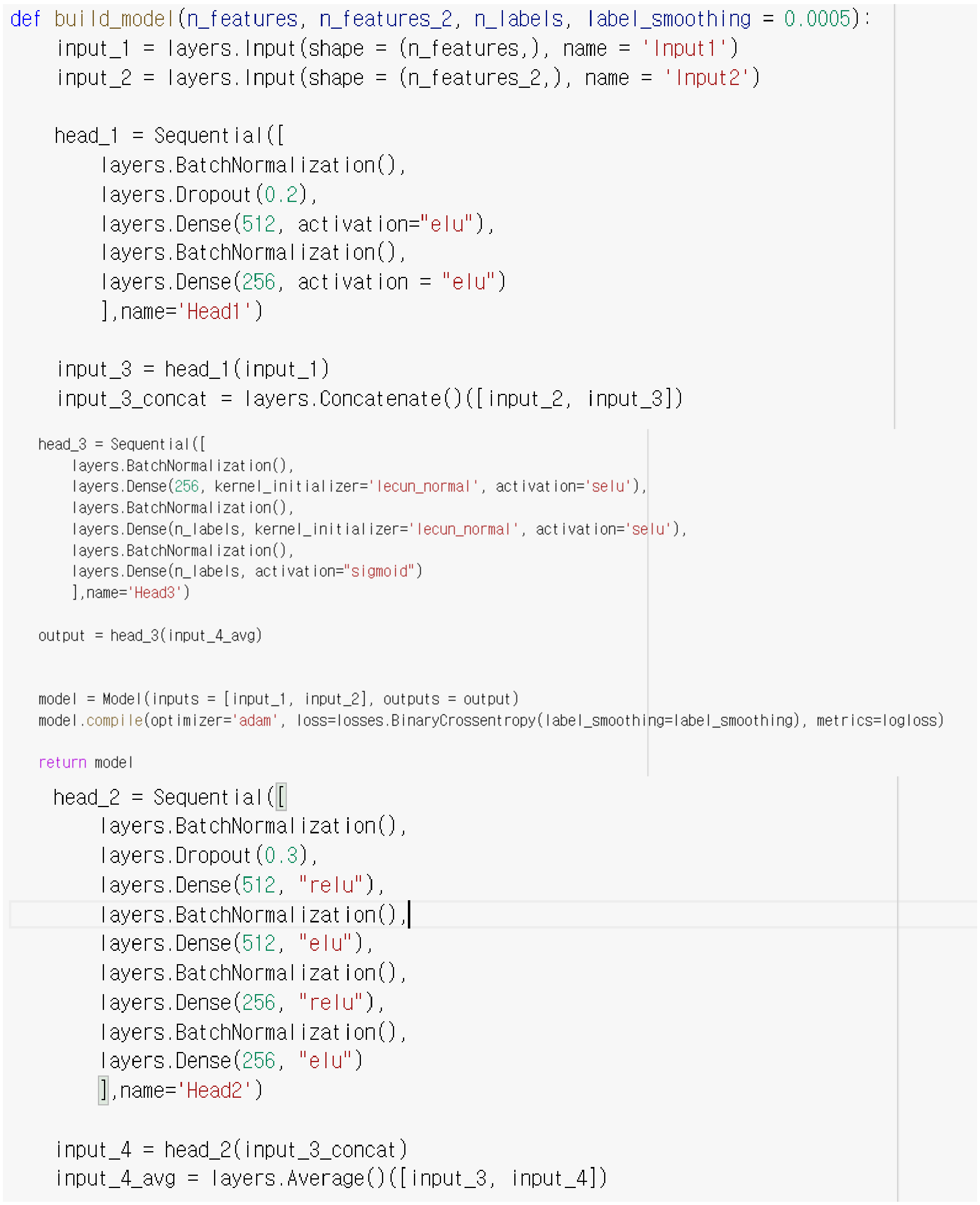
여기까지는 g-feature, c-feature 를 PCA 를 이용해 각각 2, 100 으로 feature 를 줄인다. 또한 feature 를 g-feature(772) , c-feature(100), PCA_g(2), PCA_c(100), g-mean, c-mean 로 총 976 개로 이후 모델의 input1 이 된다.



```
n_labels = train_targets.shape[1]
n_train = train_features.shape[0]
n_test = test_features.shape[0]


# Prediction Clipping Thresholds

p_min = 0.0005
p_max = 0.9995

# OOF Evaluation Metric with clipping and no label smoothing

def logloss(y_true, y_pred):
    y_pred = tf.clip_by_value(y_pred,p_min,p_max)
    return -backend.mean(y_true*backend.log(y_pred) + (1-y_true)*backend.log(1-y_pred))
```

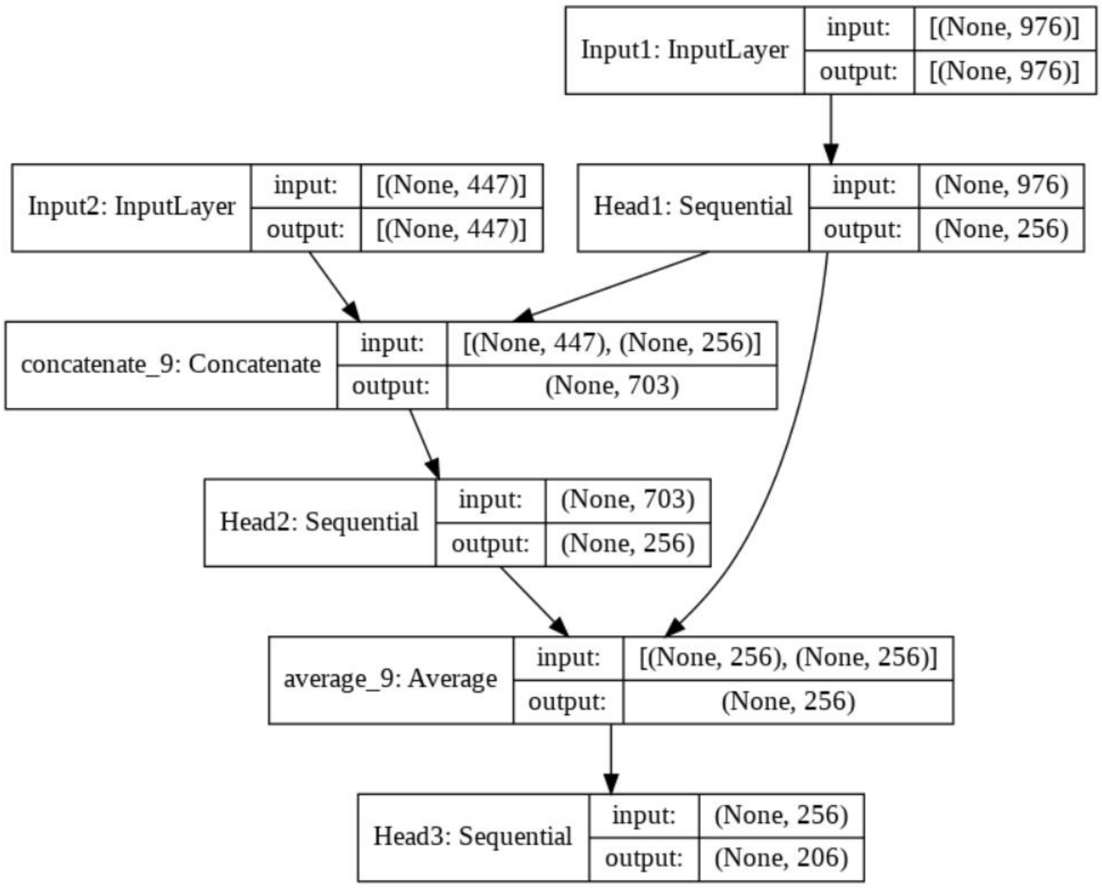
이는 oof evaluation 을 위한 logloss function 이다.

Model



```python
def build_model(n_features, n_features_2, n_labels, label_smoothing = 0.0005):
    input_1 = layers.Input(shape = (n_features,), name = 'Input1')
    input_2 = layers.Input(shape = (n_features_2,), name = 'Input2')

    head_1 = Sequential([
        layers.BatchNormalization(),
        layers.Dropout(0.2),
        layers.Dense(512, activation="elu"),
        layers.BatchNormalization(),
        layers.Dense(256, activation = "elu")
        ],name='Head1')

    input_3 = head_1(input_1)
    input_3_concat = layers.Concatenate()([input_2, input_3])

    head_3 = Sequential([
        layers.BatchNormalization(),
        layers.Dense(256, kernel_initializer='lecun_normal', activation='selu'),
        layers.BatchNormalization(),
        layers.Dense(n_labels, kernel_initializer='lecun_normal', activation='selu'),
        layers.BatchNormalization(),
        layers.Dense(n_labels, activation="sigmoid")
        ],name='Head3')

    output = head_3(input_4_avg)


    model = Model(inputs = [input_1, input_2], outputs = output)
    model.compile(optimizer='adam', loss=losses.BinaryCrossentropy(label_smoothing=label_smoothing), metrics=logloss)

    return model
    head_2 = Sequential([
        layers.BatchNormalization(),
        layers.Dropout(0.3),
        layers.Dense(512, "relu"),
        layers.BatchNormalization(),
        layers.Dense(512, "elu"),
        layers.BatchNormalization(),
        layers.Dense(256, "relu"),
        layers.BatchNormalization(),
        layers.Dense(256, "elu")
        ],name='Head2')

    input_4 = head_2(input_3_concat)
    input_4_avg = layers.Average()([input_3, input_4])
```

Model Summary



이 모델은 두개의 input 을 가진 Resnet 구조이다. Input1 에선 g-feature(772) , c-feature(100), PCA_g(2), PCA_c(100), g-mean, c-mean 로 총 976 개이고, Input 2 에선 p-value 0.0.1 이하의 predictor 로 447 개이다.

predictor 참고:https://www.kaggle.com/demetrypascal/t-test-pca-rfe-logistic-regression/output

## Train

```
n_seeds = 7
np.random.seed(1)
seeds = np.random.randint(0,100,size=n_seeds)
7*10
# Training Loop

n_folds = 10
y_pred = np.zeros((n_test,n_labels))
oof = tf.constant(0.0)
hists = []
for seed in seeds:
    fold = 0
    kf = KFold(n_splits=n_folds,shuffle=True,random_state=seed)
    for train, test in kf.split(train_features):
        X_train, X_test = preprocessor(train_features.iloc[train].values,
                                       train_features.iloc[test].values)
        _,data_test = preprocessor(train_features.iloc[train].values,
                                   test_features.drop('cp_type',axis=1).values)
        X_train_2 = train_features.iloc[train][predictors].values
        X_test_2 = train_features.iloc[test][predictors].values
        data_test_2 = test_features[predictors].values
        y_train = labels_train[train]
        y_test = labels_train[test]
        n_features = X_train.shape[1]
        n_features_2 = X_train_2.shape[1]
```

```
        model = build_model(n_features, n_features_2, n_labels)

        reduce_lr = callbacks.ReduceLROnPlateau(monitor='val_logloss', factor=0.1, patience=2, mode='min', min_lr=1E-5)
        early_stopping = callbacks.EarlyStopping(monitor='val_logloss', min_delta=1E-5, patience=10, mode='min',restore_best_weights=True)

        hist = model.fit([X_train,X_train_2],y_train, batch_size=128, epochs=192,verbose=0,validation_data = ([X_test,X_test_2],y_test),
                         callbacks=[reduce_lr, early_stopping])
        hists.append(hist)

        # Save Model
        model.save('TwoHeads_seed_'+str(seed)+'_fold_'+str(fold))
        print('Seed_'+str(seed)+'_fold_'+str(fold)+'_var_logloss_'+str(hist.history['val_logloss'][-1]))

        # OOF Score
        y_val = model.predict([X_test,X_test_2])
        oof += logloss(tf.constant(y_test,dtype=tf.float32),tf.constant(y_val,dtype=tf.float32))/(n_folds*n_seeds)

        # Run prediction
        y_pred += model.predict([data_test,data_test_2])/(n_folds*n_seeds)

        fold += 1
```

학습은 random 의 seed 값을 7 개를 정하여 각 seed 값 별로 10-fold 검증을 실시한 후 각 모델을 저장한다. 이때 callback 함수로 reduce lr, earlystopping 을 이용한다. 이렇게 검증한 결과 OOF score 는 0.0165592507 이 나왔다.