How to create model

1. the result of DNM-RF stage 3-4:

Graphical user interface

Description automatically generated

The result is in " keras-main\DNM-RF\stage3-4\15 year\_top10\_validation\_retrieved.csv". Similar files for 10 years and 5 years can also be found in the same folder.

It is the result of best model and the column is the parameter needed by the model. We only need the first row as input parameter of creating model.

1. the code of creating model:

def create\_model(mstruct, idim, drate, kinit, iacti, hacti, oacti, opti, lrate, momen ,dec,ls, L1, L2,ltype):  
 # create a model that KerasClassifier needs as an input for parameter build\_fn  
 model = Sequential()  
 if ltype==0:  
 model.add(Dense(mstruct[0], input\_dim=idim, kernel\_initializer = kinit, activation= iacti))  
 elif ltype==1:  
 model.add(Dense(mstruct[0], input\_dim=idim, kernel\_initializer = kinit, activation= iacti,kernel\_regularizer=regularizers.l1(L1)))  
 elif ltype ==2:  
 model.add(Dense(mstruct[0], input\_dim=idim, kernel\_initializer = kinit, activation= iacti,kernel\_regularizer=regularizers.l2(L2)))  
 elif ltype ==3:  
 model.add(Dense(mstruct[0], input\_dim=idim, kernel\_initializer = kinit, activation= iacti,kernel\_regularizer=regularizers.l1\_l2(l1=L1, l2=L2)))  
  
  
 model.add(Dropout(drate))  
 nlayers = len(mstruct)  
 nhiddenlayers =nlayers -2  
 for i in range(nhiddenlayers):  
 model.add(Dense(mstruct[i+1], activation=hacti))  
 model.add(Dropout(drate))  
 model.add(Dense(mstruct[nlayers-1], activation=oacti))  
 # Using 'softmax' as the activation function for the output layer will return all 0.5s when class is binary  
  
  
 cur\_opt = opti  
 if opti == 'Adagrad':  
 cur\_opt = opt.Adagrad(lr = lrate, decay=dec)  
 elif opti == 'SGD':  
 cur\_opt = opt.SGD(lr = lrate, momentum=momen, decay=dec)  
 model.compile(optimizer=cur\_opt, loss=ls, metrics = "accuracy")  
 return model  
  
  
model\_15 = create\_model((84,96,124,62,1),17,0,"glorot\_normal","relu","relu","sigmoid",'Adagrad',0.1,0.1,0.0005,"binary\_crossentropy",0.002,0.005,3)

input: the parameters derived from 5 year\_top10\_validation\_retrieved.csv. You can also create model for 10 and 5 year using corresponding parameters.

output: the multilayer classifier as a keras model

1. load dataset:

filename = os.path.join("C:/Users/jiny1/PycharmProjects/keras-main/keras-main/DNM-RF/dataset/alpha240", "LSM-15Year-I-240.txt")  
predset, target=loadandprocess(filename, predtype=1, scaled=False)

use the path of the dataset as filename variable. The other two year’s datasets are named in the same format.

Call the function ‘loadandprocess’. this function is used to divide the original data into train features(predictors) X and labels Y For trainset X:

if we set scaled = True, then all the train features will be normalized(scale to [0,1]); if we set predtype = 1, it means the last column features is target, others are predictors if we set predtype = 2, it means the first column is index(we could ignore) and the last column features is target, others are predictors

the code for loadandprocess:

def loadandprocess(file, sep='\t', predtype=1, scaled=True):  
print(file)  
 df = pd.read\_csv(file, sep, lineterminator='\n')  
 # cols=[0,532]  
 # predset = df.drop(df.columns[cols],axis=1)  
 if predtype == 1:  
 X = df.iloc[:, :-1] # all columns except for the last one are predictors  
 elif predtype == 2:  
 X = df.iloc[:, 1:-1] # all columns except for the first and last ones are predictors  
 # If scaled is true, Normalized to [0,1]. Default is true.  
 if scaled:  
 scaler = MinMaxScaler()  
 scaler.fit(predset)  
 X = scaler.transform(X)  
  
 print(f'pred shape: {X.shape}')  
 print(f'pred dimension: {X.ndim}')  
 # tarcol2 = n.array(df.columns[-1])  
 Y = df.iloc[:, -1]  
 print(Y.head(4))  
 print(f'target frame dimension: {Y.ndim}')  
 print(f'target frame shape: {Y.shape}')  
 Y = Y.to\_numpy()  
 print(f'target dimension: {Y.ndim}')  
 X = X.to\_numpy()  
 # if have a problem"numpy.ndarray' object has no attribute 'to\_numpy" when scale = True,  
 # you can just comment"predset = predset.to\_numpy()" because predset = scaler.transform(predset) will return numpy object directly  
 # which we want to make a prediction  
 return X,Y

1. fit the model with predictors and target:

model\_15.fit(predset,target,epochs=83)

the epochs parameter is also from 15 year\_top10\_validation\_retrieved.csv.

training progress:

Text

Description automatically generated

1. save the model as a .h5 file

model\_15.save("model15.h5")

the file containing model information will be saved in the same directory as the python file.

1. Load the model when using:

Copy the file to the directory of imedbot and load at file application.py

model\_15 = load\_model('model15.h5')

1. The whole code of creating model:

from tensorflow.keras.layers import Dense  
from tensorflow.keras.layers import Dropout  
from tensorflow.keras.models import Sequential  
from keras.wrappers.scikit\_learn import KerasClassifier  
import tensorflow.keras.optimizers as opt  
from tensorflow.keras import regularizers  
import numpy as np  
import os  
from utils.input import loadandprocess  
  
  
def create\_model(mstruct, idim, drate, kinit, iacti, hacti, oacti, opti, lrate, momen ,dec,ls, L1, L2,ltype):  
 # create a model that KerasClassifier needs as an input for parameter build\_fn  
 model = Sequential()  
 if ltype==0:  
 model.add(Dense(mstruct[0], input\_dim=idim, kernel\_initializer = kinit, activation= iacti))  
 elif ltype==1:  
 model.add(Dense(mstruct[0], input\_dim=idim, kernel\_initializer = kinit, activation= iacti,kernel\_regularizer=regularizers.l1(L1)))  
 elif ltype ==2:  
 model.add(Dense(mstruct[0], input\_dim=idim, kernel\_initializer = kinit, activation= iacti,kernel\_regularizer=regularizers.l2(L2)))  
 elif ltype ==3:  
 model.add(Dense(mstruct[0], input\_dim=idim, kernel\_initializer = kinit, activation= iacti,kernel\_regularizer=regularizers.l1\_l2(l1=L1, l2=L2)))  
  
  
 model.add(Dropout(drate))  
 nlayers = len(mstruct)  
 nhiddenlayers =nlayers -2  
 for i in range(nhiddenlayers):  
 model.add(Dense(mstruct[i+1], activation=hacti))  
 model.add(Dropout(drate))  
 model.add(Dense(mstruct[nlayers-1], activation=oacti))  
 # Using 'softmax' as the activation function for the output layer will return all 0.5s when class is binary  
  
  
 cur\_opt = opti  
 if opti == 'Adagrad':  
 cur\_opt = opt.Adagrad(lr = lrate, decay=dec)  
 elif opti == 'SGD':  
 cur\_opt = opt.SGD(lr = lrate, momentum=momen, decay=dec)  
 model.compile(optimizer=cur\_opt, loss=ls, metrics = "accuracy")  
 return model  
  
  
model\_15 = create\_model((84,96,124,62,1),17,0,"glorot\_normal","relu","relu","sigmoid",'Adagrad',0.1,0.1,0.0005,"binary\_crossentropy",0.002,0.005,3)  
  
filename = os.path.join("C:/Users/jiny1/PycharmProjects/keras-main/keras-main/DNM-RF/dataset/alpha240", "LSM-15Year-I-240.txt")  
predset, target=loadandprocess(filename, predtype=1, scaled=False)  
model\_15.fit(predset,target,epochs=83)  
model\_15.save("model15.h5")  
  
model\_10 = create\_model((85,151,51,193,1),18,0.15,"glorot\_normal","relu","relu","sigmoid",'Adagrad',0.07,0.4,0,"binary\_crossentropy",0.0035,0,3)  
input = [[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]]  
input = np.array(input)  
print(input)  
filename = os.path.join("C:/Users/jiny1/PycharmProjects/keras-main/keras-main/DNM-RF/dataset/alpha240", "LSM-10Year-I-240.txt")  
predset, target=loadandprocess(filename, predtype=1, scaled=False)  
model\_10.fit(predset,target,epochs=110)  
model\_10.save("model10.h5")  
res = model\_10.predict(input)  
print(res)  
  
model\_5 = create\_model((333,218,294,218,1),20,0,"glorot\_normal","relu","relu","sigmoid",'Adagrad',0.03,0.1,0,"binary\_crossentropy",0.005,0,3)  
input = [[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]]  
input = np.array(input)  
print(input)  
filename = os.path.join("C:/Users/jiny1/PycharmProjects/keras-main/keras-main/DNM-RF/dataset/alpha240", "LSM-5Year-I-240.txt")  
predset, target=loadandprocess(filename, predtype=1, scaled=False)  
model\_5.fit(predset,target,epochs=160)  
model\_5.save("model5.h5")  
res = model\_5.predict(input)  
print(res)