Chapter 13: Part 5

**Deep Learning**

**SOM: Hybrid Model**

Python Implementation

**13.5.1 Problem description**

Here we'll make a Hybrid Deep Learning Model. We'll combine two deep learning models ANN and SOM (supervised and unsupervised).

* Our data-set will be, the *credit card applications* dataset to identify the *frauds*.
* The idea is to make an even more *advanced deep learning model* where we can *predict* the *probability* that each customer *cheated*.
* It takes two parts:
* In the first part, we'll make the unsupervised deep learning branch of our hybrid deep learning model using SOM.
* And in the second part, we'll make the supervised deep learning branch using ANN.
* And, in the end, we'll get this *hybrid deep learning model* composed of both *unsupervised* and *supervised* deep learning.
* We will use the *self-organizing map* exactly as we did *previously*, to identify the *fraud*. So, there will be nothing new about that.
* But then, the *challenge* is to use the *results* of this *self-organizing* map to *ANN* model. *ANN* will take, as *input*, the *results* given by your *SOM*.
* The challenge is to *combine two models*. In the end, what you must obtain is a *ranking* of the *predicted probabilities* that each customer *cheated*.

Note that, you will get very *small* *probabilities*, that's normal. It's because there are few *frauds* identified by the *SOM*, but that *doesn't matter*. What's important is that you get this *ranking* of *probabilities*.

**13.5.2 SOM part**

* We run the SOM code to get the map. Execute all the code from beginning to ***show()*** method.
* After obtaining the map, we choose a *threshold* for *outlier neuron* and we find and select those neurons.
* We then *concatenate* those data.
* Note that we execute the following code after selecting the outlying neurons. ***(8, 6)*** , ***(1, 7)*** are just for example, yours could be different.

# *finding the Fruds*

mappins = **som.win\_map**(X) # *not 'x' its our dataset "X", capital X*

fraud\_list\_1 = mappins[(8, 6)]

fraud\_list\_2 = mappins[(1, 7)]

frauDs = **np.concatenate**((mappins[(8, 6)], mappins[(1, 7)]), axis=0)

frauDs = **sc.inverse\_transform**(frauDs)

**13.5.3 ANN part & Prediction**

Creating the Feature matrix: We'll take all the columns except first column, i.e. ***customer\_id***.

* Note: the last column *approved/rejected* is not a dependent variable here (it is done by the *Bank* they don’t know who is fraud). We create the *dependent* variable from *SOM output*.

# *creating the matrix of features*

cuStomers = dataset.iloc[:, 1:].values

* Creating the dependent variable for ANN: This dependent variable is about fraud, ***0 = no fraud*** and ***1 = fraud***.
* Since from SOM we extracted some frauds, we can use them as our dependent variable.
* We first create a vector of **690 0**'s i.e. at first we assume that there is no cheaters/fraud. Then we *generate* 1 for the Id's that *falls* into our *fraud list* from *SOM*.
* In a loop we check each customer that are inside this list of fraud.
* ***dataset.iloc[i, 0]*** means ***i***th row and 1st column (i.e. ***customer\_id***), we don’t need ***.values***, it used for ***NumPy array***.

# *creating the dependent variable*

is\_fraud = **np.zeros**(len(dataset)) # *creating 690 size vector of zeros*

**for** i **in** **range**(len(dataset)):

**if** dataset.iloc[i, 0] **in** frauDs:

        is\_fraud[i]=1

* Creating ANN model:
* Feature scaling: We replace ***X\_train*** by ***cuStomers*** and remove X\_test.

# *------------------ Feature-Scaling ----------------------*

**from** sklearn.preprocessing **import** StandardScaler

# *y dependent variable, need not to be scaled: categorical variable, 0 and 1*

st\_x= **StandardScaler**()

cuStomers= **st\_x.fit\_transform**(cuStomers)    # *replace "X\_train" by 'cuStomers'*

* We remove *2nd hidden-layer* (we keep our model simple for learning purpose, however in general this model could be far more complex.)
* We set ***units = 2*** instead of ***6*** (no. of neurons).
* Since we have ***15*** columns, so our ***input\_dim = 15***.
* Also in ***ann\_classifier.fit()*** we use as feature-matrix and as output/dependent variable/data. We also change the ***batch\_size*** and no. of ***epochs***

ann\_classifier.fit(**cuStomers**, **is\_fraud**, **batch\_size**= 1, **epochs**= 2)

Note, do not train your *Deep Learning models* in *too many epochs* if you have *few observations* and *few features*.

# *-----------------------------------  Creating ANN model ---------------------------------------------*

    # *1. importing "keras" libraries and packages*

# *from tensorflow import keras*

**import** keras # *using TensorFlow backend*

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

**from** keras.layers **import** Dense

    # *2. initialize the ANN*

ann\_classifier = **Sequential**()

    # *3. Add the "input-layer" and  "first Hidden-layer"*

# *ann\_classifier.add(Dense(output\_dim = 6, init = "uniform", activation = "relu", input\_dim = 11))*

**ann\_classifier.add**(**Dense**(units = 2, kernel\_initializer = "uniform", activation = "relu", input\_dim = 15))

    # *4. Add the "second Hidden-layer"*

# *ann\_classifier.add(Dense(units = 6, kernel\_initializer = "uniform", activation = "relu"))*

    # *5. Add the "output-layer"*

**ann\_classifier.add**(**Dense**(units = 1, kernel\_initializer = "uniform", activation = "sigmoid"))

    # *6. Compile the ANN*

**ann\_classifier.compile**(optimizer = "adam", loss = "binary\_crossentropy", metrics= ["accuracy"])

    # *7. Train the model: fit the ANN to Training-set (batch\_size and epoch)*

**ann\_classifier.fit**(cuStomers, is\_fraud, batch\_size= 1, epochs= 2)

# *----------------------------------- Part 3 : Predictions  ---------------------------------------------*

# *Predicting the probabilities of frauds*

y\_prd = **ann\_classifier.predict**(cuStomers)

**13.5.4 Ranking the customers**

* We will sort these probabilities. But before that, we add ***customer\_id*** to ***y\_pred*** because it will be easy to identify the fraud.
* It will be ***2D*** ***array*** of 1st column hold the ***ids*** and ***2nd column*** is probabilities of being a fraud.
* Concatenate the columns:



* Notice ***y\_pred*** is a NumPy array of float, of size ***(690, 1)***. So to concatenate the ***customer\_id***, it need to be a ***NumPy*** ***array*** of float.
* ***dataset.iloc[:, 0:1].values***, selects the 1st column i.e. ***customer\_id*** from the dataset. ***0:1*** and ***.values*** is used to convert it to *NumPy array*.
* Also we changed the axis. Since we now concatenate side-by-side, we set ***axis=1.***

# *concatenate columns to add customer\_id*

y\_prd = **np.concatenate**((dataset.iloc[:, 0:1].values, y\_prd), axis=1)

* Now sorting is bit-Tricky, because ***NumPy*** will ***sort*** both columns. We don't want that, we just want to ***sort*** 2nd column of ***y\_pred***, i.e. the probabilities only.
* In *Excel* it is *easy*, the ids always *stay paired* if we *sort* the *probability*. In NumPy it is different.

# *sorting trick*

y\_prd = y\_prd[y\_prd[:, 1].**argsort**()] # *y\_prd[:, 1] selects the 2nd column of y\_pred*

* ***y\_prd[:, 1]*** selects the 2nd column of ***y\_pred***.

|  |  |
| --- | --- |
| SOM | Prediction probabilities |
|  |  |

# *---------------------------- Part 3 : Predictions  -----------------------------------*

# *Predicting the probabilities of frauds*

y\_prd = **ann\_classifier.predict**(cuStomers)

# *concatenate columns to add customer\_id*

y\_prd = **np.concatenate**((dataset.iloc[:, 0:1].values, y\_prd), axis=1)

# *sorting trick*

y\_prd = y\_prd[y\_prd[:, 1].**argsort**()] # *y\_prd[:, 1] selects the 2nd column of y\_pred*

**All code at once (practiced version)**

# *--------- mega case study : Make a Hybrid Deep learning model ----------*

# *importing libraries*

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

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

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

# *-------------- Data Preprocessing -------------------*

# *importing the Data-set*

dataset = **pd.read\_csv**("Credit\_Card\_Applications.csv")

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

#*feature scaling*

**from** sklearn.preprocessing **import** MinMaxScaler

sc = **MinMaxScaler**(feature\_range= (0, 1))

X = **sc.fit\_transform**(X)

# *------ Part 1 : Identify the frauds using Self Organizing Maps (SOM) ---------*

# *Training the SOM*

**from** minisom **import** MiniSom

som = **MiniSom**(x=10, y=10, input\_len=15, sigma=1.0, learning\_rate=0.5)

**som.random\_weights\_init**(X)

**som.train\_random**(data=X, num\_iteration=100)

# *Visualize the result*

**from** pylab **import** bone, pcolor, colorbar, plot, show

**bone**()

**pcolor**(**som.distance\_map**().T)

**colorbar**()

marKers = ['o', 's']

coLors = ['r', 'g']

**for** i, x **in** **enumerate**(X):

    w= **som.winner**(x)

**plot**(

        w[0]+0.5,

        w[1]+0.5,

        marKers[y[i]],

        markeredgecolor = coLors[y[i]],

        markerfacecolor = 'None',

        markersize = 10,

        markeredgewidth = 2)

**show**()

# *finding the Fruds*

mappins = **som.win\_map**(X) # *not 'x' its our dataset "X", capital X*

fraud\_list\_1 = mappins[(5, 5)]

fraud\_list\_2 = mappins[(6, 4)]

frauDs = **np.concatenate**((mappins[(5, 5)], mappins[(6, 4)]), axis=0)

frauDs = **sc.inverse\_transform**(frauDs)

# *saving the list to a csv*

# *save numpy array as csv file*

"""

from numpy import asarray

from numpy import savetxt

# define data

data = frauDs

# save to csv file

savetxt('frauds.csv', data, delimiter=',')

"""

# *part 2: Going from Unsupervised to Supervised Deep Learning*

# *creating the matrix of features*

cuStomers = dataset.iloc[:, 1:].values

# *creating the dependent variable*

is\_fraud = **np.zeros**(len(dataset)) # *creating 690 size vector of zeros*

**for** i **in** **range**(len(dataset)):

**if** dataset.iloc[i, 0] **in** frauDs:

        is\_fraud[i]=1

# *creating ANN model*

# *------------------ Feature-Scaling ----------------------*

**from** sklearn.preprocessing **import** StandardScaler

# *y dependent variable, need not to be scaled: categorical variable, 0 and 1*

st\_x= **StandardScaler**()

cuStomers= **st\_x.fit\_transform**(cuStomers)    # *replace "X\_train" by 'cuStomers'*

# *-----------------------------------  Creating ANN model ---------------------------------------------*

    # *1. importing "keras" libraries and packages*

# *from tensorflow import keras*

**import** keras # *using TensorFlow backend*

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

**from** keras.layers **import** Dense

    # *2. initialize the ANN*

ann\_classifier = **Sequential**()

    # *3. Add the "input-layer" and  "first Hidden-layer"*

# *ann\_classifier.add(Dense(output\_dim = 6, init = "uniform", activation = "relu", input\_dim = 11))*

**ann\_classifier.add**(**Dense**(units = 2, kernel\_initializer = "uniform", activation = "relu", input\_dim = 15))

    # *4. Add the "second Hidden-layer"*

# *ann\_classifier.add(Dense(units = 6, kernel\_initializer = "uniform", activation = "relu"))*

    # *5. Add the "output-layer"*

**ann\_classifier.add**(**Dense**(units = 1, kernel\_initializer = "uniform", activation = "sigmoid"))

    # *6. Compile the ANN*

**ann\_classifier.compile**(optimizer = "adam", loss = "binary\_crossentropy", metrics= ["accuracy"])

    # *7. Train the model: fit the ANN to Training-set (batch\_size and epoch)*

**ann\_classifier.fit**(cuStomers, is\_fraud, batch\_size= 1, epochs= 2)

# *----------------------------------- Part 3 : Predictions  ---------------------------------------------*

# *Predicting the probabilities of frauds*

y\_prd = **ann\_classifier.predict**(cuStomers)

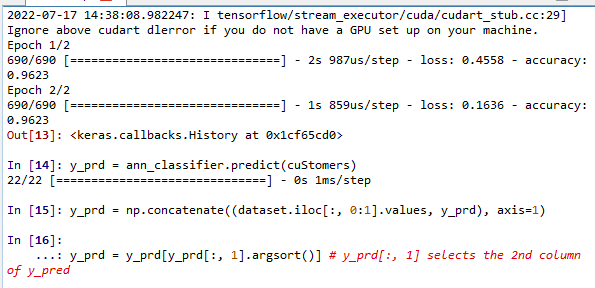
# *concatenate columns to add customer\_id*

y\_prd = **np.concatenate**((dataset.iloc[:, 0:1].values, y\_prd), axis=1)

# *sorting trick*

y\_prd = y\_prd[y\_prd[:, 1].**argsort**()] # *y\_prd[:, 1] selects the 2nd column of y\_pred*

# *python prctc\_mega\_Hybrid\_SOM\_ANN.py*



|  |  |
| --- | --- |
| Un-sorted Customer\_id & Probability | Sorted Customer\_id & Probability |
|  |  |

**Conclusion:** The purpose here to know how to *combine* two *models* to create a *Hybrid model*.