**Assignment -2**

**Healthcare Analytics - AD3002**

**Clinical Predition model using Random forest:**

**Aim:**

To implement a python program for clinical prediction model using Deep learning algorithm(random forest) .

**Program:**

import tensorflow as tf

import tensorflow\_decision\_forests as tfdf

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

import math

print("TensorFlow Decision Forests v" + tfdf.\_\_version\_\_)

print("TensorFlow v" + tf.\_\_version\_\_)

TensorFlow Decision Forests v1.2.0

TensorFlow v2.11.0

train\_proteins = pd.read\_csv("/content/drive/MyDrive/parkinsons\_dis\_pro.csv")

train\_peptides = pd.read\_csv("/content/drive/MyDrive/parkinons\_dis\_pep.csv")

train\_clinical = pd.read\_csv("/content/drive/MyDrive/parkinsons\_dis\_cli.csv")

print('clinical dataset shape: ', train\_clinical.shape)

clinical dataset shape: (2615, 8)

train\_clinical.head(5)

print('proteins dataset shape: ', train\_proteins.shape)

proteins dataset shape: (232741, 5)

train\_proteins.head(5)



print('peptides dataset shape: ', train\_peptides.shape)

peptides dataset shape: (981834, 6)

train\_peptides.head(5)



plot\_df = train\_clinical[train\_clinical["patient\_id"]==1517]

fig, ax = plt.subplots(1, 1)

ax.plot(plot\_df['visit\_month'], plot\_df['updrs\_1'], marker='o', color="blue", label="updrs\_1", linestyle="-")

ax.plot(plot\_df['visit\_month'], plot\_df['updrs\_2'], marker='o', color="red", label="updrs\_2", linestyle="-")

ax.plot(plot\_df['visit\_month'], plot\_df['updrs\_3'], marker='o', color="green", label="updrs\_3", linestyle="-")

ax.plot(plot\_df['visit\_month'], plot\_df['updrs\_4'], marker='o', color="orange", label="updrs\_4", linestyle="-")

ax.legend()

plt.show()



pro\_plot\_df = train\_proteins[train\_proteins["patient\_id"]==1517]

protein\_list = pro\_plot\_df['UniProt'].unique()

protein\_list = protein\_list[:40]

pro\_plot\_df = pro\_plot\_df[pro\_plot\_df['UniProt'].isin(protein\_list)]

unique\_month = pro\_plot\_df['visit\_month'].unique()

unique\_month.sort()

p\_len = len(unique\_month)

pro\_plot\_df['visit\_month'].unique()

n\_rows = math.ceil(p\_len/2)

figure, axis = plt.subplots(n\_rows, 2, figsize=(15, n\_rows\*6))

plt.subplots\_adjust(hspace=0.5, wspace=0.3)

for i, month **in** enumerate(unique\_month):

row = i//2

col = i % 2

plot\_df = pro\_plot\_df[pro\_plot\_df['visit\_month']==month]

bp = sns.barplot(ax=axis[row, col], x=plot\_df['UniProt'], y=plot\_df['NPX'])

bp.set\_xticklabels(bp.get\_xticklabels(), rotation=90, size = 7)

axis[row, col].set\_title('visit\_month ' + str(month))

plt.show()



def prepare\_dataset(train\_proteins, train\_peptides):

df\_protein\_grouped = train\_proteins.groupby(['visit\_id','UniProt'])['NPX'].mean().reset\_index()

df\_peptide\_grouped = train\_peptides.groupby(['visit\_id','Peptide'])['PeptideAbundance'].mean().reset\_index()

df\_protein = df\_protein\_grouped.pivot(index='visit\_id',columns = 'UniProt', values = 'NPX').rename\_axis(columns=None).reset\_index()

df\_peptide = df\_peptide\_grouped.pivot(index='visit\_id',columns = 'Peptide', values = 'PeptideAbundance').rename\_axis(columns=None).reset\_index()

pro\_pep\_df = df\_protein.merge(df\_peptide, on = ['visit\_id'], how = 'left')

return pro\_pep\_df

pro\_pep\_df.shape

(1113, 1196)

pro\_pep\_df.head(5)



def split\_dataset(dataset, test\_ratio=0.20):

test\_indices = np.random.rand(len(dataset)) < test\_ratio

return dataset[~test\_indices], dataset[test\_indices]

model\_dict = {}

mse\_dict = {}

smape\_dict = {}

target = ["updrs\_1", "updrs\_2", "updrs\_3", "updrs\_4"]

for label **in** target:

dataset\_df = pro\_pep\_df.merge(train\_clinical[['visit\_id', 'patient\_id', 'visit\_month', label]], on = ['visit\_id'], how = 'left')

feature\_list = FEATURES.copy()

feature\_list.append(label)

train\_df, valid\_df = split\_dataset(dataset\_df[feature\_list])

print("**{}** examples in training, **{}** examples in testing.".format(len(train\_df), len(valid\_df)))

train\_ds = tfdf.keras.pd\_dataframe\_to\_tf\_dataset(train\_df, label=label, task = tfdf.keras.Task.REGRESSION)

valid\_ds = tfdf.keras.pd\_dataframe\_to\_tf\_dataset(valid\_df, label=label, task = tfdf.keras.Task.REGRESSION)

rf = tfdf.keras.RandomForestModel(task = tfdf.keras.Task.REGRESSION, verbose=0)

rf.compile(metrics=["mse"])

rf.fit(x=train\_ds)

model\_dict[label] = rf

inspector = rf.make\_inspector()

inspector.evaluation()

evaluation = rf.evaluate(x=valid\_ds,return\_dict=True)

mse\_dict[label] = evaluation["mse"]

preds = rf.predict(valid\_ds)

smape\_dict[label] = smape(valid\_df[label].values.tolist(), preds.flatten())

1/1 [==============================] - 5s 5s/step - loss: 0.0000e+00 - mse: 25.7711

1/1 [==============================] - 2s 2s/step

856 examples in training, 212 examples in testing.

[INFO 2023-05-03T13:06:55.401690727+00:00 kernel.cc:1046] Use fast generic engine

1/1 [==============================] - 4s 4s/step - loss: 0.0000e+00 - mse: 29.0788

1/1 [==============================] - 2s 2s/step

845 examples in training, 213 examples in testing.

[INFO 2023-05-03T13:07:59.887008526+00:00 kernel.cc:1046] Use fast generic engine

1/1 [==============================] - 4s 4s/step - loss: 0.0000e+00 - mse: 175.8958

1/1 [==============================] - 2s 2s/step

471 examples in training, 98 examples in testing.

[INFO 2023-05-03T13:08:55.700941357+00:00 kernel.cc:1046] Use fast generic engine

1/1 [==============================] - 4s 4s/step - loss: 0.0000e+00 - mse: 9.4679

1/1 [==============================] - 2s 2s/step

for name, value **in** mse\_dict.items():

print(f"label **{**name**}**: mse **{**value**:**.4f**}**")

print("**\n**Average mse", sum(mse\_dict.values())/4)

label updrs\_1: mse 25.7711

label updrs\_2: mse 29.0788

label updrs\_3: mse 175.8958

label updrs\_4: mse 9.4679

Average mse 60.05340576171875

for name, value **in** smape\_dict.items():

print(f"label **{**name**}**: sMAPE **{**value**:**.4f**}**")

print("**\n**Average sMAPE", sum(smape\_dict.values())/4)

label updrs\_1: sMAPE 61.4490

label updrs\_2: sMAPE 95.1665

label updrs\_3: sMAPE 83.4445

label updrs\_4: sMAPE 154.8876

Average sMAPE 98.73688203627106

import matplotlib.pyplot as plt

logs = model\_dict['updrs\_1'].make\_inspector().training\_logs()

plt.plot([log.num\_trees for log **in** logs], [log.evaluation.rmse for log **in** logs])

plt.xlabel("Number of trees")

plt.ylabel("MSE (out-of-bag)")

plt.show()



import amp\_pd\_peptide\_310

env = amp\_pd\_peptide\_310.make\_env()

iter\_test = env.iter\_test()

In [30]:

def get\_predictions(test\_df, test\_ds, model\_dict):

target = ["updrs\_1", "updrs\_2", "updrs\_3", "updrs\_4"]

temp = test\_df[FEATURES].copy()

mask = test\_df[FEATURES].sum(axis=1) != 0

new\_test\_df = temp.loc[mask]

test\_ds = tfdf.keras.pd\_dataframe\_to\_tf\_dataset(new\_test\_df, task = tfdf.keras.Task.REGRESSION)

for u **in** target:

test\_df['result\_' + str(u)] = 0

if u != 'updrs\_4':

print(model\_dict[u])

test\_df.loc[mask, 'result\_' + str(u)] = model\_dict[u].predict(test\_ds)

result = pd.DataFrame()

for m **in** [0, 6, 12, 24]:

for u **in** [1, 2, 3, 4]:

temp = test\_df[["visit\_id", "result\_updrs\_" + str(u)]].copy()

temp["prediction\_id"] = temp["visit\_id"] + "\_updrs\_" + str(u) + "\_plus\_" + str(m) + "\_months"

temp["rating"] = temp["result\_updrs\_" + str(u)]

temp = temp [['prediction\_id', 'rating']]

result = result.append(temp)

result = result.drop\_duplicates(subset=['prediction\_id', 'rating'])

return result

for (test, test\_peptides, test\_proteins, sample\_submission) **in** iter\_test:

pro\_pep\_df = prepare\_dataset(test\_proteins, test\_peptides)

test\_proteins = test\_proteins.pivot(index=['visit\_id', 'visit\_month', 'patient\_id'],

columns='UniProt', values='NPX').reset\_index().rename\_axis(None, axis=1).drop(columns=['visit\_month', 'patient\_id'])

merged\_df = test.merge(test\_proteins, how='left', on='visit\_id')

test\_df = test.merge(pro\_pep\_df, how='left', on='visit\_id')

for col **in** FEATURES:

if col **not** **in** test\_df.columns:

test\_df[col] = 0

test\_df = test\_df[['visit\_id', 'patient\_id'] + FEATURES]

test\_ds = tfdf.keras.pd\_dataframe\_to\_tf\_dataset(test\_df, task = tfdf.keras.Task.REGRESSION)

result = get\_predictions(test\_df, valid\_ds, model\_dict)

print(result)

env.predict(result)

<tensorflow\_decision\_forests.keras.RandomForestModel object at 0x7ad3d87398a0>

1/1 [==============================] - 9s 9s/step

<tensorflow\_decision\_forests.keras.RandomForestModel object at 0x7ad3d6e123e0>

1/1 [==============================] - 9s 9s/step

<tensorflow\_decision\_forests.keras.RandomForestModel object at 0x7ad3924fba00>

1/1 [==============================] - 8s 8s/step

**Result:**

Thus, to implementation of python code for clinical prediction model using Deep learning algorithm(random forest) is executed successfully.