Kaggle_Final_Submission(2)

November 30, 2021

0.1 Mounting the drive and navigating to the folder

[4]:
```
# mouting the google drive
from google.colab import drive
drive.mount("/gdrive")
```

```
Mounted at /gdrive
```

[6]:
```
# Navigating to the folder in google drive where the data is stored
%cd '/gdrive/MyDrive/Colab Notebooks/Project'
```

```
/gdrive/MyDrive/Colab Notebooks/Project
```

0.1.1 Loading all the libraries

[7]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from scipy.stats import zscore
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
import time
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.model_selection import RandomizedSearchCV
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.neural_network import MLPClassifier
```

0.1.2 Reading the file

[8]:
```
train_data=pd.read_csv('train.csv')
```

[9]:
```
train_data.head(3)
```

[9]:
```
   Unnamed: 0  lepton_1_pT  lepton_1_eta  …  dPhi_r_b  cos(theta_r1)  class
0           0     0.841381      1.832647  …  0.461542       0.005710    0.0
1           1     0.663798      2.058290  …  1.455247       0.101246    0.0
2           2     1.792225     -1.099978  …  0.721326       0.613326    1.0

[3 rows x 20 columns]
```

0.1.3 Splitting the data to inputs and outputs

- It can be seen that the first column of the dataset is the index. Since this does not contain any essential information we can drop this column.
- The last column in the dataset is the output class. This has to be seperated from the other parameters for testing and training purposes.

[ ]:
```python
# Reading the names of columns in the dataset
train_data.columns
```

[ ]:
```
Index(['Unnamed: 0', 'lepton_1_pT', 'lepton_1_eta', 'lepton_1_phi',
       'lepton_2_pT', 'lepton_2_eta', 'lepton_2_phi',
       'missing_energy_magnitude', 'missing_energy_phi', 'MET_rel',
       'axial_MET', 'M_R', 'M_TR_2', 'R', 'MT2', 'S_R', 'M_Delta_R',
       'dPhi_r_b', 'cos(theta_r1)', 'class'],
      dtype='object')
```

[10]:
```python
# Dropping the firsy column which is the index and the last column which is the␣
 ↪output
# Output is then moved to a different variable called y_train whereas the input␣
 ↪is stored in x_train
x_train=train_data.drop([train_data.columns[0],'class'],axis=1)
y_train=train_data['class']
```

0.2 Exploratory data analysis

[ ]:
```python
x_train.head()
```

[ ]:
```
   lepton_1_pT  lepton_1_eta  lepton_1_phi  …  M_Delta_R  dPhi_r_b
cos(theta_r1)
0     0.841381      1.832647     -0.689286  …   0.342497  0.461542
0.005710
1     0.663798      2.058290      0.681435  …   0.333800  1.455247
0.101246
2     1.792225     -1.099978      0.088109  …   0.645729  0.721326
0.613326
3     0.893018      0.297782     -1.274870  …   0.298163  0.803802
0.038902
4     1.338997      0.350023     -1.518510  …   0.330327  0.717237
```

```
0.003147

[5 rows x 18 columns]
```

0.2.1 Finding missing or invalid entries in the dataset

```
[ ]: x_train.isna().sum() # Checking if there are missing values in the dataset
```

```
[ ]: lepton_1_pT                 0
     lepton_1_eta                0
     lepton_1_phi                0
     lepton_2_pT                 0
     lepton_2_eta                0
     lepton_2_phi                0
     missing_energy_magnitude    0
     missing_energy_phi          0
     MET_rel                     0
     axial_MET                   0
     M_R                         0
     M_TR_2                      0
     R                           0
     MT2                         0
     S_R                         0
     M_Delta_R                   0
     dPhi_r_b                    0
     cos(theta_r1)               0
     dtype: int64
```

It can be seen that there is no missing data in the dataset. Hence , there is no process needed to handle null values. However, we might still have missing values or wrong data

```
[ ]: x_train.dtypes # Checking the datatype of all columns in the dataset
```

```
[ ]: lepton_1_pT                 float64
     lepton_1_eta                float64
     lepton_1_phi                float64
     lepton_2_pT                 float64
     lepton_2_eta                float64
     lepton_2_phi                float64
     missing_energy_magnitude    float64
     missing_energy_phi          float64
     MET_rel                     float64
     axial_MET                   float64
     M_R                         float64
     M_TR_2                      float64
     R                           float64
     MT2                         float64
```

```
S_R                           float64
M_Delta_R                     float64
dPhi_r_b                      float64
cos(theta_r1)                 float64
dtype: object
```

Findings from dtypes

- All the datatypes are float64 except the Unnamed 0, which means that there is no data entered as NAN, or as other string formats.

0.3 Understanding the statistical distribution of data

[ ]:
```
np.round(x_train.describe(),2)
```

[ ]:

	lepton_1_pT	lepton_1_eta	…	dPhi_r_b	cos(theta_r1)
count	3500000.00	3500000.00	…	3500000.00	3500000.00
mean	1.00	0.00	…	1.00	0.22
std	0.69	1.00	…	0.44	0.20
min	0.25	-2.10	…	0.00	0.00
25%	0.56	-0.76	…	0.69	0.07
50%	0.79	0.00	…	1.09	0.17
75%	1.20	0.76	…	1.37	0.33
max	20.55	2.10	…	1.59	1.00

[8 rows x 18 columns]

Findings from describe table

- From the above table, it can be understood that the Unnamed : 0 is the index
- lepton_1_pt is left is right skewed as the mean and standard deviation and inclined towards the lower value of the range of values
- missing_energy_magnitude,MET_rel,M_R,R,S_R,M_delta_R,dphi_r_b also has a right skewness
- The last column is the output column and it has to be seperated.
- To understand the dependency of other columns we will have to try other process

0.3.1 Checking for duplicate values

[ ]:
```
x_train.duplicated().sum() # Checking if there are any duplicate entries in the␣
↪dataset
```

[ ]: 0

There are no duplicate entries in the database

0.3.2 Identifying the correlation of data

[ ]:
```
# Set a threshold to identify pairs more than that
correlation=x_train.corr().T

# Identifying columns with high correlation

high_correlation_list=[]
threshold=0.7
# Iterating through the rows and columns of the correlation matrix to check if␣
↪there are any columns or parameters that are highly correlated

for x_iter  in correlation.columns:
  for y_iter in correlation.index:
    if x_iter!=y_iter:
      if (correlation[x_iter][y_iter]>threshold) &␣
↪(correlation[x_iter][y_iter]>(threshold*-1)):
        high_correlation_list.
↪append((x_iter,y_iter,correlation[x_iter][y_iter]))

for element in high_correlation_list:
    print(element)
```

```
('lepton_1_pT', 'M_R', 0.8516937529653643)
('lepton_1_pT', 'M_TR_2', 0.7242294648547011)
('lepton_1_pT', 'S_R', 0.8116162751408211)
('lepton_2_pT', 'M_R', 0.7974893534011629)
('lepton_2_pT', 'S_R', 0.79932499694437)
('missing_energy_magnitude', 'MET_rel', 0.7058940094586493)
('missing_energy_magnitude', 'M_TR_2', 0.7217469748772937)
('MET_rel', 'missing_energy_magnitude', 0.7058940094586493)
('MET_rel', 'M_Delta_R', 0.74856879269838)
('M_R', 'lepton_1_pT', 0.8516937529653643)
('M_R', 'lepton_2_pT', 0.7974893534011629)
('M_R', 'S_R', 0.9813072099983999)
('M_TR_2', 'lepton_1_pT', 0.7242294648547011)
('M_TR_2', 'missing_energy_magnitude', 0.7217469748772937)
('MT2', 'M_Delta_R', 0.808811052489197)
('S_R', 'lepton_1_pT', 0.8116162751408211)
('S_R', 'lepton_2_pT', 0.79932499694437)
('S_R', 'M_R', 0.9813072099983999)
('M_Delta_R', 'MET_rel', 0.74856879269838)
('M_Delta_R', 'MT2', 0.808811052489197)
```

0.3.3 Checking for skewness of the data

[ ]:
```
# It was visible from the datat distribution that certain parameters were␣
↪skewed to the right.
# Using the nibuily skew function to check if the parameters are skewed or not

for column in x_train.columns:
  if (train_data[column].skew()>1) | (train_data[column].skew()<-1):
    print(column,'  ',train_data[column].skew())
print( ' The above attributes are skewed ')
```

```
lepton_1_pT    2.860452539757907
lepton_2_pT    3.522050030748477
missing_energy_magnitude    3.1158771071898284
MET_rel    2.266883264480619
axial_MET    1.53082620504899
M_R    2.8762447920635874
M_TR_2    2.381011555541814
S_R    2.8965969645166187
cos(theta_r1)    1.1415481567512238
 The above attributes are skewed
```

0.3.4 Next steps

- Although it high correaltion and skewness are seen in the above exploration, we have'nt dropeed them as we would like to see the performance of the models with and without these corrections made.

0.3.5 Dividing the dataset into train and test.

- A random state of 7 is used which will be followed throughout in this process

[11]:
```
x_train,x_test,y_train,y_test=train_test_split(x_train,y_train,test_size=0.
↪25,random_state=7)
```

0.3.6 Creating a subset of the dataset to understand which models perform better and which doesn't

[13]:
```
x_train_subset,x_test_subset,y_train_subset,y_test_subset=train_test_split(x_train,y_train,tes
↪40,random_state=7)
```

0.3.7 Trying different models to see its performance

Here we have tried using * Logistic Regression * Decision Tree classifier * Random Forests of Logistic Regressions * Random Forests of Decision Tree Classifiers * Multi Layered Perceptron using sklearn * Multi Layered Perceptron using tensorflow

[ ]:
```
x_train_subset.shape
```

[ ]: (1575000, 18)

```
[ ]: #Creating lists to store results
     # This is later used to create a dataframe to summarise all the models created
     model_name=[]
     score=[]
     run_time=[]
```

Logistic regression and Hyper parameter Tuning

```
[ ]: start=time.time()
     lgR=LogisticRegression(random_state=7)
     lgR.fit(x_train_subset,y_train_subset)
     t=time.time()-start
     model_name.append('Logistic Regression')
     score.append(lgR.score(x_test_subset,y_test_subset))
     run_time.append(t)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

Decision tree classifier

```
[ ]: start=time.time()
     dtClassifier=DecisionTreeClassifier(random_state=7)
     dtClassifier.fit(x_train_subset,y_train_subset)
     t=time.time()-start
     model_name.append('Decision Tree')
     score.append(dtClassifier.score(x_test_subset,y_test_subset))
     run_time.append(t)
```

Random Forest of Linear Regressions

```
[ ]: start=time.time()
     lr_random_forest=BaggingClassifier(base_estimator=lgR,n_estimators=10,random_state=17)
     lr_random_forest.fit(x_train_subset,y_train_subset)
     t=time.time()-start
     model_name.append('Random Forest of Logistic Regressions')
     score.append(lr_random_forest.score(x_test_subset,y_test_subset))
     run_time.append(t)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

Random Forest of Decision Trees

[ ]:
```
start=time.time()
lr_random_forest=BaggingClassifier(base_estimator=dtClassifier,n_estimators=10,random_state=17
lr_random_forest.fit(x_train_subset,y_train_subset)
t=time.time()-start
model_name.append('Random Forest of Decsion Tree Classifiers')
score.append(lr_random_forest.score(x_test_subset,y_test_subset))
run_time.append(t)
```

0.3.8 Multi Layered Perceptron

[ ]:
```
model_name
```

[ ]:
```
['Logistic Regression',
 'Decision Tree',
 'Random Forest of Logistic Regressions',
 'Random Forest of Decsion Tree Classifiers']
```

[15]:
```
start=time.time()
mlp=MLPClassifier(random_state=7,max_iter=100,solver='sgd')
mlp.fit(x_train_subset,y_train_subset)
t=time.time()-start
model_name.append('Multi Layered Perceptron')
score.append(mlp.score(x_test_subset,y_test_subset))
run_time.append(t)
```

[ ]:
```
results=pd.DataFrame()
results['Model']=model_name
results['Score']=score
results['Train time']=run_time
results
```

[ ]:

	Model	Score	Train time
0	Logistic Regression	0.787643	28.858265
1	Decision Tree	0.714670	129.081156
2	Random Forest of Logistic Regressions	0.787667	288.906142
3	Random Forest of Decsion Tree Classifiers	0.781822	838.045714
4	Multi Layered Perceptron	0.801927	445.749412

0.4 Findings

- Out of all the models tried, Multi Layered Perceptron seems to be having the best prediction. Hence we will be using the same model for further fine tuning and hypter parameter adjustments.
- It can also be seen that the MLP takes lower time than a random forest of decision trees and gives the best accuracy on the training subset data

1 Using tensorflow

[ ]:
```python
# Creating a keras model

import tensorflow as tf
import keras
from keras import Sequential,layers



model = keras.Sequential([
    layers.Dense(19, activation='sigmoid', input_shape=(x_train.shape[1],)),
    layers.Dense(100,activation='sigmoid'),
    layers.Dense(50, activation='sigmoid'),
    layers.Dense(25, activation='sigmoid'),
    layers.Dense(10, activation='sigmoid'),
    layers.Dense(1, activation='sigmoid'),
])

model.compile(optimizer='adam',loss='logistic',metrics=['Accuracy'])
```

[ ]:
```python
dlModel = model.fit(
    x_train, y_train,
    validation_data=(x_test, y_test),
    batch_size=100,
    epochs=50,
    verbose=1
)
```

```
Epoch 1/50
26250/26250 [==============================] - 76s 3ms/step - loss: 0.4425 -
Accuracy: 0.7948 - val_loss: 0.4335 - val_Accuracy: 0.7992
Epoch 2/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4341 -
Accuracy: 0.7992 - val_loss: 0.4323 - val_Accuracy: 0.8001
Epoch 3/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4325 -
Accuracy: 0.8000 - val_loss: 0.4317 - val_Accuracy: 0.8003
Epoch 4/50
```

```
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4316 -
Accuracy: 0.8005 - val_loss: 0.4312 - val_Accuracy: 0.8012
Epoch 5/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4309 -
Accuracy: 0.8009 - val_loss: 0.4302 - val_Accuracy: 0.8013
Epoch 6/50
26250/26250 [==============================] - 70s 3ms/step - loss: 0.4305 -
Accuracy: 0.8010 - val_loss: 0.4293 - val_Accuracy: 0.8013
Epoch 7/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4301 -
Accuracy: 0.8012 - val_loss: 0.4302 - val_Accuracy: 0.8006
Epoch 8/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4298 -
Accuracy: 0.8014 - val_loss: 0.4292 - val_Accuracy: 0.8016
Epoch 9/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4295 -
Accuracy: 0.8016 - val_loss: 0.4285 - val_Accuracy: 0.8019
Epoch 10/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4292 -
Accuracy: 0.8017 - val_loss: 0.4289 - val_Accuracy: 0.8020
Epoch 11/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4290 -
Accuracy: 0.8018 - val_loss: 0.4285 - val_Accuracy: 0.8015
Epoch 12/50
26250/26250 [==============================] - 71s 3ms/step - loss: 0.4287 -
Accuracy: 0.8020 - val_loss: 0.4281 - val_Accuracy: 0.8021
Epoch 13/50
26250/26250 [==============================] - 72s 3ms/step - loss: 0.4285 -
Accuracy: 0.8020 - val_loss: 0.4282 - val_Accuracy: 0.8019
Epoch 14/50
26250/26250 [==============================] - 70s 3ms/step - loss: 0.4283 -
Accuracy: 0.8022 - val_loss: 0.4299 - val_Accuracy: 0.8015
Epoch 15/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4281 -
Accuracy: 0.8021 - val_loss: 0.4276 - val_Accuracy: 0.8028
Epoch 16/50
26250/26250 [==============================] - 70s 3ms/step - loss: 0.4278 -
Accuracy: 0.8025 - val_loss: 0.4267 - val_Accuracy: 0.8031
Epoch 17/50
26250/26250 [==============================] - 71s 3ms/step - loss: 0.4276 -
Accuracy: 0.8026 - val_loss: 0.4268 - val_Accuracy: 0.8029
Epoch 18/50
26250/26250 [==============================] - 70s 3ms/step - loss: 0.4274 -
Accuracy: 0.8026 - val_loss: 0.4273 - val_Accuracy: 0.8027
Epoch 19/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4272 -
Accuracy: 0.8028 - val_loss: 0.4271 - val_Accuracy: 0.8025
Epoch 20/50
```

```
26250/26250 [==============================] - 68s 3ms/step - loss: 0.4270 -
Accuracy: 0.8028 - val_loss: 0.4268 - val_Accuracy: 0.8029
Epoch 21/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4269 -
Accuracy: 0.8029 - val_loss: 0.4273 - val_Accuracy: 0.8025
Epoch 22/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4267 -
Accuracy: 0.8030 - val_loss: 0.4259 - val_Accuracy: 0.8034
Epoch 23/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4265 -
Accuracy: 0.8030 - val_loss: 0.4255 - val_Accuracy: 0.8035
Epoch 24/50
26250/26250 [==============================] - 68s 3ms/step - loss: 0.4264 -
Accuracy: 0.8032 - val_loss: 0.4263 - val_Accuracy: 0.8031
Epoch 25/50
26250/26250 [==============================] - 68s 3ms/step - loss: 0.4264 -
Accuracy: 0.8031 - val_loss: 0.4256 - val_Accuracy: 0.8035
Epoch 26/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4263 -
Accuracy: 0.8032 - val_loss: 0.4255 - val_Accuracy: 0.8035
Epoch 27/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4262 -
Accuracy: 0.8032 - val_loss: 0.4255 - val_Accuracy: 0.8036
Epoch 28/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4261 -
Accuracy: 0.8033 - val_loss: 0.4258 - val_Accuracy: 0.8034
Epoch 29/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4260 -
Accuracy: 0.8034 - val_loss: 0.4258 - val_Accuracy: 0.8034
Epoch 30/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4260 -
Accuracy: 0.8033 - val_loss: 0.4255 - val_Accuracy: 0.8034
Epoch 31/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4259 -
Accuracy: 0.8033 - val_loss: 0.4252 - val_Accuracy: 0.8036
Epoch 32/50
26250/26250 [==============================] - 73s 3ms/step - loss: 0.4258 -
Accuracy: 0.8034 - val_loss: 0.4259 - val_Accuracy: 0.8035
Epoch 33/50
26250/26250 [==============================] - 70s 3ms/step - loss: 0.4257 -
Accuracy: 0.8035 - val_loss: 0.4258 - val_Accuracy: 0.8034
Epoch 34/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4257 -
Accuracy: 0.8035 - val_loss: 0.4264 - val_Accuracy: 0.8030
Epoch 35/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4256 -
Accuracy: 0.8036 - val_loss: 0.4250 - val_Accuracy: 0.8035
Epoch 36/50
```

```
26250/26250 [==============================] - 70s 3ms/step - loss: 0.4255 -
Accuracy: 0.8038 - val_loss: 0.4250 - val_Accuracy: 0.8037
Epoch 37/50
26250/26250 [==============================] - 70s 3ms/step - loss: 0.4254 -
Accuracy: 0.8036 - val_loss: 0.4249 - val_Accuracy: 0.8036
Epoch 38/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4254 -
Accuracy: 0.8036 - val_loss: 0.4251 - val_Accuracy: 0.8039
Epoch 39/50
26250/26250 [==============================] - 70s 3ms/step - loss: 0.4253 -
Accuracy: 0.8037 - val_loss: 0.4250 - val_Accuracy: 0.8038
Epoch 40/50
26250/26250 [==============================] - 70s 3ms/step - loss: 0.4253 -
Accuracy: 0.8037 - val_loss: 0.4248 - val_Accuracy: 0.8036
Epoch 41/50
26250/26250 [==============================] - 70s 3ms/step - loss: 0.4252 -
Accuracy: 0.8038 - val_loss: 0.4247 - val_Accuracy: 0.8041
Epoch 42/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4252 -
Accuracy: 0.8037 - val_loss: 0.4266 - val_Accuracy: 0.8028
Epoch 43/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4252 -
Accuracy: 0.8037 - val_loss: 0.4245 - val_Accuracy: 0.8041
Epoch 44/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4251 -
Accuracy: 0.8038 - val_loss: 0.4253 - val_Accuracy: 0.8033
Epoch 45/50
26250/26250 [==============================] - 68s 3ms/step - loss: 0.4250 -
Accuracy: 0.8038 - val_loss: 0.4246 - val_Accuracy: 0.8039
Epoch 46/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4250 -
Accuracy: 0.8039 - val_loss: 0.4251 - val_Accuracy: 0.8036
Epoch 47/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4249 -
Accuracy: 0.8039 - val_loss: 0.4254 - val_Accuracy: 0.8029
Epoch 48/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4249 -
Accuracy: 0.8039 - val_loss: 0.4249 - val_Accuracy: 0.8041
Epoch 49/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4248 -
Accuracy: 0.8039 - val_loss: 0.4244 - val_Accuracy: 0.8040
Epoch 50/50
26250/26250 [==============================] - 69s 3ms/step - loss: 0.4248 -
Accuracy: 0.8041 - val_loss: 0.4248 - val_Accuracy: 0.8041
```

[ ]:
```
df=pd.read_csv('test.csv')
out=pd.DataFrame()
```

```
out['Id']=df[df.columns[0]]
df.drop([df.columns[0]],axis=1,inplace=True)
prediction=model.predict(df)
output=[]
for pred in prediction:
  if pred<=0.80:
    output.append(float(0))
  else:
    output.append(float(1))

out['class']=output
out.to_csv('tensorflow81.csv',index=False)
```

Although the model was able to get good accuracy, the performance of the model on the test dataset in kaggle was pretty low.Hence excluding this model from the final submission

#Creating a multi layered perceptron

1.0.1 Creating a Multi Layered Perceptron for testing the model

- Training the model with more data

[ ]:
```
x_train_subset.shape
```

[ ]:
```
(1575000, 18)
```

Since the MLP took 443 seconds to train on a dataset of size 1.5Million, taking a further subset of the dataset using stratified sampling to find the best hyper parameters.

[18]:
```
#Creating a smaller subset of data
x_train_micro,x_test_micro,y_train_micro,y_test_micro=train_test_split(x_train_subset,y_train_
↪8,stratify=y_train_subset)
```

[ ]:
```
x_train_micro.shape
```

[ ]:
```
(315000, 18)
```

1.1 Using Gridsearch cv for hyper parameter tuning of the MLP model

[16]:
```
hidden_layer_sizes=[10,12,15,18,20]
solver=['sgd','adam']
activation=['identify','logistic','relu','tanh']
batch_size=[100,300,600,1000]
learning_rate=['constant','invscaling','adaptive']

grid={'hidden_layer_sizes':hidden_layer_sizes,
      'solver':solver,
      'activation':activation,
```

```
      'batch_size':batch_size,
      'learning_rate':learning_rate}

random_search=RandomizedSearchCV(estimator=mlp,param_distributions=grid,
                                 n_iter=30,random_state=7,n_jobs=-1,verbose=1)
```

[19]:
```
mlp_search=random_search.fit(x_train_micro,y_train_micro)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
```

```
/usr/local/lib/python3.7/dist-
packages/sklearn/model_selection/_validation.py:372: FitFailedWarning:
20 fits failed out of a total of 150.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
20 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-
packages/sklearn/model_selection/_validation.py", line 681, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.7/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py", line 752, in fit
    return self._fit(X, y, incremental=False)
  File "/usr/local/lib/python3.7/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py", line 384, in _fit
    self._validate_hyperparameters()
  File "/usr/local/lib/python3.7/dist-
packages/sklearn/neural_network/_multilayer_perceptron.py", line 495, in
_validate_hyperparameters
    % (self.activation, list(sorted(ACTIVATIONS)))
ValueError: The activation 'identify' is not supported. Supported activations
are ['identity', 'logistic', 'relu', 'softmax', 'tanh'].

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_search.py:972:
UserWarning: One or more of the test scores are non-finite: [        nan
0.79847619 0.59259048 0.79998095 0.7980381  0.79724127
 0.79859048 0.80124127 0.80086349        nan 0.80119365 0.79782857
 0.80164762 0.8001873  0.80098413        nan 0.80098413 0.73623492
        nan 0.80138095 0.80069841 0.78293651 0.79809841 0.68734603
 0.79482222 0.63067937 0.80029841 0.79844762 0.79931746 0.80014921]
  category=UserWarning,
```

[20]:
```
mlp_search.best_params_
```

[20]:
```
{'activation': 'tanh',
 'batch_size': 300,
 'hidden_layer_sizes': 18,
 'learning_rate': 'constant',
 'solver': 'adam'}
```

2 Creating a model with the best params as specified by grid search

[21]:
```
model=MLPClassifier(hidden_layer_sizes=18,random_state=7,max_iter=100,solver='sgd',batch_size=
```

[22]:
```
model.fit(x_train_micro,y_train_micro)
```

[22]:
```
MLPClassifier(batch_size=100, hidden_layer_sizes=18, max_iter=100,
              random_state=7, solver='sgd')
```

[23]:
```
model.score(x_test_micro,y_test_micro)
```

[23]:
```
0.8005539682539683
```

2.1 Tweaking he hyper parameters more, as the Gridsearch CV Results was not performing well on the test data

[ ]:
```
model_full_data_tuned=MLPClassifier(hidden_layer_sizes=18,random_state=7,max_iter=100,solver='
model_full_data_tuned.fit(x_train,y_train)
model_full_data_tuned.score(x_test,y_test)
```

[ ]:
```
0.8019097142857143
```

[ ]:
```
model_full_data_tuned1=MLPClassifier(hidden_layer_sizes=15,random_state=7,max_iter=200,solver=
model_full_data_tuned1.fit(x_train,y_train)
model_full_data_tuned1.score(x_test,y_test)
```

[ ]:
```
0.8018697142857143
```

[ ]:
```
model_full_data_tuned2=MLPClassifier(hidden_layer_sizes=18,random_state=7,max_iter=100,solver=
model_full_data_tuned2.fit(x_train,y_train)
model_full_data_tuned2.score(x_test,y_test)
```

[ ]:
```
0.80184
```

[ ]:
```
model_full_data_tuned3=MLPClassifier(hidden_layer_sizes=18,random_state=7,max_iter=100,solver=
model_full_data_tuned3.fit(x_train,y_train)
model_full_data_tuned3.score(x_test,y_test)
```

[ ]: 0.80184

[ ]:
```
model_full_data4=MLPClassifier(random_state=7,max_iter=100,solver='sgd')
model_full_data4.fit(x_train,y_train)
model_full_data4.score(x_test,y_test)
```

[ ]: 0.8028171428571429

[ ]:
```
model_full_data5=MLPClassifier(random_state=7,max_iter=50,solver='sgd')
model_full_data5.fit(x_train,y_train)
model_full_data5.score(x_test,y_test)
```

[ ]: 0.8028171428571429

[ ]:
```
model_full_data6=MLPClassifier(random_state=7,max_iter=100,solver='sgd')
model_full_data6.fit(x_train,y_train)
model_full_data6.score(x_test,y_test)
```

[ ]: 0.8028171428571429

[ ]:
```
model_full_data_7=MLPClassifier(random_state=7,max_iter=100,solver='adam')
model_full_data_7.fit(x_train,y_train)
model_full_data_7.score(x_test,y_test)
```

[ ]: 0.8035085714285715

[ ]:
```
model_full_data_8=MLPClassifier(random_state=7,max_iter=100,solver='adam',activation='logistic
model_full_data_8.fit(x_train,y_train)
model_full_data_8.score(x_test,y_test)
```

[ ]: 0.8037977142857143

[ ]:
```
model_full_data_9=MLPClassifier(random_state=7,max_iter=100,solver='adam',activation='logistic
model_full_data_9.fit(x_train,y_train)
model_full_data_9.score(x_test,y_test)
```

[ ]: 0.8037977142857143

[ ]:
```
# Normalising the data and applying the above model again
```

[ ]:
```
x_train=train_data.drop([train_data.columns[0],'class'],axis=1)
y_train=train_data['class']
x_train=x_train.apply(zscore)
x_train,x_test,y_train,y_test=train_test_split(x_train,y_train,test_size=0.
↪25,random_state=7)

model_full_data_10=MLPClassifier(random_state=7,max_iter=100,solver='adam',activation='logisti
```

```
model_full_data_10.fit(x_train,y_train)
model_full_data_10.score(x_test,y_test)
```

[ ]: 0.8039668571428571

3 Submission using the result obtained in las titeration where the accuracy was the maximum so far.

3.1 submission1(1).csv

[ ]:
```
df=pd.read_csv('test.csv')
out=pd.DataFrame()
out['Id']=df[df.columns[0]]
df.drop([df.columns[0]],axis=1,inplace=True)
prediction=model_full_data6.predict(df)
out['class']=prediction
```

[ ]:
```
out.to_csv('submission1.csv',index=False)
```

3.2 Submission2.csv

[ ]:
```
df=pd.read_csv('test.csv')
out=pd.DataFrame()
out['Id']=df[df.columns[0]]
df.drop([df.columns[0]],axis=1,inplace=True)
prediction=model_full_data_7.predict(df)
out['class']=prediction
out.to_csv('submission2.csv',index=False)
```

```
# This is formatted as code
```

3.3 Submission3.csv

[ ]:
```
df=pd.read_csv('test.csv')
out=pd.DataFrame()
out['Id']=df[df.columns[0]]
df.drop([df.columns[0]],axis=1,inplace=True)
prediction=model_full_data_8.predict(df)
out['class']=prediction
out.to_csv('submission3.csv',index=False)
```

3.4 Submission4.csv

[ ]:
```
df=pd.read_csv('test.csv')
out=pd.DataFrame()
out['Id']=df[df.columns[0]]
df.drop([df.columns[0]],axis=1,inplace=True)
prediction=model_full_data_10.predict(df)
out['class']=prediction
out.to_csv('submission4.csv',index=False)
```