[Run in Google Colab]

*This is an example Notebook for running training on Higgs vs background signal classification. *

Background: High-energy collisions at the Large Hadron Collider (LHC) produce particles that interact with particle detectors. One important task is to classify different types of collisions based on their physics content, allowing physicists to find patterns in the data and to potentially unravel new discoveries.

Problem statement: The discovery of the Higgs boson by CMS and ATLAS Collaborations was announced at CERN in 2012. In this challenge, we focus on the potential of Machine Learning in detecting potential Higgs signal from one of the background processes that mimics it.

Dataset: The dataset is made available by the Center for Machine Learning and Intelligent Systems at University of California, Irvine. The dataset can be found on the UCI Machine learning Repository

Description: The dataset consists of a total of 11 million labeled samples of Higgs vs background events produced by Monte Carlo simulations. Each sample consists of 28 features. The first 21 features are kinematic properties measured at the level of the detectors. The last seven are functions of the first 21.

Steps to load the training dataset

1. Download the dataset from the UCI website.

```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/00280/HIGGS.csv.gz
```

```
--2022-02-22 19:39:17--  https://archive.ics.uci.edu/ml/machine-learning-databases/00280/HIGGS.csv.gz
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2816407858 (2.6G) [application/x-httpd-php]
Saving to: 'HIGGS.csv.gz'

HIGGS.csv.gz        100%[===================>]   2.62G  56.5MB/s    in 55s

2022-02-22 19:40:12 (49.1 MB/s) - 'HIGGS.csv.gz' saved [2816407858/2816407858]
```

2. Unzip the dataset folder

```
!gzip -d HIGGS.csv.gz
```

```
pip install plot-metric ## For plotting ROC at the end
```

```
Collecting plot-metric
  Downloading plot_metric-0.0.6-py3-none-any.whl (13 kB)
Requirement already satisfied: seaborn>=0.9.0 in /usr/local/lib/python3.7/dist-packages (from plot-metri
Requirement already satisfied: colorlover>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from plot-me
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.7/dist-packages (from plot-metric
Requirement already satisfied: matplotlib>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from plot-me
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from plot-metric)
Requirement already satisfied: scikit-learn>=0.21.2 in /usr/local/lib/python3.7/dist-packages (from plot
Requirement already satisfied: pandas>=0.23.4 in /usr/local/lib/python3.7/dist-packages (from plot-metri
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplot
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matp
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.23
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scik
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn
Installing collected packages: plot-metric
Successfully installed plot-metric-0.0.6
```

```
from sklearn.datasets import make_gaussian_quantiles
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
```

```
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
```

```
import numpy as np
np.random.seed(1337)  # for reproducibility
import h5py
from keras.models import Sequential
from keras.optimizers import adam_v2
from keras.initializers import TruncatedNormal
from keras.layers import Input, Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.callbacks import ReduceLROnPlateau

from sklearn.metrics import roc_curve, auc

import matplotlib.pyplot as plt
```

Load the file using pandas library

```
data=pd.read_csv('./HIGGS.csv')
```

Assign first column 0 to class labels (labeled 1 for signal, 0 for background) and all others to feature matrix X.

For demonstration, here we only use 1000 samples. To train on the entire dataset, uncomment the lines below.

```
X=data.iloc[:,1:]#data.iloc[:,1:]
y=data.iloc[:,0]#data.iloc[:,0]
```

Split your data into training and validation samples, where the fraction of the data used for validation is 20%.

```
X_train1, X_test, y_train1, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train1, y_train1, test_size=0.2, random_state=42)
```

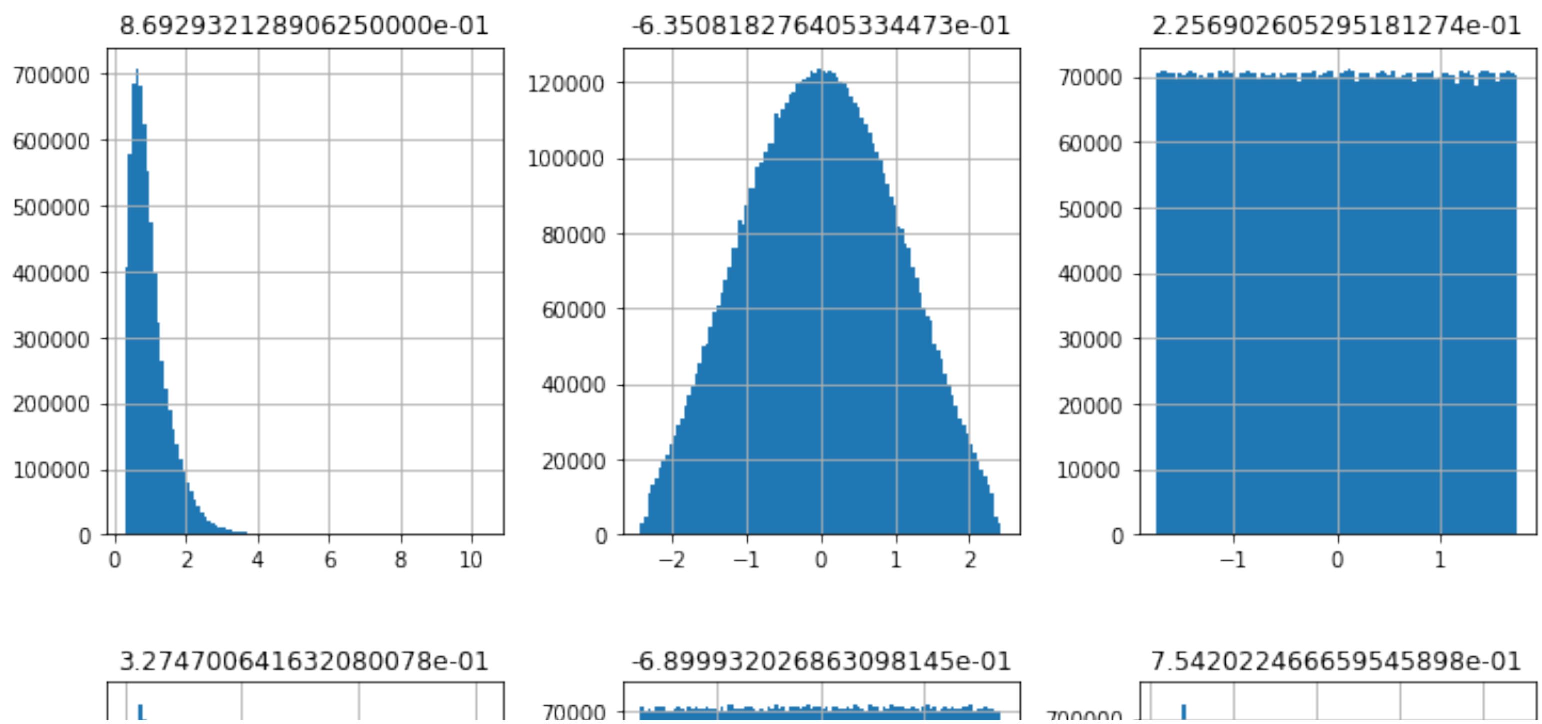
Visualize your data - One histogram per feature column

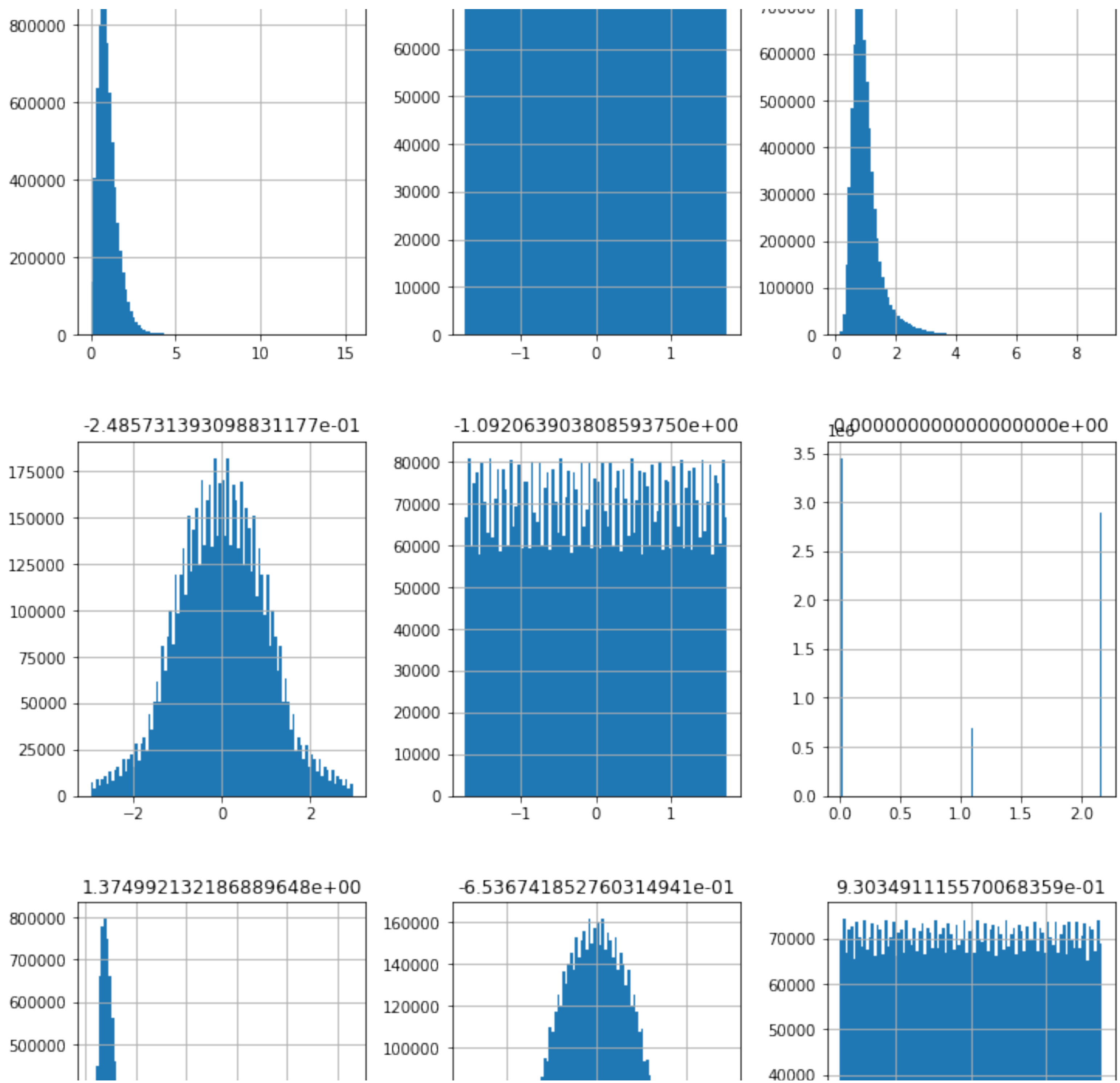
Detailed information on what each feature column is can be found in *Attribute Information* section on the [UCI Machine learning Repositery](). For further information, refer to the [paper]() by Baldi et. al

```
from itertools import combinations
import matplotlib.pyplot as plt

fig, axes = plt.subplots(len(X_train.columns)//3, 3, figsize=(12, 48))

i = 0
for triaxis in axes:
    for axis in triaxis:
        X_train.hist(column = X_train.columns[i], bins = 100, ax=axis)
        i = i+1
```




-2.485731393098831177e-01
-1.092063903808593750e+00
0.000000000000000000e+00
1e6
1.374992132186889648e+00
-6.536741852760314941e-01
9.303491115570068359e-01

Boosted Decision Tree model

Setup the Boosted Decision Tree model

```
# classifier = AdaBoostClassifier(
#     DecisionTreeClassifier(max_depth=1),
#     n_estimators=400
# )
```

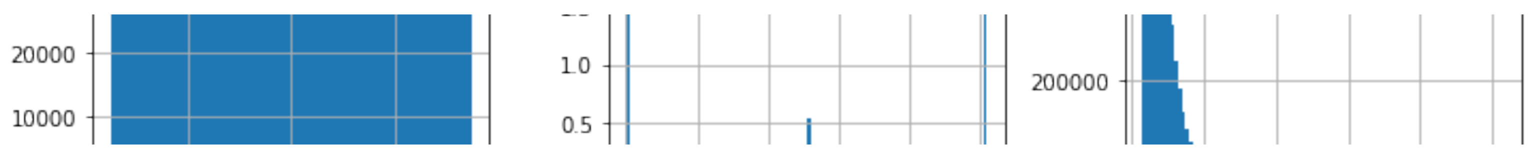
Train the Boosted Decision Tree model

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

```
# classifier.fit(X_train, y_train)
```

Predict on new testing data

```
# predictions = classifier.predict(X_test)
```



```
# y_hat = classifier.predict_proba(X_test)[:, 1]
```

Print confusion matrix and plot ROC curve.

```
# confusion_matrix(y_test, predictions)
```

New Section

```
# from plot_metric.functions import BinaryClassification
# # Visualisation with plot_metric
# bc = BinaryClassification(y_test, y_hat, labels=["Class 1", "Class 2"])

# # Figures
# plt.figure(figsize=(5,5))
# bc.plot_roc_curve()
# plt.show()
```

Shallow Neural Networks (optional for those who are familiar with NNs)

Setup the Neural Network (some useful info here)

```
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import keras
```

```
# model_nn = Sequential()
# model_nn.add(Dense(28, input_dim=28, activation='relu'))
# model_nn.add(Dense(8, activation='relu'))
# model_nn.add(Dense(1, activation='sigmoid'))
```

Train the Neural Network and save your model weights in a h5 file

(Train for more epochs than shown here and play around with the architecture)


```
model_nn = Sequential()
model_nn.add(Dense(300, input_dim=28, activation='relu'))
model_nn.add(Dropout(.05))
model_nn.add(Dense(250, activation='relu'))
model_nn.add(Dropout(.05))
model_nn.add(Dense(200, activation='relu'))
model_nn.add(Dropout(.05))
model_nn.add(Dense(150, activation='relu'))
model_nn.add(Dropout(.05))
model_nn.add(Dense(100, activation='relu'))
model_nn.add(Dropout(.025))
model_nn.add(Dense(50, activation='relu'))
model_nn.add(Dense(1, activation='sigmoid'))
# compile the keras model
model_nn.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=1.e-3), metrics=['accuracy'])
model_nn.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 300)               8700

 dropout (Dropout)           (None, 300)               0

 dense_1 (Dense)             (None, 250)               75250

 dropout_1 (Dropout)         (None, 250)               0

 dense_2 (Dense)             (None, 200)               50200

 dropout_2 (Dropout)         (None, 200)               0

 dense_3 (Dense)             (None, 150)               30150

 dropout_3 (Dropout)         (None, 150)               0

 dense_4 (Dense)             (None, 100)               15100

 dropout_4 (Dropout)         (None, 100)               0

 dense_5 (Dense)             (None, 50)                5050

 dense_6 (Dense)             (None, 1)                 51

=================================================================
Total params: 184,501
Trainable params: 184,501
Non-trainable params: 0
_________________________________________________________________
```

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_delta=0.0001, min_lr=1e-10, mod
checkpoint_cb = keras.callbacks.ModelCheckpoint("model_nn.h5", save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=4, restore_best_weights = True)
history=model_nn.fit(X_train, y_train,\
```

```
        batch_size=1000,\
        epochs=50,\
        validation_data=(X_val, y_val),\
        callbacks=[reduce_lr, checkpoint_cb, early_stopping_cb],\
        verbose=1, shuffle=True, initial_epoch=0
        )
model_nn = keras.models.load_model("model_nn.h5")
```

```
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4851 - accuracy: 0.7609 - val_loss
Epoch 5/50
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4815 - accuracy: 0.7632 - val_loss
Epoch 6/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4785 - accuracy: 0.7651 - val_loss
Epoch 7/50
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4765 - accuracy: 0.7664 - val_loss
Epoch 8/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4745 - accuracy: 0.7677 - val_loss
Epoch 9/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4730 - accuracy: 0.7686 - val_loss
Epoch 10/50
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4716 - accuracy: 0.7696 - val_loss
Epoch 11/50
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4705 - accuracy: 0.7702 - val_loss
Epoch 12/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4693 - accuracy: 0.7708 - val_loss
Epoch 13/50
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4683 - accuracy: 0.7716 - val_loss
Epoch 14/50
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4675 - accuracy: 0.7722 - val_loss
Epoch 15/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4668 - accuracy: 0.7725 - val_loss
Epoch 16/50
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4661 - accuracy: 0.7729 - val_loss
Epoch 17/50
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4653 - accuracy: 0.7735 - val_loss
Epoch 18/50
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4648 - accuracy: 0.7738 - val_loss
Epoch 19/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4577 - accuracy: 0.7781 - val_loss
Epoch 20/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4565 - accuracy: 0.7787 - val_loss
Epoch 21/50
```

```
Epoch 21/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4560 - accuracy: 0.7791 - val_loss
Epoch 22/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4558 - accuracy: 0.7791 - val_loss
Epoch 23/50
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4554 - accuracy: 0.7794 - val_loss
Epoch 24/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4553 - accuracy: 0.7795 - val_loss
Epoch 25/50
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4551 - accuracy: 0.7796 - val_loss
Epoch 26/50
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4548 - accuracy: 0.7798 - val_loss
Epoch 27/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4547 - accuracy: 0.7798 - val_loss
Epoch 28/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4545 - accuracy: 0.7799 - val_loss
Epoch 29/50
7040/7040 [==============================] - 62s 9ms/step - loss: 0.4544 - accuracy: 0.7802 - val_loss
Epoch 30/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4527 - accuracy: 0.7810 - val_loss
Epoch 31/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4526 - accuracy: 0.7810 - val_loss
Epoch 32/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4525 - accuracy: 0.7812 - val_loss
Epoch 33/50
7040/7040 [==============================] - 61s 9ms/step - loss: 0.4524 - accuracy: 0.7812 - val_loss
```

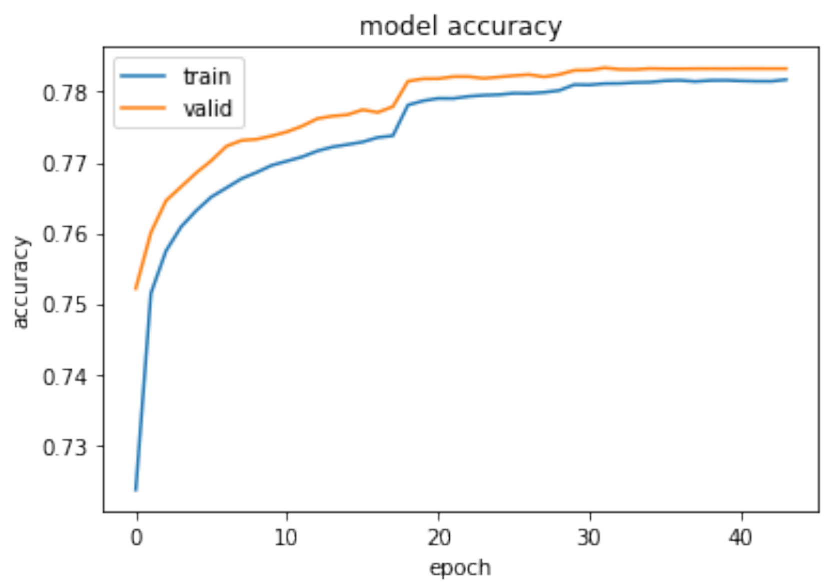
```
# # compile the keras model
# model_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# # fit the keras model on the dataset
# history=model_nn.fit(X, y,validation_data=(X_val,y_val),epochs=5, batch_size=10)
# # evaluate the keras model
# _, accuracy = model_nn.evaluate(X, y)
# model_nn.save('my_model.h5') ##Saving model weights
# print('Accuracy: %.2f' % (accuracy*100))
```

```
# list all data in history
print(history.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy', 'lr'])
```

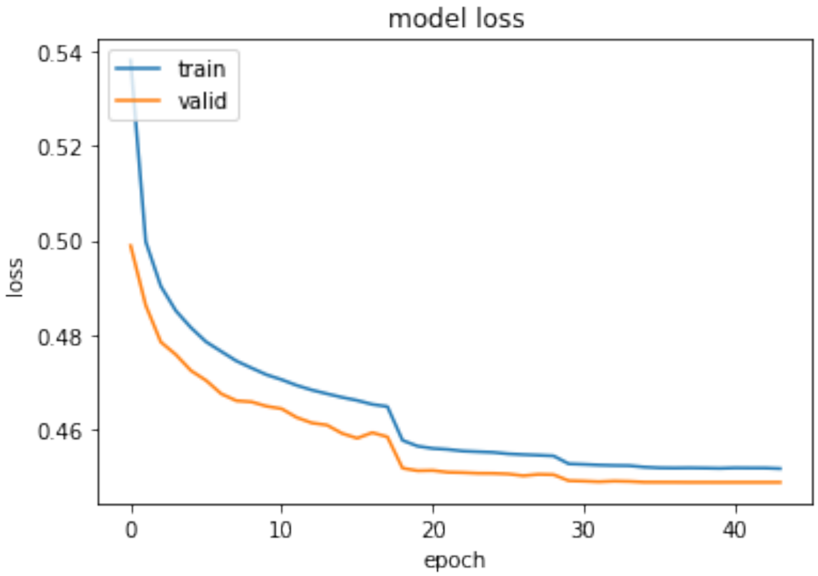
Plot accuracy wrt number of epochs

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
```



Plot training loss wrt number of epochs

```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
```




```
y_pred=model_nn.predict(X_test)
```

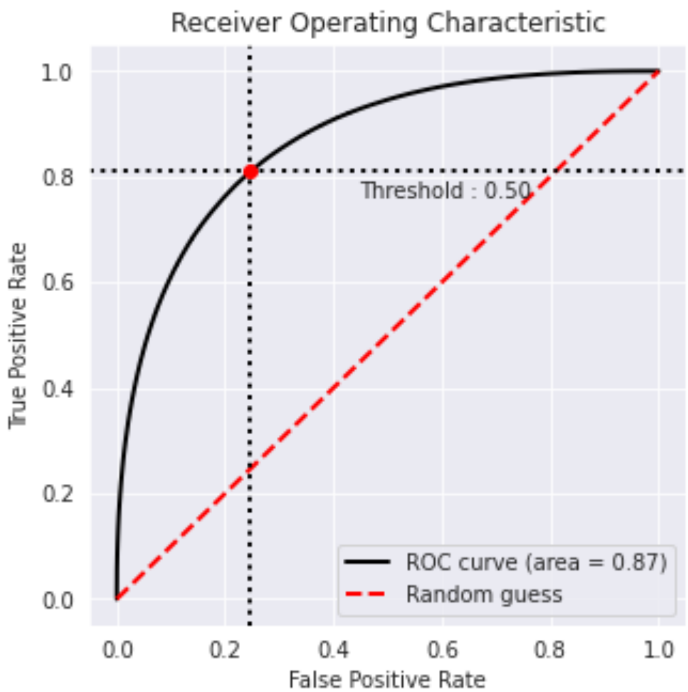
```
confusion_matrix(y_test, y_pred.round())
```

```
    array([[778699, 255774],
           [221438, 944089]])
```

Plot the ROC (Receiver Operating Characteristic) Curve (more info on ROC could be found here)

```
from plot_metric.functions import BinaryClassification
# Visualisation with plot_metric
bc = BinaryClassification(y_test, y_pred, labels=["Class 1", "Class 2"])

# Figures
plt.figure(figsize=(5,5))
bc.plot_roc_curve()
plt.show()
```



Goal: Please train your own machine learning model (or modify provided examples) with the goal of attaining the top classifier performance.

Deliverables:

Please submit the following:

- Your full notebook (pdf and `.ipynb`) used for training including the ROC Curves, loss and accuracy plots wrt number of epochs.
- for Neural Networks: model weights (`.h5`)

References:

Baldi, P., Sadowski P., and Whiteson D. “Searching for Exotic Particles in High-energy Physics with Deep Learning.” Nature Communications 5 (July 2, 2014).