Hackathon-01

February 23, 2022

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.

[5]:
```
!wget https://archive.ics.uci.edu/ml/machine-learning-databases/00280/HIGGS.csv.
↪gz
```

```
--2022-02-23 11:22:11--  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  9,63MB/s    in 6m 6s

2022-02-23 11:28:19 (7,34 MB/s) - ‘HIGGS.csv.gz’ saved [2816407858/2816407858]
```

2. Unzip the dataset folder

[6]:
```
!gzip -d HIGGS.csv.gz
```

[1]:
```
pip install plot-metric ## For plotting ROC at the end
```

```
Requirement already satisfied: plot-metric in /usr/local/lib/python3.8/site-
packages (0.0.6)
Requirement already satisfied: seaborn>=0.9.0 in /usr/local/lib/python3.8/site-
packages (from plot-metric) (0.10.1)
Requirement already satisfied: matplotlib>=3.0.2 in
/usr/local/lib/python3.8/site-packages (from plot-metric) (3.5.1)
Requirement already satisfied: colorlover>=0.3.0 in
/usr/local/lib/python3.8/site-packages (from plot-metric) (0.3.0)
Requirement already satisfied: scikit-learn>=0.21.2 in
/usr/local/lib/python3.8/site-packages (from plot-metric) (1.0.2)
Requirement already satisfied: pandas>=0.23.4 in /usr/local/lib/python3.8/site-
packages (from plot-metric) (1.4.0)
Requirement already satisfied: numpy>=1.15.4 in /usr/local/lib/python3.8/site-
packages (from plot-metric) (1.22.2)
Requirement already satisfied: scipy>=1.1.0 in
/Users/thalesoliveira/Library/Python/3.8/lib/python/site-packages (from plot-
metric) (1.4.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/site-
packages (from matplotlib>=3.0.2->plot-metric) (0.10.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/site-
packages (from matplotlib>=3.0.2->plot-metric) (21.3)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.8/site-
packages (from matplotlib>=3.0.2->plot-metric) (9.0.1)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.8/site-packages (from matplotlib>=3.0.2->plot-metric)
(2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.8/site-packages (from matplotlib>=3.0.2->plot-metric)
(1.3.2)
Requirement already satisfied: pyparsing>=2.2.1 in
/usr/local/lib/python3.8/site-packages (from matplotlib>=3.0.2->plot-metric)
(2.4.7)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.8/site-packages (from matplotlib>=3.0.2->plot-metric)
(4.29.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.8/site-
packages (from scikit-learn>=0.21.2->plot-metric) (0.16.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.8/site-packages (from scikit-learn>=0.21.2->plot-metric)
(2.1.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.8/site-
packages (from pandas>=0.23.4->plot-metric) (2021.3)
```

```
Requirement already satisfied: six in /usr/local/lib/python3.8/site-packages
(from cycler>=0.10->matplotlib>=3.0.2->plot-metric) (1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

[7]:
```
## Scikit learn imports
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.datasets import load_breast_cancer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve, auc
```

[8]:
```
# Standard imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[9]:
```
np.random.seed(1337)  # for reproducibility
import h5py

## tensorflow/keras imports
import tensorflow.keras as keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.initializers import TruncatedNormal
from tensorflow.keras.layers import Input, Dense, Dropout, Flatten, Conv2D,␣
↪MaxPooling2D
from tensorflow.keras.callbacks import ReduceLROnPlateau
```

Load the file using pandas library

[10]:
```
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.

[11]:
```
X=data.iloc[:1000,1:]#data.iloc[:,1:]
y=data.iloc[:1000,0]#data.iloc[:,0]
```

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

[12]:
```
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)
```

[13]:
```
# 640 for training
# 160 for validation
# 200 for testing
```

Visualize your data - One histogram per feature column

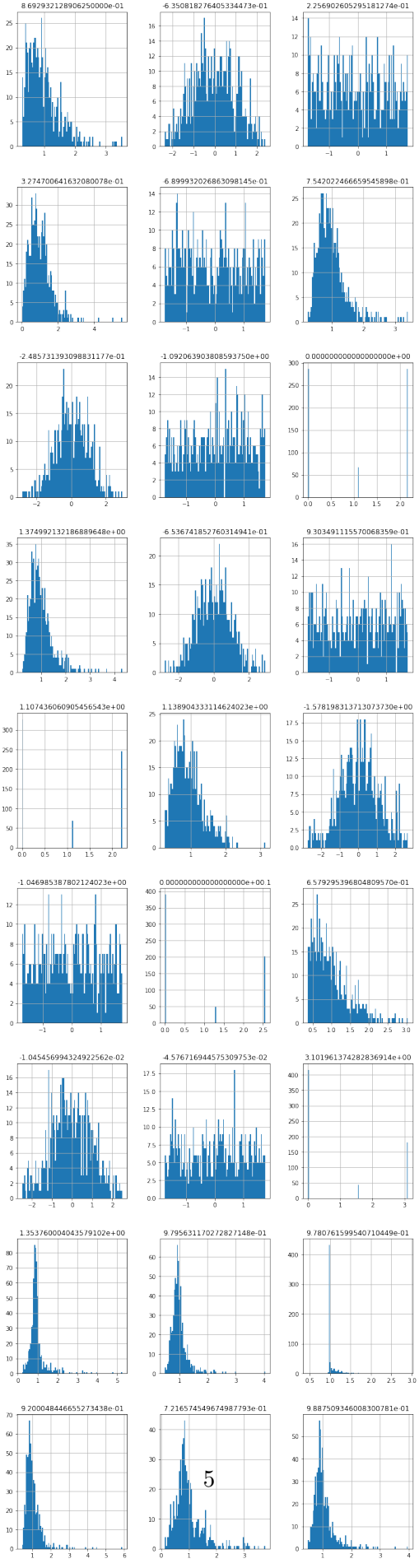
Detailed information on what each feature column is can be found in *Attribute Information* section on the UCI Machine learning Repository. For further information, refer to the paper by Baldi et. al

[14]:
```
#lepton pT, lepton eta, lepton phi, missing energy magnitude, missing energy␣
↪phi,
#jet 1 pt, jet 1 eta, jet 1 phi, jet 1 b-tag, jet 2 pt, jet 2 eta, jet 2 phi,␣
↪jet 2 b-tag,
#jet 3 pt, jet 3 eta, jet 3 phi, jet 3 b-tag, jet 4 pt, jet 4 eta, jet 4 phi,␣
↪jet 4 b-tag,
#m_jj, m_jjj, m_lv, m_jlv, m_bb, m_wbb, m_wwbb
```

[15]:
```
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
```


8.692932128906250000e-01
-6.350818276405334473e-01
2.256902605295181274e-01
3.274700641632080078e-01
-6.899932026863098145e-01
7.542022466659545898e-01
-2.485731393098831177e-01
-1.092063903808593750e+00
0.000000000000000000e+00
1.374992132186889648e+00
-6.536741852760314941e-01
9.303491115570068359e-01
1.107436060905456543e+00
1.138904333114624023e+00
-1.578198313713073730e+00
-1.046985387802124023e+00
0.000000000000000000e+00.1
6.579295396804809570e-01
-1.045456994324922562e-02
-4.576716944575309753e-02
3.101961374282836914e+00
1.353760004043579102e+00
9.795631170272827148e-01
9.780761599540710449e-01
9.200048446655273438e-01
7.216574549674987793e-01
9.887509346008300781e-01
5

1 Boosted Decision Tree model

Setup the Boosted Decision Tree model

[16]:
```
classifier = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1),
    n_estimators=200)
```

Train the Boosted Decision Tree model

[17]:
```
classifier.fit(X_train, y_train)
```

[17]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   n_estimators=200)
```

Predict on new testing data

[18]:
```
predictions = classifier.predict(X_test)
```

[19]:
```
y_hat = classifier.predict_proba(X_test)[:, 1]
```

Print confusion matrix and plot ROC curve.

[20]:
```
confusion_matrix(y_test, predictions)
```

[20]:
```
array([[49, 43],
       [28, 80]])
```

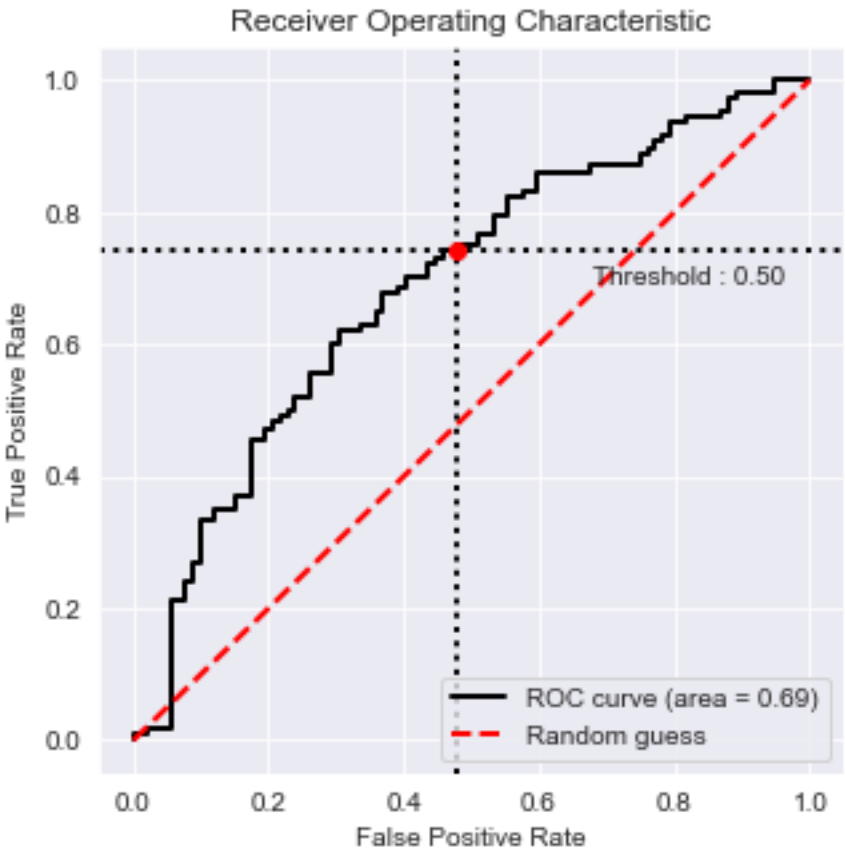
[31]:
```
from sklearn.metrics import accuracy_score

accuracy_score(y_test,predictions)
```

[31]:
```
0.645
```

[21]:
```
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()
```



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

Setup the Neural Network (some useful info here)

[22]:
```
# Keras NN
#model_nn = Sequential()
#model_nn.add(Dense(28, input_dim=28, activation='relu'))  # input layer
#model_nn.add(Dense(8, activation='relu'))
#model_nn.add(Dense(1, activation='sigmoid'))              # output layer
```

[50]:
```
model_nn = Sequential()
model_nn.add(Dense(28, input_dim=28, activation='relu'))
model_nn.add(Dense(14, activation='relu'))
model_nn.add(Dense(7, activation='relu'))
#model_nn.add(Dense(3, 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)

[51]:
```
# 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=30, batch_size=5)
# evaluate the keras model
_, accuracy = model_nn.evaluate(X, y)
model_nn.save('my_model.h5') ##Saving model weights
print('Accuracy: %.2f' % (accuracy*100))
```

```
Epoch 1/30
200/200 [==============================] - 1s 3ms/step - loss: 0.7109 -
accuracy: 0.4960 - val_loss: 0.6831 - val_accuracy: 0.5312
Epoch 2/30
200/200 [==============================] - 1s 3ms/step - loss: 0.6772 -
accuracy: 0.5830 - val_loss: 0.6563 - val_accuracy: 0.6125
Epoch 3/30
200/200 [==============================] - 1s 3ms/step - loss: 0.6645 -
accuracy: 0.5980 - val_loss: 0.6344 - val_accuracy: 0.6250
Epoch 4/30
200/200 [==============================] - 0s 2ms/step - loss: 0.6476 -
accuracy: 0.5970 - val_loss: 0.6222 - val_accuracy: 0.6562
Epoch 5/30
200/200 [==============================] - ETA: 0s - loss: 0.6328 - accuracy:
0.62 - 1s 3ms/step - loss: 0.6324 - accuracy: 0.6300 - val_loss: 0.6063 -
val_accuracy: 0.6812
Epoch 6/30
200/200 [==============================] - 1s 3ms/step - loss: 0.6227 -
accuracy: 0.6380 - val_loss: 0.6091 - val_accuracy: 0.6500
Epoch 7/30
200/200 [==============================] - 1s 3ms/step - loss: 0.6063 -
accuracy: 0.6720 - val_loss: 0.5809 - val_accuracy: 0.6750
Epoch 8/30
200/200 [==============================] - 1s 3ms/step - loss: 0.5874 -
accuracy: 0.6840 - val_loss: 0.5684 - val_accuracy: 0.6875
Epoch 9/30
200/200 [==============================] - 1s 3ms/step - loss: 0.5724 -
accuracy: 0.6840 - val_loss: 0.5537 - val_accuracy: 0.7063
Epoch 10/30
200/200 [==============================] - 1s 3ms/step - loss: 0.5571 -
accuracy: 0.7090 - val_loss: 0.5505 - val_accuracy: 0.7188
Epoch 11/30
200/200 [==============================] - 1s 3ms/step - loss: 0.5454 -
accuracy: 0.7110 - val_loss: 0.5247 - val_accuracy: 0.6938
Epoch 12/30
200/200 [==============================] - 1s 3ms/step - loss: 0.5282 -
```

```
accuracy: 0.7180 - val_loss: 0.4997 - val_accuracy: 0.7500
Epoch 13/30
200/200 [==============================] - 1s 3ms/step - loss: 0.5096 -
accuracy: 0.7530 - val_loss: 0.4831 - val_accuracy: 0.7500
Epoch 14/30
200/200 [==============================] - 1s 3ms/step - loss: 0.5016 -
accuracy: 0.7620 - val_loss: 0.4928 - val_accuracy: 0.8000
Epoch 15/30
200/200 [==============================] - 1s 3ms/step - loss: 0.4811 -
accuracy: 0.7740 - val_loss: 0.4534 - val_accuracy: 0.8062
Epoch 16/30
200/200 [==============================] - 1s 3ms/step - loss: 0.4654 -
accuracy: 0.7890 - val_loss: 0.4578 - val_accuracy: 0.7750
Epoch 17/30
200/200 [==============================] - 1s 3ms/step - loss: 0.4501 -
accuracy: 0.8010 - val_loss: 0.4583 - val_accuracy: 0.7563
Epoch 18/30
200/200 [==============================] - 1s 3ms/step - loss: 0.4354 -
accuracy: 0.7960 - val_loss: 0.4451 - val_accuracy: 0.7750
Epoch 19/30
200/200 [==============================] - 1s 4ms/step - loss: 0.4238 -
accuracy: 0.8240 - val_loss: 0.4308 - val_accuracy: 0.8125
Epoch 20/30
200/200 [==============================] - 1s 3ms/step - loss: 0.4167 -
accuracy: 0.8100 - val_loss: 0.4149 - val_accuracy: 0.8125
Epoch 21/30
200/200 [==============================] - 1s 3ms/step - loss: 0.3985 -
accuracy: 0.8240 - val_loss: 0.3850 - val_accuracy: 0.8375
Epoch 22/30
200/200 [==============================] - 1s 3ms/step - loss: 0.3850 -
accuracy: 0.8330 - val_loss: 0.4124 - val_accuracy: 0.8062
Epoch 23/30
200/200 [==============================] - 0s 2ms/step - loss: 0.3729 -
accuracy: 0.8330 - val_loss: 0.3902 - val_accuracy: 0.8375
Epoch 24/30
200/200 [==============================] - 1s 5ms/step - loss: 0.3614 -
accuracy: 0.8440 - val_loss: 0.3693 - val_accuracy: 0.8438
Epoch 25/30
200/200 [==============================] - 1s 3ms/step - loss: 0.3523 -
accuracy: 0.8390 - val_loss: 0.3554 - val_accuracy: 0.8250
Epoch 26/30
200/200 [==============================] - 1s 3ms/step - loss: 0.3447 -
accuracy: 0.8490 - val_loss: 0.3378 - val_accuracy: 0.8500
Epoch 27/30
200/200 [==============================] - 1s 4ms/step - loss: 0.3313 -
accuracy: 0.8620 - val_loss: 0.3373 - val_accuracy: 0.8562
Epoch 28/30
200/200 [==============================] - 1s 4ms/step - loss: 0.3215 -
```

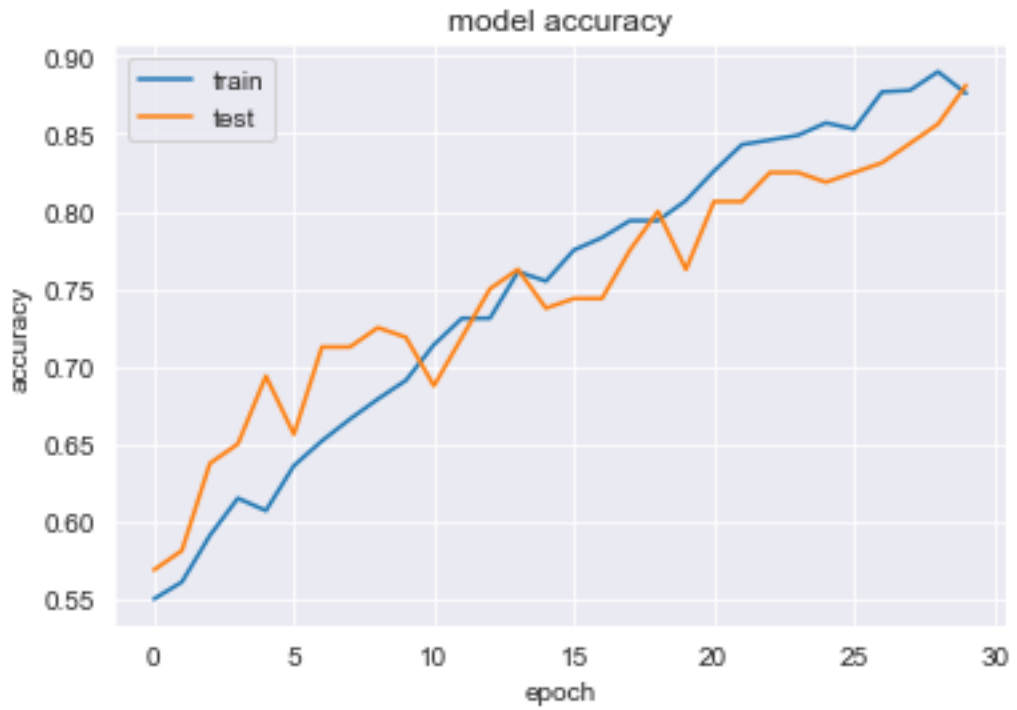
```
accuracy: 0.8730 - val_loss: 0.3202 - val_accuracy: 0.8562
Epoch 29/30
200/200 [==============================] - 1s 3ms/step - loss: 0.3126 -
accuracy: 0.8710 - val_loss: 0.3284 - val_accuracy: 0.8625
Epoch 30/30
200/200 [==============================] - 1s 4ms/step - loss: 0.2983 -
accuracy: 0.8840 - val_loss: 0.2931 - val_accuracy: 0.8750
32/32 [==============================] - 0s 3ms/step - loss: 0.2573 - accuracy:
0.9070
Accuracy: 90.70
```

[122]:
```
# list all data in history
print(history.history.keys())
```

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

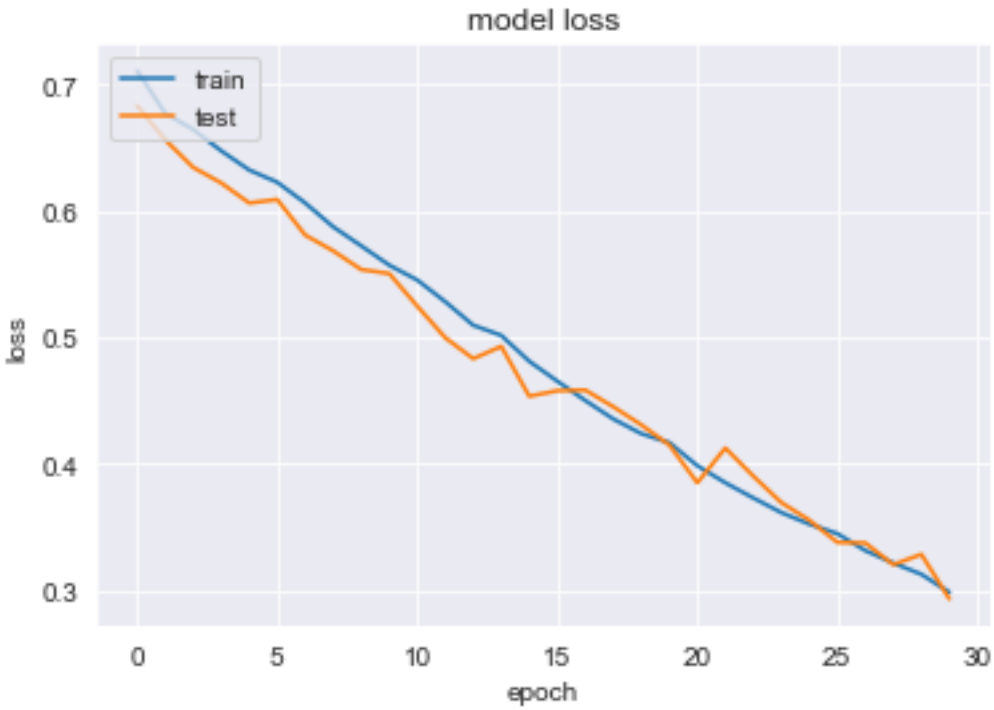
Plot accuracy wrt number of epochs

[48]:
```
# 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', 'test'], loc='upper left')
plt.show()
```



Plot training loss wrt number of epochs

[52]:
```
# 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', 'test'], loc='upper left')
plt.show()
```



[53]:
```
y_pred=model_nn.predict(X_test)
```

[54]:
```
confusion_matrix(y_test, y_pred.round())
```

[54]:
```
array([[ 81,  11],
       [  6, 102]])
```

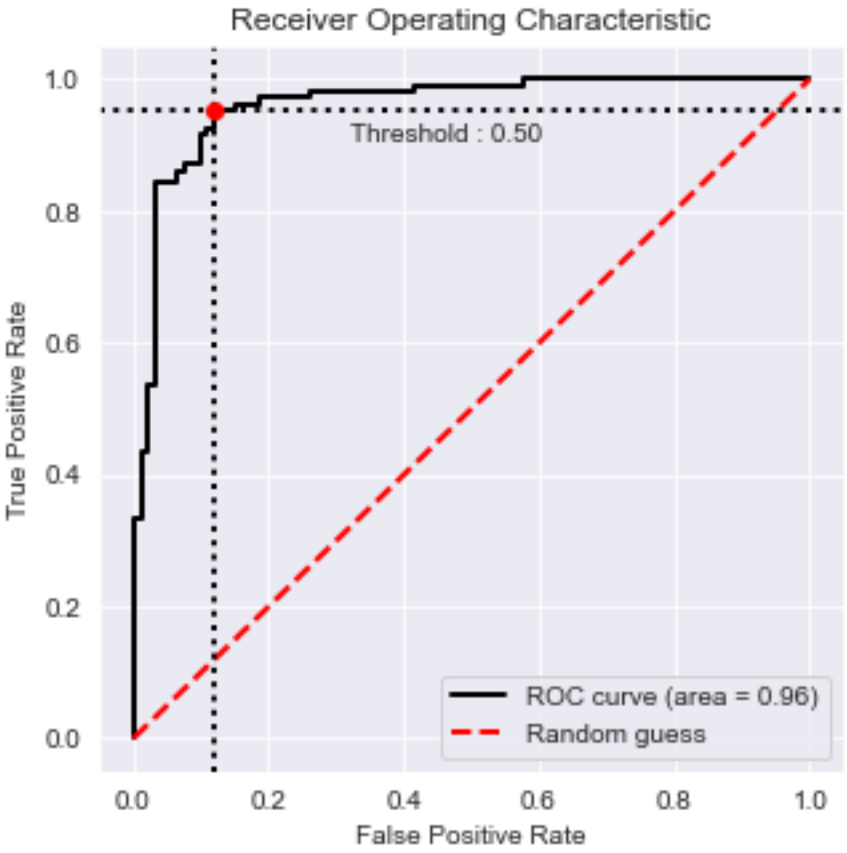
Plot the ROC (Receiver Operating Characteristic) Curve (more info on ROC could be found here)

[56]:
```
from plot_metric.functions import BinaryClassification
# Visualisation with plot_metric
```

```
bc = BinaryClassification(y_test, y_pred, labels=["Class 1", "Class 2"])
bc_bdt = BinaryClassification(y_test, y_hat, labels=["Class 1", "Class 2"])

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