Data Reading and Preprocessing

```python
import os
import copy
import itertools
import numpy as np
import seaborn as sns
import scipy
import tables as tb
from mpl_toolkits.mplot3d import Axes3D
import warnings
warnings.filterwarnings('ignore')

%matplotlib inline
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```python
# Reading data
data = os.path.join('drive','My Drive','dl_data','QIS_EXAM_200Events')
test_input = np.load(os.path.join(data,'test_input.npy'),allow_pickle=True)
train_input = np.load(os.path.join(data,'training_input.npy'),allow_pickle=True)

test_input = test_input[()]
train_input = train_input[()]

# Segregating the test and train data
y_train = np.array([np.array([0])]*50 + [np.array([1])]*50)
y_test = np.array([np.array([0])]*50 + [np.array([1])]*50)

x_train = np.concatenate([train_input['0'],train_input['1']])
x_test = np.concatenate([test_input['0'],test_input['1']])
```

```python
# Shuffling the dataset

def get_indices(_seed, _len):
    np.random.seed(_seed)
    indices = np.random.permutation(_len)
```

```
    indices = np.random.permutation(_len)
    return indices

indices_train = get_indices(0,len(x_train))
indices_test = get_indices(1,len(x_test))

x_train = x_train[indices_train]
y_train = y_train[indices_train]

x_test = x_train[indices_test]
y_test = y_test[indices_test]

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
x_train_scale = scaler.fit_transform(x_train)
x_test_scale = scaler.fit_transform(x_test)
```

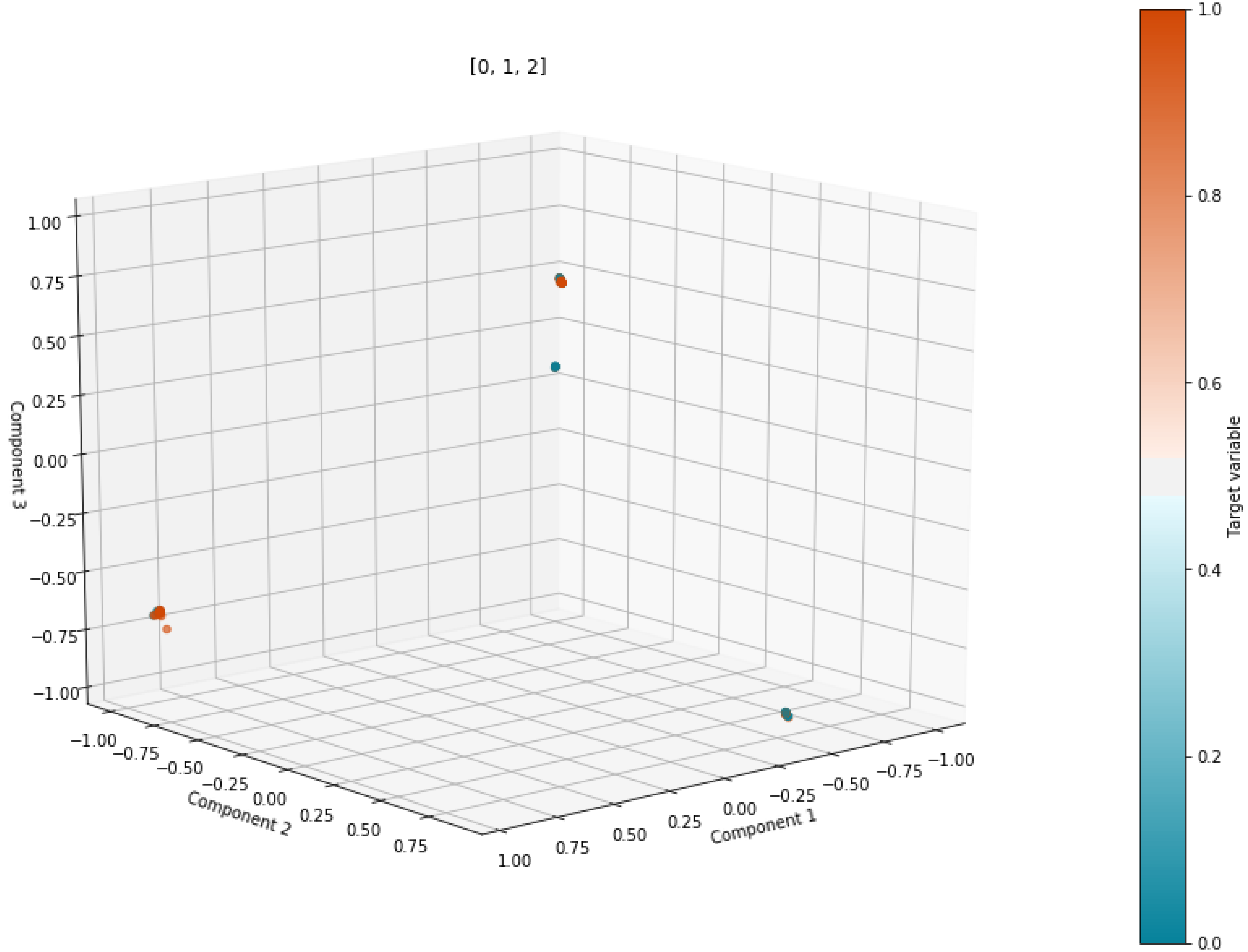
Exploratory Data Analysis

```
# 3D visualization of data points

CMAP = sns.diverging_palette(220, 20, s=99, as_cmap=True, n=2500)

def plot3D(X, target, elev=0, azim=0, title=None, sub=111):
    x = X[:, 0]
    y = X[:, 1]
    z = X[:, 2]

    fig = plt.figure(figsize=(12, 8))
    ax = Axes3D(fig)
    mappab = ax.scatter(x, y, z, c=target, cmap=CMAP)

    if title is not None:
        ax.set_title(title)
    ax.set_xlabel('Component 1')
    ax.set_ylabel('Component 2')
    ax.set_zlabel('Component 3')

    # to change your point of view
    ax.view_init(elev=elev, azim=azim)
    fig.colorbar(mappable=mappab, label='Target variable')
```

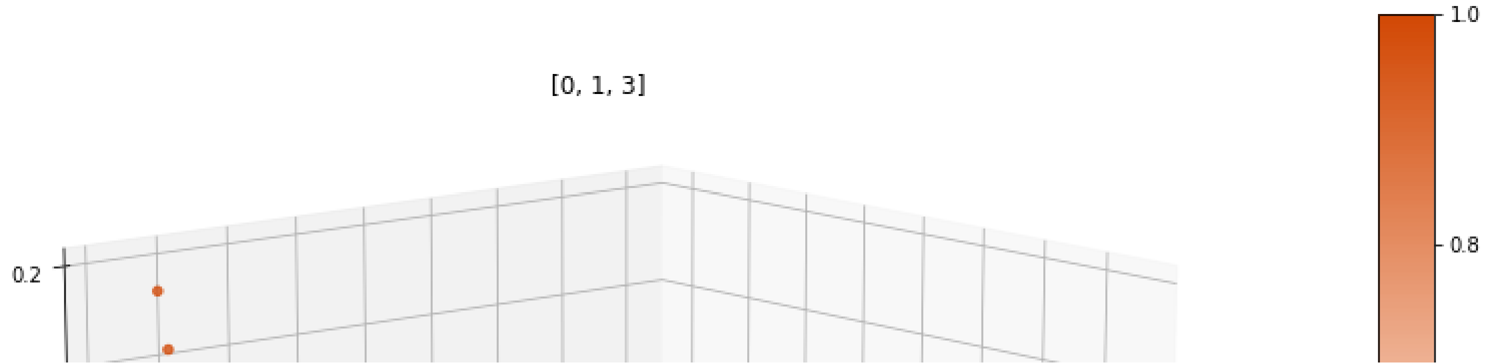
```
    fig.colorbar(mappable=mappab, label='Target variable')
    plt.show()

```

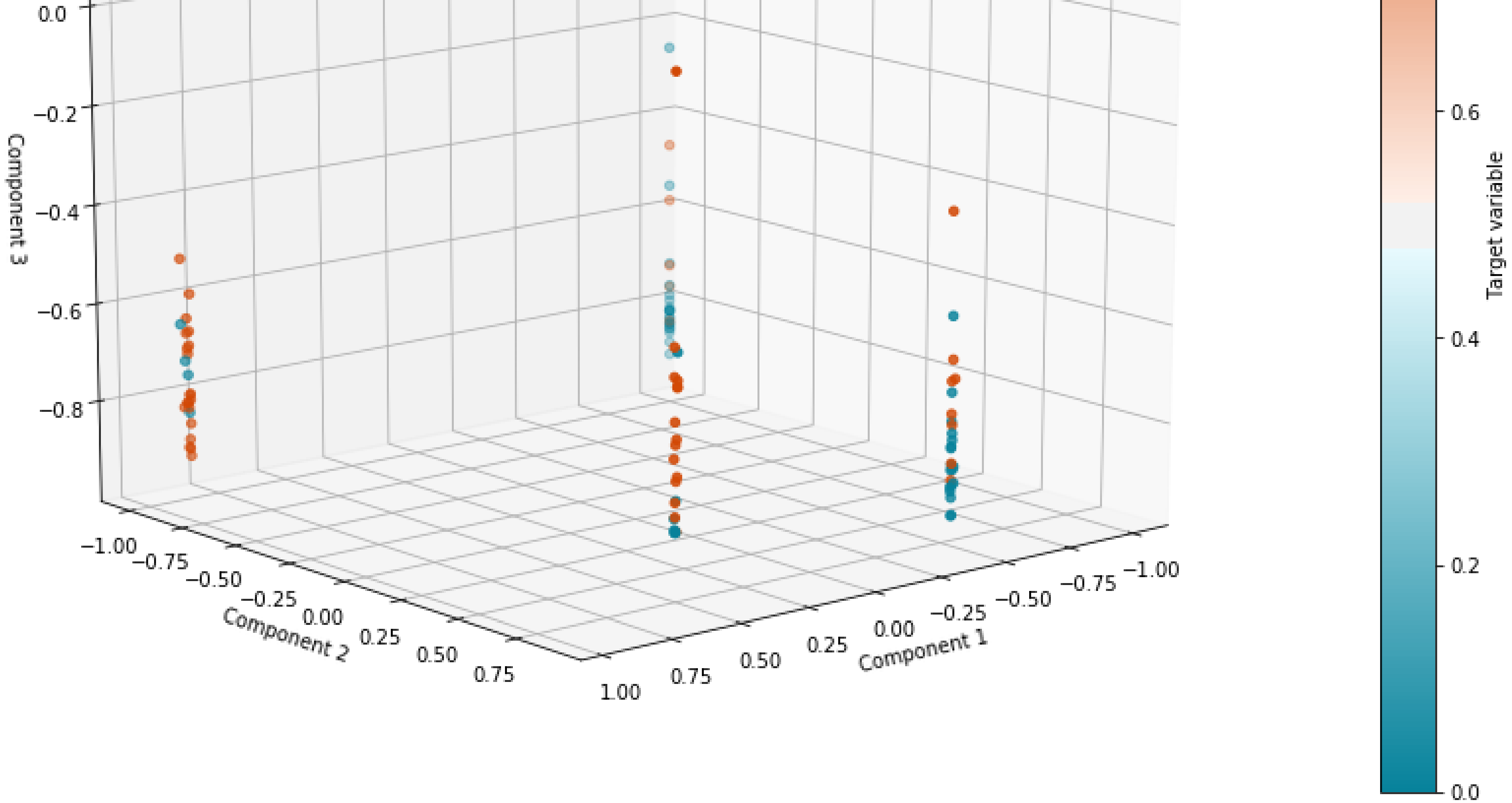
```
# Visualizing all the combinations of data to get a clear picture of classes

y = y_train.squeeze(1)
for i in range(5):
    for j in range(i+1,5):
        for k in range(j+1,5):
            arr = x_train[:,[i,j,k]]
            plot3D(arr, y, elev=15, azim=50, title=str([i,j,k]))
```

[0, 1, 2]

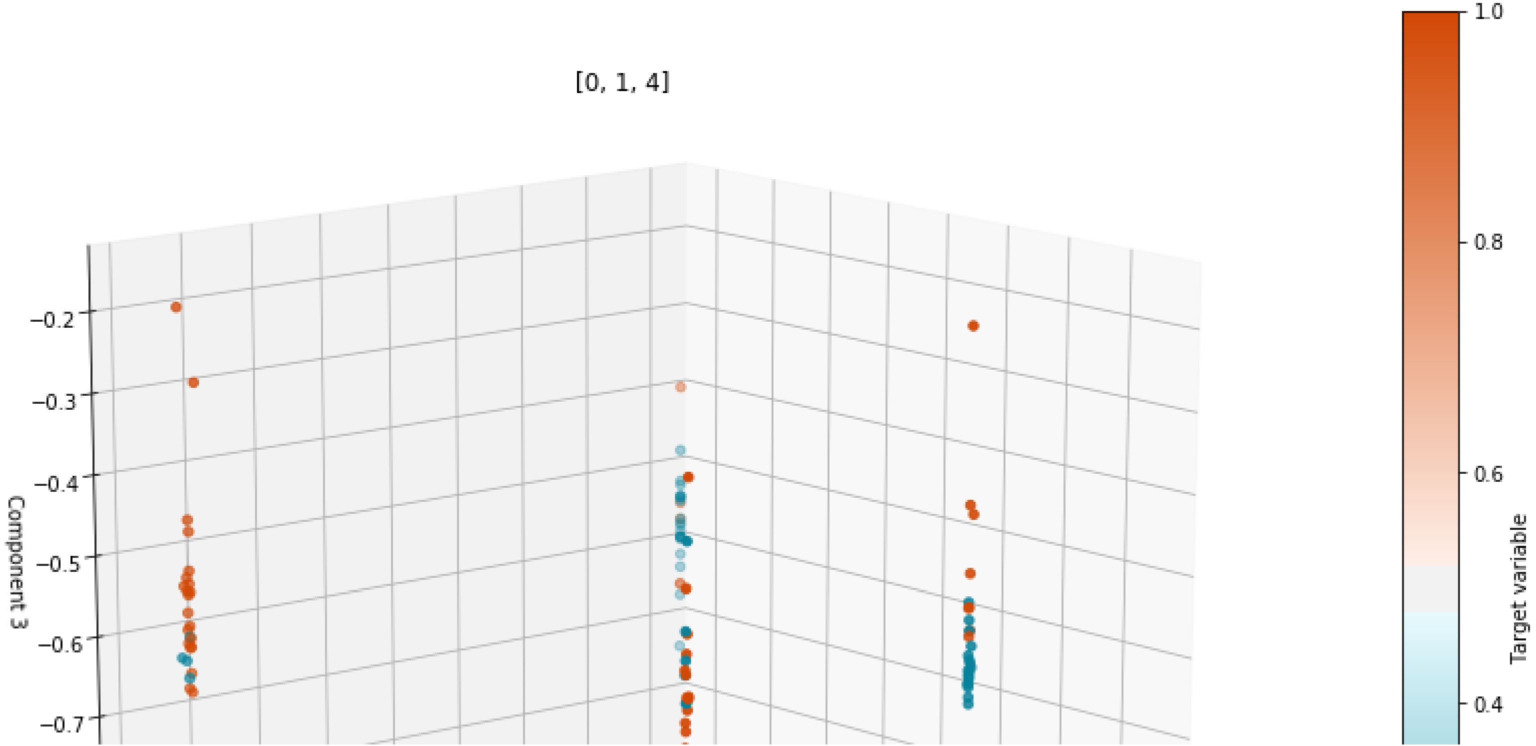

1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Component 3
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
Component 2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Component 1
1.0
0.8
0.6
0.4
0.2
0.0
Target variable

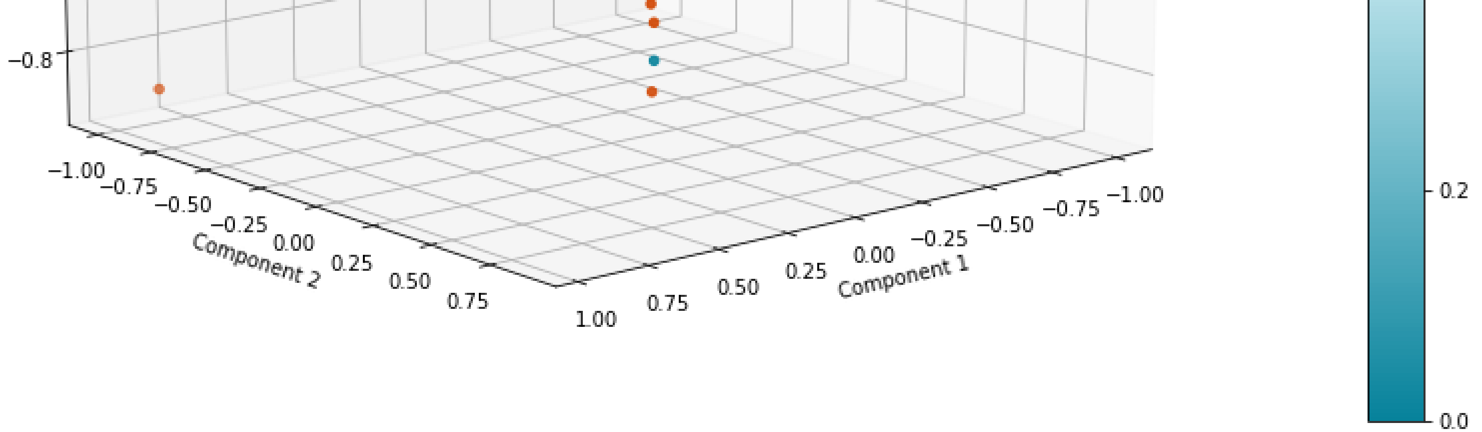
[0, 1, 3]


0.2
1.0
0.8

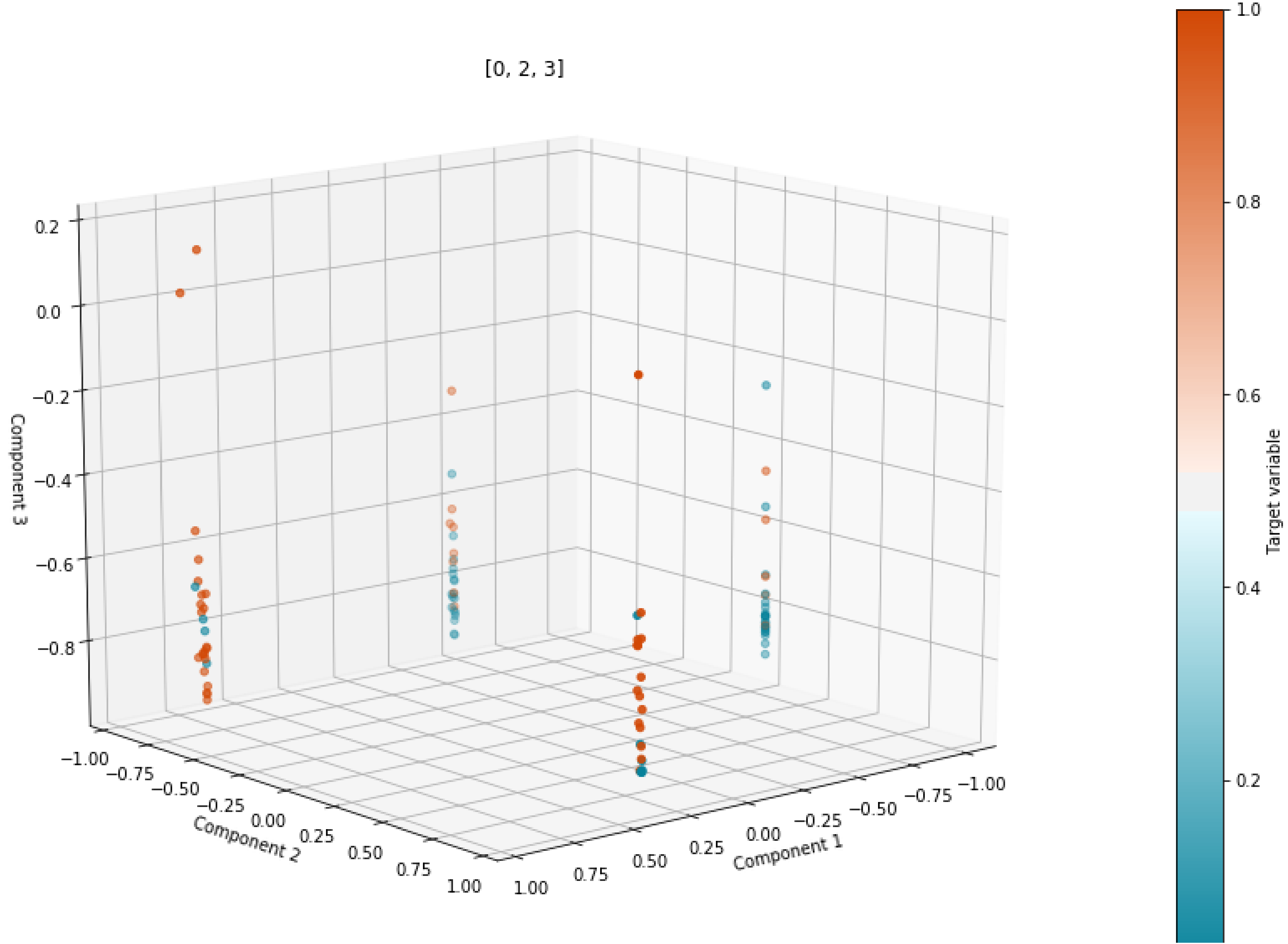

Component 3
0.0
−0.2
−0.4
−0.6
−0.8
Component 2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Component 1
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Target variable
0.0
0.2
0.4
0.6

[0, 1, 4]


Component 3
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
Target variable
1.0
0.8
0.6
0.4

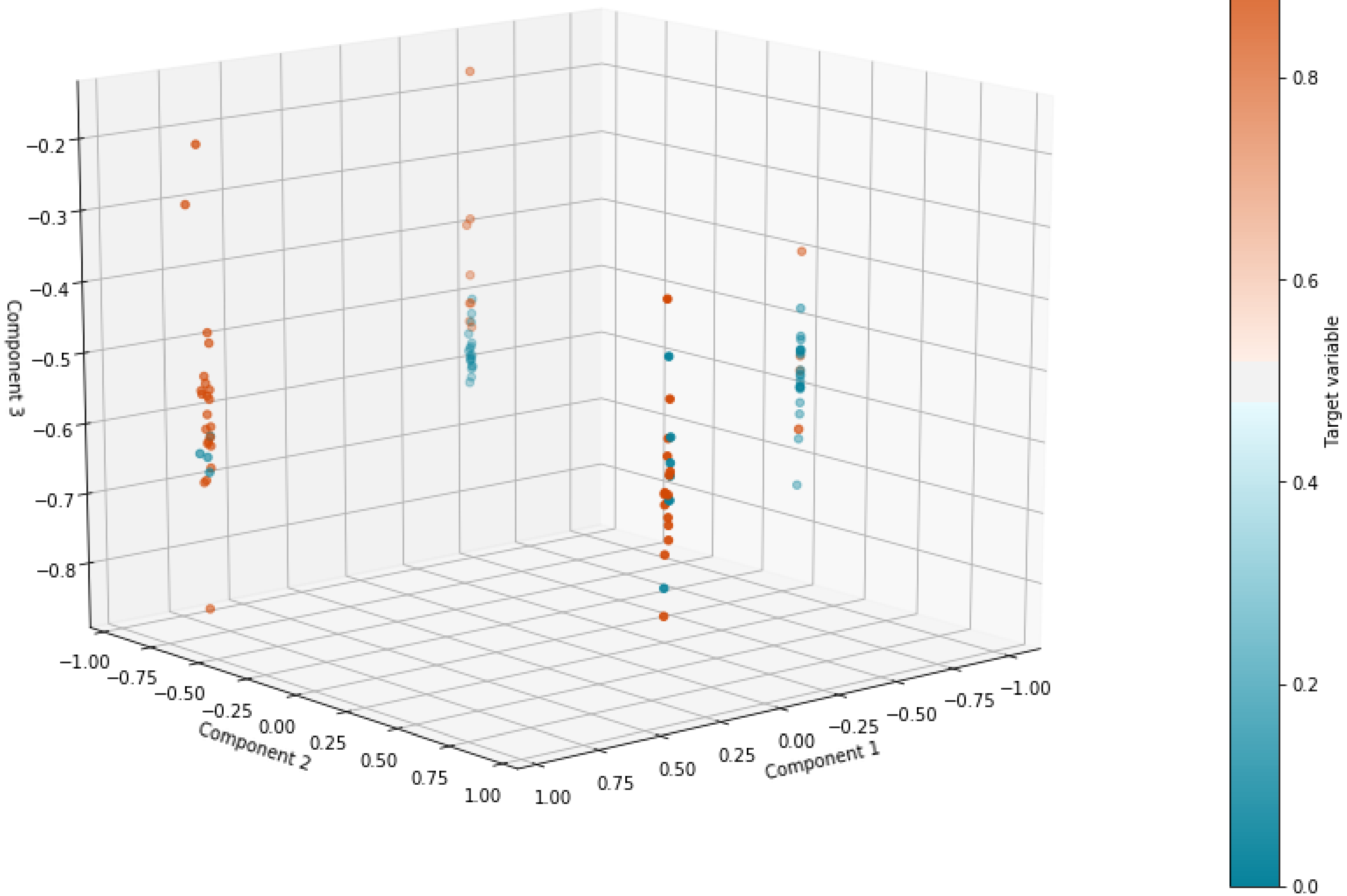

Component 2
Component 1
−0.8
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
0.0
0.2

[0, 2, 3]

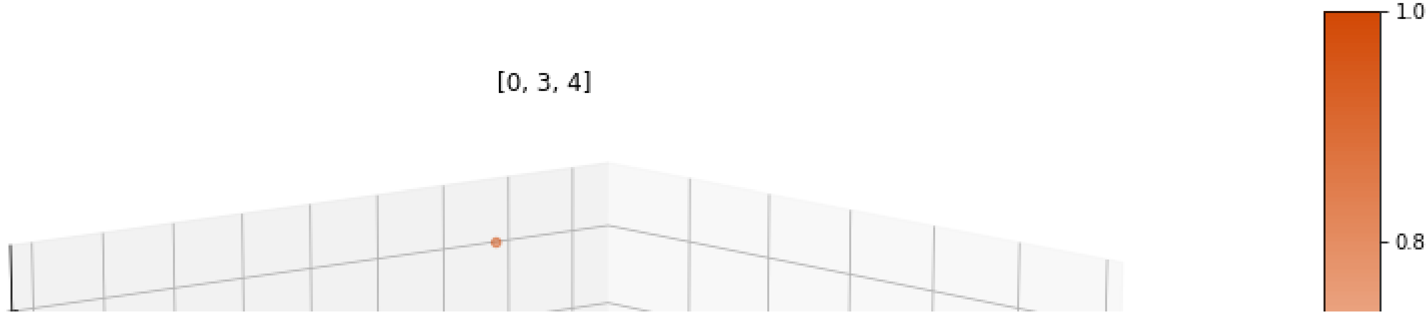

Component 3
Component 2
Component 1
Target variable
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
1.0
0.8
0.6
0.4
0.2

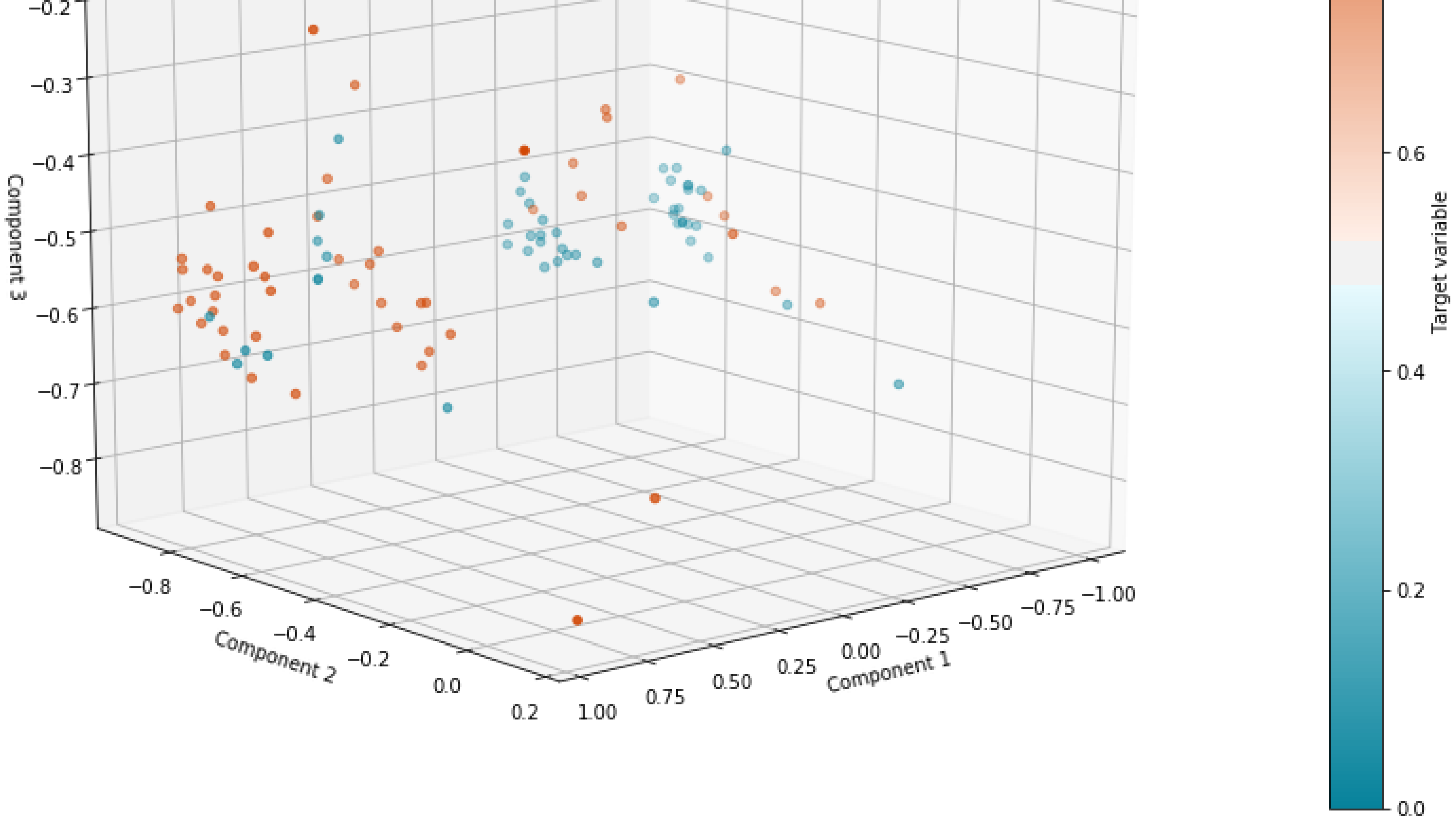

0.0

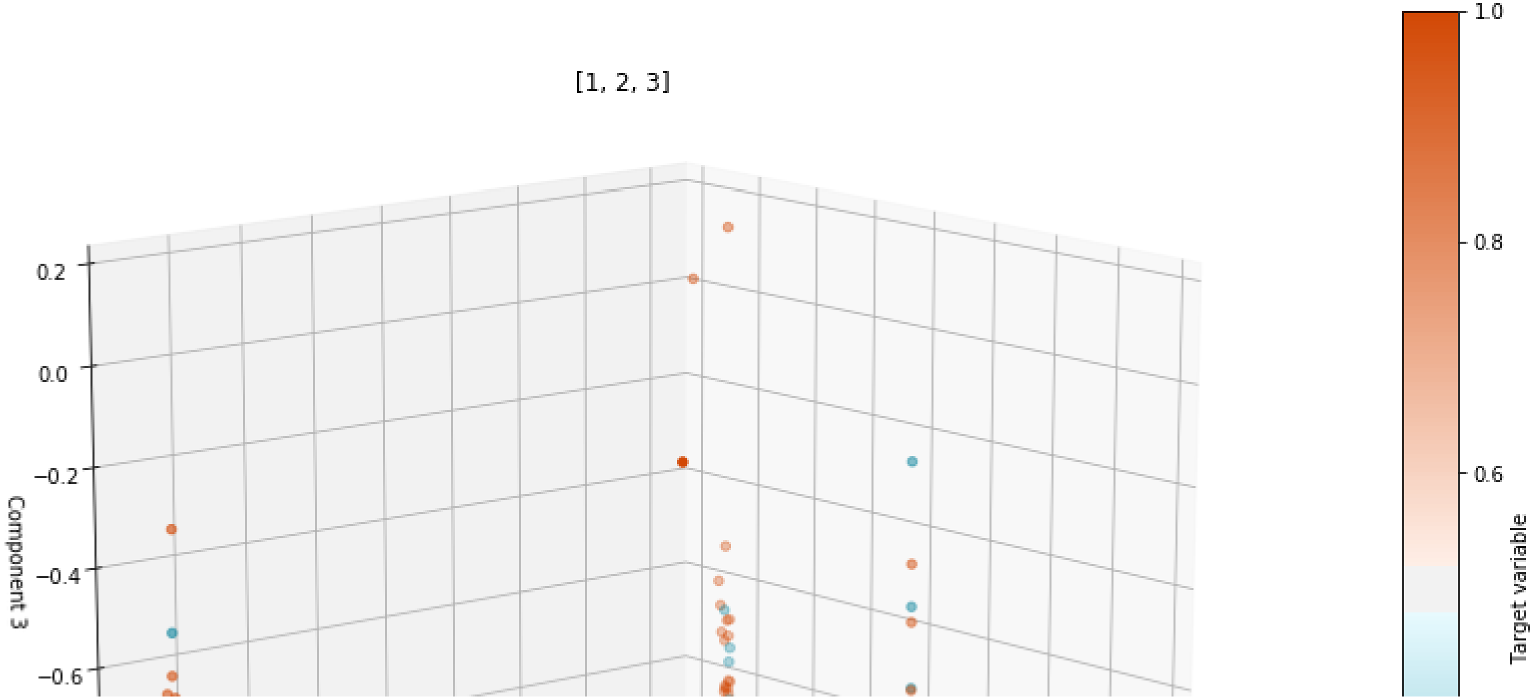
[0, 2, 4]

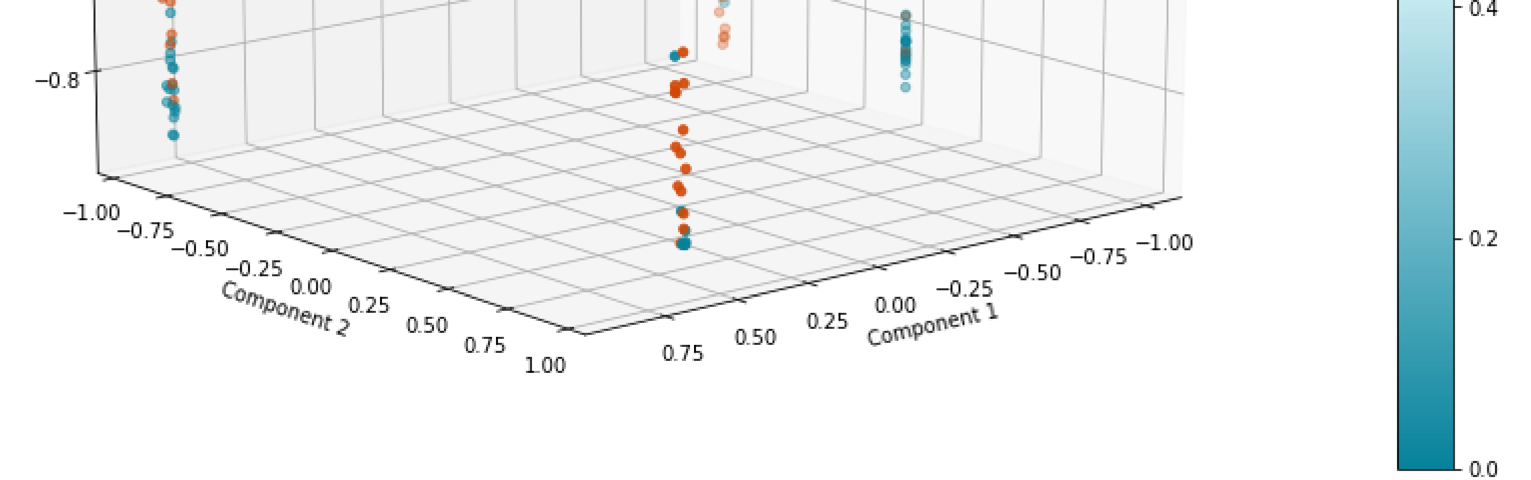

−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
−0.8
Component 3
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Component 2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Component 1
1.0
0.8
0.6
0.4
0.2
0.0
Target variable

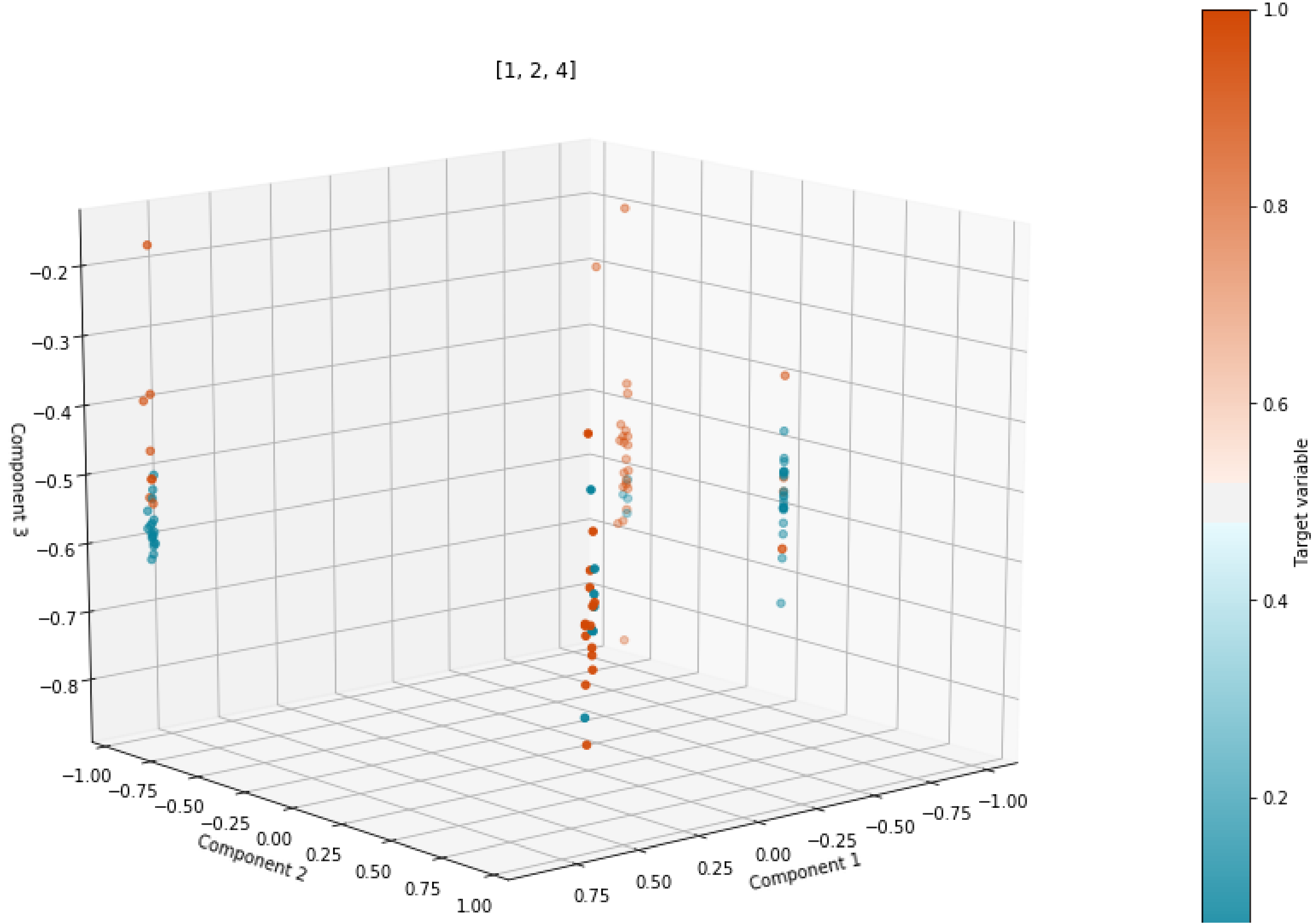
[0, 3, 4]


1.0
0.8


Component 1
Component 2
Component 3
Target variable

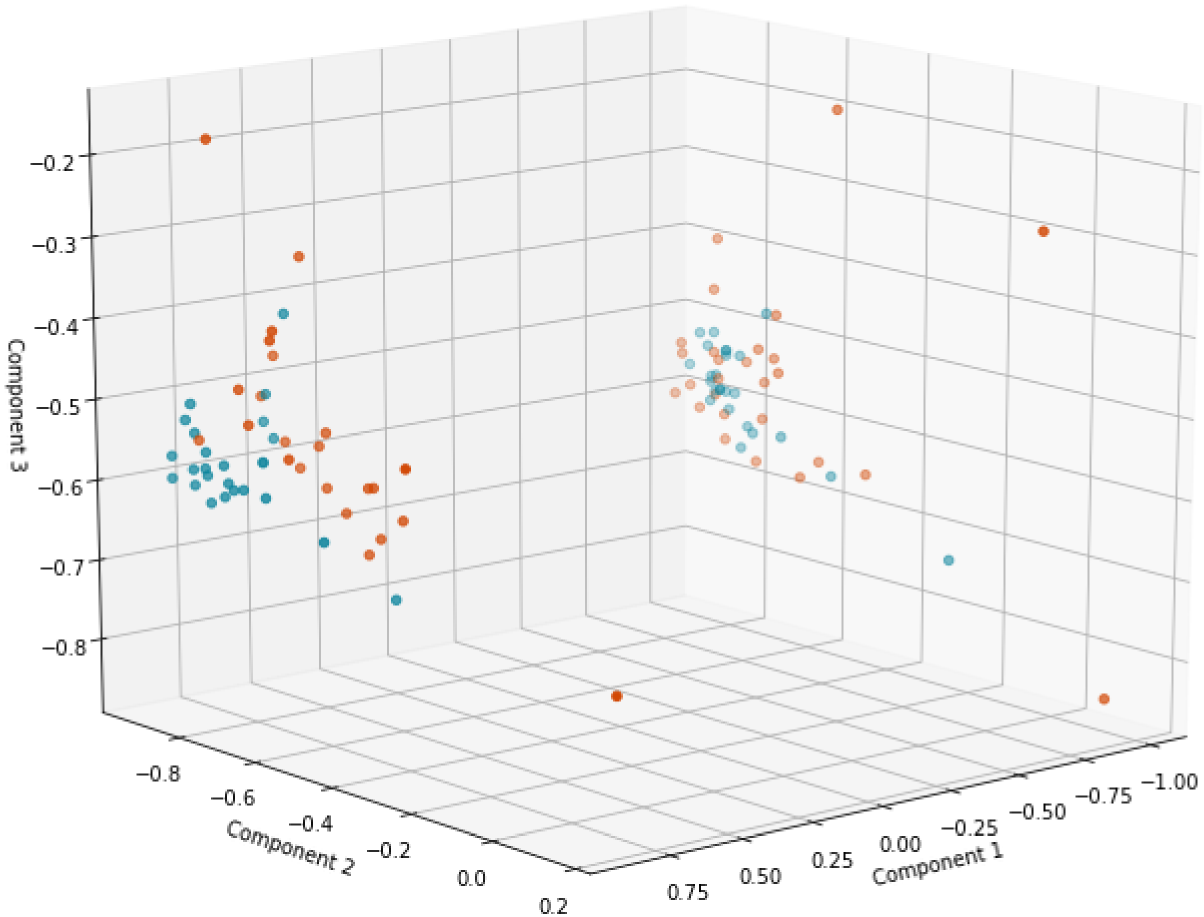

[1, 2, 3]
Component 3
Target variable

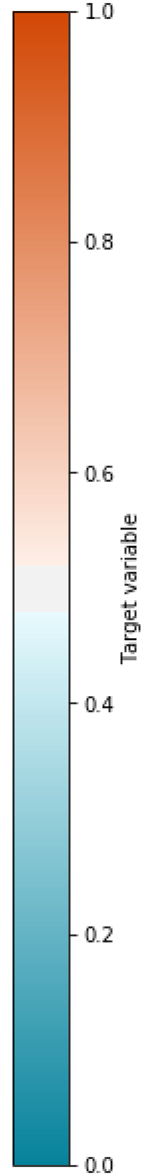

−0.8
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
Component 2
0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00
Component 1
0.4
0.2
0.0


[1, 2, 4]
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
−0.8
Component 3
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00
Component 2
0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75 −1.00
Component 1
1.0
0.8
0.6
0.4
0.2
Target variable

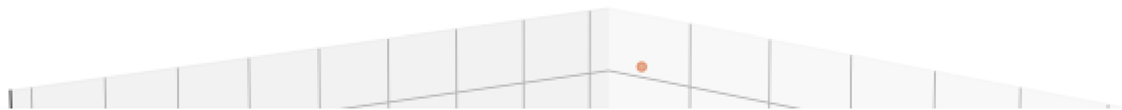

0.0

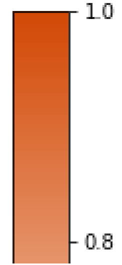
[1, 3, 4]

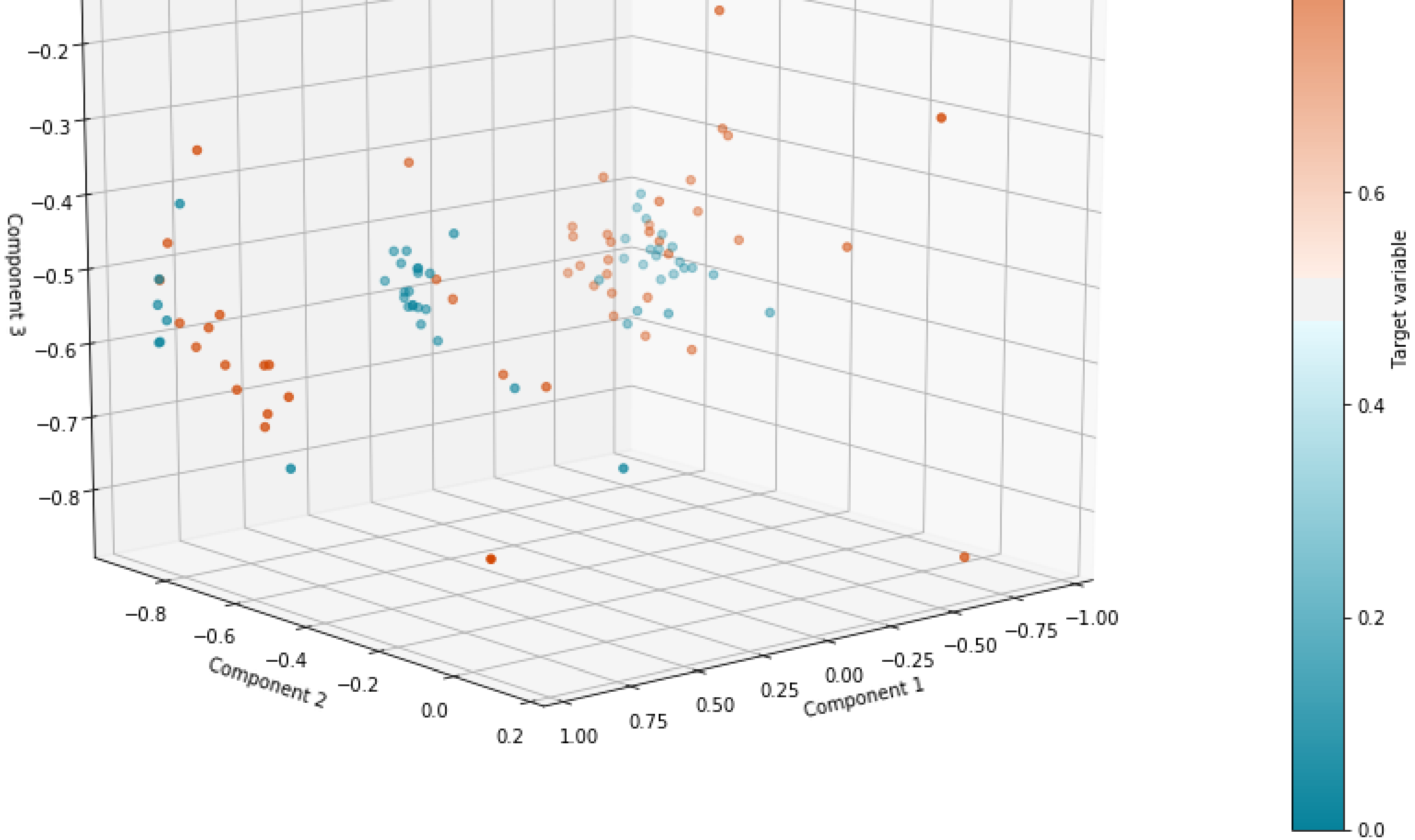

Component 1
Component 2
Component 3
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
−0.8
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


Target variable
1.0
0.8
0.6
0.4
0.2
0.0

[2, 3, 4]

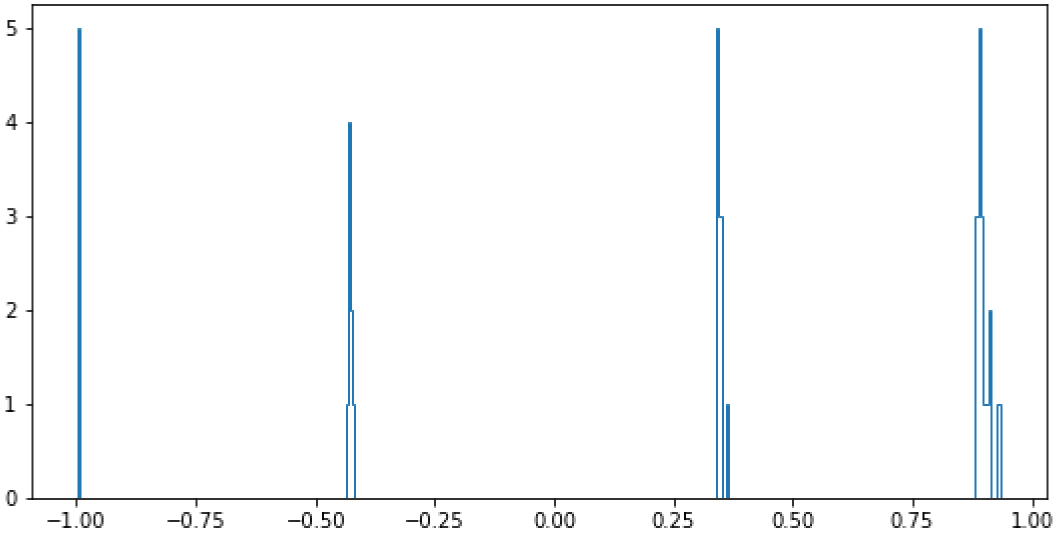


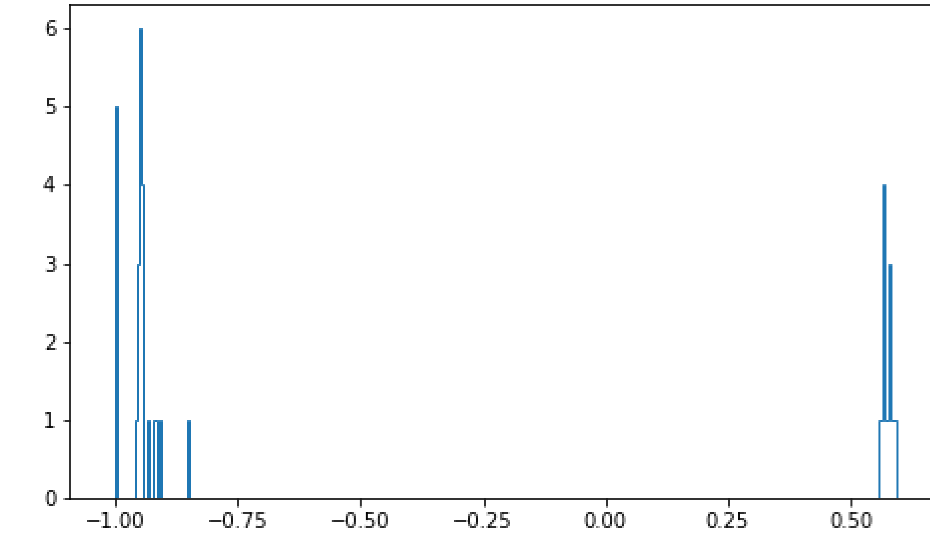

1.0
0.8

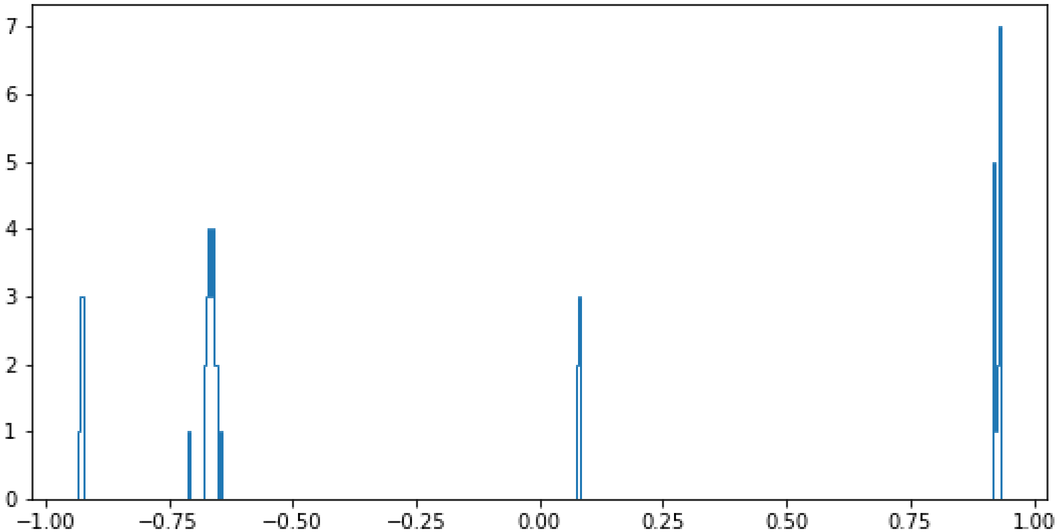


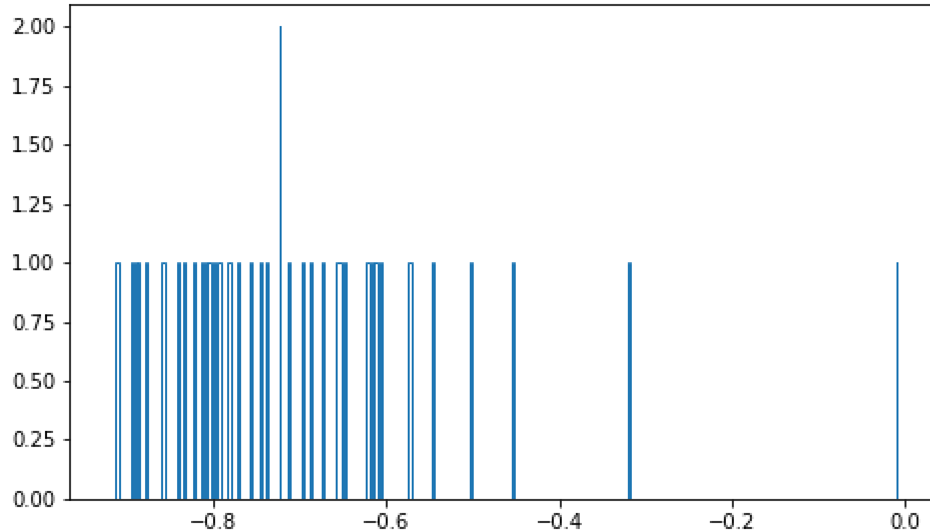

```
# signal basetrack distribution along the axis
axis = 'X'

fig = plt.figure(figsize = [20, 10])
fig.add_subplot(221)
plt.hist(x_train[y==1][:,[0]], bins=500, histtype='step')
fig.add_subplot(222)
plt.hist(x_train[y==1][:,[1]], bins=500, histtype='step')
fig.add_subplot(223)
plt.hist(x_train[y==1][:,[2]], bins=500, histtype='step')
fig.add_subplot(224)
plt.hist(x_train[y==1][:,[3]], bins=500, histtype='step')
plt.show()
fig.add_subplot(321)
fig = plt.figure(figsize = [10, 5])
```

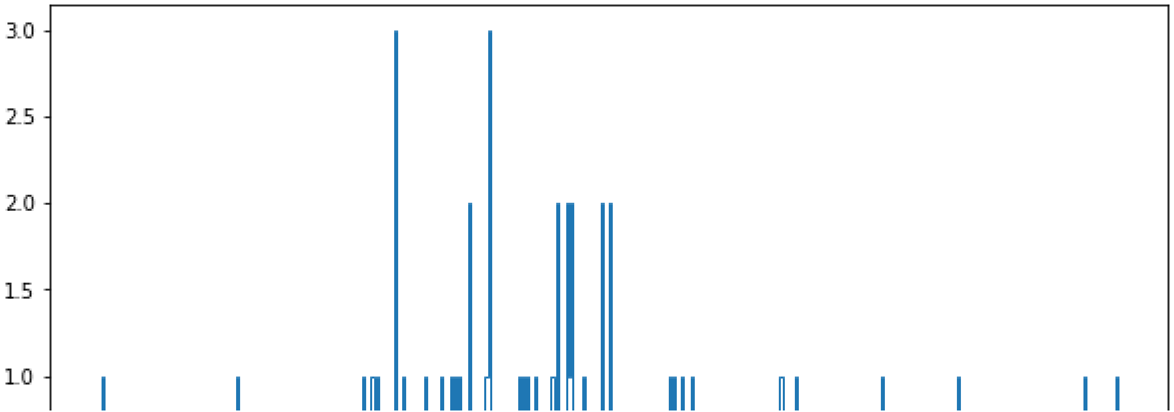
```
plt.hist(x_train[y==1][:,[4]], bins=500, histtype='step')
plt.show()
```

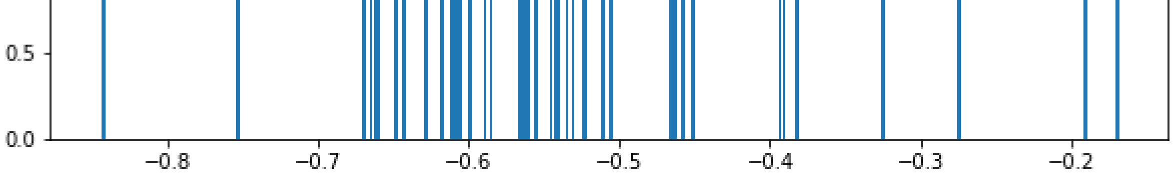

0 1 2 3 4 5
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


0 1 2 3 4 5 6
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50

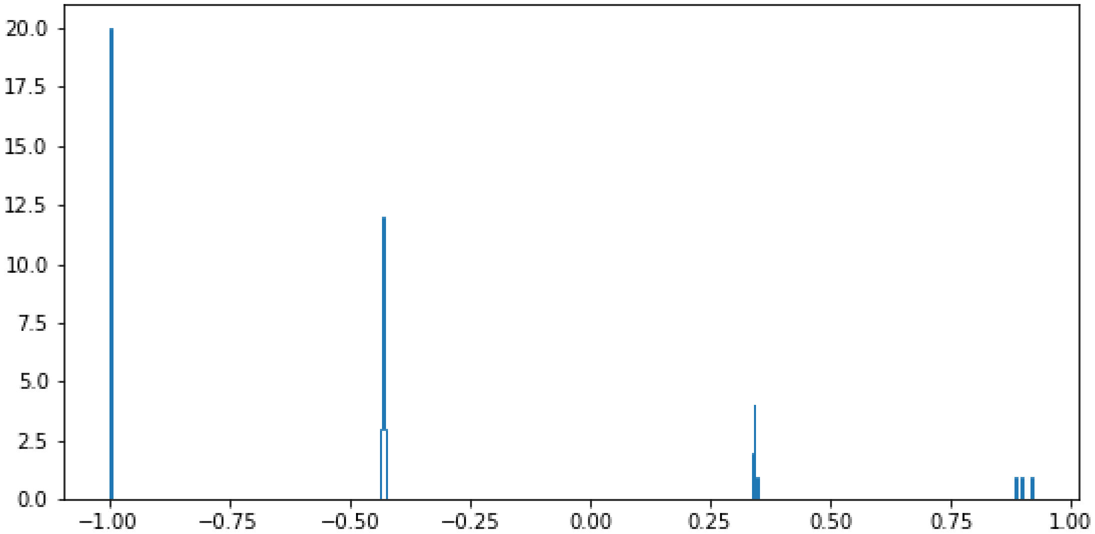

0 1 2 3 4 5 6 7
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

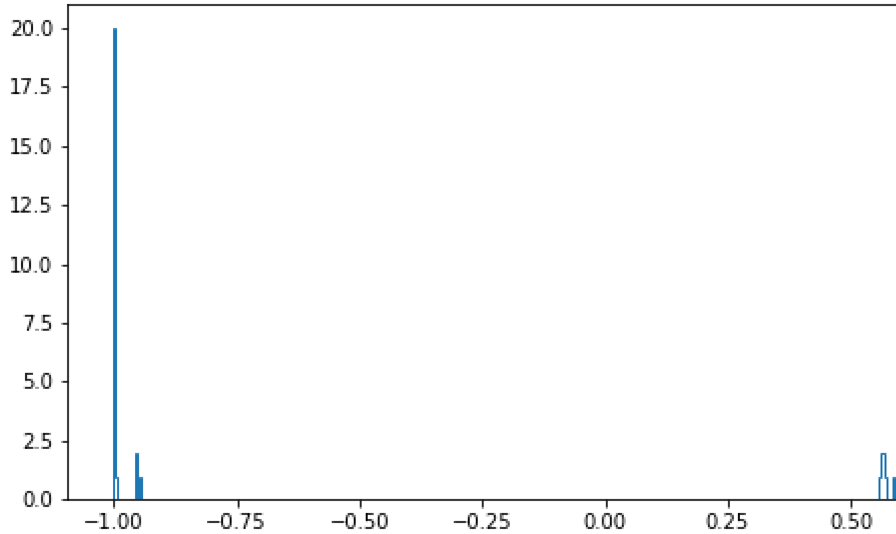

0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00
−0.8 −0.6 −0.4 −0.2 0.0

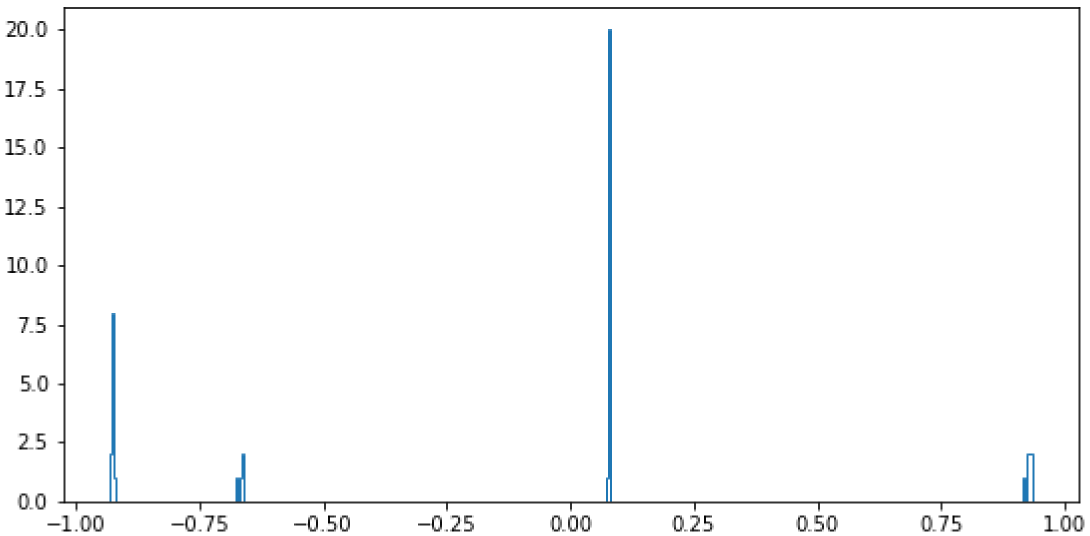

1.0 1.5 2.0 2.5 3.0

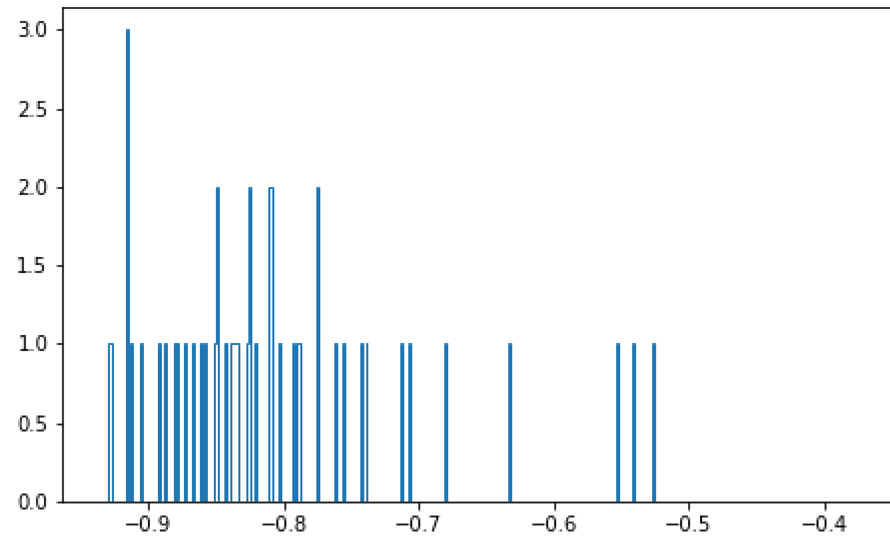


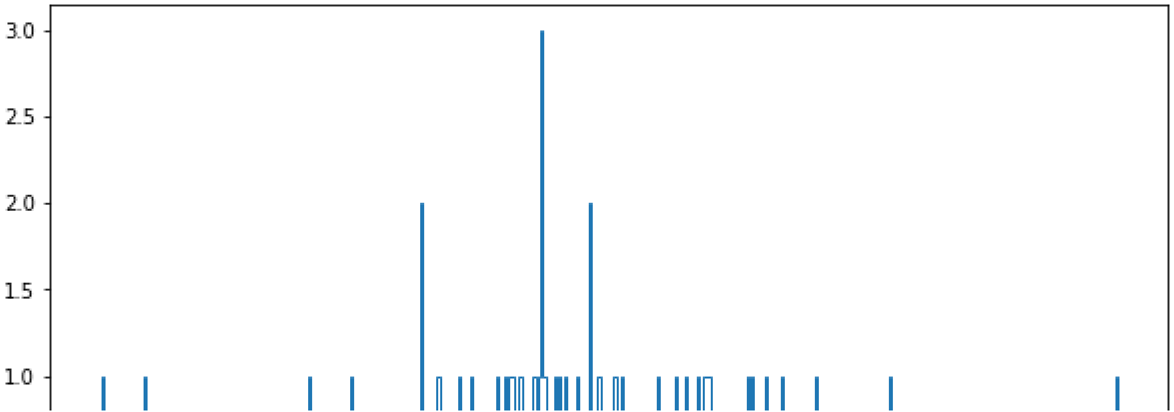

```
# background basetrack distribution along the axis
axis = 'X'

fig = plt.figure(figsize = [20, 10])
fig.add_subplot(221)
plt.hist(x_train[y==0][:,[0]], bins=500, histtype='step')
fig.add_subplot(222)
plt.hist(x_train[y==0][:,[1]], bins=500, histtype='step')
fig.add_subplot(223)
plt.hist(x_train[y==0][:,[2]], bins=500, histtype='step')
fig.add_subplot(224)
plt.hist(x_train[y==0][:,[3]], bins=500, histtype='step')
plt.show()
fig.add_subplot(321)
fig = plt.figure(figsize = [10, 5])
plt.hist(x_train[y==0][:,[4]], bins=500, histtype='step')
plt.show()
```

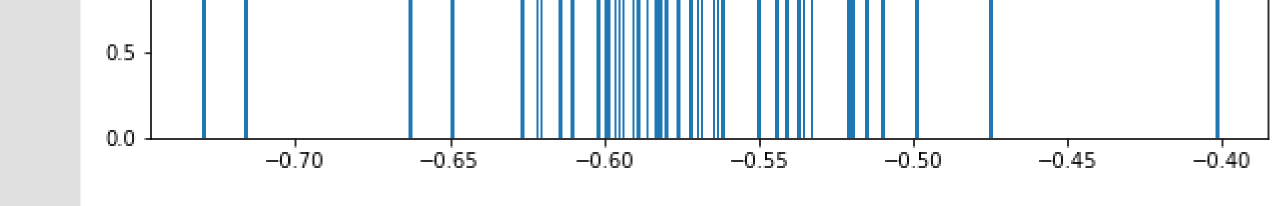

20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00


3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.9 −0.8 −0.7 −0.6 −0.5 −0.4


3.0
2.5
2.0
1.5
1.0



Classification

```
import xgboost as xg
from xgboost import XGBClassifier
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler, Normalizer
from sklearn import svm
from sklearn import metrics
from sklearn.metrics import (
    classification_report,
    confusion_matrix, roc_curve,
    roc_auc_score,
    log_loss
    )

from keras.layers.core import Dense, Activation, Dropout
from keras.models import Sequential
from keras.optimizers import Adam
from keras.utils import np_utils
from keras.callbacks import EarlyStopping, ModelCheckpoint

```

```
# utility function for plotting ROC-AUC
def plot_metrics(y_true, y_pred):
    fpr, tpr, thresholds = roc_curve(y_true, y_pred)
    roc_auc = roc_auc_score(y_true, y_pred)

    plt.plot(fpr, tpr, label='ROC AUC=%f' % roc_auc)
    plt.xlabel("FPR")
    plt.ylabel("TPR")
```

```
    plt.legend()
    plt.title("ROC Curve")

# Utility function to add noise
def add_noise(array, level=0.15, random_seed=34):
    numpy.random.seed(random_seed)
    return level * numpy.random.random(size=array.size) + (1 - level) * array

# Utility function to plot the confusion matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Keras DNN

```python
# Creating a simple deep neural network
def nn_model(input_dim):
    model = Sequential()
    model.add(Dense(256, input_dim=input_dim))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))

    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))

    model.add(Dense(64))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))

    model.add(Dense(1))
    model.add(Activation('sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer=Adam(),metrics=['accuracy'])
    return model
```

```python
# Concatenating the dataset to increase the dataset
x_train_nn = np.concatenate([x_train,x_test])
y_train_nn = np.concatenate([y_train,y_test])

# Normalizing the dataset
transformer = Normalizer()
X_train_norm = transformer.fit_transform(x_train_nn)
X_train_norm = x_train_nn

# Using 20% data for testing
x_test_nn = X_train_norm[int(0.8*len(X_train_norm)):]
y_test_nn = y_train_nn[int(0.8*len(X_train_norm)):]

# Using 80% data for training
x_train_nn = X_train_norm[:int(0.8*len(X_train_norm))]
y_train_nn = y_train_nn[:int(0.8*len(X_train_norm))]


callbacks = [EarlyStopping(monitor='val_loss', min_delta=0, patience=100, verbose=0, mode='auto'),
             ModelCheckpoint('output/{val_loss:.4f}.hdf5', monitor='val_loss', verbose=2, save_best_only=True, mode='auto')]
```

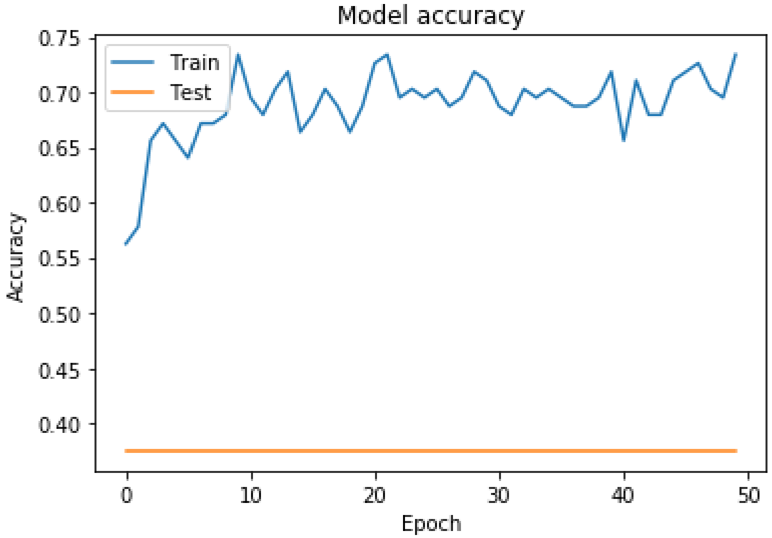
```
# Creating model
nn = nn_model(x_train_nn.shape[1])

# Training. Using 20% of data for validation
history = nn.fit(x_train_nn, y_train_nn, validation_split=0.2, epochs=50,
       verbose=1, batch_size=8, shuffle=True)

```

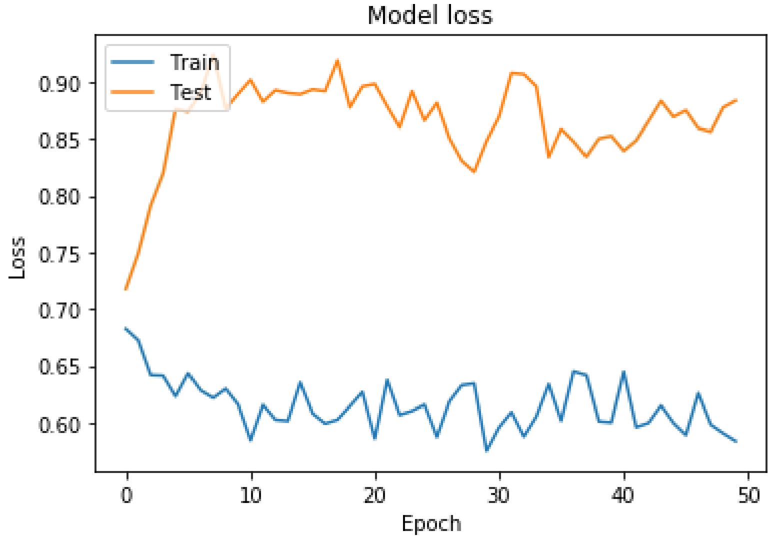
```
Train on 128 samples, validate on 32 samples
Epoch 1/50
128/128 [==============================] - 1s 5ms/step - loss: 0.6825 - acc: 0.5625 - val_loss: 0.7178 - val_acc: 0.3750
Epoch 2/50
128/128 [==============================] - 0s 901us/step - loss: 0.6725 - acc: 0.5781 - val_loss: 0.7493 - val_acc: 0.3750
Epoch 3/50
128/128 [==============================] - 0s 786us/step - loss: 0.6417 - acc: 0.6562 - val_loss: 0.7913 - val_acc: 0.3750
Epoch 4/50
128/128 [==============================] - 0s 863us/step - loss: 0.6413 - acc: 0.6719 - val_loss: 0.8197 - val_acc: 0.3750
Epoch 5/50
128/128 [==============================] - 0s 826us/step - loss: 0.6230 - acc: 0.6562 - val_loss: 0.8762 - val_acc: 0.3750
Epoch 6/50
128/128 [==============================] - 0s 779us/step - loss: 0.6432 - acc: 0.6406 - val_loss: 0.8738 - val_acc: 0.3750
Epoch 7/50
128/128 [==============================] - 0s 761us/step - loss: 0.6282 - acc: 0.6719 - val_loss: 0.8906 - val_acc: 0.3750
Epoch 8/50
128/128 [==============================] - 0s 852us/step - loss: 0.6217 - acc: 0.6719 - val_loss: 0.9242 - val_acc: 0.3750
Epoch 9/50
128/128 [==============================] - 0s 830us/step - loss: 0.6298 - acc: 0.6797 - val_loss: 0.8764 - val_acc: 0.3750
Epoch 10/50
128/128 [==============================] - 0s 848us/step - loss: 0.6160 - acc: 0.7344 - val_loss: 0.8894 - val_acc: 0.3750
Epoch 11/50
128/128 [==============================] - 0s 791us/step - loss: 0.5845 - acc: 0.6953 - val_loss: 0.9018 - val_acc: 0.3750
Epoch 12/50
128/128 [==============================] - 0s 860us/step - loss: 0.6157 - acc: 0.6797 - val_loss: 0.8825 - val_acc: 0.3750
Epoch 13/50
128/128 [==============================] - 0s 1ms/step - loss: 0.6025 - acc: 0.7031 - val_loss: 0.8927 - val_acc: 0.3750
Epoch 14/50
128/128 [==============================] - 0s 966us/step - loss: 0.6015 - acc: 0.7188 - val_loss: 0.8902 - val_acc: 0.3750
Epoch 15/50
128/128 [==============================] - 0s 871us/step - loss: 0.6353 - acc: 0.6641 - val_loss: 0.8891 - val_acc: 0.3750
Epoch 16/50
128/128 [==============================] - 0s 836us/step - loss: 0.6080 - acc: 0.6797 - val_loss: 0.8934 - val_acc: 0.3750
Epoch 17/50
128/128 [==============================] - 0s 756us/step - loss: 0.5992 - acc: 0.7031 - val_loss: 0.8919 - val_acc: 0.3750
Epoch 18/50
128/128 [==============================] - 0s 775us/step - loss: 0.6026 - acc: 0.6875 - val_loss: 0.9190 - val_acc: 0.3750
Epoch 19/50
128/128 [==============================] - 0s 795us/step - loss: 0.6146 - acc: 0.6641 - val_loss: 0.8777 - val_acc: 0.3750
Epoch 20/50
128/128 [==============================] - 0s 839us/step - loss: 0.6270 - acc: 0.6875 - val_loss: 0.8961 - val_acc: 0.3750
Epoch 21/50
128/128 [==============================] - 0s 876us/step - loss: 0.5860 - acc: 0.7266 - val_loss: 0.8982 - val_acc: 0.3750
Epoch 22/50
```

```
Epoch 22/50
128/128 [==============================] - 0s 871us/step - loss: 0.6375 - acc: 0.7344 - val_loss: 0.8786 - val_acc: 0.3750
Epoch 23/50
128/128 [==============================] - 0s 767us/step - loss: 0.6067 - acc: 0.6953 - val_loss: 0.8600 - val_acc: 0.3750
Epoch 24/50
128/128 [==============================] - 0s 835us/step - loss: 0.6102 - acc: 0.7031 - val_loss: 0.8919 - val_acc: 0.3750
Epoch 25/50
128/128 [==============================] - 0s 909us/step - loss: 0.6159 - acc: 0.6953 - val_loss: 0.8661 - val_acc: 0.3750
Epoch 26/50
128/128 [==============================] - 0s 835us/step - loss: 0.5874 - acc: 0.7031 - val_loss: 0.8816 - val_acc: 0.3750
Epoch 27/50
128/128 [==============================] - 0s 861us/step - loss: 0.6186 - acc: 0.6875 - val_loss: 0.8506 - val_acc: 0.3750
Epoch 28/50
128/128 [==============================] - 0s 797us/step - loss: 0.6329 - acc: 0.6953 - val_loss: 0.8307 - val_acc: 0.3750
Epoch 29/50
128/128 [==============================] - 0s 769us/step - loss: 0.6344 - acc: 0.7188 - val_loss: 0.8211 - val_acc: 0.3750
Epoch 30/50
128/128 [==============================] - 0s 819us/step - loss: 0.5753 - acc: 0.7109 - val_loss: 0.8482 - val_acc: 0.3750
Epoch 31/50
128/128 [==============================] - 0s 848us/step - loss: 0.5959 - acc: 0.6875 - val_loss: 0.8694 - val_acc: 0.3750
Epoch 32/50
128/128 [==============================] - 0s 813us/step - loss: 0.6092 - acc: 0.6797 - val_loss: 0.9078 - val_acc: 0.3750
Epoch 33/50
128/128 [==============================] - 0s 782us/step - loss: 0.5877 - acc: 0.7031 - val_loss: 0.9068 - val_acc: 0.3750
Epoch 34/50
128/128 [==============================] - 0s 886us/step - loss: 0.6054 - acc: 0.6953 - val_loss: 0.8961 - val_acc: 0.3750
Epoch 35/50
128/128 [==============================] - 0s 867us/step - loss: 0.6338 - acc: 0.7031 - val_loss: 0.8340 - val_acc: 0.3750
Epoch 36/50
128/128 [==============================] - 0s 872us/step - loss: 0.6017 - acc: 0.6953 - val_loss: 0.8586 - val_acc: 0.3750
Epoch 37/50
128/128 [==============================] - 0s 948us/step - loss: 0.6450 - acc: 0.6875 - val_loss: 0.8476 - val_acc: 0.3750
Epoch 38/50
128/128 [==============================] - 0s 803us/step - loss: 0.6416 - acc: 0.6875 - val_loss: 0.8343 - val_acc: 0.3750
Epoch 39/50
128/128 [==============================] - 0s 788us/step - loss: 0.6011 - acc: 0.6953 - val_loss: 0.8502 - val_acc: 0.3750
Epoch 40/50
128/128 [==============================] - 0s 828us/step - loss: 0.6001 - acc: 0.7188 - val_loss: 0.8525 - val_acc: 0.3750
Epoch 41/50
128/128 [==============================] - 0s 936us/step - loss: 0.6447 - acc: 0.6562 - val_loss: 0.8394 - val_acc: 0.3750
Epoch 42/50
128/128 [==============================] - 0s 790us/step - loss: 0.5960 - acc: 0.7109 - val_loss: 0.8486 - val_acc: 0.3750
Epoch 43/50
128/128 [==============================] - 0s 804us/step - loss: 0.5998 - acc: 0.6797 - val_loss: 0.8658 - val_acc: 0.3750
Epoch 44/50
```

```
Epoch 44/50
128/128 [==============================] - 0s 781us/step - loss: 0.6151 - acc: 0.6797 - val_loss: 0.8834 - val_acc: 0.3750
Epoch 45/50
128/128 [==============================] - 0s 836us/step - loss: 0.5997 - acc: 0.7109 - val_loss: 0.8693 - val_acc: 0.3750
Epoch 46/50
128/128 [==============================] - 0s 870us/step - loss: 0.5891 - acc: 0.7188 - val_loss: 0.8750 - val_acc: 0.3750
Epoch 47/50
128/128 [==============================] - 0s 787us/step - loss: 0.6259 - acc: 0.7266 - val_loss: 0.8593 - val_acc: 0.3750
Epoch 48/50
128/128 [==============================] - 0s 798us/step - loss: 0.5984 - acc: 0.7031 - val_loss: 0.8562 - val_acc: 0.3750
Epoch 49/50
128/128 [==============================] - 0s 835us/step - loss: 0.5907 - acc: 0.6953 - val_loss: 0.8776 - val_acc: 0.3750
Epoch 50/50
128/128 [==============================] - 0s 786us/step - loss: 0.5838 - acc: 0.7344 - val_loss: 0.8836 - val_acc: 0.3750
```

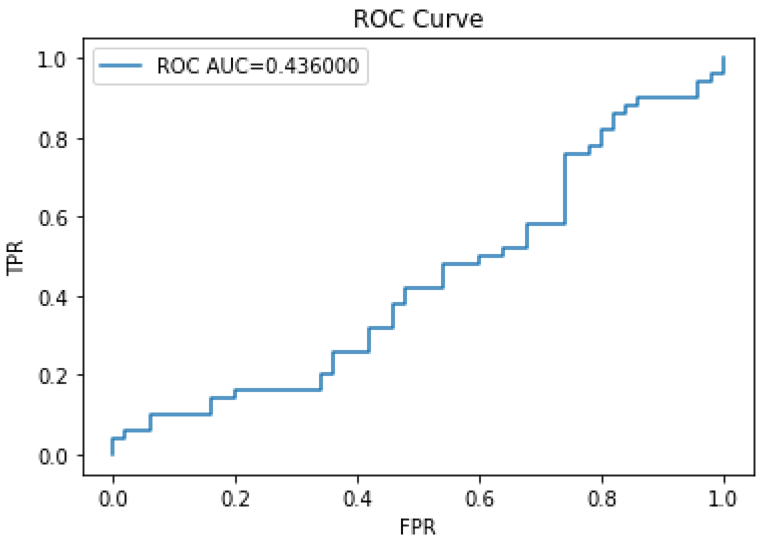
```
# Plot training & validation accuracy values
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation loss values
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()
```



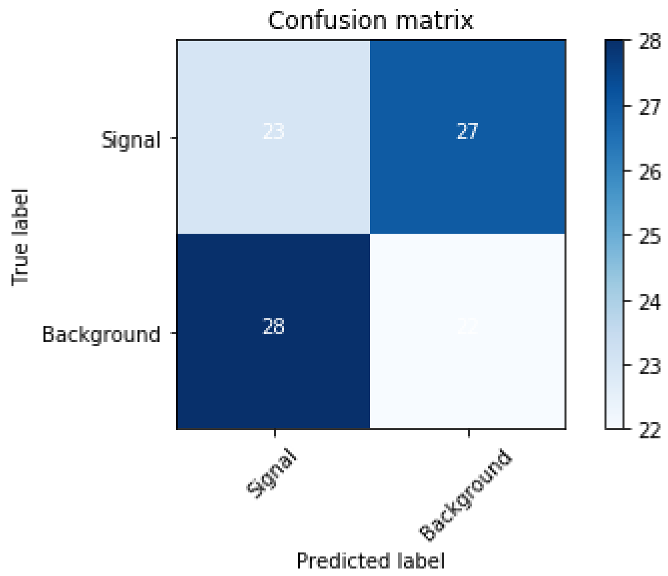



```
# Calculating test loss and accuracy
score = nn.evaluate(x_test, y_test, verbose=2)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

# plotting ROC-AUC curve
y_pred_nn = nn.predict_proba(x_test)
plot_metrics(y_test, y_pred_nn)
```

```
Test loss: 0.8740641403198243
Test accuracy: 0.45
```




```
# creating classification data
y_class = y_pred_nn.copy()
y_class[y_class>0.5]=1
y_class[y_class<=0.5]=0

# Confusing matrix
confusion_mtx = confusion_matrix(y_test, y_class)
plot_confusion_matrix(confusion_mtx, ['Signal','Background'])
```



XGBClassifier

```
param_grid = {
        'n_estimators':[10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
        'max_depth':[100],

}

class XGBClassifier_tmp(XGBClassifier):
    def predict(self, X):
        return XGBClassifier.predict_proba(self, X)[:, 1]

clf = GridSearchCV(
    XGBClassifier_tmp(learning_rate=0.005, subsample=0.8,
                      colsample_bytree=0.8, n_jobs=20),
    param_grid=param_grid, n_jobs=5,
    scoring='roc_auc',
    cv=StratifiedKFold(7, shuffle=True, random_state=0),
    verbose=7)
```

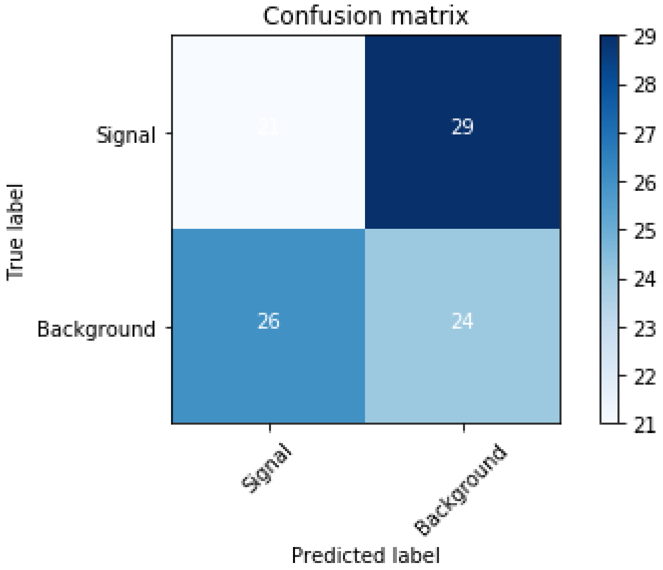
```
clf.fit(x_train, y_train)
clf.best_estimator
```

```
clf.best_estimator_
```

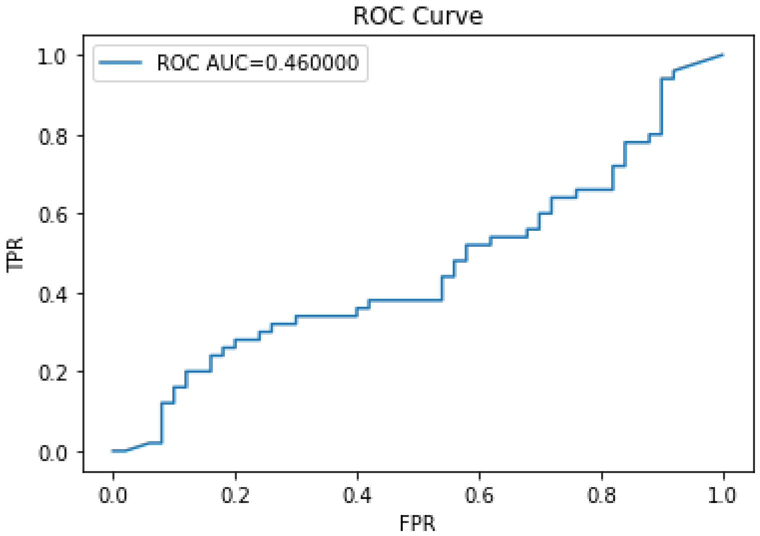
```
Fitting 7 folds for each of 10 candidates, totalling 70 fits
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  22 tasks      | elapsed:    5.7s
[Parallel(n_jobs=5)]: Done  70 out of  70 | elapsed:   11.8s finished
XGBClassifier_tmp(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                  colsample_bynode=1, colsample_bytree=0.8, gamma=0,
                  learning_rate=0.005, max_delta_step=0, max_depth=100,
                  min_child_weight=1, missing=None, n_estimators=80, n_jobs=20,
                  nthread=None, objective='binary:logistic', random_state=0,
                  reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                  silent=None, subsample=0.8, verbosity=1)
```

```
y_pred_xgb = clf.predict(x_test)
y_class = y_pred_xgb.copy()
y_class[y_class>0.5]=1
y_class[y_class<=0.5]=0
print("Accuracy:",metrics.accuracy_score(y_test.squeeze(1), y_class))
print("Precision:",metrics.precision_score(y_test, y_class))
print("Recall:",metrics.recall_score(y_test, y_class))

# confusing matrix
confusion_mtx = confusion_matrix(y_test, y_class)
plot_confusion_matrix(confusion_mtx, ['Signal','Background'])
```

```
Accuracy: 0.45
Precision: 0.4528301886792453
Recall: 0.48
```




```
# plotting ROC-AUC curve
y_pred_nn = nn.predict_proba(x_test)
plot_metrics(y_test, y_pred_xgb)
```

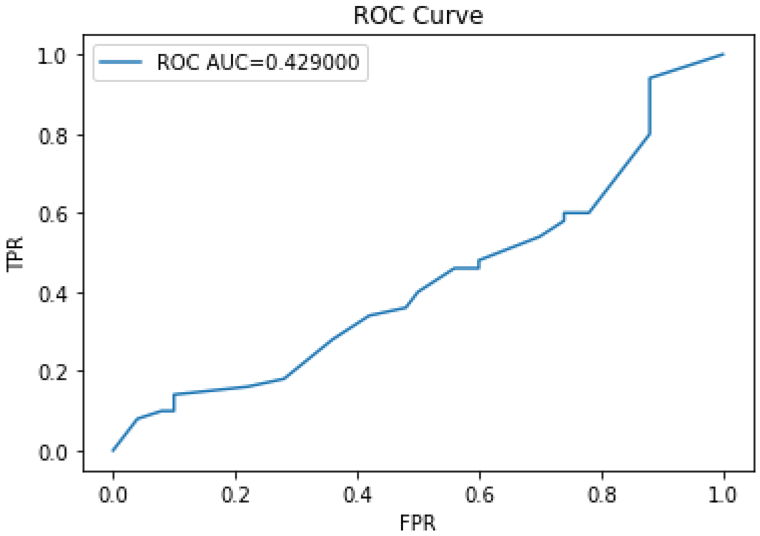


AdaBoostClassifier

```
clf = AdaBoostClassifier(n_estimators=120, learning_rate=0.009, random_state=13,
                              base_estimator=DecisionTreeClassifier(max_depth=19, min_samples_leaf=40, max_features=2,
                                                                       random_state=13))
clf.fit(x_train, y_train.squeeze(1))
```

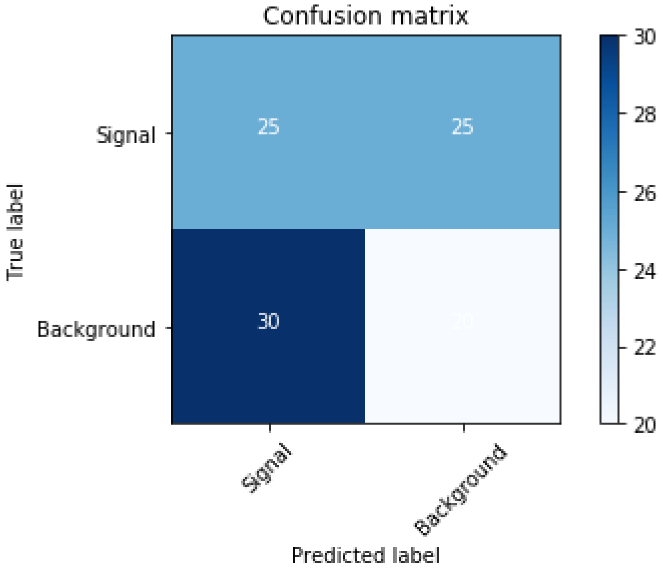
```
AdaBoostClassifier(algorithm='SAMME.R',
                   base_estimator=DecisionTreeClassifier(ccp_alpha=0.0,
                                                         class_weight=None,
                                                         criterion='gini',
                                                         max_depth=19,
                                                         max_features=2,
                                                         max_leaf_nodes=None,
                                                         min_impurity_decrease=0.0,
                                                         min_impurity_split=None,
                                                         min_samples_leaf=40,
                                                         min_samples_split=2,
                                                         min_weight_fraction_leaf=0.0,
                                                         presort='deprecated',
                                                         random_state=13,
                                                         splitter='best'),
                   learning_rate=0.009, n_estimators=120, random_state=13)
```

```
y_pred_ada = clf.predict_proba(x_test)[:, 1]
plot_metrics(y_test, y_pred_ada)
```

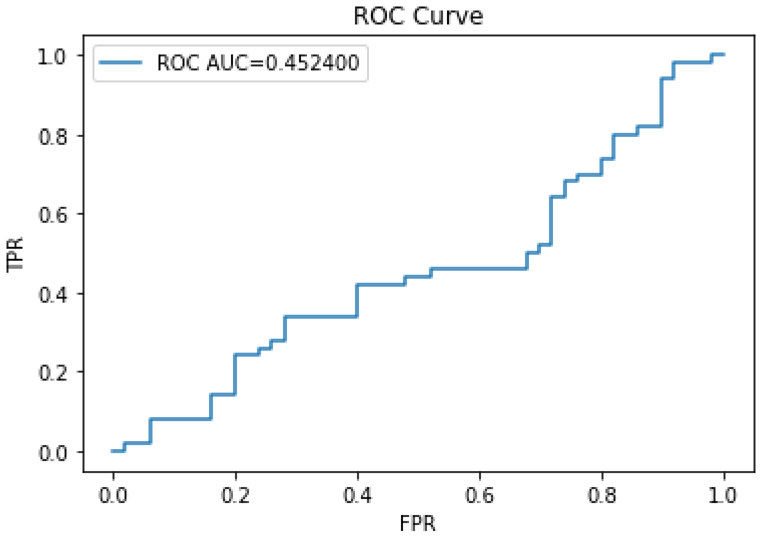



```
y_class = y_pred_ada.copy()
y_class[y_class>0.5]=1
y_class[y_class<=0.5]=0
print("Accuracy:",metrics.accuracy_score(y_test.squeeze(1), y_class))
print("Precision:",metrics.precision_score(y_test, y_class))
print("Recall:",metrics.recall_score(y_test, y_class))

# confusing matrix
confusion_mtx = confusion_matrix(y_test, y_class)
plot_confusion_matrix(confusion_mtx, ['Signal','Background'])
```

```
Accuracy: 0.45
Precision: 0.4444444444444444
Recall: 0.4
```




```
# Adding Noise
y_pred_ada = add_noise(clf.predict_proba(x_test)[:, 1])
plot_metrics(y_test, y_pred_ada)
```



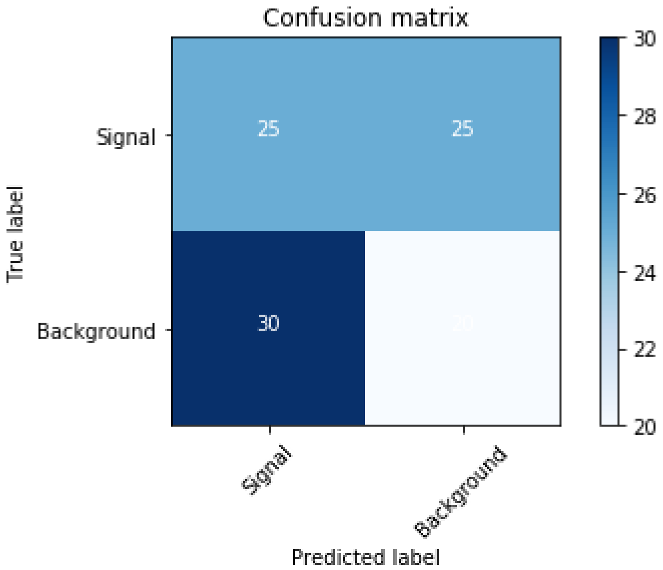
GradientBoostingClassifier

```
%time
gb = GradientBoostingClassifier(learning_rate=0.1, n_estimators=50, subsample=0.8, random_state=13,
                                min_samples_leaf=1, max_depth=3)
gb.fit(x_train_scale, y_train)
```

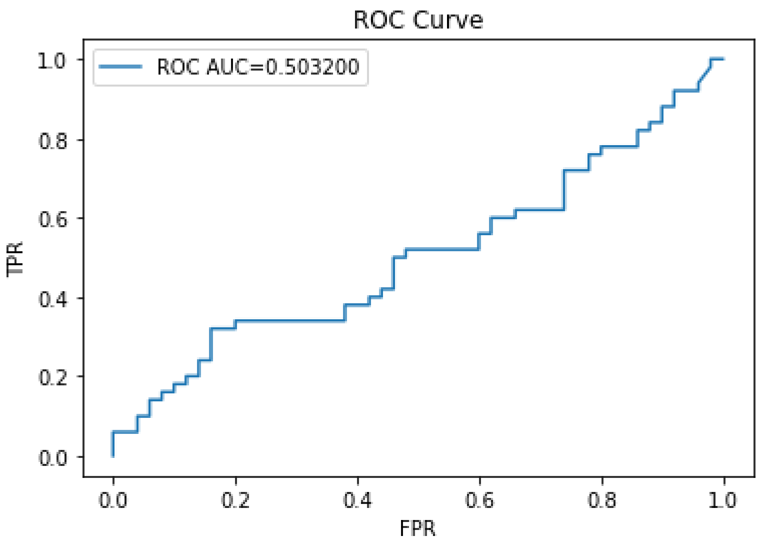
```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 8.58 µs
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=50,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=13, subsample=0.8, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

```
proba_gb = gb.predict_proba(x_test_scale)
proba_class = proba_gb.copy()
proba_class = proba_class[:, 1]
proba_class[proba_class>0.5]=1
proba_class[proba_class<=0.5]=0

print ("Log Loss:", log_loss(y_test, proba_gb))
print("Accuracy:",metrics.accuracy_score(y_test.squeeze(1), proba_class))
print("Precision:",metrics.precision_score(y_test, proba_class))
print("Recall:",metrics.recall_score(y_test, proba_class))

# confusing matrix
confusion_mtx = confusion_matrix(y_test, y_class)
plot_confusion_matrix(confusion_mtx, ['Signal','Background'])
```

```
Log Loss: 1.3733057294268327
Accuracy: 0.52
Precision: 0.52
Recall: 0.52
```



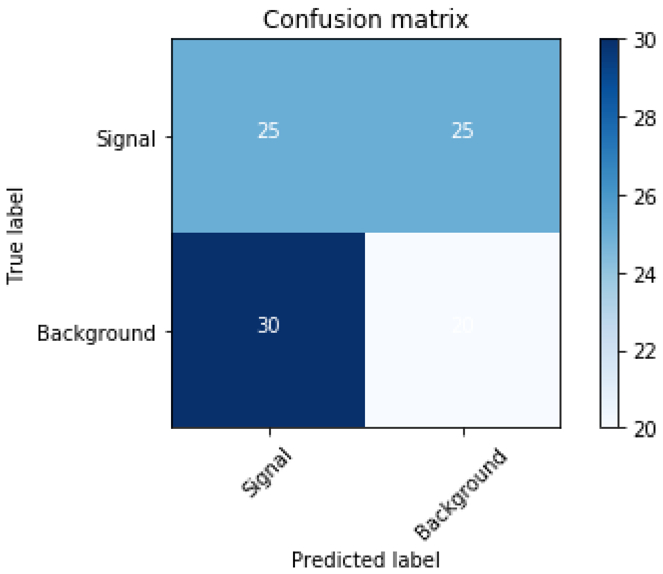

```
1  plot_metrics(y_test, proba_gb[:,1])
```



Support Vector Machine

```
clf = svm.SVC(kernel='linear') # Linear Kernel
clf.fit(x_train_scale, y_train)
y_pred_svm = clf.predict(x_test_scale)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_svm))
print("Precision:",metrics.precision_score(y_test, y_pred_svm))
print("Recall:",metrics.recall_score(y_test, y_pred_svm))

# confusing matrix
confusion_mtx = confusion_matrix(y_test, y_class)
plot_confusion_matrix(confusion_mtx, ['Signal','Background'])
```

```
Accuracy: 0.46
Precision: 0.46296296296296297
Recall: 0.5
```

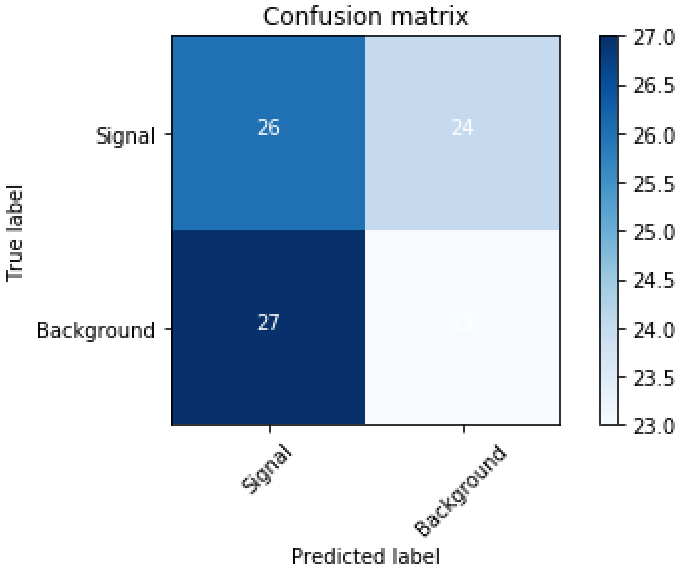


KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier()
clf.fit(x_train_scale, y_train)
y_pred_knn = clf.predict(x_test_scale)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_knn))
print("Precision:",metrics.precision_score(y_test, y_pred_knn))
```

```
print("Recall:",metrics.recall_score(y_test, y_pred_knn))

# confusing matrix
confusion_mtx = confusion_matrix(y_test, y_pred_knn)
plot_confusion_matrix(confusion_mtx, ['Signal','Background'])
```

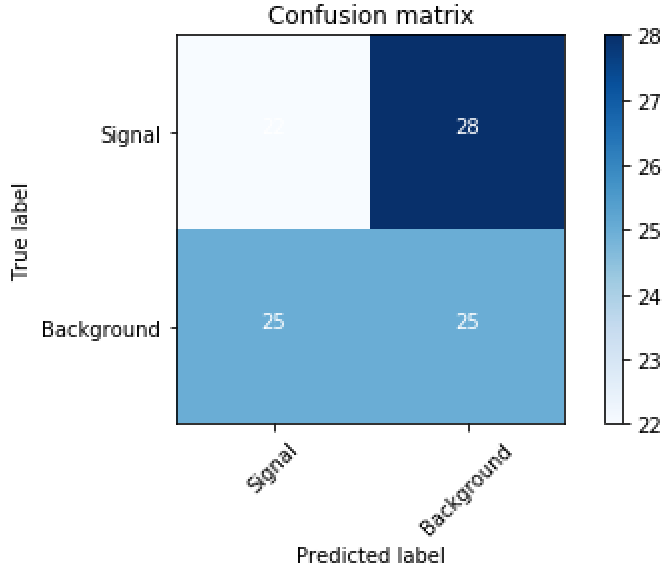
```
Accuracy: 0.49
Precision: 0.48936170212765956
Recall: 0.46
```



LogisticRegression

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(x_train_scale,y_train)
y_pred_lr = clf.predict(x_test_scale)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_lr))
print("Precision:",metrics.precision_score(y_test, y_pred_lr))
print("Recall:",metrics.recall_score(y_test, y_pred_lr))

# confusing matrix
confusion_mtx = confusion_matrix(y_test, y_pred_lr)
plot_confusion_matrix(confusion_mtx, ['Signal','Background'])
```

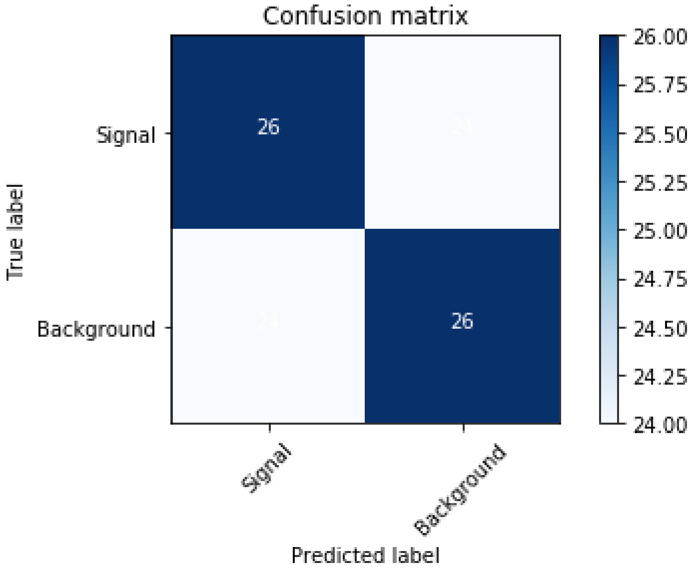
```
Accuracy: 0.47
Precision: 0.4716981132075472
Recall: 0.5
```



DecisionTreeClassifier

```
clf = DecisionTreeClassifier()
clf.fit(x_train,y_train)
y_pred_dt = clf.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_dt))
print("Precision:",metrics.precision_score(y_test, y_pred_dt))
print("Recall:",metrics.recall_score(y_test, y_pred_dt))

# confusing matrix
confusion_mtx = confusion_matrix(y_test, y_pred_dt)
plot_confusion_matrix(confusion_mtx, ['Signal','Background'])
```

```
Accuracy: 0.52
Precision: 0.52
Recall: 0.52
```



RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=1000)
clf.fit(x_train, y_train)
y_pred_rfc = clf.predict(x_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred_rfc))
print("Precision:",metrics.precision_score(y_test, y_pred_rfc))
print("Recall:",metrics.recall_score(y_test, y_pred_rfc))

# confusing matrix
confusion_mtx = confusion_matrix(y_test, y_pred_dt)
plot_confusion_matrix(confusion_mtx, ['Signal','Background'])
```

```
Accuracy: 0.52
Precision: 0.52
Recall: 0.52
```