Higgs Boson Detection - Extracting excotic particles

Data Set Information:

The data has been produced using Monte Carlo simulations. The first 21 features (columns 2-22) are kinematic properties measured by the particle detectors in the accelerator. The last seven features are functions of the first 21 features; these are high-level features derived by physicists to help discriminate between the two classes. There is an interest in using deep learning methods to obviate the need for physicists to manually develop such features. Benchmark results using Bayesian Decision Trees from a standard physics package and 5-layer neural networks are presented in the original paper. The last 500,000 examples are used as a test set.

Attribute Information:

The first column is the class label (1 for signal, 0 for background), followed by the 28 features (21 low-level features then 7 high-level features): 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. For more detailed information about each feature see the original paper.

Ref : https://www.openml.org/search?type=data&status=active&id=4532 (https://www.openml.org/search?type=data&status=active&id=4532)

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
```

In [ ]:
```
import os

# TODO: Fill in the Google Drive path where you uploaded the lab ma
# Example: GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'Colab Notebooks/Lab m

GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'ColabNotebooks/NN/CourseWork/Hig
GOOGLE_DRIVE_PATH = os.path.join('drive', 'My Drive', GOOGLE_DRIVE_
# print(os.listdir(GOOGLE_DRIVE_PATH))
```

In [ ]:
```
higgs_df = pd.read_csv(GOOGLE_DRIVE_PATH+'/phpZLgL9q.csv')
higgs_df.shape
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshe
ll.py:2882: DtypeWarning: Columns (20,21,22,23,24,25,26,27,28) hav
e mixed types.Specify dtype option on import or set low_memory=Fal
se.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[5]: (98050, 29)

In [ ]:
```
X = higgs_df.drop(columns = 'class')
y = higgs_df['class']
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
```

In [ ]:
```
higgs_df = X_train
higgs_df['class'] = y_train
higgs_df.shape
```

Out[8]: (78440, 29)

In [ ]:
```
higgs_df.head()
```

Out[29]:

	lepton_pT	lepton_eta	lepton_phi	missing_energy_magnitude	missing_energy_phi	
2803	2.893923	-0.523075	1.367595	1.396493	1.540824	0.
92448	2.484349	0.380768	1.103481	1.055930	0.778968	0.
50172	0.925843	-0.342891	0.395478	0.236633	1.192825	0.
97304	1.622742	-0.256208	-1.667602	1.772357	-1.415502	2.
96449	0.494308	1.920612	-0.830871	1.064397	0.223475	0.

5 rows × 29 columns

In [ ]:
```
higgs_df.describe()
```

Out[30]:

	lepton_pT	lepton_eta	lepton_phi	missing_energy_magnitude	missing_energ
count	73537.000000	73537.000000	73537.000000	73537.000000	73537.0(
mean	0.989887	-0.003045	-0.005444	0.994044	-0.0(
std	0.561446	1.004373	1.006901	0.593334	1.0(
min	0.274697	-2.434976	-1.742508	0.001283	-1.7
25%	0.591485	-0.739296	-0.876925	0.576567	-0.8
50%	0.854835	-0.002976	-0.003570	0.889649	-0.0(
75%	1.235311	0.736266	0.866000	1.288357	0.8(
max	7.000281	2.433894	1.743236	7.074050	1.7

In [ ]:
```
higgs_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 73537 entries, 2803 to 92634
Data columns (total 29 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   lepton_pT                 73537 non-null  float64
 1   lepton_eta                73537 non-null  float64
 2   lepton_phi                73537 non-null  float64
 3   missing_energy_magnitude  73537 non-null  float64
 4   missing_energy_phi        73537 non-null  float64
 5   jet1pt                    73537 non-null  float64
 6   jet1eta                   73537 non-null  float64
 7   jet1phi                   73537 non-null  float64
 8   jet1b-tag                 73537 non-null  float64
 9   jet2pt                    73537 non-null  float64
 10  jet2eta                   73537 non-null  float64
 11  jet2phi                   73537 non-null  float64
 12  jet2b-tag                 73537 non-null  float64
 13  jet3pt                    73537 non-null  float64
 14  jet3eta                   73537 non-null  float64
 15  jet3phi                   73537 non-null  float64
 16  jet3b-tag                 73537 non-null  float64
 17  jet4pt                    73537 non-null  float64
 18  jet4eta                   73537 non-null  float64
 19  jet4phi                   73537 non-null  object 
 20  jet4b-tag                 73537 non-null  object 
 21  m_jj                      73537 non-null  object 
 22  m_jjj                     73537 non-null  object 
 23  m_lv                      73537 non-null  object 
 24  m_jlv                     73537 non-null  object 
 25  m_bb                      73537 non-null  object 
 26  m_wbb                     73537 non-null  object 
 27  m_wwbb                    73537 non-null  object 
 28  class                     73537 non-null  int64  
dtypes: float64(19), int64(1), object(9)
memory usage: 16.8+ MB
```

In [ ]:
```
higgs_df['m_lv'].value_counts()
```

Out[32]:
```
0.988105714321136    13
0.98750513792038     11
0.989273726940155    11
0.987684428691864    11
0.98950582742691     11
                     ..
1.14196169376373      1
1.62598371505737      1
1.046271443367        1
1.17211437225342      1
0.986775577068329     1
Name: m_lv, Length: 46365, dtype: int64
```

On analysis, it was identified that one record had values of '?' for columns jet4phi , jet4b-tag , m_jj , m_jjj , m_lv , m_jlv , m_bb , m_wbb','m_wwbb

The solution would be to remove that one record, as there are more than 90,000 records, the removal of this one record would not impact

In [ ]:
```
higgs_df = higgs_df[higgs_df['jet4phi'] != '?']
```

In [ ]:
```
temp_col = ['jet4phi','jet4b-tag','m_jj', 'm_jjj','m_lv','m_jlv', '
for i in temp_col:
  higgs_df[i] = higgs_df[i].astype(float)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pa
ndas-docs/stable/user_guide/indexing.html#returning-a-view-versus-
a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/in
dexing.html#returning-a-view-versus-a-copy)
  This is separate from the ipykernel package so we can avoid doin
g imports until
```

In [ ]:
```
higgs_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 78440 entries, 70570 to 92634
Data columns (total 29 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   lepton_pT                 78440 non-null  float64
 1   lepton_eta                78440 non-null  float64
 2   lepton_phi                78440 non-null  float64
 3   missing_energy_magnitude  78440 non-null  float64
 4   missing_energy_phi        78440 non-null  float64
 5   jet1pt                    78440 non-null  float64
 6   jet1eta                   78440 non-null  float64
 7   jet1phi                   78440 non-null  float64
 8   jet1b-tag                 78440 non-null  float64
 9   jet2pt                    78440 non-null  float64
 10  jet2eta                   78440 non-null  float64
 11  jet2phi                   78440 non-null  float64
 12  jet2b-tag                 78440 non-null  float64
 13  jet3pt                    78440 non-null  float64
 14  jet3eta                   78440 non-null  float64
 15  jet3phi                   78440 non-null  float64
 16  jet3b-tag                 78440 non-null  float64
 17  jet4pt                    78440 non-null  float64
 18  jet4eta                   78440 non-null  float64
 19  jet4phi                   78440 non-null  object 
 20  jet4b-tag                 78440 non-null  object 
 21  m_jj                      78440 non-null  object 
 22  m_jjj                     78440 non-null  object 
 23  m_lv                      78440 non-null  object 
 24  m_jlv                     78440 non-null  object 
 25  m_bb                      78440 non-null  object 
 26  m_wbb                     78440 non-null  object 
 27  m_wwbb                    78440 non-null  object 
 28  class                     78440 non-null  int64  
dtypes: float64(19), int64(1), object(9)
memory usage: 20.0+ MB
```

In [ ]:
```
higgs_df.skew()
```

Out[11]:
```
lepton_pT                   1.728784
lepton_eta                  0.001857
lepton_phi                  0.000199
missing_energy_magnitude    1.473417
missing_energy_phi          0.006269
jet1pt                      1.926783
jet1eta                    -0.005140
jet1phi                     0.002865
jet1b-tag                   0.166342
jet2pt                      2.035213
jet2eta                    -0.001060
jet2phi                    -0.004792
jet2b-tag                   0.181471
jet3pt                      1.779549
jet3eta                     0.003308
jet3phi                    -0.004275
jet3b-tag                   0.430144
jet4pt                      1.699018
jet4eta                     0.007927
jet4phi                     0.007642
jet4b-tag                   0.773208
m_jj                        6.004986
m_jjj                       4.672780
m_lv                        4.695684
m_jlv                       2.790971
m_bb                        2.440266
m_wbb                       2.606528
m_wwbb                      2.468990
class                      -0.112445
dtype: float64
```

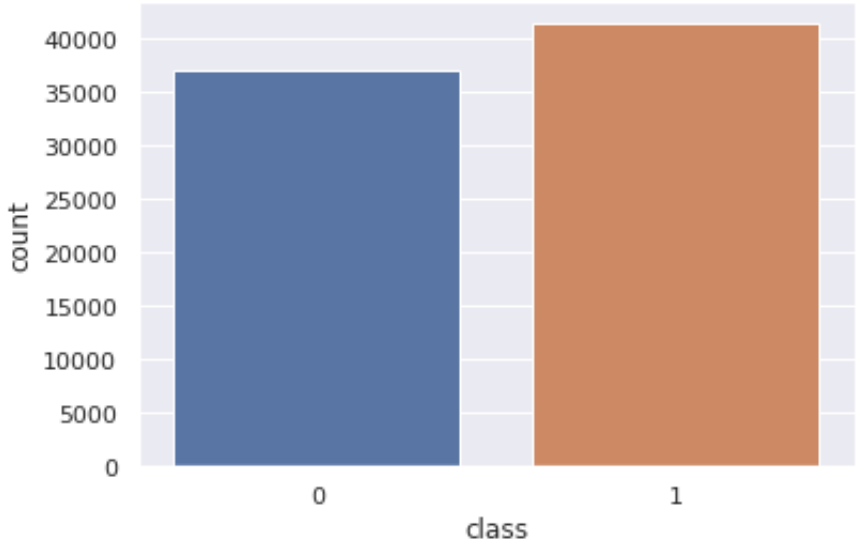
In [ ]:
```
higgs_df['class'].value_counts()
```

Out[37]:
```
1    38864
0    34672
Name: class, dtype: int64
```

DATA BALANCE CHECK

It can be observed that data is almost balanced with 53% of higgs signal and 47% of background

In [ ]:
```
sns.set_theme(style="darkgrid")
ax = sns.countplot(x="class", data=higgs_df)
```

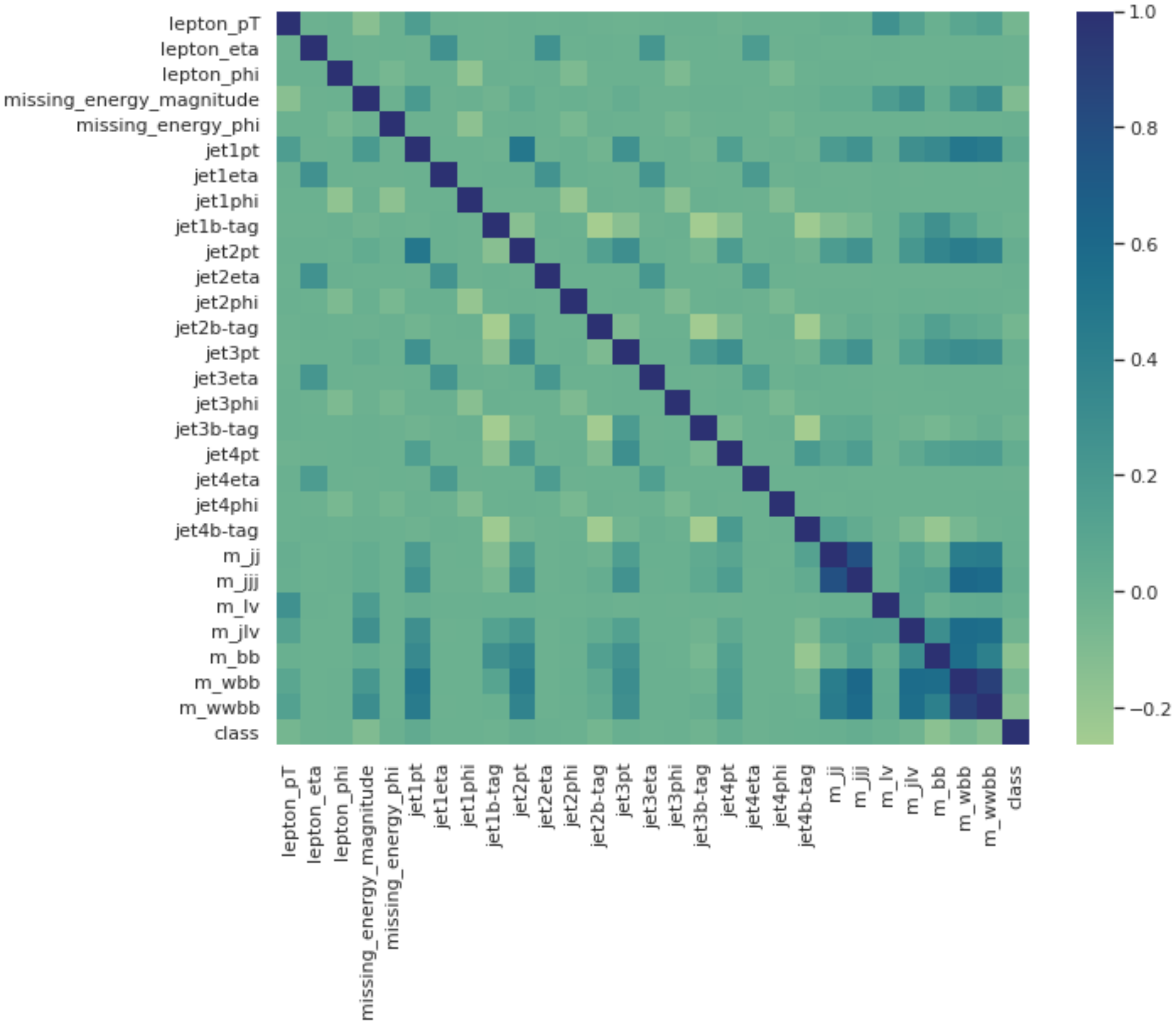


CORELATION CHECK

It is a good practice to check if data is corelated or not, it can be observed that few variables are corelated. these are the calculated variables.

In [ ]:
```
fig, ax = plt.subplots(figsize=(10,8))
sns.heatmap(higgs_df.corr(),cmap="crest", ax=ax)
```

Out[39]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3293baa610>
```

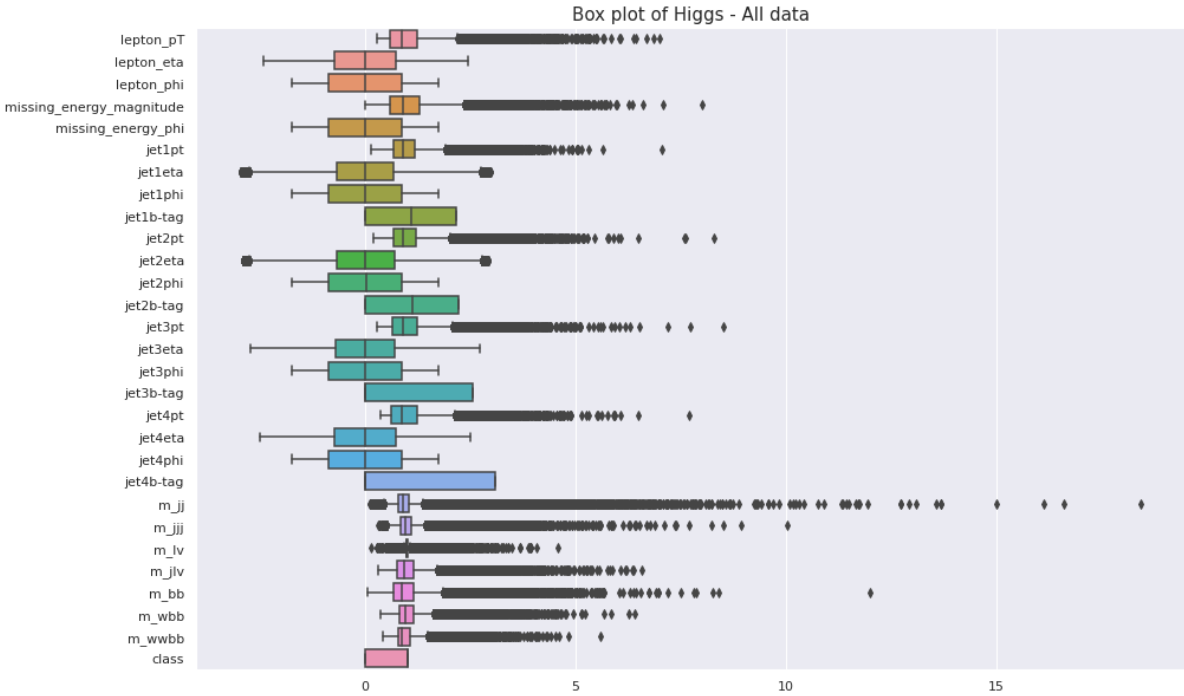


OUTLIERS CHECK

Presence of outliers can skwe the model, hence removing outliers using Z-score and also seeting the score ina way that there is minimal loss of data

In [ ]:
```
plt.figure(figsize=(15,10))
sns.boxplot(data=higgs_df, orient="h")
plt.title('Box plot of Higgs - All data',  fontsize = 15)
```

Out[16]:
```
Text(0.5, 1.0, 'Box plot of Higgs - All data')
```

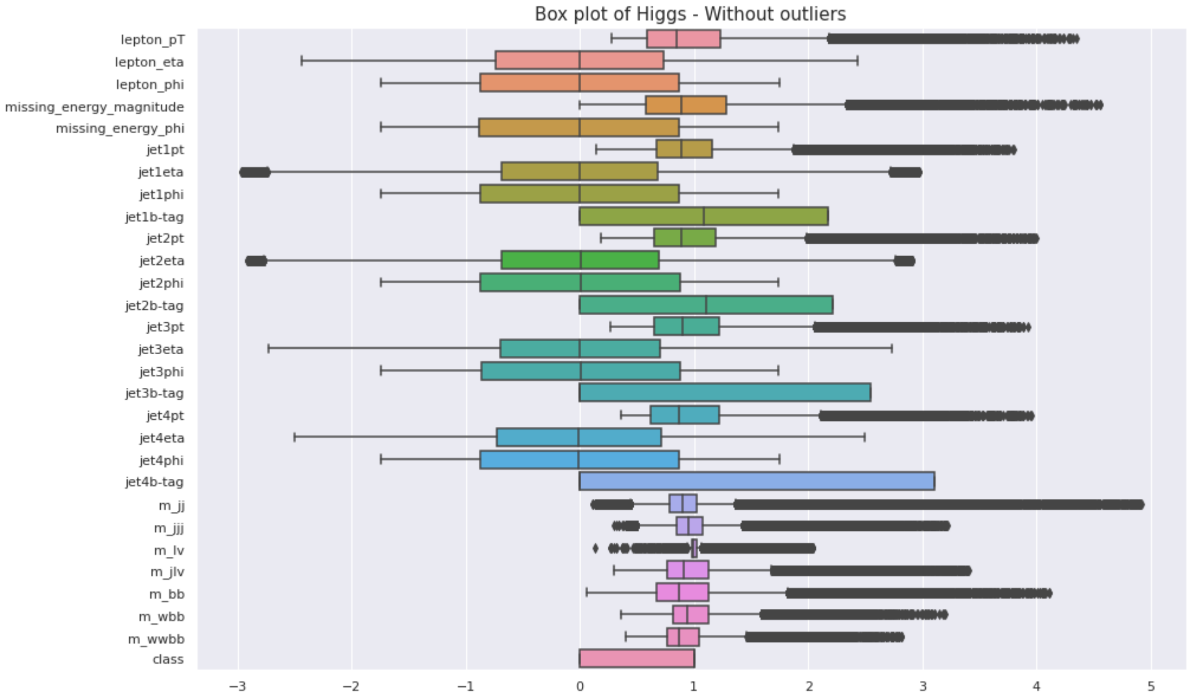


In [ ]:
```
from scipy import stats
zT  = np.abs(stats.zscore(higgs_df))
df_higgs_std = higgs_df[(zT < 6).all(axis =1)]
df_higgs_std.shape
```

Out[18]:
```
(77006, 29)
```

In [ ]:
```
plt.figure(figsize=(15,10))
sns.boxplot(data=df_higgs_std, orient="h")
plt.title('Box plot of Higgs - Without outliers',  fontsize = 15)
```

Out[19]: Text(0.5, 1.0, 'Box plot of Higgs - Without outliers')



In [ ]:
```
df_higgs_sample = higgs_df.sample(n = 8000)
df_higgs_sample.shape
```

Out[36]: (8000, 29)

In [ ]:

PARALLEL COORDI

In [ ]:
```
import plotly.express as px
fig = px.parallel_coordinates(df_higgs_sample, color="class",
                              )
fig.show()
```

Output hidden; open in https://colab.research.google.com (https://colab.research.google.com) to view.

In [ ]:
```
from sklearn import preprocessing

norm_col = ['lepton_pT', 'lepton_eta', 'lepton_phi', 'missing_energ
       'missing_energy_phi', 'jet1pt', 'jet1eta', 'jet1phi', 'jet1b
       'jet2pt', 'jet2eta', 'jet2phi', 'jet2b-tag', 'jet3pt', 'jet3
       'jet3phi', 'jet3b-tag', 'jet4pt', 'jet4eta', 'jet4phi', 'jet
       'm_jj', 'm_jjj', 'm_lv', 'm_jlv', 'm_bb', 'm_wbb', 'm_wwbb']

# normalized = preprocessing.normalize(df_higgs_sample)
normalized = df_higgs_sample

# for i in norm_col :
#   normalized[i] = preprocessing.normalize(normalized[i])

from sklearn.preprocessing import StandardScaler

normalized[norm_col] = preprocessing.normalize(normalized[norm_col]
```

In [ ]:
```
import plotly.express as px
fig = px.parallel_coordinates(normalized, color="class",
                              )
fig.show()
```

Output hidden; open in https://colab.research.google.com (https://colab.research.google.com) to view.

In [ ]:
```
from sklearn import preprocessing

norm_col = ['lepton_pT', 'lepton_eta', 'lepton_phi', 'missing_energ
       'missing_energy_phi', 'jet1pt', 'jet1eta', 'jet1phi', 'jet1b
       'jet2pt', 'jet2eta', 'jet2phi', 'jet2b-tag', 'jet3pt', 'jet3
       'jet3phi', 'jet3b-tag', 'jet4pt', 'jet4eta', 'jet4phi', 'jet
       'm_jj', 'm_jjj', 'm_lv', 'm_jlv', 'm_bb', 'm_wbb', 'm_wwbb']

df_higgs_m = df_higgs_sample[['class', 'm_jj', 'm_jjj', 'm_lv', 'm_
fig = px.parallel_coordinates(df_higgs_m, color="class",
                              )
fig.show()
```

In [ ]:
```
df_higgs_std.to_csv(GOOGLE_DRIVE_PATH+'/HiggsPreprocessedData.csv',

X_test.to_csv(GOOGLE_DRIVE_PATH+'/X_test_data.csv', index=False)
y_test.to_csv(GOOGLE_DRIVE_PATH+'/y_test_data.csv', index=False)
```

In [ ]:
```
X_test.shape
```

Out[40]:
```
(19610, 28)
```

In [ ]:
```
y_test.shape
```
Out[41]:
```
(19610,)
```

In [ ]:
```
df_higgs_std.shape
```
Out[42]:
```
(77006, 29)
```

In [ ]:

Multi-Layer Perceptron - Higgs Detection

Higgs signal is an exotic signal present all over the world but difficuilt to differentiate from background signals

A Multi-Layer Perceptron is a feedforward Artificial Neural Network (ANN). that consists of three types of layers: an input layer, hidden layer(s), and output layer.

The Model creation and evaluation is all done in Ipython and Scikit-learn Deep learning Libraries. MLPClassifier from sklearn library has a preset of classes and parameters which could be called and set accorging to needs.

We aim to create the best MLP model that would classify Higgs signal (class - 1) from background (class - 0). This is a binary classification problem with 29 features in total. The train and test set was split even before preprocessing the training data. Hence test data is not introdced until the final best model is retrived.

Connecting to Google Drive (comment the below cell if training in Jupyter notebook)

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Imports

All Import required for model creation and evaluation

In [3]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
import os
from sklearn.metrics import accuracy_score
import joblib
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
```

In [4]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.metrics import accuracy_score

from sklearn.pipeline import make_pipeline
```

In [5]:
```
import os

# TODO: Fill in the Google Drive path where you uploaded the lab ma
# Example: GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'Colab Notebooks/Lab m

GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'ColabNotebooks/NN/CourseWork/Hig
GOOGLE_DRIVE_PATH = os.path.join('drive', 'My Drive', GOOGLE_DRIVE_
# print(os.listdir(GOOGLE_DRIVE_PATH))
```

In [6]:
```
import joblib
GOOGLE_MODELS_SAVED = GOOGLE_DRIVE_PATH + '/SavedModels/MLP'
```

Preprocessed Data

HiggsPreprocessedData.csv is the training data which was preprocessed in EDA_HiggsDetection.ipynb file and stored in csv format. This has all the features including target variable 'class'.

This preprocessed training data is inturn split into training and validation data (10%) to evaluate the model

Test data is introduced only to text on the best model

In [7]:
```
higgs_df_train = pd.read_csv(GOOGLE_DRIVE_PATH + '/HiggsPreprocesse
higgs_df_train.shape
```

Out[7]:
```
(77006, 29)
```

In [8]:
```
y_train = higgs_df_train['class']
X_train = higgs_df_train.drop(columns='class')
```

In [9]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
```

In [45]:
```
X_train.shape
```

Out[45]:
```
(56136, 28)
```

Basic Model

A Basic model to understand the computational time and expense is created below with hidden_layer_size = 2.

On training the model with train data, the model is evaluated with validation data

In [24]:
```
MLPClf_ = MLPClassifier(hidden_layer_sizes=2).fit(X_train, y_train)
print('MLP Classifier trained :')

y_preds=MLPClf_.predict(X_val)
print('Accuracy : ')
print(MLPClf_.score(X_val, y_val))
print('Validation set results : ')
print(classification_report(y_val, y_preds))
```

```
MLP Classifier trained :
Accuracy :
0.6531619270224646
Validation set results :
              precision    recall  f1-score   support

           0       0.65      0.54      0.59      3573
           1       0.65      0.75      0.70      4128

    accuracy                           0.65      7701
   macro avg       0.65      0.65      0.64      7701
weighted avg       0.65      0.65      0.65      7701
```

In [33]:
```
clf_1 = MLPClassifier( hidden_layer_sizes= 2, activation='logistic'

MLPClf_3 = clf_1.fit(X_train, y_train)

print('MLP Classifier trained with hidden layers nodes : 5, Activat
```

```
MLP Classifier trained with hidden layers nodes : 5, Activation fu
nction : logistic

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

In [34]:
```
y_preds1=MLPClf_3.predict(X_val)
print('Accuracy : ')
print(MLPClf_3.score(X_val, y_val))
print('Validation set results with with hidden layers nodes : 5, Ac
print(classification_report(y_val, y_preds1))
```

```
Accuracy :
0.6695234385144786
Validation set results with with hidden layers nodes : 5, Activati
on function : logistic
              precision    recall  f1-score   support

           0       0.65      0.62      0.64      3573
           1       0.69      0.71      0.70      4128

    accuracy                           0.67      7701
   macro avg       0.67      0.67      0.67      7701
weighted avg       0.67      0.67      0.67      7701
```

In [ ]:
```
clf_2 = MLPClassifier(random_state=20, hidden_layer_sizes=5, activa

MLPClf_4 = clf_2.fit(X_train, y_train)

print('MLP Classifier trained with hidden layers : 5, Activation fu
```

```
MLP Classifier trained with hidden layers : 5, Activation function
: relu, random state = 20

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

In [ ]:
```
y_preds2=MLPClf_4.predict(X_val)
print('Accuracy : ')
print(MLPClf_4.score(X_val, y_val))
print('Validation set results with 2 Hidden layers : ')
print(classification_report(y_val, y_preds2))
```

```
Accuracy :
0.6768831168831169
Validation set results with 2 Hidden layers :
              precision    recall  f1-score   support

           0       0.66      0.62      0.64      2663
           1       0.69      0.73      0.71      3112

    accuracy                           0.68      5775
   macro avg       0.67      0.67      0.67      5775
weighted avg       0.68      0.68      0.68      5775
```

Grid Search for Ativation function

A grid search is performed on activation function to identify the best activation for this dataset. GridSearchCV() in sklearner is an existing library that can be used to tune the parameter.

In [ ]:
```
parameters = {'activation' : ['logistic', 'relu', 'tanh']}

clf3 = GridSearchCV(MLPClassifier(), parameters, n_jobs=-1, verbose
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

In [ ]:
```
print(clf3.best_params_)
```

```
{'activation': 'logistic'}
```

In [ ]:
```
result1 = clf3.cv_results_
result1
```

Out[24]:
```
{'mean_fit_time': array([107.88673348,  84.08345866, 106.29938221]
),
 'mean_score_time': array([0.06941824, 0.02834787, 0.07051129]),
 'mean_test_score': array([0.69443547, 0.69332718, 0.68707766]),
 'param_activation': masked_array(data=['logistic', 'relu', 'tanh'
],
              mask=[False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'activation': 'logistic'},
  {'activation': 'relu'},
  {'activation': 'tanh'}],
 'rank_test_score': array([1, 2, 3], dtype=int32),
 'split0_test_score': array([0.70376356, 0.69514354, 0.69606711]),
 'split1_test_score': array([0.69160317, 0.69183406, 0.6882937 ]),
 'split2_test_score': array([0.69291157, 0.69437389, 0.68190564]),
 'split3_test_score': array([0.6880628 , 0.69460479, 0.68321404]),
 'split4_test_score': array([0.69583622, 0.6906796 , 0.6859078 ]),
 'std_fit_time': array([ 0.75909086,  1.55973135, 13.6399145 ]),
 'std_score_time': array([0.00702866, 0.0026133 , 0.01221008]),
 'std_test_score': array([0.00529019, 0.00174735, 0.00500669])}
```

GRID SEARCH - 1 RESULTS

The result showed that logistic is the best activation function for this dataset, but when combined with other set of parameters we have to identify which is the best activation function and also hidden layer

We split the grid search because the data is huge, the search become very time consuming as the number of fits increases with many combinations are avaibale.

In [ ]:
```
parameters = {  'hidden_layer_sizes': [10, 20, 50, 100], 'activatio

clf4 = GridSearchCV(MLPClassifier(), parameters, n_jobs=-1, verbose
print(clf4.best_params_)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
{'activation': 'relu', 'hidden_layer_sizes': 50}

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

In [ ]:
```
result2 = clf4.cv_results_
result2
```

Out[26]:
```
{'mean_fit_time': array([ 49.07641449,  60.05024347,  72.43873796,
```

```
102.13406796,
         42.83842516,  47.89261079,  60.37309618,  74.67126994]),
 'mean_score_time': array([0.01353493, 0.02847543, 0.03616247, 0.0
643549 , 0.00904212,
        0.01092434, 0.01723175, 0.02626619]),
 'mean_test_score': array([0.683291  , 0.68946356, 0.68830909, 0.6
9243439, 0.68874009,
        0.69586701, 0.6969599 , 0.69209574]),
 'param_activation': masked_array(data=['logistic', 'logistic', 'l
ogistic', 'logistic', 'relu',
                    'relu', 'relu', 'relu'],
              mask=[False, False, False, False, False, False, Fals
e, False],
        fill_value='?',
             dtype=object),
 'param_hidden_layer_sizes': masked_array(data=[10, 20, 50, 100, 1
0, 20, 50, 100],
              mask=[False, False, False, False, False, False, Fals
e, False],
        fill_value='?',
             dtype=object),
 'params': [{'activation': 'logistic', 'hidden_layer_sizes': 10},
  {'activation': 'logistic', 'hidden_layer_sizes': 20},
  {'activation': 'logistic', 'hidden_layer_sizes': 50},
  {'activation': 'logistic', 'hidden_layer_sizes': 100},
  {'activation': 'relu', 'hidden_layer_sizes': 10},
  {'activation': 'relu', 'hidden_layer_sizes': 20},
  {'activation': 'relu', 'hidden_layer_sizes': 50},
  {'activation': 'relu', 'hidden_layer_sizes': 100}],
 'rank_test_score': array([8, 5, 7, 3, 6, 2, 1, 4], dtype=int32),
 'split0_test_score': array([0.6917571 , 0.69475872, 0.69506657, 0
.70114677, 0.69160317,
        0.69352728, 0.7002232 , 0.69645194]),
 'split1_test_score': array([0.68205957, 0.68575387, 0.68552297, 0
.68513815, 0.68752405,
        0.69098745, 0.69806819, 0.6869853 ]),
 'split2_test_score': array([0.68306011, 0.69106442, 0.6893712 , 0
.69129531, 0.68660048,
        0.69876087, 0.69876087, 0.69160317]),
 'split3_test_score': array([0.67705688, 0.68875548, 0.68429154, 0
.68890941, 0.68575387,
        0.69729855, 0.68967906, 0.69221889]),
 'split4_test_score': array([0.68252136, 0.6869853 , 0.68729316, 0
.69568229, 0.69221889,
        0.69876087, 0.69806819, 0.69321943]),
 'std_fit_time': array([4.75617058, 5.01181014, 0.49296561, 0.4692
361 , 1.55719781,
        0.25983617, 0.15828078, 9.40225158]),
 'std_score_time': array([0.00057265, 0.01151204, 0.0011906 , 0.00
291598, 0.000336  ,
        0.00087551, 0.00056326, 0.00349956]),
 'std_test_score': array([0.00474766, 0.00319394, 0.00378753, 0.00
553918, 0.00265607,
```

```
        0.00310105, 0.00372451, 0.00305331])}
```

GRID SEARCH - 2 RESULTS

On combining activation function and hidden layers, it can be observed that activation : relu and hidden layers nodes : 50 gives a better result

Further performing grid search with multi layerd hidden layers to see if the accuracy of validation set increases. Again activation is given both logistic and relu for grid search to identify the best parameter as in the previous runs one gave logistic and other gave relu

In [ ]:
```
parameters = {  'hidden_layer_sizes': [5 , (5, 30), 10, (10, 30), 2
# parameters = {'activation' : ['logistic', 'relu', 'tanh']}

clf_5_1 = GridSearchCV(MLPClassifier(), parameters, n_jobs=-1, verb
print(clf_5_1.best_params_)

joblib.dump(clf_5_1, GOOGLE_MODELS_SAVED + '/clf_5_1_hidden_and_nod
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
{'activation': 'relu', 'hidden_layer_sizes': (25, 35)}

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

Out[135]:
```
['drive/My Drive/ColabNotebooks/NN/CourseWork/HiggsDetection_Neura
lComputing/SavedModels/MLP/clf_5_1_hidden_and_nodes.pkl']
```

In [ ]:
```
result3 = clf_5_1.cv_results_
result3
```

Out[136]:
```
{'mean_fit_time': array([ 48.44974761,  71.96911707,  48.34943953,
76.28526993,
         58.39257383,  90.89141831,  73.60890355, 105.32863612,
         28.12802196,  66.40368104,  46.74141932,  70.66350675,
         58.30057931,  88.64038324,  69.88222361,  82.09790931]),
 'mean_score_time': array([0.01270595, 0.03076129, 0.01482687, 0.0
3579006, 0.02375202,
        0.04776564, 0.04050307, 0.07368007, 0.00958476, 0.01975927
,
        0.01110468, 0.02292967, 0.01710653, 0.03337393, 0.02484856
,
        0.03448677]),
 'mean_test_score': array([0.68267529, 0.67965828, 0.68773955, 0.6
8592319, 0.68889402,
        0.69100285, 0.68955591, 0.6893712 , 0.67126914, 0.69974602
,
        0.68384515, 0.70188563, 0.69526668, 0.70625722, 0.69528207
```

```
,
        0.69051027]),
 'param_activation': masked_array(data=['logistic', 'logistic', 'l
ogistic', 'logistic',
                    'logistic', 'logistic', 'logistic', 'logistic'
, 'relu',
                    'relu', 'relu', 'relu', 'relu', 'relu', 'relu'
, 'relu'],
              mask=[False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False, False, False, Fals
e, False],
        fill_value='?',
             dtype=object),
 'param_hidden_layer_sizes': masked_array(data=[5, (5, 30), 10, (1
0, 30), 25, (25, 35), 50, 100, 5,
                    (5, 30), 10, (10, 30), 25, (25, 35), 50, 100],
              mask=[False, False, False, False, False, False, Fals
e, False,
                    False, False, False, False, False, False, Fals
e, False],
        fill_value='?',
             dtype=object),
 'params': [{'activation': 'logistic', 'hidden_layer_sizes': 5},
  {'activation': 'logistic', 'hidden_layer_sizes': (5, 30)},
  {'activation': 'logistic', 'hidden_layer_sizes': 10},
  {'activation': 'logistic', 'hidden_layer_sizes': (10, 30)},
  {'activation': 'logistic', 'hidden_layer_sizes': 25},
  {'activation': 'logistic', 'hidden_layer_sizes': (25, 35)},
  {'activation': 'logistic', 'hidden_layer_sizes': 50},
  {'activation': 'logistic', 'hidden_layer_sizes': 100},
  {'activation': 'relu', 'hidden_layer_sizes': 5},
  {'activation': 'relu', 'hidden_layer_sizes': (5, 30)},
  {'activation': 'relu', 'hidden_layer_sizes': 10},
  {'activation': 'relu', 'hidden_layer_sizes': (10, 30)},
  {'activation': 'relu', 'hidden_layer_sizes': 25},
  {'activation': 'relu', 'hidden_layer_sizes': (25, 35)},
  {'activation': 'relu', 'hidden_layer_sizes': 50},
  {'activation': 'relu', 'hidden_layer_sizes': 100}],
 'rank_test_score': array([14, 15, 11, 12, 10,  6,  8,  9, 16,  3,
13,  2,  5,  1,  4,  7],
       dtype=int32),
 'split0_test_score': array([0.68667744, 0.67567152, 0.68644655, 0
.68906334, 0.69506657,
        0.70137766, 0.69799123, 0.69614408, 0.64896483, 0.70453321
,
        0.67990456, 0.70561071, 0.69460479, 0.70461017, 0.69876087
,
        0.6882937 ]),
 'split1_test_score': array([0.67928885, 0.68382975, 0.6869853 , 0
.683291  , 0.68375279,
        0.68983299, 0.69021781, 0.69129531, 0.68367583, 0.69806819
,
```

```
        0.68767798, 0.70106981, 0.69114138, 0.70930501, 0.68667744
,
        0.69191103]),
 'split2_test_score': array([0.68467636, 0.66989918, 0.68713923, 0
.68406065, 0.68990995,
        0.68544601, 0.68713923, 0.68729316, 0.66959132, 0.70168552
,
        0.68252136, 0.70407142, 0.70076195, 0.71045948, 0.69991534
,
        0.69152621]),
 'split3_test_score': array([0.68044332, 0.68275225, 0.69006388, 0
.6856769 , 0.68944816,
        0.68683137, 0.68636958, 0.68421458, 0.67382437, 0.69629801
,
        0.68513815, 0.69845301, 0.68898638, 0.70360964, 0.69614408
,
        0.69083353]),
 'split4_test_score': array([0.68229046, 0.68613869, 0.6880628 , 0
.68752405, 0.68629262,
        0.69152621, 0.68606173, 0.68790887, 0.68028939, 0.69814516
,
        0.68398368, 0.7002232 , 0.70083891, 0.70330178, 0.69491265
,
        0.68998692]),
 'std_fit_time': array([ 1.68392092,  1.07631875,  0.61161379,  0.
7731899 ,  0.89975652,
         0.51610048,  0.28815459,  1.29158689, 10.40385579,  0.911
22254,
         0.5632603 ,  4.07842092,  2.61351735,  4.50674765,  1.584
08085,
        14.13293896]),
 'std_score_time': array([0.00182973, 0.00310499, 0.00023393, 0.00
500297, 0.00018315,
        0.00139235, 0.00089777, 0.00621745, 0.00041541, 0.00226164
,
        0.00015752, 0.00523047, 0.00292106, 0.00916144, 0.00360793
,
        0.00632059]),
 'std_test_score': array([0.00270626, 0.00600173, 0.00127343, 0.00
213923, 0.00381154,
        0.00561354, 0.00446755, 0.00406501, 0.01218292, 0.00296464
,
        0.0025947 , 0.00260254, 0.00486099, 0.00301347, 0.00465808
,
        0.00128731])}
```

BEST PARAMETERS - 1

The best parameters after tuning is observed to be 2 hidden layers of each having 25 and 35 nodes and activation function as 'relu'

Going forword we will be working with relu as activation function

A model is trained and fit with these parameters and validated with the validation set. the previous two grid searchs resulted in two results, one was one layer with 50 nodes, other best hidden layer is two layers with 25, 30 nodes respectively, so performing cross validation with these parameters to yeild better results

In [ ]:
```
MLPClf_51 = MLPClassifier(hidden_layer_sizes = (25, 35), activation

y_preds3=MLPClf_51.predict(X_val)
print('Accuracy : ')
print(MLPClf_51.score(X_val, y_val))
print('Validation set results with 2 Hidden layers : ')
print(classification_report(y_val, y_preds3))
```

```
Accuracy :
0.7110403102922842
Validation set results with 2 Hidden layers :
              precision    recall  f1-score   support

           0       0.70      0.65      0.68      3350
           1       0.72      0.76      0.74      3869

    accuracy                           0.71      7219
   macro avg       0.71      0.71      0.71      7219
weighted avg       0.71      0.71      0.71      7219


/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

In [ ]:
```
MPL_cv_scores_51 = cross_val_score(MLPClf_51, X_train, y_train, cv=

MPL_cv_scores_51
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

Out[139]:
```
array([0.70822751, 0.70245517, 0.70430232, 0.70337874, 0.70599554]
)
```

The best set of parameters at {'activation': 'relu', 'hidden_layer_sizes': 50}

In [ ]:
```
# best_param_ = {'activation': 'relu', 'hidden_layer_sizes': 50}

MLPClf_5 = MLPClassifier(hidden_layer_sizes = 50, activation = 'rel

y_preds3=MLPClf_5.predict(X_val)
print('Accuracy : ')
print(MLPClf_5.score(X_val, y_val))
print('Validation set results with 2 Hidden layers : ')
print(classification_report(y_val, y_preds3))
```

```
Accuracy :
0.7027289098213049
Validation set results with 2 Hidden layers :
              precision    recall  f1-score   support

           0       0.69      0.65      0.67      3350
           1       0.71      0.75      0.73      3869

    accuracy                           0.70      7219
   macro avg       0.70      0.70      0.70      7219
weighted avg       0.70      0.70      0.70      7219
```

In [ ]:
```
MPL_cv_scores_1 = cross_val_score(MLPClf_5, X_train, y_train, cv=5)

MPL_cv_scores_1
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

Out[11]:
```
array([0.68429154, 0.69614408, 0.69622104, 0.6989148 , 0.69645194]
)
```

It can be observed that two hidden layer with 25, 30 nodes respectively with relu activation function gives better result

Standerdiseing Data for PCA

Ref : https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60 (https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60)

Ref : https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)

It is a good practice to standerdise data before performing MLP, and also in an aim to improve performance a principal component analysis is done on the training set with different coverage (0.95, 0.97, 0.99) of attributes. Also models are evaluated with standerdised PCA and without standerdised PCA with hidden layers (25, 25) and activation function relu

In [ ]:
```
scaler = StandardScaler()
# Fit on training set only.
scaler.fit(X_train)
# Apply transform to both the training set and the test set.
train_scaler = scaler.transform(X_train)
```

In [ ]:
```
train_scaler
```

Out[46]:
```
array([[ 1.03077323e+00, -6.31659065e-01,  5.55961714e-02, ...,
         1.75612352e+00,  6.08805799e-01,  3.29194842e-01],
       [ 5.48485343e-01,  1.14011025e+00,  1.15568781e+00, ...,
        -5.98142196e-01, -6.46739946e-01, -8.31684074e-01],
       [-6.24776559e-01,  6.54521429e-01, -1.25634733e+00, ...,
        -5.71941778e-01, -9.65298216e-01, -7.91088968e-01],
       ...,
       [ 3.64270755e-01,  1.48578533e-01, -4.66153352e-01, ...,
         9.62141612e-01,  9.30115098e-01,  4.97511467e-01],
       [-6.99889707e-01, -1.60608771e-01,  3.56122487e-01, ...,
         7.57223982e-04,  8.67932428e-01,  4.70681001e-01],
       [-9.99662850e-01, -6.00643380e-01,  1.04319733e+00, ...,
         2.05164122e-01,  1.51644239e-01, -1.04560114e-01]])
```

In [ ]:
```
pca = PCA(.95)
pca1 = pca.fit_transform(train_scaler)
pca1.shape
```

Out[51]:
```
(64965, 23)
```

In [ ]:
```
pca.explained_variance_ratio_
```

Out[52]:
```
array([0.13885902, 0.06736003, 0.06379252, 0.04946378, 0.04840618,
       0.04636658, 0.04524005, 0.044358  , 0.04295407, 0.03888531,
       0.03868175, 0.03775829, 0.03685797, 0.03155603, 0.03089586,
       0.02978946, 0.02802315, 0.02638866, 0.02595262, 0.02254196,
       0.02129574, 0.01989115, 0.01770979])
```

In [ ]:
```
std_clf = make_pipeline(StandardScaler(), PCA(n_components=2), MLPC
std_clf.fit(X_train, y_train)
pred_test_std = std_clf.predict(X_val)
```

In [ ]:
```
print("\nPrediction accuracy for the normal test dataset with PCA")
print(f"{accuracy_score(y_val, pred_test_std):.2%}\n")
```

```

Prediction accuracy for the normal test dataset with PCA
56.55%
```

In [ ]:
```
print('Accuracy : ')
print(std_clf.score(X_val, y_val))
print('Validation set results with 2 Hidden layers : ')
print(classification_report(y_val, pred_test_std))
```

```
Accuracy : 
0.5654522787089624
Validation set results with 2 Hidden layers : 
              precision    recall  f1-score   support

           0       0.54      0.43      0.48      3350
           1       0.58      0.69      0.63      3869

    accuracy                           0.57      7219
   macro avg       0.56      0.56      0.55      7219
weighted avg       0.56      0.57      0.56      7219
```

In [ ]:
```
std_clf51 = make_pipeline(StandardScaler(), PCA(0.95), MLPClassifie
std_clf51.fit(X_train, y_train)
pred_val_std = std_clf51.predict(X_val)

print('Accuracy : ')
print(std_clf51.score(X_val, y_val))
print('Validation set results with 2 Hidden layers : ')
print(classification_report(y_val, pred_val_std))
```

```
Accuracy : 
0.6786258484554647
Validation set results with 2 Hidden layers : 
              precision    recall  f1-score   support

           0       0.65      0.66      0.66      3350
           1       0.70      0.70      0.70      3869

    accuracy                           0.68      7219
   macro avg       0.68      0.68      0.68      7219
weighted avg       0.68      0.68      0.68      7219
```

In [ ]:
```
std_clf1 = make_pipeline(StandardScaler(), PCA(n_components=20), ML
std_clf1.fit(X_train, y_train)
pred_test_std = std_clf1.predict(X_val)
```

In [ ]:
```
print('Accuracy : ')
print(std_clf1.score(X_val, y_val))
print('Validation set results with 2 Hidden layers : ')
print(classification_report(y_val, pred_test_std))
```

```
Accuracy :
0.6511982269012329
Validation set results with 2 Hidden layers :
              precision    recall  f1-score   support

           0       0.64      0.56      0.60      3350
           1       0.66      0.73      0.69      3869

    accuracy                           0.65      7219
   macro avg       0.65      0.65      0.65      7219
weighted avg       0.65      0.65      0.65      7219
```

In [ ]:
```
pca = PCA(0.95)

x_pca = pca.fit_transform(X_train)

print(x_pca.shape)
pca.explained_variance_ratio_
```

```
(64965, 19)
```

Out[141]:
```
array([0.11857291, 0.099966  , 0.08644029, 0.07255706, 0.06639667,
       0.05796887, 0.05759269, 0.05616057, 0.05485031, 0.04616458,
       0.04175469, 0.03899919, 0.03846471, 0.03113024, 0.0264493 ,
       0.01914977, 0.01880117, 0.01498932, 0.01381715])
```

In [ ]:
```
pca = PCA(0.97)

x_pca = pca.fit_transform(X_train)

print(x_pca.shape)
pca.explained_variance_ratio_
```

```
(64965, 20)
```

Out[142]:
```
array([0.11857291, 0.099966  , 0.08644029, 0.07255706, 0.06639667,
       0.05796887, 0.05759269, 0.05616057, 0.05485031, 0.04616458,
       0.04175469, 0.03899919, 0.03846471, 0.03113024, 0.0264493 ,
       0.01914977, 0.01880117, 0.01498932, 0.01381715, 0.01000232]
)
```

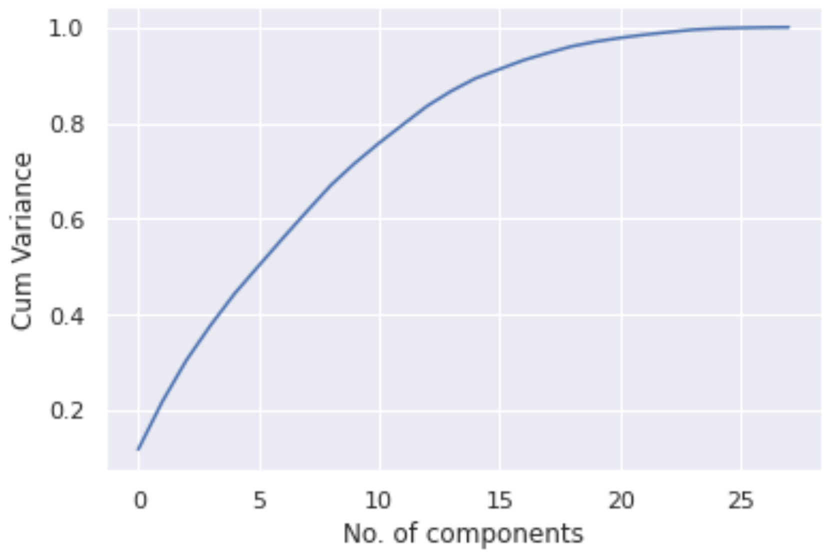
In [ ]:
```
pca = PCA(0.99)

x_pca = pca.fit_transform(X_train)

print(x_pca.shape)
pca.explained_variance_ratio_
```

```
(64965, 24)
```

Out[148]:
```
array([0.11857291, 0.099966  , 0.08644029, 0.07255706, 0.06639667,
       0.05796887, 0.05759269, 0.05616057, 0.05485031, 0.04616458,
       0.04175469, 0.03899919, 0.03846471, 0.03113024, 0.0264493 ,
       0.01914977, 0.01880117, 0.01498932, 0.01381715, 0.01000232,
       0.00759765, 0.0065112 , 0.00544163, 0.00500464])
```

In [ ]:
```
# from sklearn.decomposition import PCA
pca = PCA(n_components = 28)
pca.fit(X_train)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel ("No. of components")
plt.ylabel ( "Cum Variance")
```

Out[145]:
```
Text(0, 0.5, 'Cum Variance')
```



The below code was run with different percentage of PCA (0.95, 0.97, 0.98, 0.99)

In [ ]:
```
for pca_per in [0.95, 0.97, 0.98, 0.99] :
  print('PCA percentage set to : ' , pca_per)
  pca_clf_51 = make_pipeline( PCA(pca_per), MLPClassifier(hidden_la
  pca_clf_51.fit(X_train, y_train)
  pred_test_pca = pca_clf_51.predict(X_val)

  print('Accuracy : ')
  print(pca_clf_51.score(X_val, y_val))
  print('Validation set results with 2 Hidden layers : ')
  print(classification_report(y_val, pred_test_pca))
```

```
PCA percentage set to :  0.95
Accuracy :
0.6403934062889597
Validation set results with 2 Hidden layers :
              precision    recall  f1-score   support

           0       0.63      0.56      0.59      3350
           1       0.65      0.71      0.68      3869

    accuracy                           0.64      7219
   macro avg       0.64      0.63      0.63      7219
weighted avg       0.64      0.64      0.64      7219

PCA percentage set to :  0.97

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,

Accuracy :
0.6457958165950963
Validation set results with 2 Hidden layers :
              precision    recall  f1-score   support

           0       0.63      0.59      0.61      3350
           1       0.66      0.70      0.68      3869

    accuracy                           0.65      7219
   macro avg       0.64      0.64      0.64      7219
weighted avg       0.64      0.65      0.64      7219

PCA percentage set to :  0.98

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,

Accuracy :
0.6560465438426375
Validation set results with 2 Hidden layers :
              precision    recall  f1-score   support

           0       0.63      0.63      0.63      3350
           1       0.68      0.68      0.68      3869

    accuracy                           0.66      7219
   macro avg       0.65      0.65      0.65      7219
weighted avg       0.66      0.66      0.66      7219
```

```
PCA percentage set to :  0.99
Accuracy :
0.6834741653968693
Validation set results with 2 Hidden layers :
              precision    recall  f1-score   support

           0       0.67      0.64      0.65      3350
           1       0.70      0.72      0.71      3869

    accuracy                           0.68      7219
   macro avg       0.68      0.68      0.68      7219
weighted avg       0.68      0.68      0.68      7219
```

In [ ]:
```
pca_clf = make_pipeline( PCA(n_components=23), MLPClassifier(hidden
pca_clf.fit(X_train, y_train)
pred_test_pca = pca_clf.predict(X_val)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

In [ ]:
```
print('Accuracy : ')
print(pca_clf.score(X_val, y_val))
print('Validation set results with 2 Hidden layers : ')
print(classification_report(y_val, pred_test_pca))
```

```
Accuracy :
0.662003047513506
Validation set results with 2 Hidden layers :
              precision    recall  f1-score   support

           0       0.65      0.59      0.62      3350
           1       0.67      0.73      0.70      3869

    accuracy                           0.66      7219
   macro avg       0.66      0.66      0.66      7219
weighted avg       0.66      0.66      0.66      7219
```

Trying different components - a trial and error method

In [ ]:
```
pca_clf_2 = make_pipeline( PCA(n_components = 'mle', svd_solver = '
pca_clf_2.fit(X_train, y_train)
pred_test_pca2 = pca_clf_2.predict(X_val)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

In [ ]:
```
print('Accuracy : ')
print(pca_clf_2.score(X_val, y_val))
print('Validation set results with 2 Hidden layers : ')
print(classification_report(y_val, pred_test_pca2))
```

```
Accuracy :
0.7050838066214157
Validation set results with 2 Hidden layers :
              precision    recall  f1-score   support

           0       0.68      0.70      0.69      3350
           1       0.73      0.71      0.72      3869

    accuracy                           0.71      7219
   macro avg       0.70      0.70      0.70      7219
weighted avg       0.71      0.71      0.71      7219
```

Ref : [https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html#sphx-glr-auto-examples-preprocessing-plot-scaling-importance-py (https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html#sphx-glr-auto-examples-preprocessing-plot-scaling-importance-py)](https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html#sphx-glr-auto-examples-preprocessing-plot-scaling-importance-py)

Though 0.95 percentage of data is being coverd by 19 variables it is not enough when trying to validate data, 0.99 has 27 attributes and performs better in validations set after being standerdised.

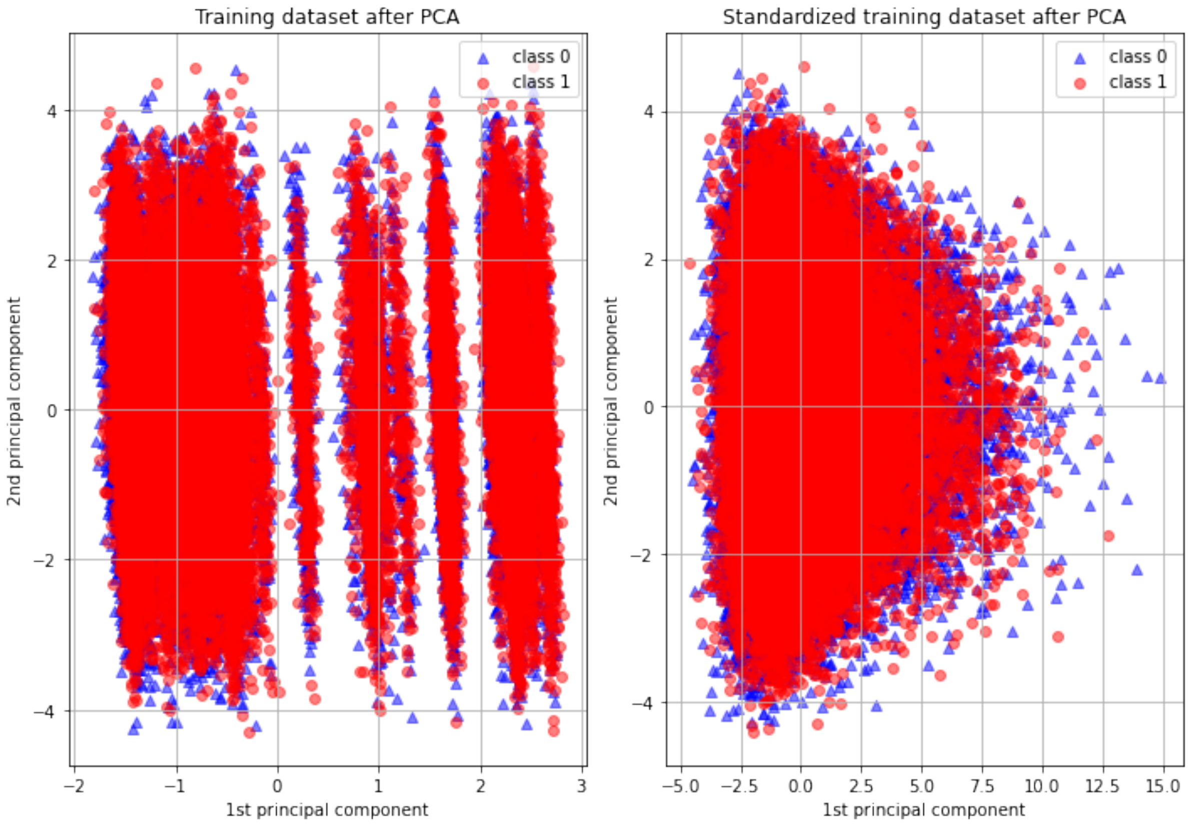
In [ ]:
```
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_wine
from sklearn.pipeline import make_pipeline
```

```
# Code source: Tyler Lanigan <tylerlanigan@gmail.com>
#              Sebastian Raschka <mail@sebastianraschka.com>

# License: BSD 3 clause

RANDOM_STATE = 42
FIG_SIZE = (10, 7)

# Fit to data and predict using pipelined GNB and PCA
unscaled_clf = make_pipeline(PCA(0.99), MLPClassifier(hidden_layer_
unscaled_clf.fit(X_train, y_train)
pred_val = unscaled_clf.predict(X_val)

# Fit to data and predict using pipelined scaling, GNB and PCA
std_clf = make_pipeline(StandardScaler(), PCA(0.99), MLPClassifier(
std_clf.fit(X_train, y_train)
pred_val_std = std_clf.predict(X_val)

# Show prediction accuracies in scaled and unscaled data.
print("\nPrediction accuracy for the normal test dataset with PCA")
print(f"{accuracy_score(y_val, pred_val):.2%}\n")

print("\nPrediction accuracy for the standardized test dataset with
print(f"{accuracy_score(y_val, pred_val_std):.2%}\n")

# Extract PCA from pipeline
pca = unscaled_clf.named_steps["pca"]
pca_std = std_clf.named_steps["pca"]

# Show first principal components
print(f"\nPC 1 without scaling:\n{pca.components_[0]}")
print(f"\nPC 1 with scaling:\n{pca_std.components_[0]}")

# Use PCA without and with scale on X_train data for visualization.
X_train_transformed = pca.transform(X_train)

scaler = std_clf.named_steps["standardscaler"]
scaled_X_train = scaler.transform(X_train)
X_train_std_transformed = pca_std.transform(scaled_X_train)

# visualize standardized vs. untouched dataset with PCA performed
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=FIG_SIZE)

target_classes = [0, 1]
colors = ("blue", "green")
markers = ("^", "o")

for target_class, color, marker in zip(target_classes, colors, mark
    ax1.scatter(
        x=X_train_transformed[y_train == target_class, 0],
        y=X_train_transformed[y_train == target_class, 1],
        color=color,
        label=f"class {target_class}",
        alpha=0.5,
```

```
        marker=marker,
    )

    ax2.scatter(
        x=X_train_std_transformed[y_train == target_class, 0],
        y=X_train_std_transformed[y_train == target_class, 1],
        color=color,
        label=f"class {target_class}",
        alpha=0.5,
        marker=marker,
    )

ax1.set_title("Training dataset after PCA")
ax2.set_title("Standardized training dataset after PCA")

for ax in (ax1, ax2):
    ax.set_xlabel("1st principal component")
    ax.set_ylabel("2nd principal component")
    ax.legend(loc="upper right")
    ax.grid()

plt.tight_layout()

plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (200) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,


Prediction accuracy for the normal test dataset with PCA
67.20%


Prediction accuracy for the standardized test dataset with PCA
70.36%


PC 1 without scaling:
[-7.03595148e-04  1.46454412e-02  1.64279089e-03  9.47974777e-04
  1.77170442e-03 -1.84975855e-03  1.35567298e-02 -6.66445639e-04
 -1.30711783e-01 -6.43604351e-03  5.13640744e-03 -9.25436632e-04
 -1.48526542e-01 -2.19166424e-02  2.16475334e-02 -2.17961652e-03
 -3.70944613e-01  7.45842500e-02  7.99694557e-03  1.81517625e-03
  8.99539320e-01  3.52627210e-02  2.60750925e-03 -1.03971660e-03
 -2.06626867e-02 -7.05069535e-02 -1.86641038e-02 -4.87246879e-03]

PC 1 with scaling:
[ 4.73842721e-02  1.76831036e-03  3.76020105e-03  1.20971354e-01
  3.74171843e-04  2.99602017e-01 -9.03509189e-04 -4.92521952e-03
  1.22575684e-02  2.86545905e-01 -6.23945297e-03  3.66827452e-03
  3.84292173e-02  2.27035850e-01  4.63020402e-04  5.97554442e-03
```

```
 5.03946609e-03  1.49930156e-01  4.44525694e-04 -1.69035493e-03
-2.93564451e-02  2.38845821e-01  3.16361943e-01  2.82667698e-02
 2.94895153e-01  2.89644430e-01  4.59875745e-01  4.40799674e-01]
```



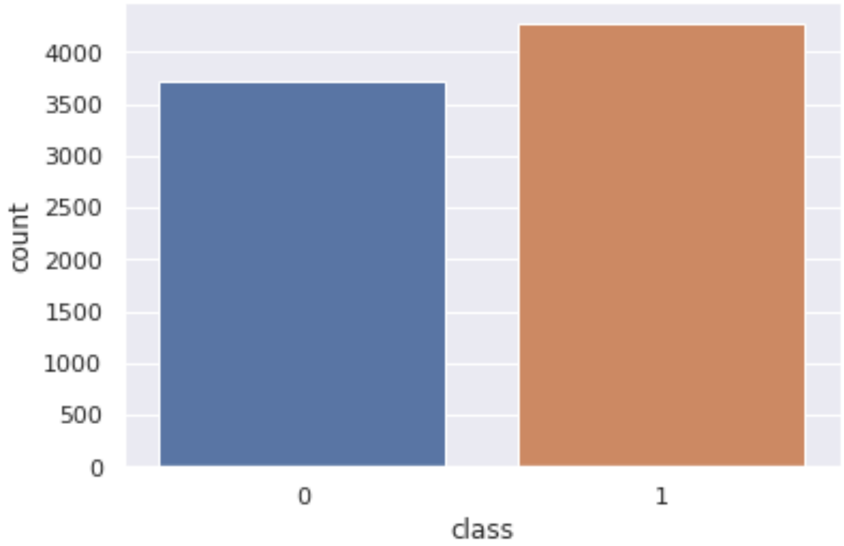
MLP model building on training set with the whole dataset is very time consuming, so subsampling few records to run grid search to get the best set of parameters. This best set of parameters willl be used to retrain the final model

PCA (0.99) with standerdised data gives better results hence sampleling 8000 records and performing standardization and PCA and run grid search to identify other parameters

In [ ]:
```
higgs_df_sample = higgs_df_train.sample(n = 8000)
higgs_df_sample.shape
```

Out[20]:
```
(8000, 29)
```

In [ ]:
```
sns.set_theme(style="darkgrid")
ax = sns.countplot(x="class", data=higgs_df_sample)
```



In [ ]:
```
y_sample = higgs_df_sample['class']
X_sample = higgs_df_sample.drop(columns='class')

X_train_sample, X_val_sample, y_train_sample, y_val_sample = train_
```

In [ ]:
```
scaler = StandardScaler()
# Fit on training set only.
scaler.fit(X_train_sample)
# Apply transform to both the training set and the test set.
X_train_sample = scaler.transform(X_train_sample)
X_val_sample = scaler.transform(X_val_sample)

from sklearn.decomposition import PCA
# Make an instance of the Model
pca = PCA(.99)

pca.fit(X_train_sample)

X_train_sample = pca.transform(X_train_sample)
X_val_sample = pca.transform(X_val_sample)

X_train_sample.shape
```

Out[107]: (7200, 23)

In [ ]:
```
import joblib
GOOGLE_MODELS_SAVED = GOOGLE_DRIVE_PATH + '/SavedModels/MLP'
```

In [ ]:
```
parameters = { 'max_iter': [100, 300, 500, 1000 ],  'random_state':
               'hidden_layer_sizes': [50], 'activation' : ['relu'] }
# parameters = {'activation' : ['logistic', 'relu', 'tanh']}

clf5 = GridSearchCV(MLPClassifier(), parameters, n_jobs=-1, verbose
print(clf5.best_params_)


GOOGLE_MODELS_SAVED = GOOGLE_DRIVE_PATH + '/SavedModels/MLP'

joblib.dump(clf5, GOOGLE_MODELS_SAVED + '/MLP_Grid_clf5_afterPCASca
```

```
Fitting 5 folds for each of 144 candidates, totalling 720 fits
{'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': 50, '
learning_rate_init': 0.001, 'max_iter': 100, 'random_state': 5}

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (100) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

Out[108]:
```
['drive/My Drive/ColabNotebooks/NN/CourseWork/HiggsDetection_Neura
lComputing/SavedModels/MLP/MLP_Grid_clf5_afterPCAScalar.pkl']
```

The best set of parameters on sampled data {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': 50, 'learning_rate_init': 0.001, 'max_iter': 100, 'random_state': 5}

Dataset Columns

Data set columns are ['lepton_pT', 'lepton_eta', 'lepton_phi', 'missing_energy_magnitude', 'missing_energy_phi', 'jet1pt', 'jet1eta', 'jet1phi', 'jet1b-tag', 'jet2pt', 'jet2eta', 'jet2phi', 'jet2b-tag', 'jet3pt', 'jet3eta', 'jet3phi', 'jet3b-tag', 'jet4pt', 'jet4eta', 'jet4phi', 'jet4b-tag'] are the actual

where as these are calucalated ['m_jj', 'm_jjj', 'm_lv', 'm_jlv', 'm_bb', 'm_wbb', 'm_wwbb']

The first set of attributes are not highly corelated but the second set of values which are the calculated fields arr highly corelated, experimenting on these columns to check if they seperatly have impacts on models

In [ ]:
```
X_train.shape
```

Out[177]:
```
(64965, 28)
```

In [ ]:
```
X_train.columns
```

Out[167]:
```
Index(['lepton_pT', 'lepton_eta', 'lepton_phi', 'missing_energy_ma
gnitude',
       'missing_energy_phi', 'jet1pt', 'jet1eta', 'jet1phi', 'jet1
b-tag',
       'jet2pt', 'jet2eta', 'jet2phi', 'jet2b-tag', 'jet3pt', 'jet
3eta',
       'jet3phi', 'jet3b-tag', 'jet4pt', 'jet4eta', 'jet4phi', 'je
t4b-tag',
       'm_jj', 'm_jjj', 'm_lv', 'm_jlv', 'm_bb', 'm_wbb', 'm_wwbb'
],
      dtype='object')
```

In [ ]:
```
X_train_high = X_train[['lepton_pT', 'lepton_eta', 'lepton_phi', 'm
       'missing_energy_phi', 'jet1pt', 'jet1eta', 'jet1phi', 'jet1b
       'jet2pt', 'jet2eta', 'jet2phi', 'jet2b-tag', 'jet3pt', 'jet3
       'jet3phi', 'jet3b-tag', 'jet4pt', 'jet4eta', 'jet4phi', 'jet

X_val_high = X_val[['lepton_pT', 'lepton_eta', 'lepton_phi', 'missi
       'missing_energy_phi', 'jet1pt', 'jet1eta', 'jet1phi', 'jet1b
       'jet2pt', 'jet2eta', 'jet2phi', 'jet2b-tag', 'jet3pt', 'jet3
       'jet3phi', 'jet3b-tag', 'jet4pt', 'jet4eta', 'jet4phi', 'jet
```

In [ ]:
```
MLPClf_6 = MLPClassifier(hidden_layer_sizes = (25, 30), activation

y_preds6=MLPClf_6.predict(X_val_high)
print('Accuracy : ')
print(MLPClf_6.score(X_val_high, y_val))
print('Validation set results with 2 Hidden layers : ')
print(classification_report(y_val, y_preds6))
```

```
Accuracy :
0.6395622662418617
Validation set results with 2 Hidden layers :
              precision    recall  f1-score   support

           0       0.62      0.57      0.59      3350
           1       0.65      0.70      0.68      3869

    accuracy                           0.64      7219
   macro avg       0.64      0.63      0.64      7219
weighted avg       0.64      0.64      0.64      7219


/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (100) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

In [ ]:
```
X_train_cal = X_train[['m_jj', 'm_jjj', 'm_lv', 'm_jlv', 'm_bb', 'm

X_val_cal = X_val[['m_jj', 'm_jjj', 'm_lv', 'm_jlv', 'm_bb', 'm_wbb

X_train_cal.shape
```

Out[11]:
```
(64965, 7)
```

In [ ]:
```
MLPClf_7 = MLPClassifier(hidden_layer_sizes = (25, 30), activation

y_preds7=MLPClf_7.predict(X_val_cal)
print('Accuracy : ')
print(MLPClf_7.score(X_val_cal, y_val))
print('Validation set results : ')
print(classification_report(y_val, y_preds7))
```

```
Accuracy :
0.6956642194209725
Validation set results :
              precision    recall  f1-score   support

           0       0.67      0.67      0.67      3350
           1       0.72      0.71      0.72      3869

    accuracy                           0.70      7219
   macro avg       0.69      0.69      0.69      7219
weighted avg       0.70      0.70      0.70      7219


/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (100) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

In [ ]:
```
X_train_cal = X_train_sample[['m_jj', 'm_jjj', 'm_lv', 'm_jlv', 'm_

X_val_cal = X_val_sample[['m_jj', 'm_jjj', 'm_lv', 'm_jlv', 'm_bb',

X_train_cal.shape
```

Out[24]:
```
(7200, 7)
```

In [ ]:
```
parameters = { 'max_iter': [100, 300, 500  ],  'random_state':[5, 1
               'hidden_layer_sizes': [5 , 10, (10, 30), 25 , (25, 35
# parameters = {'activation' : ['logistic', 'relu', 'tanh']}

clf7 = GridSearchCV(MLPClassifier(), parameters, n_jobs=-1, verbose
print(clf7.best_params_)


GOOGLE_MODELS_SAVED = GOOGLE_DRIVE_PATH + '/SavedModels/MLP'

joblib.dump(clf7, GOOGLE_MODELS_SAVED + '/MLP_Grid_clf7_aftercol_ca
```

```
Fitting 5 folds for each of 648 candidates, totalling 3240 fits
{'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': (25,
35), 'learning_rate_init': 0.005, 'max_iter': 300, 'random_state':
10}
```

Out[25]:
```
['drive/My Drive/ColabNotebooks/NN/CourseWork/HiggsDetection_Neura
lComputing/SavedModels/MLP/MLP_Grid_clf7_aftercol_calcol.pkl']
```

In [ ]:
```
pd.DataFrame(clf7.cv_results_)
```

Out[31]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_activation	param
0	0.440281	0.252509	0.003711	0.000639	relu	
1	1.271991	0.279760	0.003469	0.000522	relu	
2	1.832831	0.338331	0.003322	0.000289	relu	
3	0.816501	0.502040	0.004472	0.002829	relu	
4	2.220341	0.349802	0.003038	0.000404	relu	
...	...	...	...	...	...	
643	7.738694	0.574507	0.003953	0.000178	relu	

644	7.591255	0.527575	0.003832	0.000234	relu
645	8.316577	0.739338	0.003941	0.000134	relu
646	8.772695	2.099852	0.004692	0.001384	relu
647	7.582081	1.077411	0.003580	0.000618	relu

648 rows × 19 columns

RESULTS

In the above implementation it can be observed that the calculated fields alone ['m_jj', 'm_jjj', 'm_lv', 'm_jlv', 'm_bb', 'm_wbb', 'm_wwbb'] produce 70% accuracy on validation set. Because they are highly corelated, this could lead to overfitting and so we are not going to use only these variables to create model, we are going to stick with the previous approch where PCA was 0.99

BEST SET OF PARAMETERS TRAINING

From all the above observations the best set of parameters are chosen and MLP is trained

In [ ]:
```
MLPClf_8 = MLPClassifier(hidden_layer_sizes = (25, 35), activation

y_preds8=MLPClf_8.predict(X_val)
print('Accuracy : ')
print(MLPClf_8.score(X_val, y_val))
print('Validation set results with 2 Hidden layers : ')
print(classification_report(y_val, y_preds8))
```

```
Accuracy :
0.7168582906219698
Validation set results with 2 Hidden layers :
              precision    recall  f1-score   support

           0       0.69      0.71      0.70      3350
           1       0.74      0.72      0.73      3869

    accuracy                           0.72      7219
   macro avg       0.72      0.72      0.72      7219
weighted avg       0.72      0.72      0.72      7219
```

Model parameter chosen from paper

These set of parameters are chosen from the ref paper "*Searching for exotic particles in high-energy physics with deep learning by P. Baldi, P. Sadowski & D. Whiteson*". not all parameters are mentioned, so building on ehat is mentioned and the rest we are choosing from the above observed parameters

In [ ]:
```
MLPClf_10 = MLPClassifier(hidden_layer_sizes = (300, 300, 300, 300,

y_preds10=MLPClf_10.predict(X_val_sample)
print('Accuracy : ')
print(MLPClf_10.score(X_val_sample, y_val_sample))
print('Validation set results with 2 Hidden layers : ')
print(classification_report(y_val_sample, y_preds10))
```

```
Accuracy :
0.64625
Validation set results with 2 Hidden layers :
              precision    recall  f1-score   support

           0       0.59      0.82      0.68       373
           1       0.76      0.49      0.60       427

    accuracy                           0.65       800
   macro avg       0.67      0.66      0.64       800
weighted avg       0.68      0.65      0.64       800
```

In [ ]:

Best Model Training

Best set of parameters in whole dataset {'activation': 'relu', 'alpha': 0.001, 'hidden_layer_sizes': 50, 'learning_rate_init': 0.001, 'max_iter': 100, 'random_state': 5}

Standardize and PCA 0.99

In [11]:
```
scaler = StandardScaler()
scaler.fit(X_train)
# Apply transform to both the training set and the test set.
X_train_F = scaler.transform(X_train)
X_val_F = scaler.transform(X_val)

from sklearn.decomposition import PCA
# Make an instance of the Model
pca = PCA(.99)

pca.fit(X_train_F)

X_train_F = pca.transform(X_train_F)
X_val_F = pca.transform(X_val_F)

X_train_F.shape
```

Out[11]:
```
(69305, 27)
```

In [38]:
```
MLPClf_final = MLPClassifier(activation =  'relu', alpha = 0.001, h

y_preds5=MLPClf_final.predict(X_val_F)
print('Accuracy : ')
print(MLPClf_final.score(X_val_F, y_val))
print('Validation set results  : ')
print(classification_report(y_val, y_preds5))
```

```
Accuracy :
0.714582521750422
Validation set results  :
              precision    recall  f1-score   support

           0       0.70      0.68      0.69      3565
           1       0.73      0.74      0.74      4136

    accuracy                           0.71      7701
   macro avg       0.71      0.71      0.71      7701
weighted avg       0.71      0.71      0.71      7701
```

In [ ]:
```
MLPClf_final.get_params()
```

Out[45]:
```
{'activation': 'relu',
 'alpha': 0.001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (25, 30),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 100,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': 10,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
```

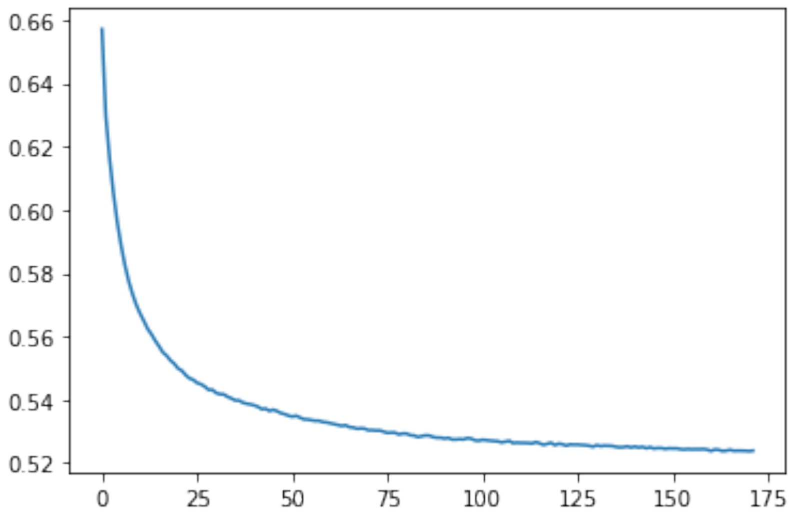
In [15]:
```
MPL_Final_cv_scores = cross_val_score(MLPClf_final, X_train, y_trai

MPL_Final_cv_scores
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (300) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (300) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (300) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_mul
tilayer_perceptron.py:696: ConvergenceWarning: Stochastic Optimize
r: Maximum iterations (300) reached and the optimization hasn't co
nverged yet.
  ConvergenceWarning,
```

Out[15]:
```
array([0.7055768 , 0.71351273, 0.70961691, 0.70781329, 0.70644254]
)
```

In [14]:
```
plt.plot(MLPClf_final.loss_curve_)
```

Out[14]:
```
[<matplotlib.lines.Line2D at 0x7f44854c6090>]
```

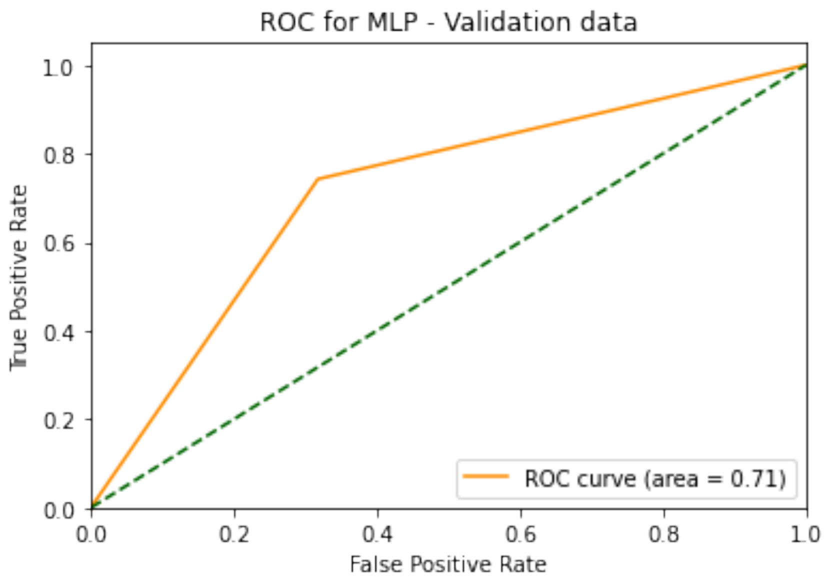


In [39]:
```
GOOGLE_MODELS_SAVED = GOOGLE_DRIVE_PATH + '/SavedModels/MLP'

joblib.dump(MLPClf_final, GOOGLE_MODELS_SAVED + '/MLP_FinalModelwit
```

Out[39]:
```
['drive/My Drive/ColabNotebooks/NN/CourseWork/HiggsDetection_Neura
lComputing/SavedModels/MLP/MLP_FinalModelwithBestParameters.pkl']
```

Ref : [https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html (https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html)](https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html)

In [50]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score

MLPVal_fpr, MLPVal_tpr, MLPVal_thresholds = roc_curve(y_val, y_pred
roc_auc = auc(MLPVal_fpr, MLPVal_tpr)

plt.figure()
plt.plot(MLPVal_fpr, MLPVal_tpr, color="darkorange",
label="ROC curve (area = %0.2f)" % roc_auc,)
plt.plot([0, 1], [0, 1], color="darkgreen",  linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC for MLP - Validation data")
plt.legend(loc="lower right")
plt.show()
```



In [22]:
```
#------------------------------------------------------END OF TRAI
```

In [ ]:
```
```

Testing

Now that the best set of parameters and best model is created, this model is saved and can be loaded any where by using **load(pkl_file_path)** . This need not be trained going further, any data can be directly tested with the below lined of code

In [58]:
```
from joblib import dump, load


TEST_GOOGLE_FOLDER = GOOGLE_DRIVE_PATH + '/SavedModels/TestingBestM
```

In [59]:
```
X_test = pd.read_csv(TEST_GOOGLE_FOLDER+'/X_test_data.csv')
y_test = pd.read_csv(TEST_GOOGLE_FOLDER+'/y_test_data.csv')
```

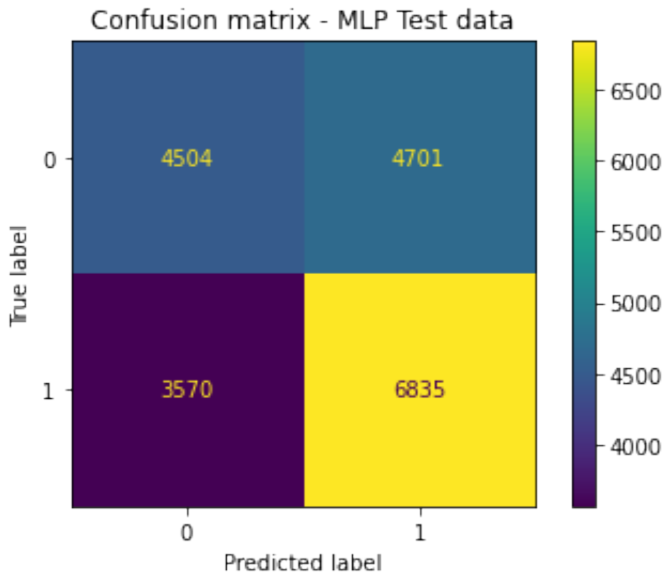
In [60]:
```
BestClassifier_MLP = load(TEST_GOOGLE_FOLDER + '/MLP_FinalModelwith
```

In [61]:
```
scaler_test = StandardScaler()
scaler_test.fit(X_test)

X_test = scaler_test.transform(X_test)


from sklearn.decomposition import PCA
# Make an instance of the Model
pca = PCA(n_components = 27)

pca.fit(X_test)

X_test = pca.transform(X_test)


y_preds_test = BestClassifier_MLP.predict(X_test)

print('MLP Accuracy on Test data: ')
print(BestClassifier_MLP.score(X_test, y_test))

print('MLP Results for Test data  : ')
print(classification_report(y_test, y_preds_test))
```

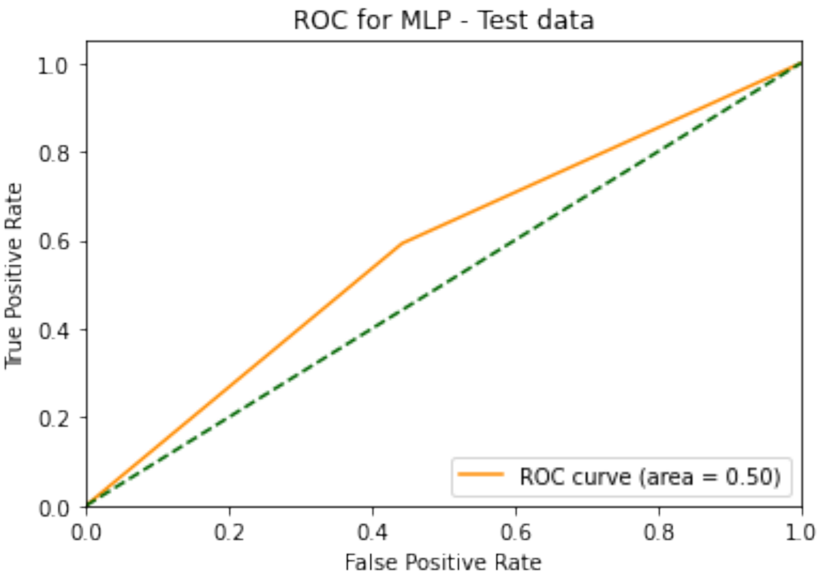
```
MLP Accuracy on Test data: 
0.5782253952065273
MLP Results for Test data  : 
              precision    recall  f1-score   support

           0       0.56      0.49      0.52      9205
           1       0.59      0.66      0.62     10405

    accuracy                           0.58     19610
   macro avg       0.58      0.57      0.57     19610
weighted avg       0.58      0.58      0.58     19610
```

In [62]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(BestClassifier_MLP, X_test, y_test)
plt.title('Confusion matrix - MLP Test data')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.p
y:87: FutureWarning: Function plot_confusion_matrix is deprecated;
Function `plot_confusion_matrix` is deprecated in 1.0 and will be
removed in 1.2. Use one of the class methods: ConfusionMatrixDispl
ay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



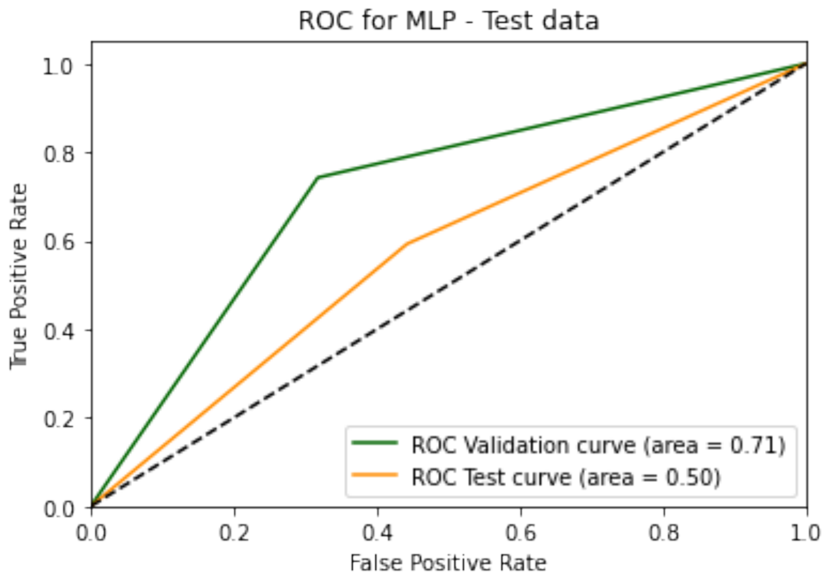
In [63]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score

MLPTEST_fpr, MLPTEST_tpr, MLPTEST_thresholds = roc_curve(y_preds_te
roc_auc_TEST = auc(MLPTEST_fpr, MLPTEST_fpr)

plt.figure()
plt.plot(MLPTEST_fpr, MLPTEST_tpr, color="darkorange",
label="ROC curve (area = %0.2f)" % roc_auc_TEST,)
plt.plot([0, 1], [0, 1], color="darkgreen",  linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC for MLP - Test data")
plt.legend(loc="lower right")
plt.show()
```



ROC - Validation and Test

In [ ]:

In [64]:
```
MLPTEST_fpr, MLPTEST_tpr, MLPTEST_thresholds = roc_curve(y_preds_te
roc_auc_TEST = auc(MLPTEST_fpr, MLPTEST_fpr)

plt.figure()
plt.plot(MLPVal_fpr, MLPVal_tpr, color="darkgreen",
label="ROC Validation curve (area = %0.2f)" % roc_auc,)
plt.plot(MLPTEST_fpr, MLPTEST_tpr, color="darkorange",
label="ROC Test curve (area = %0.2f)" % roc_auc_TEST,)
plt.plot([0, 1], [0, 1], color="black",  linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC for MLP - Test data")
plt.legend(loc="lower right")
plt.show()
```



In [ ]:
```
plt.plot(MLPVal_fpr, MLPVal_tpr, color="darkorange",
label="ROC curve (area = %0.2f)" % roc_auc,)
```

SVM Higgs Detection

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly re
mount, call drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
import os
from sklearn.metrics import accuracy_score
import joblib
from sklearn.model_selection import GridSearchCV
```

In [ ]:
```
import os

# TODO: Fill in the Google Drive path where you uploaded the lab ma
# Example: GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'Colab Notebooks/Lab m

GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'ColabNotebooks/NN/CourseWork/Hig
GOOGLE_DRIVE_PATH = os.path.join('drive', 'My Drive', GOOGLE_DRIVE_
# print(os.listdir(GOOGLE_DRIVE_PATH))
```

In [ ]:
```
higgs_df_train = pd.read_csv(GOOGLE_DRIVE_PATH + '/HiggsPreprocesse
higgs_df_train.shape
```

Out[5]:
```
(77006, 29)
```

In [ ]:
```
y_train = higgs_df_train['class']
X_train = higgs_df_train.drop(columns='class')
```

In [ ]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
```

In [ ]:
```
svm_model1 = SVC(kernel = 'linear')
#Fit the model for the data

SVMClf_1 = svm_model1.fit(X_train, y_train)
```

In [ ]:
```
y_preds=SVMClf_1.predict(X_val)
print('Summary for validation set with base model : ')
print(classification_report(y_val, y_preds))
```

```
Summary for validation set with base model :
              precision    recall  f1-score   support

           0       0.67      0.44      0.53      3350
           1       0.63      0.82      0.71      3869

    accuracy                           0.64      7219
   macro avg       0.65      0.63      0.62      7219
weighted avg       0.65      0.64      0.63      7219
```

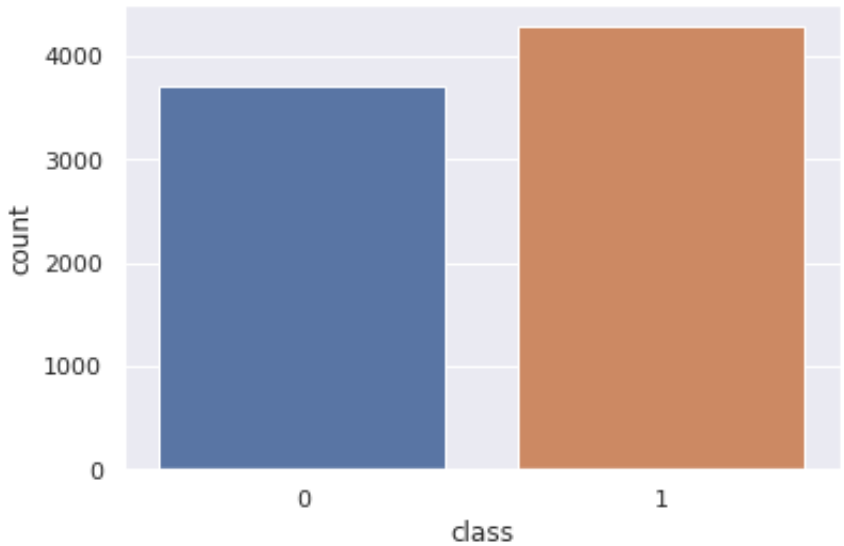
In [ ]:
```
import joblib
#save your model or results
GOOGLE_MODELS_SAVED = GOOGLE_DRIVE_PATH + '/SavedModels/SVM'
joblib.dump(SVMClf_1, GOOGLE_MODELS_SAVED + '/basic_model1_SVM_Higg
```

SVM basic linear model fit alone took upto 48 minutes to fit and predict. With a large dataset computational time for SVM is more and is not efficient. Further creating multiple models and performing grid search withh be very time consuming. For the purpose of course work, will sample fewer records from the dataset to create and compare models and will fit the whole test set with the best model.

In [ ]:
```
higgs_df_sample = higgs_df_train.sample(n = 8000)
higgs_df_sample.shape
```

Out[8]: (8000, 29)

In [ ]:
```
sns.set_theme(style="darkgrid")
ax = sns.countplot(x="class", data=higgs_df_sample)
```



In [ ]:
```
y_sample = higgs_df_sample['class']
X_sample = higgs_df_sample.drop(columns='class')

X_train_sample, X_val_sample, y_train_sample, y_val_sample = train_
```

In [ ]:
```
X_train_sample.shape
```

Out[16]:
```
(7200, 28)
```

In [ ]:
```
svm_model2 = SVC(kernel = 'linear', C=0.1, gamma=0.1)
#Fit the model for the data

SVMClf_2 = svm_model2.fit(X_train_sample, y_train_sample)

y_preds=SVMClf_2.predict(X_val_sample)
print('Summary on Validation set : ')
print(classification_report(y_val_sample, y_preds))

joblib.dump(SVMClf_2, GOOGLE_MODELS_SAVED + '/SVMClf_2 _model_SVM_H
```

```
Summary on Validation set :
              precision    recall  f1-score   support

           0       0.67      0.39      0.50       347
           1       0.65      0.85      0.74       453

    accuracy                           0.65       800
   macro avg       0.66      0.62      0.62       800
weighted avg       0.66      0.65      0.63       800
```

Out[17]:
```
['drive/My Drive/ColabNotebooks/NN/CourseWork/HiggsDetection_Neura
lComputing/SavedModels/SVM/SVMClf_2 _model_SVM_Higgs.pkl']
```

In [ ]:
```
# defining parameter range
param_grid = {'kernel': ['linear', 'rbf', 'poly', 'sigmoid']}

grid_SVM_model_kernal = GridSearchCV(SVC(), param_grid, refit = Tru

# fitting the model for grid search
grid_SVM_model_kernal = grid_SVM_model_kernal.fit(X_train_sample, y
print()
print("Best parameters set found on development set:")
print()
print(grid_SVM_model_kernal.best_params_)
joblib.dump(grid_SVM_model_kernal, GOOGLE_MODELS_SAVED + '/grid_SVM
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[CV 1/5] END .....................kernel=linear;, score=0.659 tota
l time=   6.1s
[CV 2/5] END .....................kernel=linear;, score=0.643 tota
```

```
l time=   3.5s
[CV 3/5] END .....................kernel=linear;, score=0.633 tota
l time=   3.5s
[CV 4/5] END .....................kernel=linear;, score=0.642 tota
l time=   3.5s
[CV 5/5] END .....................kernel=linear;, score=0.640 tota
l time=   3.4s
[CV 1/5] END ........................kernel=rbf;, score=0.642 tota
l time=   3.2s
[CV 2/5] END ........................kernel=rbf;, score=0.615 tota
l time=   3.2s
[CV 3/5] END ........................kernel=rbf;, score=0.631 tota
l time=   3.2s
[CV 4/5] END ........................kernel=rbf;, score=0.628 tota
l time=   3.2s
[CV 5/5] END ........................kernel=rbf;, score=0.624 tota
l time=   3.2s
[CV 1/5] END .......................kernel=poly;, score=0.624 tota
l time=   2.5s
[CV 2/5] END .......................kernel=poly;, score=0.608 tota
l time=   2.5s
[CV 3/5] END .......................kernel=poly;, score=0.612 tota
l time=   2.5s
[CV 4/5] END .......................kernel=poly;, score=0.617 tota
l time=   2.6s
[CV 5/5] END .......................kernel=poly;, score=0.617 tota
l time=   2.5s
[CV 1/5] END ....................kernel=sigmoid;, score=0.507 tota
l time=   3.6s
[CV 2/5] END ....................kernel=sigmoid;, score=0.492 tota
l time=   2.8s
[CV 3/5] END ....................kernel=sigmoid;, score=0.503 tota
l time=   3.4s
[CV 4/5] END ....................kernel=sigmoid;, score=0.474 tota
l time=   3.3s
[CV 5/5] END ....................kernel=sigmoid;, score=0.508 tota
l time=   3.2s

Best parameters set found on development set:

{'kernel': 'linear'}
```

Out[18]:

```
['drive/My Drive/ColabNotebooks/NN/CourseWork/HiggsDetection_Neura
lComputing/SavedModels/SVM/grid_SVM_model_bestkernal.pkl']
```

In [ ]:

```
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100],
              'gamma': [1, 0.1, 0.01],
              'kernel': ['linear']}

grid_SVM_model_C_gamma = GridSearchCV(SVC(), param_grid, refit = Tr

# fitting the model for grid search
grid_SVM_model_C_gamma.fit(X_train_sample,
```

```
grid_SVM_model_C_gamma = grid_SVM_model_C_gamma.fit(X_train_sample,
print()
print("Best parameters set found on development set:")
print()
print(grid_SVM_model_C_gamma.best_params_)
joblib.dump(grid_SVM_model_C_gamma, GOOGLE_DRIVE_PATH + '/grid_SVM_
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV 1/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.644 tota
l time=   3.7s
[CV 2/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.624 tota
l time=   3.2s
[CV 3/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.630 tota
l time=   2.0s
[CV 4/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.640 tota
l time=   2.1s
[CV 5/5] END .....C=0.1, gamma=1, kernel=linear;, score=0.628 tota
l time=   2.0s
[CV 1/5] END ...C=0.1, gamma=0.1, kernel=linear;, score=0.644 tota
l time=   2.1s
[CV 2/5] END ...C=0.1, gamma=0.1, kernel=linear;, score=0.624 tota
l time=   2.1s
[CV 3/5] END ...C=0.1, gamma=0.1, kernel=linear;, score=0.630 tota
l time=   2.1s
[CV 4/5] END ...C=0.1, gamma=0.1, kernel=linear;, score=0.640 tota
l time=   2.1s
[CV 5/5] END ...C=0.1, gamma=0.1, kernel=linear;, score=0.628 tota
l time=   2.0s
[CV 1/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.644 tota
l time=   2.1s
[CV 2/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.624 tota
l time=   2.1s
[CV 3/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.630 tota
l time=   2.1s
[CV 4/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.640 tota
l time=   2.1s
[CV 5/5] END ..C=0.1, gamma=0.01, kernel=linear;, score=0.628 tota
l time=   2.0s
[CV 1/5] END .......C=1, gamma=1, kernel=linear;, score=0.659 tota
l time=   3.4s
[CV 2/5] END .......C=1, gamma=1, kernel=linear;, score=0.643 tota
l time=   3.4s
[CV 3/5] END .......C=1, gamma=1, kernel=linear;, score=0.633 tota
l time=   3.5s
[CV 4/5] END .......C=1, gamma=1, kernel=linear;, score=0.642 tota
l time=   3.4s
[CV 5/5] END .......C=1, gamma=1, kernel=linear;, score=0.640 tota
l time=   3.5s
[CV 1/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.659 tota
l time=   3.4s
[CV 2/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.643 tota
l time=   3.4s
[CV 3/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.633 tota
l time=   3.5s
```

```
[CV 4/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.642 tota
l time=   3.4s
[CV 5/5] END .....C=1, gamma=0.1, kernel=linear;, score=0.640 tota
l time=   3.4s
[CV 1/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.659 tota
l time=   3.4s
[CV 2/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.643 tota
l time=   3.4s
[CV 3/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.633 tota
l time=   3.5s
[CV 4/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.642 tota
l time=   3.4s
[CV 5/5] END ....C=1, gamma=0.01, kernel=linear;, score=0.640 tota
l time=   3.4s
[CV 1/5] END ......C=10, gamma=1, kernel=linear;, score=0.659 tota
l time=  10.2s
[CV 2/5] END ......C=10, gamma=1, kernel=linear;, score=0.644 tota
l time=  10.1s
[CV 3/5] END ......C=10, gamma=1, kernel=linear;, score=0.639 tota
l time=  10.3s
[CV 4/5] END ......C=10, gamma=1, kernel=linear;, score=0.642 tota
l time=  10.3s
[CV 5/5] END ......C=10, gamma=1, kernel=linear;, score=0.640 tota
l time=  10.3s
[CV 1/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.659 tota
l time=  10.2s
[CV 2/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.644 tota
l time=  10.1s
[CV 3/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.639 tota
l time=  10.2s
[CV 4/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.642 tota
l time=  10.3s
[CV 5/5] END ....C=10, gamma=0.1, kernel=linear;, score=0.640 tota
l time=  10.4s
[CV 1/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.659 tota
l time=  10.2s
[CV 2/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.644 tota
l time=  10.1s
[CV 3/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.639 tota
l time=  10.4s
[CV 4/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.642 tota
l time=  10.3s
[CV 5/5] END ...C=10, gamma=0.01, kernel=linear;, score=0.640 tota
l time=  10.3s
[CV 1/5] END .....C=100, gamma=1, kernel=linear;, score=0.658 tota
l time= 1.1min
[CV 2/5] END .....C=100, gamma=1, kernel=linear;, score=0.644 tota
l time= 1.1min
[CV 3/5] END .....C=100, gamma=1, kernel=linear;, score=0.638 tota
l time= 1.1min
[CV 4/5] END .....C=100, gamma=1, kernel=linear;, score=0.642 tota
l time= 1.1min
[CV 5/5] END .....C=100, gamma=1, kernel=linear;, score=0.639 tota
```

```
l time= 1.1min
[CV 1/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.658 tota
l time= 1.1min
[CV 2/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.644 tota
l time= 1.1min
[CV 3/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.638 tota
l time= 1.1min
[CV 4/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.642 tota
l time= 1.1min
[CV 5/5] END ...C=100, gamma=0.1, kernel=linear;, score=0.639 tota
l time= 1.1min
[CV 1/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.658 tota
l time= 1.1min
[CV 2/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.644 tota
l time= 1.1min
[CV 3/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.638 tota
l time= 1.1min
[CV 4/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.642 tota
l time= 1.1min
[CV 5/5] END ..C=100, gamma=0.01, kernel=linear;, score=0.639 tota
l time= 1.1min

Best parameters set found on development set:

{'C': 10, 'gamma': 1, 'kernel': 'linear'}
```

Out[19]:
```
['drive/My Drive/ColabNotebooks/NN/CourseWork/HiggsDetection_Neura
lComputing/grid_SVM_model_C_and_gamma.pkl']
```

In [ ]:
```python
svm_model3 = SVC(kernel = 'linear', C = 10 , gamma = 1)
#Fit the model for the data

SVMClf_3 = svm_model3.fit(X_train_sample, y_train_sample)

y_preds = SVMClf_3.predict(X_val_sample)
print('Summary on Validation set : ')
print(classification_report(y_val_sample, y_preds))

joblib.dump(SVMClf_3, GOOGLE_MODELS_SAVED + '/SVMClf_3 _model_SVM_H
```

```
Summary on Validation set :
              precision    recall  f1-score   support

           0       0.66      0.44      0.53       347
           1       0.66      0.83      0.73       453

    accuracy                           0.66       800
   macro avg       0.66      0.63      0.63       800
weighted avg       0.66      0.66      0.64       800
```

Out[21]:
```
['drive/My Drive/ColabNotebooks/NN/CourseWork/HiggsDetection_Neura
lComputing/SavedModels/SVM/SVMClf_3 _model_SVM_Higgs.pkl']
```

Training on normalized data

In [ ]:
```
from sklearn import preprocessing

normalized_X_train = X_train_sample
normalized_X_val = X_val_sample

normalized_X_train = preprocessing.normalize(normalized_X_train)
normalized_X_val = preprocessing.normalize(normalized_X_val)

normalized_X_val = preprocessing.normalize(normalized_X_val)
```

In [ ]:
```
svm_model31 = SVC(kernel = 'linear', C = 10 , gamma = 1)
#Fit the model for the data

SVMClf_31 = svm_model31.fit(normalized_X_train, y_train_sample)

y_preds = SVMClf_31.predict(normalized_X_val)
print('Summary on Validation set : ')
print(classification_report(y_val_sample, y_preds))

# joblib.dump(SVMClf_3, GOOGLE_MODELS_SAVED + '/SVMClf_3 _model_SVM
```

```
Summary on Validation set :
              precision    recall  f1-score   support

           0       0.66      0.49      0.56       380
           1       0.63      0.77      0.69       420

    accuracy                           0.64       800
   macro avg       0.64      0.63      0.63       800
weighted avg       0.64      0.64      0.63       800
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit on training set only.
scaler.fit(X_train_sample)
# Apply transform to both the training set and the test set.
X_train_sample = scaler.transform(X_train_sample)
X_val_sample = scaler.transform(X_val_sample)

from sklearn.decomposition import PCA
# Make an instance of the Model
pca = PCA(.95)

pca.fit(X_train_sample)

X_train_sample = pca.transform(X_train_sample)
X_val_sample = pca.transform(X_val_sample)

X_train_sample.shape
```

Out[22]:
```
(7200, 23)
```

In [ ]:
```
svm_model4 = SVC(kernel = 'linear', C = 10 , gamma = 1)
#Fit the model for the data

SVMClf_4 = svm_model4.fit(X_train_sample, y_train_sample)

y_preds = SVMClf_4.predict(X_val_sample)
print('Summary on Validation set : ')
print(classification_report(y_val_sample, y_preds))

# joblib.dump(SVMClf_4, GOOGLE_MODELS_SAVED + '/SVMClf_4 _model_SVM
```

```
Summary on Validation set :
              precision    recall  f1-score   support

           0       0.62      0.38      0.47       347
           1       0.63      0.82      0.72       453

    accuracy                           0.63       800
   macro avg       0.63      0.60      0.59       800
weighted avg       0.63      0.63      0.61       800
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
# Fit on training set only.
scaler.fit(X_train_sample)
# Apply transform to both the training set and the test set.
X_train_sample = scaler.transform(X_train_sample)
X_val_sample = scaler.transform(X_val_sample)

from sklearn.decomposition import PCA
# Make an instance of the Model
pca = PCA(.99)

pca.fit(X_train_sample)

X_train_sample = pca.transform(X_train_sample)
X_val_sample = pca.transform(X_val_sample)

X_train_sample.shape
```

Out[28]:
```
(7200, 27)
```

In [ ]:
```
svm_model4 = SVC(kernel = 'linear', C = 10 , gamma = 1)
#Fit the model for the data

SVMClf_4 = svm_model4.fit(X_train_sample, y_train_sample)

y_preds = SVMClf_4.predict(X_val_sample)
print('Summary on Validation set : ')
print(classification_report(y_val_sample, y_preds))

# joblib.dump(SVMClf_4, GOOGLE_MODELS_SAVED + '/SVMClf_4 _model_SVM
```

```
Summary on Validation set :
              precision    recall  f1-score   support

           0       0.59      0.39      0.47       347
           1       0.63      0.79      0.70       453

    accuracy                           0.62       800
   macro avg       0.61      0.59      0.59       800
weighted avg       0.61      0.62      0.60       800
```

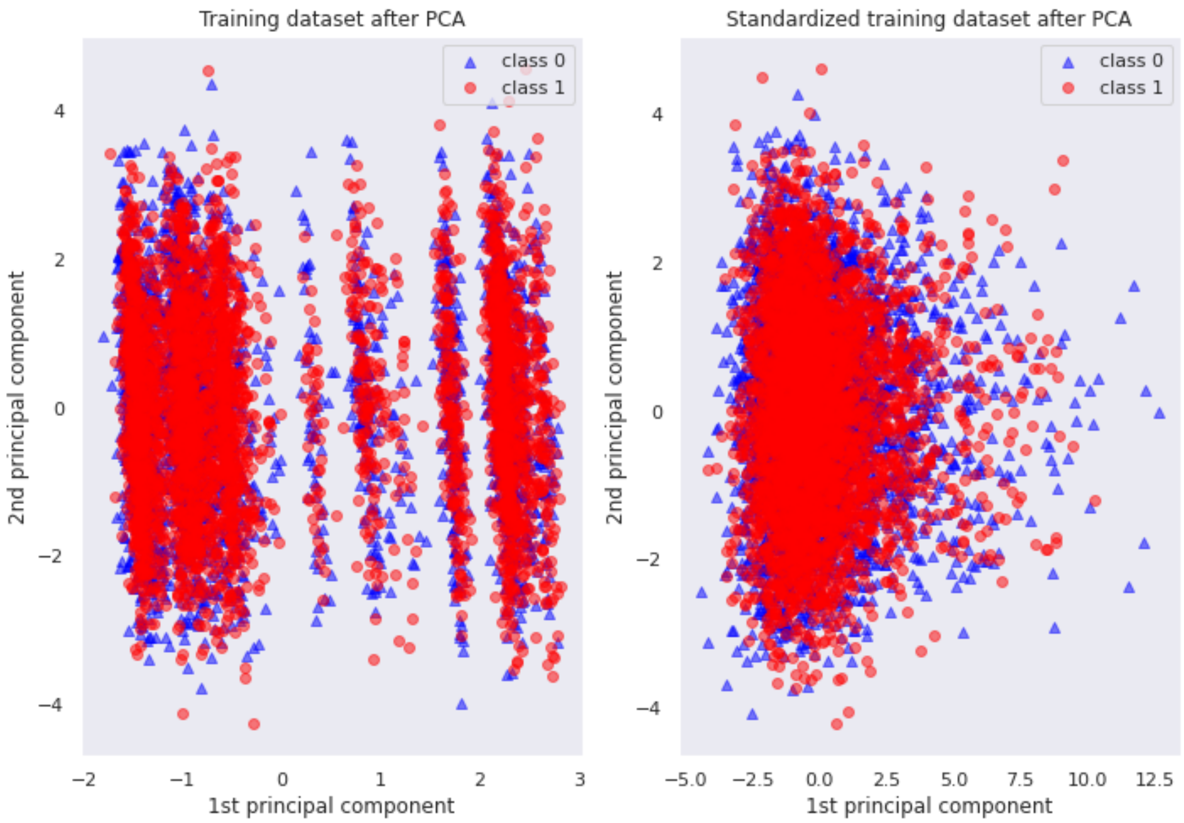
In [ ]:
```
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_wine
from sklearn.pipeline import make_pipeline
```

```
# Code source: Tyler Lanigan <tylerlanigan@gmail.com>
#              Sebastian Raschka <mail@sebastianraschka.com>

# License: BSD 3 clause

RANDOM_STATE = 42
FIG_SIZE = (10, 7)


# features, target = load_wine(return_X_y=True)

# Make a train/test split using 30% test size
# X_train, X_test, y_train, y_test = train_test_split(
#     features, target, test_size=0.30, random_state=RANDOM_STATE
# )

# Fit to data and predict using pipelined GNB and PCA
unscaled_clf = make_pipeline(PCA(0.99), SVC(kernel = 'linear', C =
unscaled_clf.fit(X_train_sample, y_train_sample)
pred_val = unscaled_clf.predict(X_val_sample)

# Fit to data and predict using pipelined scaling, GNB and PCA
std_clf = make_pipeline(StandardScaler(), PCA(0.99), SVC(kernel = '
std_clf.fit(X_train_sample, y_train_sample)
pred_val_std = std_clf.predict(X_val_sample)

# Show prediction accuracies in scaled and unscaled data.
print("\nPrediction accuracy for the normal test dataset with PCA")
print(f"{accuracy_score(y_val, pred_val):.2%}\n")

print("\nPrediction accuracy for the standardized test dataset with
print(f"{accuracy_score(y_val, pred_val_std):.2%}\n")

# Extract PCA from pipeline
pca = unscaled_clf.named_steps["pca"]
pca_std = std_clf.named_steps["pca"]

# Show first principal components
print(f"\nPC 1 without scaling:\n{pca.components_[0]}")
print(f"\nPC 1 with scaling:\n{pca_std.components_[0]}")

# Use PCA without and with scale on X_train data for visualization.
X_train_transformed = pca.transform(X_train_sample)

scaler = std_clf.named_steps["standardscaler"]
scaled_X_train = scaler.transform(X_train_sample)
X_train_std_transformed = pca_std.transform(scaled_X_train)

# visualize standardized vs. untouched dataset with PCA performed
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=FIG_SIZE)

target_classes = [0, 1]
colors = ("blue", "red")
markers = ("^", "o")
```

```
for target_class, color, marker in zip(target_classes, colors, mark
    ax1.scatter(
        x=X_train_transformed[y_train == target_class, 0],
        y=X_train_transformed[y_train == target_class, 1],
        color=color,
        label=f"class {target_class}",
        alpha=0.5,
        marker=marker,
    )

    ax2.scatter(
        x=X_train_std_transformed[y_train == target_class, 0],
        y=X_train_std_transformed[y_train == target_class, 1],
        color=color,
        label=f"class {target_class}",
        alpha=0.5,
        marker=marker,
    )

ax1.set_title("Training dataset after PCA")
ax2.set_title("Standardized training dataset after PCA")

for ax in (ax1, ax2):
    ax.set_xlabel("1st principal component")
    ax.set_ylabel("2nd principal component")
    ax.legend(loc="upper right")
    ax.grid()

plt.tight_layout()

plt.show()
```

```
Prediction accuracy for the normal test dataset with PCA
62.12%


Prediction accuracy for the standardized test dataset with PCA
62.00%


PC 1 without scaling:
[-3.24309922e-03  1.49926486e-02 -1.21846566e-02  7.57171355e-03
  6.00918268e-03 -3.27394097e-03  4.01161438e-02  1.62444608e-04
 -1.59162895e-01 -5.60269113e-03 -1.31761061e-02  7.94920888e-03
 -1.54566946e-01 -1.70598669e-02  3.12528197e-02 -2.89719883e-03
 -3.46989067e-01  7.34971606e-02  2.03995363e-03 -4.04930711e-03
  9.01891437e-01  3.44184169e-02 -7.39968491e-04  1.01493880e-03
 -2.08037336e-02 -7.64407069e-02 -2.02484338e-02 -5.51970138e-03]

PC 1 with scaling:
[ 6.51715631e-02  1.34457039e-02 -4.04833670e-04  1.12997877e-01
```

```
-1.70648508e-02  3.05128526e-01 -5.75391013e-03 -1.03181145e-03
 2.82347857e-02  2.89190867e-01 -1.13235763e-02  5.82754807e-04
 3.61390083e-02  2.26780798e-01 -1.00250514e-02  1.31923359e-02
-9.30866187e-03  1.43341213e-01  7.94130220e-03  7.11955952e-03
-3.57464722e-02  2.40850872e-01  3.16425394e-01  3.29860564e-02
 2.87638024e-01  2.94571478e-01  4.57889526e-01  4.37480515e-01]
```



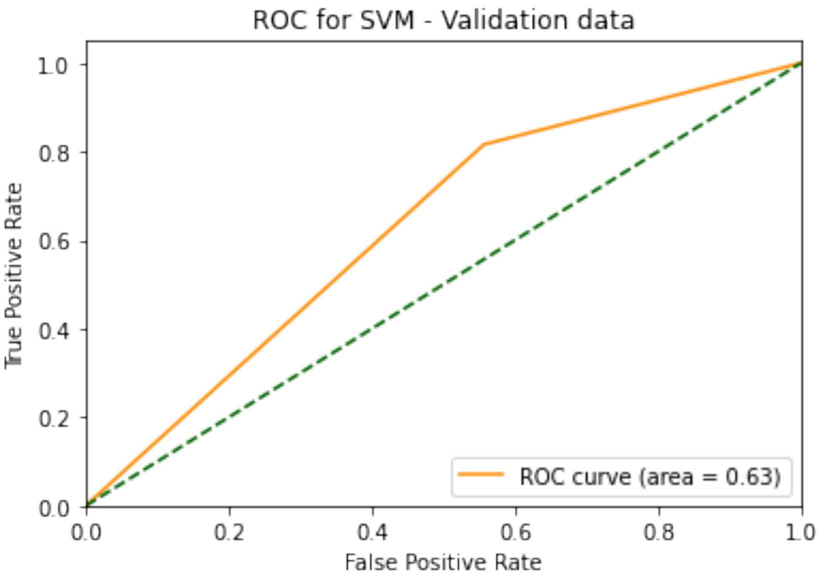
BEst SVM Model

In [ ]:
```
svm_model_final = SVC(kernel = 'linear', C = 10 )
# Fit the model for the data

SVMClf_Final = svm_model_final.fit(X_train, y_train)

y_preds = SVMClf_Final.predict(X_val)
print('Summary on Validation set : ')
print(classification_report(y_val, y_preds))

joblib.dump(SVMClf_Final, GOOGLE_MODELS_SAVED + '/SVMClf_Final _mod
```

```
Summary on Validation set :
              precision    recall  f1-score   support

           0       0.68      0.44      0.54      3573
           1       0.63      0.82      0.71      4128

    accuracy                           0.64      7701
   macro avg       0.65      0.63      0.62      7701
weighted avg       0.65      0.64      0.63      7701
```

Out[17]:
```
['drive/My Drive/ColabNotebooks/NN/CourseWork/HiggsDetection_Neura
lComputing/SavedModels/SVM/SVMClf_Final _model_SVM_Higgs.pkl']
```

In [ ]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score

SVMVal_fpr, SVMVal_tpr, SVMVal_thresholds = roc_curve(y_val, y_pred
roc_auc = auc(SVMVal_fpr, SVMVal_tpr)

plt.figure()
plt.plot(SVMVal_fpr, SVMVal_tpr, color="darkorange",
label="ROC curve (area = %0.2f)" % roc_auc,)
plt.plot([0, 1], [0, 1], color="darkgreen",  linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC for SVM - Validation data")
plt.legend(loc="lower right")
plt.show()
```



In [ ]:
```
svm_model_final = SVC(kernel = 'linear', C=10)
#Fit the model for the data

SVMClf_21 = svm_model_final.fit(X_train_sample, y_train_sample)

y_preds=SVMClf_21.predict(X_val_sample)
print('Summary on Validation set : ')
print(classification_report(y_val_sample, y_preds))

# joblib.dump(SVMClf_2, GOOGLE_MODELS_SAVED + '/SVMClf_21 _model_SV
```

```
Summary on Validation set :
              precision    recall  f1-score   support

           0       0.68      0.46      0.55       369
           1       0.64      0.82      0.72       431

    accuracy                           0.65       800
   macro avg       0.66      0.64      0.64       800
weighted avg       0.66      0.65      0.64       800
```

TESTING

In [ ]:
```
from joblib import dump, load
TEST_GOOGLE_FOLDER = GOOGLE_DRIVE_PATH + '/SavedModels/Testing'
```

In [ ]:
```
X_test = pd.read_csv(TEST_GOOGLE_FOLDER+'/X_test_data.csv')
y_test = pd.read_csv(TEST_GOOGLE_FOLDER+'/y_test_data.csv')
```

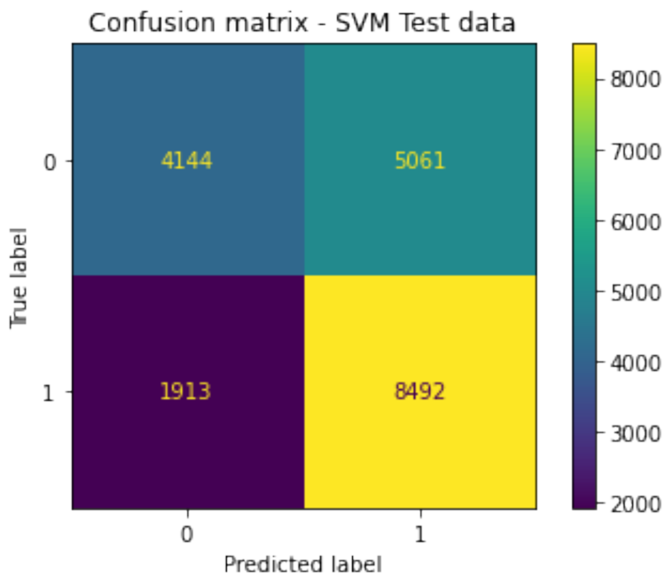
In [ ]:
```
BestClassifier_SVM = load(TEST_GOOGLE_FOLDER + '/SVMClf_Final _mode
```

In [ ]:
```
y_preds_test = BestClassifier_SVM.predict(X_test)

print('MLP Accuracy on Test data: ')
print(BestClassifier_SVM.score(X_test, y_test))

print('MLP Results for Test data  : ')
print(classification_report(y_test, y_preds_test))
```

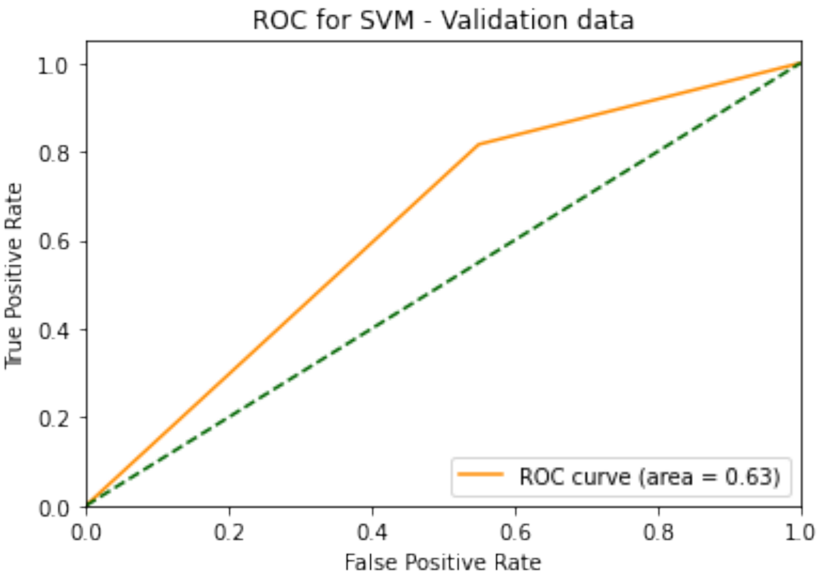
```
MLP Accuracy on Test data:
0.644365119836818
MLP Results for Test data  :
              precision    recall  f1-score   support

           0       0.68      0.45      0.54      9205
           1       0.63      0.82      0.71     10405

    accuracy                           0.64     19610
   macro avg       0.66      0.63      0.63     19610
weighted avg       0.65      0.64      0.63     19610
```

In [ ]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(BestClassifier_SVM, X_test, y_test)
plt.title('Confusion matrix - SVM Test data')
plt.show()
```

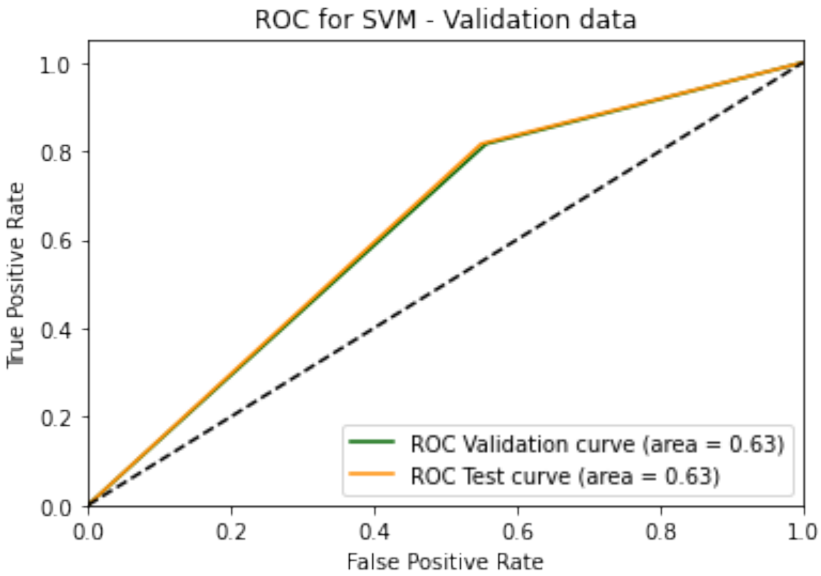
```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.p
y:87: FutureWarning: Function plot_confusion_matrix is deprecated;
Function `plot_confusion_matrix` is deprecated in 1.0 and will be
removed in 1.2. Use one of the class methods: ConfusionMatrixDispl
ay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



In [ ]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score

SVMTEST_fpr, SVMTEST_tpr, SVMTEST_thresholds = roc_curve(y_test, y_
roc_auc = auc(SVMTEST_fpr, SVMTEST_tpr)

plt.figure()
plt.plot(SVMTEST_fpr, SVMTEST_tpr, color="darkorange",
label="ROC curve (area = %0.2f)" % roc_auc,)
plt.plot([0, 1], [0, 1], color="darkgreen",  linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC for SVM - Validation data")
plt.legend(loc="lower right")
plt.show()
```



In [ ]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score

SVMTEST_fpr, SVMTEST_tpr, SVMTEST_thresholds = roc_curve(y_test, y_
roc_auc = auc(SVMTEST_fpr, SVMTEST_tpr)

plt.figure()
plt.plot(SVMVal_fpr, SVMVal_tpr, color="darkgreen",
label="ROC Validation curve (area = %0.2f)" % roc_auc,)
plt.plot(SVMTEST_fpr, SVMTEST_tpr, color="darkorange",
label="ROC Test curve (area = %0.2f)" % roc_auc,)
plt.plot([0, 1], [0, 1], color="black",  linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC for SVM - Validation data")
plt.legend(loc="lower right")
plt.show()
```



In [ ]:

TESTING BEST MODELS - MLP vs SVM

IMPORTS

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
import os
from sklearn.metrics import accuracy_score
import joblib
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
```

In [10]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.metrics import accuracy_score

from sklearn.pipeline import make_pipeline
from joblib import dump, load
```

In [6]:
```
import os
# code to run in my colab notebook
GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'ColabNotebooks/NN/CourseWork/Hig
GOOGLE_DRIVE_PATH = os.path.join('drive', 'My Drive', GOOGLE_DRIVE_
print(os.listdir(GOOGLE_DRIVE_PATH))
```

```
['TestingBestModels-MLP_SVM-HiggsDetection.ipynb', 'MLP_FinalModel
withBestParameters.pkl', 'y_test_data.csv', 'X_test_data.csv', 'SV
MClf_Final _model_SVM_Higgs.pkl']
```

If running in Google Drive uncomment and run the below code

In [ ]:
```
# If running in Google Drive uncomment and run the below code

#GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'Colab Notebooks/HiggsDetection_
```

In [9]:
```
X_test = pd.read_csv(GOOGLE_DRIVE_PATH + '/X_test_data.csv')
y_test = pd.read_csv(GOOGLE_DRIVE_PATH + '/y_test_data.csv')
```

In [ ]:
```
```

If running the same in Jupyter Notebook

run the below cells

In [ ]:
```
#If running the same in JUPYTER NOTEBOOK uncomment and run this cel

# X_test = pd.read_csv('X_test_data.csv')
# y_test = pd.read_csv('y_test_data.csv')
```

MLP Testing

Test data needs to be scaled and PCA of 27 components needs to be extracted

In [11]:
```
BestClassifier_MLP = load(GOOGLE_DRIVE_PATH + '/MLP_FinalModelwithB
```

In [ ]:
```
#If running the same in JUPYTER NOTEBOOK uncomment and run this cel

# BestClassifier_MLP = load('MLP_FinalModelwithBestParameters.pkl')
```

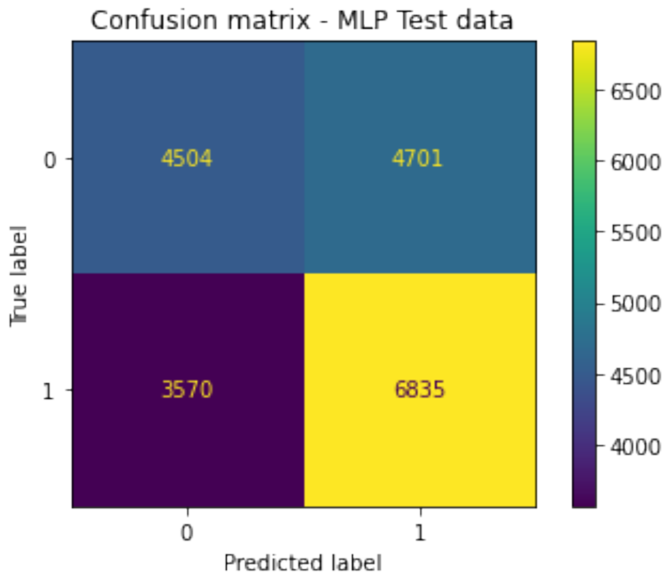
In [12]:

```
scaler_test = StandardScaler()
scaler_test.fit(X_test)

X_test = scaler_test.transform(X_test)


from sklearn.decomposition import PCA
# Make an instance of the Model
pca = PCA(n_components = 27)

pca.fit(X_test)

X_test = pca.transform(X_test)


y_preds_test = BestClassifier_MLP.predict(X_test)

print('MLP Accuracy on Test data: ')
print(BestClassifier_MLP.score(X_test, y_test))

print('MLP Results for Test data  : ')
print(classification_report(y_test, y_preds_test))
```

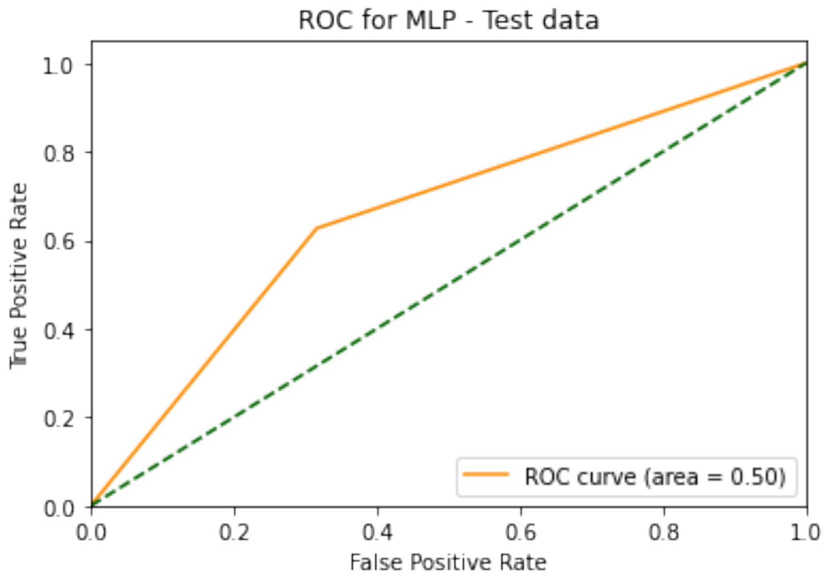
```
MLP Accuracy on Test data: 
0.5782253952065273
MLP Results for Test data  : 
              precision    recall  f1-score   support

           0       0.56      0.49      0.52      9205
           1       0.59      0.66      0.62     10405

    accuracy                           0.58     19610
   macro avg       0.58      0.57      0.57     19610
weighted avg       0.58      0.58      0.58     19610
```

In [13]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(BestClassifier_MLP, X_test, y_test)
plt.title('Confusion matrix - MLP Test data')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.p
y:87: FutureWarning: Function plot_confusion_matrix is deprecated;
Function `plot_confusion_matrix` is deprecated in 1.0 and will be
removed in 1.2. Use one of the class methods: ConfusionMatrixDispl
ay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



In [23]:
```
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score

MLPTEST_fpr, MLPTEST_tpr, MLPTEST_thresholds = roc_curve(y_preds_te
roc_auc_TEST = auc(MLPTEST_fpr, MLPTEST_fpr)

plt.figure()
plt.plot(MLPTEST_fpr, MLPTEST_tpr, color="darkorange",
label="ROC curve (area = %0.2f)" % roc_auc_TEST,)
plt.plot([0, 1], [0, 1], color="darkgreen",  linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC for MLP - Test data")
plt.legend(loc="lower right")
plt.show()
```



In [ ]:

SVM Testing

In [15]:
```
BestClassifier_SVM = load(GOOGLE_DRIVE_PATH + '/SVMClf_Final _model
```

In [ ]:
```
#If running the same in JUPYTER NOTEBOOK uncomment and run this cel

# BestClassifier_SVM = load('SVMClf_Final _model_SVM_Higgs.pkl')
```

In [ ]:
```
#If running the same in JUPYTER NOTEBOOK uncomment and run this cel

# X_test = pd.read_csv('X_test_data.csv')
# y_test = pd.read_csv('y_test_data.csv')
```

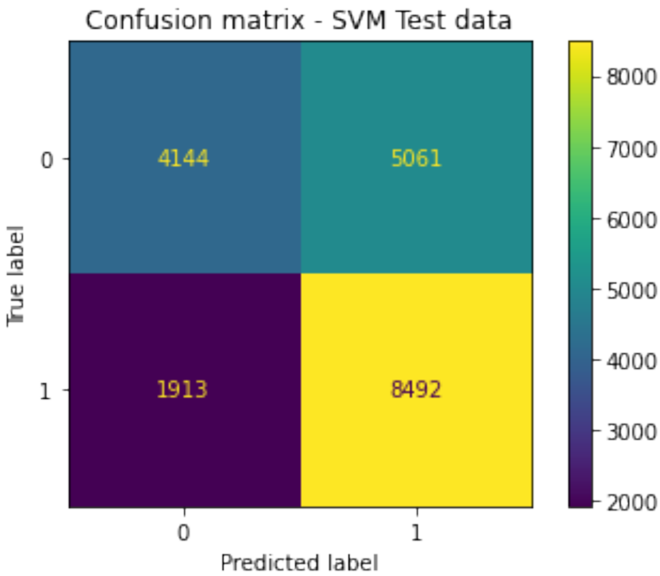
In [17]:
```
X_test = pd.read_csv(GOOGLE_DRIVE_PATH + '/X_test_data.csv')
y_test = pd.read_csv(GOOGLE_DRIVE_PATH + '/y_test_data.csv')
```

In [18]:
```
y_preds_test = BestClassifier_SVM.predict(X_test)

print('MLP Accuracy on Test data: ')
print(BestClassifier_SVM.score(X_test, y_test))

print('MLP Results for Test data  : ')
print(classification_report(y_test, y_preds_test))
```

```
MLP Accuracy on Test data: 
0.644365119836818
MLP Results for Test data  : 
              precision    recall  f1-score   support

           0       0.68      0.45      0.54      9205
           1       0.63      0.82      0.71     10405

    accuracy                           0.64     19610
   macro avg       0.66      0.63      0.63     19610
weighted avg       0.65      0.64      0.63     19610
```

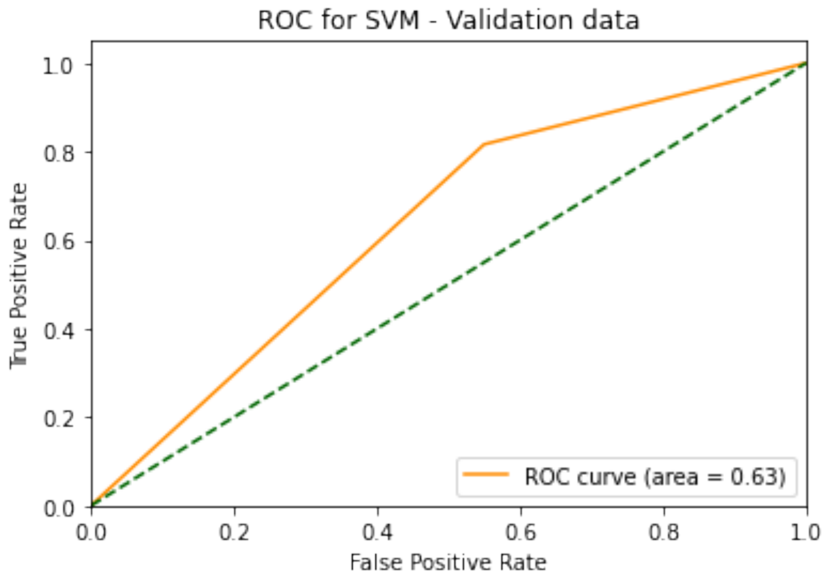
In [21]:
```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(BestClassifier_SVM, X_test, y_test)
plt.title('Confusion matrix - SVM Test data')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.p
y:87: FutureWarning: Function plot_confusion_matrix is deprecated;
Function `plot_confusion_matrix` is deprecated in 1.0 and will be
removed in 1.2. Use one of the class methods: ConfusionMatrixDispl
ay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



In [22]:

```
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score

SVMTEST_fpr, SVMTEST_tpr, SVMTEST_thresholds = roc_curve(y_test, y_
roc_auc = auc(SVMTEST_fpr, SVMTEST_tpr)

plt.figure()
plt.plot(SVMTEST_fpr, SVMTEST_tpr, color="darkorange",
label="ROC curve (area = %0.2f)" % roc_auc,)
plt.plot([0, 1], [0, 1], color="darkgreen",  linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC for SVM - Validation data")
plt.legend(loc="lower right")
plt.show()
```



In [ ]: