```
import itertools
import sys
import os
sys.path.insert(0, os.path.dirname(os.path.realpath('__file__')));
import pandas as pd
from sklearn.metrics import confusion_matrix, classification_report, precision_score, recall_score, log_loss, f1_score, mean_squared_error, roc_auc_score, accu
import time
from services.plotting_service import *
from services.neural_network_service import *
from services.preprocessing_service import *
from features.get_features import *
from keras.optimizers import SGD, Adam, RMSprop, Adagrad
from hyperopt import hp, Trials, fmin, tpe

#Query and prepreocess data
nn_data =  NeuralNetworkService().get_nn_data();
categorical_features = Features().get_categorical_features();
numerical_features = Features().get_numerical_features();
imputed_numerical_features = impute_missing_values(numerical_features);

X_train = nn_data['mortality_data']['X_train'];
X_test = nn_data['mortality_data']['X_test'];
Y_train = nn_data['mortality_data']['Y_train'];
Y_test = nn_data['mortality_data']['Y_test'];

def f(params):
    return NeuralNetworkService().train_neural_network(X_train, Y_train, X_test, Y_test, params);

params_space = {'n_layers': hp.choice('n_layers', range(2,8)),
                    'n_neurons': hp.choice('n_neurons', [8, 16, 32, 64]),
                    'optimizer': hp.choice('optimizer', ['SGD', 'Adam', 'RMSprop', 'Adagrad']),
                    'epochs': hp.choice('epochs', [10, 25, 35]),
                    'batch_size': hp.choice('batch_size', [1, 25, 50])
                };

trials_mse = Trials()

best = fmin(fn=f, space=params_space, algo=tpe.suggest, max_evals=50, trials=trials_mse);

# Training neural network with optimal paremeters
params = {'n_layers':6,
          'n_neurons':16,
          'optimizer': 'Adam',
          'epochs':50,
          'batch_size':50
          };

keras_model = NeuralNetworkService().train_neural_network(X_train, Y_train, X_test, Y_test, params);
patient_categorical_features = {'gender': 'M',
                                'marital_status':'SINGLE',
                                'religion':'CHRISTIAN',
                                'ethnicity':'WHITE',
                                'service':'CSURG',
                                'icd9_group':'diseases of the circulatory system',
                                'SURGERY_FLAG':'NARROW'
                               };

patient_numerical_features = {'age':80,
                              'total_icu_time':10,
                              'total_los_days':12,
                              'admissions_count':3,
                              'procedure_count':4,
                              'oasis_avg':40,
                              'sofa_avg':7,
                              'saps_avg':20,
                              'gcs':9,
                              'total_mech_vent_time':130
                              };

patient_lab_tests = {'blood_urea_nitrogen': [23, 24, 24],
                     'platelet_count':[230, 240],
                     'hematocrit':[33, 35],
                     'potassium': [3.9,3.8,4.4],
                     'sodium':[140, 139],
                     'creatinine':[1.3,1.2,1.3],
                     'bicarbonate':[25,26],
                     'white_blood_cells':[8.5,9,13],
                     'blood_glucose':[130, 135,140],
                     'albumin':[3.5, 3.4]
                     };

patient_physio_measures = {'heart_rate':[100,108,105,99],
                           'resp_rate':[22,25,23],
                           'sys_press':[120, 121, 115],
                           'dias_press':[70,80,85],
                           'temp':[98, 98.2, 97.8],
                           'spo2':[97,97.8,98]
                           };
patient_features = {
        'patient_categorical_features':patient_categorical_features,
        'patient_numerical_features':patient_numerical_features,
        'patient_lab_tests':patient_lab_tests,
        'patient_physio_measures':patient_physio_measures
        };


# Preprocess prediction data
prediction_data = NeuralNetworkService().preprocess_prediction_data(patient_features, imputed_numerical_features, categorical_features);
prediction = keras_model.predict(prediction_data);

n_clases = 3
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    Y_test = np.array(Y_test)
```

```
    Y_pred = np.array(Y_pred)
    fpr[i], tpr[i], _ = roc_curve(Y_test[:, i], Y_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

import seaborn as sns

sns.regplot(x=fpr[0], y = tpr[0])


import seaborn as sns
sns.set('talk', 'whitegrid', 'dark', font_scale=0.70, font='Ricty',
        rc={"lines.linewidth": 2, 'grid.linestyle': '--'})

def plot_roc_auroc_curves(tpr, fpr, roc_auc, title):
    plt.figure()
    plt.plot(fpr[1], tpr[1], color='green',
         lw=lw, label='ROC Curve (AUC = %0.2f)' % roc_auc[1])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, 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(title)
    plt.legend(loc="lower right")
    plt.show()
    plt.savefig('roc_auc.png')
    plt.close()

plot_roc_auroc_curves(tpr[2], fpr[2], roc_auc[2], 'ROC Curve for >12 months mortality')

all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])
plt.figure()

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='Curva ROC clase {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Ratio Falsos Positivos')
plt.ylabel('Ratio Veraderos Positivos')
plt.title('Curvas ROC para clasificación multiclase')
plt.legend(loc="lower right")
plt.show()



from mlxtend.evaluate import confusion_matrix

Y_test_ravel = Y_test.argmax(axis = 1)
Y_pred_ravel = Y_pred.argmax(axis = 1)

cm = confusion_matrix(y_target=Y_test_ravel,
                      y_predicted=Y_pred_ravel,
                      binary=False)

def plot_confusion_matrix_seaborn(cm):
    cm_norm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    r = sns.heatmap(cm_norm, annot=True,
            yticklabels=['<1 mes', '1-12 meses', '>12 meses'],
            xticklabels=['<1 mes', '1-12 meses', '>12 meses'],
            cbar = False)
    r.set_title('Matriz de confusión normalizada');
    r.set(xlabel='Valor Predicho', ylabel='Valor Real')
plot_confusion_matrix_seaborn(cm)

print(classification_report(Y_test_ravel, Y_pred_ravel))
```