A comparative study on optimized machine learning and deep learning models for the detection of electricity theft

MSc Research Project – Configuration Manual
MSc in Data Analytics

Oindrila Saha
Student ID: X21196061

School of Computing
National College of Ireland

Supervisor: Prof. Taimur Hafeez

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Oindrila Saha
Student ID:	X21196061
Programme:	MSc in Data Analytics **Year:** Sept 2022 - 2023
Module:	Research Project – Configuration Manual
Lecturer:	Prof. Taimur Hafeez
Submission Due Date:	09/11/2023
Project Title:	A comparative study on optimized machine learning and deep learning models for the detection of electricity theft
Word Count:	1182 **Page Count:** 19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Oindrila Saha

Date: 09/11/2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Oindrila Saha
X21196061

1 Introduction

This document is a detailed guide that outlines the step-by-step processes, necessary for successfully carrying out the research project named "A comparative study on optimized machine learning and deep learning models for the detection of electricity theft". The manual offers extensive details on the data resources, system requirements, code, and libraries used for implementing and evaluating research projects.

Section 2 outlines the essential system requirements needed for the research. Section 3 offers a comprehensive outline of the data collection method. Section 4 provides a detailed explanation of the step-by-step techniques involved in the data pre-processing process. The fifth section provides a detailed analysis of the procedural processes involved in the implementation, as well as the evaluation of various models. The report's conclusion is outlined in the last section.

2 System Requirements

This section provides information on the necessary hardware and software requirements for executing the project.

2.1 Specifications for hardware

Chip	Apple M2
Memory	8 GB
Start-up Disk	Macintosh HD
macOS	Ventura 13.2

Table1. Specifications of Hardware

2.2 Specifications for software

The project was executed using Python and the code was created under the Google Colab environment to take advantage of the complementary GPU provided. The default GPU of Google Colab, a Tesla T4 GPU, is utilized in this project.

```
!pip install pyswarms
```

```
Collecting pyswarms
  Downloading pyswarms-1.3.0-py2.py3-none-any.whl (104 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 104.1/104.1 kB 2.6 MB/s eta 0:00:00
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from pyswarms) (1.11.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pyswarms) (1.23.5)
Requirement already satisfied: matplotlib>=1.3.1 in /usr/local/lib/python3.10/dist-packages (from pyswarms) (3.7.1)
Requirement already satisfied: attrs in /usr/local/lib/python3.10/dist-packages (from pyswarms) (23.1.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from pyswarms) (4.66.1)
Requirement already satisfied: future in /usr/local/lib/python3.10/dist-packages (from pyswarms) (0.18.3)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from pyswarms) (6.0.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (1.1.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (4.43.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib>=1.3.1->pyswarms) (1.16.0)
Installing collected packages: pyswarms
Successfully installed pyswarms-1.3.0
```

```
from tensorflow.keras import backend as K
K.clear_session()
```

```
pip install np_utils
```

```
Collecting np_utils
  Downloading np_utils-0.6.0.tar.gz (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 1.4 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.0 in /usr/local/lib/python3.10/dist-packages (from np_utils) (1.23.5)
Building wheels for collected packages: np_utils
  Building wheel for np_utils (setup.py) ... done
  Created wheel for np_utils: filename=np_utils-0.6.0-py3-none-any.whl size=56438 sha256=654addd682e87bb7a295cb8197d76f406e2f19c7dd6e7fc2c89e994767fec6de
  Stored in directory: /root/.cache/pip/wheels/b6/c7/50/2307607f44366dd021209f660045f8d51cb976514d30be7cc7
Successfully built np_utils
Installing collected packages: np_utils
Successfully installed np_utils-0.6.0
```

```
#importing all libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
import joblib
import sys
import pickle
import plotly.graph_objects as go
import plotly.express as px
import plotly.figure_factory as ff
import imblearn
sys.modules['sklearn.externals.joblib'] = joblib
from sklearn.metrics import confusion_matrix,accuracy_score
from imblearn.under_sampling import RandomUnderSampler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn import ensemble
from keras.models import Sequential
from keras.layers import Conv1D
from keras.layers import MaxPooling1D
from keras.layers import Layer
from keras.layers import Flatten
from keras.layers import Dense
# from keras.utils import np_utils
from keras.callbacks import ModelCheckpoint
from keras.callbacks import EarlyStopping
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Activation
from keras.layers import Convolution1D
import tensorflow as tf
from tensorflow.keras.optimizers import Adam, SGD
import keras
from keras import Model, Sequential, backend
from keras.layers import LSTM, Dense, Dropout, Bidirectional
from keras.callbacks import EarlyStopping
from keras.layers import Input
from keras.utils import to_categorical
```

Fig 1. Necessary Python libraries for model implementation

The libraries that were employed for the implementation are depicted in Fig 1. Particle Swarm Optimization (PSO) algorithms are implemented using Pyswarms, whereas the development of deep learning models is accomplished with Keras and TensorFlow. In addition to facilitating the development of machine learning models, scikit-learn (sklearn) is employed to partition and pre-process data. Plotly, seaborn, and matplotlib are libraries that are employed to visualize data.

3 Data Acquisition

The information required to detect larceny in a smart grid system is gleaned from Mendeley data, which is available to the public via their official website. Synthetic data has been employed by the author in order to address concerns related to privacy. The dataset consists of energy consumption information pertaining to sixteen unique user categories. The initial dataset comprises a multitude of energy consumption measurements collected for a variety of clients over the course of a year, which includes twelve months. In this dataset, hourly observations are documented. The data utilized in this study is predominantly obtained from the OEDI platform.

4 Pre-processing of Data

This section encompasses multiple steps pertaining to data pre-processing. The collected data undergoes a cleansing process to remove any inconsistencies or inaccuracies. Prior to meaningful comparisons across various attributes, the data underwent formatting and normalization procedures in order to ensure consistency.

```
#checking for null values
df.isna().sum()
```

```
Electricity:Facility [kW](Hourly)              0
Fans:Electricity [kW](Hourly)                  0
Cooling:Electricity [kW](Hourly)               0
Heating:Electricity [kW](Hourly)               0
InteriorLights:Electricity [kW](Hourly)        0
InteriorEquipment:Electricity [kW](Hourly)     0
Gas:Facility [kW](Hourly)                      0
Heating:Gas [kW](Hourly)                       0
InteriorEquipment:Gas [kW](Hourly)             0
Water Heater:WaterSystems:Gas [kW](Hourly)     0
Class                                          0
theft                                          0
dtype: int64
```

```
#dropping unappropriate values from the dataset
df = df[df['Class'] != '0']
```

Fig 2. Performing a null value check and removing any improper values

Data Preprocessing

```
[ ] col = df.select_dtypes(exclude=['float64','int64']).columns.tolist()
    col

    ['Class', 'theft']
```

```
[ ] #label encoding(converting categorical data to number)
    le = LabelEncoder()
    df[col] = df[col].apply(le.fit_transform)
```

```
df.head()
```

riorLights:Electricity [kW](Hourly)	InteriorEquipment:Electricity [kW](Hourly)	Gas:Facility [kW](Hourly)	Heating:Gas [kW] (Hourly)	InteriorEquipment:Gas [kW](Hourly)	Water Heater:WaterSystems:Gas [kW](Hourly)	Class	theft
4.589925	8.1892	136.585903	123.999076	3.33988	9.246947	0	0
1.529975	7.4902	3.359880	0.000000	3.33988	0.020000	0	0
1.529975	7.4902	3.359880	0.000000	3.33988	0.020000	0	0
1.529975	7.4902	3.931932	0.000000	3.33988	0.592052	0	0
1.529975	7.4902	3.359880	0.000000	3.33988	0.020000	0	0

Fig 3. Label encoding from categorical to numerical.

Figure 3 illustrates the label encoding process for two categorical variables, Class, and Theft.

```
#splitting data into X and y.
X = df.drop('theft',axis='columns')
Y = df['theft']
```

```
#data is imbalanced
df['theft'].value_counts()
```

```
0     331824
1      51083
3      44349
4      41460
6      35413
5      33553
2      22958
Name: theft, dtype: int64
```

```
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.1,stratify=Y)
```

Fig 4. Dividing the data to training and testing sets and assessing the data imbalance

The data was partitioned into two segments. The subsequent coding stage utilizes the X_train and y_train training sets, as well as the X_test and y_test testing sets.

```
#model feature importance
fearture_name = X_train.columns.values
model = ensemble.ExtraTreesRegressor(n_estimators=25, max_depth=30, max_features=0.3, n_jobs=-1, random_state=0)
model.fit(X_train,y_train)
#plot imp
importance = model.feature_importances_
std = np.std([tree.feature_importances_ for tree in model.estimators_],axis=0)
indices = np.argsort(importance)[::-1][:10]
plt.figure(figsize=(7,7))
plt.title("Feature importances")
plt.bar(range(len(indices)), importance[indices], color="b")
plt.xticks(range(len(indices)), fearture_name[indices], rotation='vertical')
plt.xlim([-1, len(indices)])
plt.show()
```

Fig 5. Feature Importance of the model

Feature importance analysis is conducted to assess the influence or significance of various features (columns) in a dataset in relation to the target variable. Gaining insight into the elements that have the greatest impact on achieving precise forecasts or classifications is beneficial.

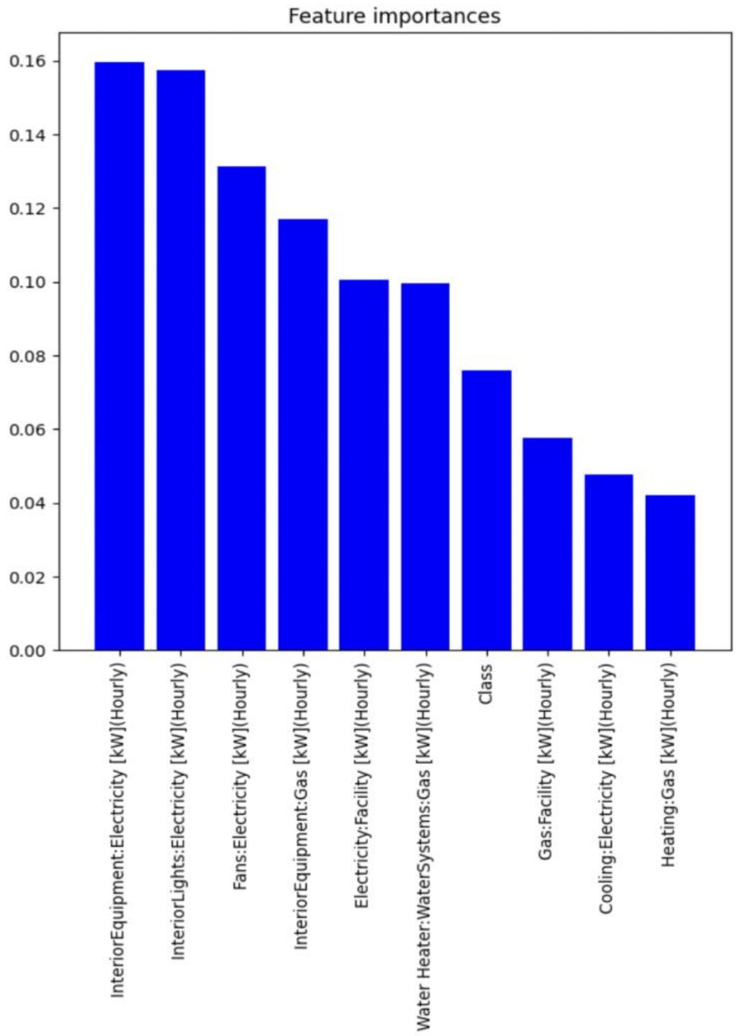


Fig 6. Feature extraction

Fig 6 clearly demonstrates that the InteriorEquipment:Electricity [kw](hourly) column has the most impact on predicting the target.

5 Implementation and Evaluation

This part offers a comprehensive summary of the diverse models employed in this study. Advanced machine learning and deep learning techniques were utilized to detect illicit activities. There are three ML and two DL models, optimized by PSO, were employed in this paper: XGBoost with PSO, Random Forest with PSO, Decision Tree with PSO, CNN with PSO and LSTM with PSO. The XGBoost with PSO model achieved the highest accuracy. Random Forest with PSO and LSTM with PSO achieved an accuracy of 83% and 82% respectively. The Decision Tree Classifier with PSO achieved comparatively poor accuracy which is 67%. The CNN with PSO model performed poorly and only yielded 59% accuracy which is the lowest one among all the models.

- <u>XGBoost with PSO</u>

```python
#XGB
import xgboost as xgb
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
pip install pyswarm
```

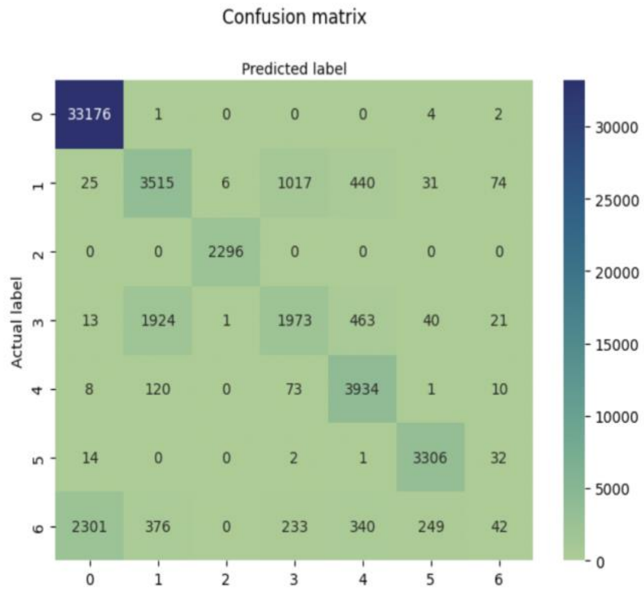
```
Collecting pyswarm
  Downloading pyswarm-0.6.tar.gz (4.3 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pyswarm) (1.23.5)
Building wheels for collected packages: pyswarm
  Building wheel for pyswarm (setup.py) ... done
  Created wheel for pyswarm: filename=pyswarm-0.6-py3-none-any.whl size=4464 sha256=587587cbd7d8d00bb2e00119837feec69bcf6bafd1f886c70585996f3b483ff9
  Stored in directory: /root/.cache/pip/wheels/71/67/40/62fa158f497f942277cbab8199b05cb61c571ab324e67ad0d6
Successfully built pyswarm
Installing collected packages: pyswarm
Successfully installed pyswarm-0.6
```

```python
#
# Define the objective function to optimize XGBoost hyperparameters using PSO
def objective_function(params):
    # max_depth, learning_rate, n_estimators, gamma, min_child_weight = params
    max_depth, learning_rate = params

    model = xgb.XGBClassifier(
        max_depth=int(max_depth),
        learning_rate=learning_rate,
        objective='multi:softmax',  # Multiclass classification
        num_class=len(np.unique(y_train))  # Number of classes
    )
    print("-----")
    # Fit the model to the training data
    model.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = model.predict(X_test)

    # Calculate the negative accuracy (PSO minimizes, so we negate accuracy)
    accuracy = -accuracy_score(y_test, y_pred)

    return accuracy


# Define the search space for hyperparameters
lb = [1, 0.01]  # Lower bounds for max_depth, learning_rate
ub = [10, 0.3] # Upper bounds for max_depth, learning_rate

# Use PSO to optimize the hyperparameters
# best_params, _ = pso(objective_function, lb, ub, swarmsize=10, maxiter=50)
best_params, _ = pso(objective_function, lb, ub, swarmsize=10, maxiter=1)

# Extract the best hyperparameters
best_max_depth, best_learning_rate = best_params

# Train the final XGBoost model with the best hyperparameters
final_model = xgb.XGBClassifier(
    max_depth=int(best_max_depth),
    learning_rate=best_learning_rate,
    objective='multi:softmax',  # Multiclass classification
    num_class=len(np.unique(y_train))  # Number of classes
)
final_model.fit(X_train, y_train)
```

```python
# Make predictions on the test data with the final model
y_pred = final_model.predict(X_test)

# Evaluate the final model's performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with Best Hyperparameters: {accuracy}")
```

```
Stopping search: maximum iterations reached --> 1
Accuracy with Best Hyperparameters: 0.8604808789954338
```

```python
#confusion Matrix
matrix =confusion_matrix(y_test, y_pred)
class_names=[1,2,3,4]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="crest" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
#Classification Report
print(classification_report(y_test, y_pred))
```




```
              precision    recall  f1-score   support

           0       0.93      1.00      0.97     33183
           1       0.59      0.69      0.64      5108
           2       1.00      1.00      1.00      2296
           3       0.60      0.44      0.51      4435
           4       0.76      0.95      0.84      4146
           5       0.91      0.99      0.95      3355
           6       0.23      0.01      0.02      3541

    accuracy                           0.86     56064
   macro avg       0.72      0.73      0.70     56064
weighted avg       0.82      0.86      0.83     56064
```

Figure 7. Construction and assessment of XGBoost with PSO

- <u>Random Forest with Particle Swarm Optimization</u>

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(504576, 11) (56064, 11) (504576,) (56064,)
```

```
# Random forest Tree
# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score


# Define the fitness function
def fitness_function(params):
    n_estimators = int(params[0])
    max_depth = int(params[1])
    print("-----")

    # Create a Random Forest classifier with the specified hyperparameters
    clf = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth, criterion='gini',random_state=42)
    # model.fit(X_train, y_train)

    # Calculate cross-validation accuracy
    # accuracy = np.mean(cross_val_score(clf, X_train, y_train, cv=5))
    clf.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = clf.predict(X_test)
    # Since PSO minimizes, we want to maximize accuracy, so we return its negative
    accuracy = -accuracy_score(y_test, y_pred)
    classification_report(y_test, y_pred)
    print(matrix)
    return -accuracy

# Define the parameter space
# For simplicity, let's consider n_estimators in the range of [10, 100] and max_depth in the range of [1, 20]
lb = [1, 1]  # Lower bounds for n_estimators and max_depth
ub = [2, 20]  # Upper bounds for n_estimators and max_depth

# Use PSO to find the optimal hyperparameters
best_params, _ = pso(fitness_function, lb, ub, swarmsize=10, maxiter=1)
# best_max_depth, best_learning_rate = best_params

# Extract the best hyperparameters
best_n_estimators = int(best_params[0])
best_max_depth = int(best_params[1])
```

```
Stopping search: maximum iterations reached --> 1
Accuracy with Best Hyperparameters: 0.6328303367579908
```

```
best_n_estimators,best_max_depth
```

```
(66, 1)
```

```
# Train the final Random Forest classifier with the best hyperparameters
final_classifier = RandomForestClassifier(n_estimators=best_n_estimators, max_depth=best_max_depth, criterion='gini',random_state=42)
# final_classifier = RandomForestClassifier(n_estimators=2, criterion='gini',random_state=42)
final_classifier.fit(X_train, y_train)

# Make predictions on the test data with the final model
y_pred = final_classifier.predict(X_test)

# Evaluate the final model's performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with Best Hyperparameters: {accuracy}")
```

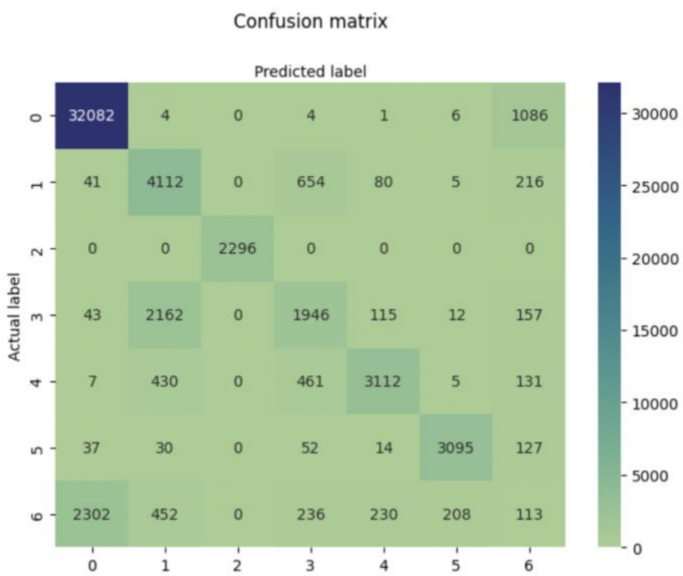
```
Accuracy with Best Hyperparameters: 0.8339754566210046
```

```
# y_pred
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95     33183
           1       0.57      0.81      0.67      5108
           2       1.00      1.00      1.00      2296
           3       0.58      0.44      0.50      4435
           4       0.88      0.75      0.81      4146
           5       0.93      0.92      0.93      3355
           6       0.06      0.03      0.04      3541

    accuracy                           0.83     56064
   macro avg       0.71      0.70      0.70     56064
weighted avg       0.81      0.83      0.82     56064
```

```
#confusion Matrix
matrix =confusion_matrix(y_test, y_pred)
class_names=[1,2,3,4]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="crest" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
#Classification Report
print(classification_report(y_test, y_pred))
```




```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95     33183
           1       0.57      0.81      0.67      5108
           2       1.00      1.00      1.00      2296
           3       0.58      0.44      0.50      4435
           4       0.88      0.75      0.81      4146
           5       0.93      0.92      0.93      3355
           6       0.06      0.03      0.04      3541

    accuracy                           0.83     56064
   macro avg       0.71      0.70      0.70     56064
weighted avg       0.81      0.83      0.82     56064
```

Figure 8. Construction and assessment of Random Forest with PSO

- <u>Decision Tree with Particle Swarm Optimization</u>

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(504576, 11) (56064, 11) (504576,) (56064,)
```

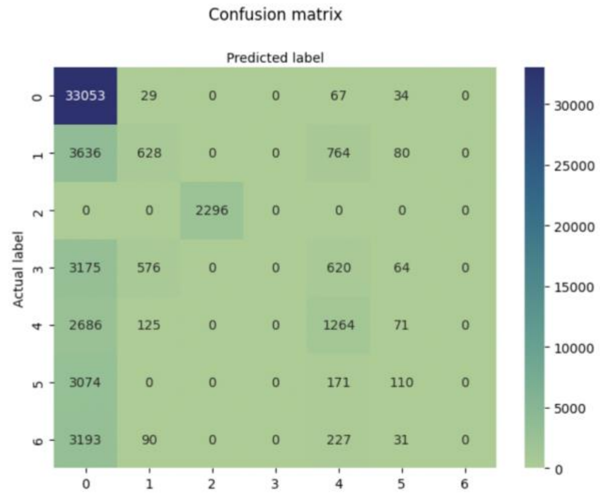
```
#Decision Tree
def fitness_function(params):
    max_depth = int(params[0])
    min_samples_split = int(params[1])

    clf = DecisionTreeClassifier(max_depth=max_depth, min_samples_split=min_samples_split, random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    # Since PSO minimizes, we want to maximize accuracy, so we return its negative
    accuracy = -accuracy_score(y_test, y_pred)
    print(-accuracy)
    return -accuracy  # Negative because PSO minimizes
lb = [5, 2]  # Lower bounds for max_depth and min_samples_split
ub = [10, 10]  # Upper bounds for max_depth and min_samples_split
best_params, _ = pso(fitness_function, lb, ub, swarmsize=10, maxiter=1)

best_max_depth = int(best_params[0])
best_min_samples_split = int(best_params[1])

final_classifier = DecisionTreeClassifier(max_depth=best_max_depth, min_samples_split=best_min_samples_split, random_state=42)
final_classifier.fit(X_train, y_train)
```

```
0.6861800799086758
0.7312357305936074
0.712810359589041
0.712810359589041
0.7312357305936074
0.6943849885844748
0.6943849885844748
0.7312357305936074
0.7312357305936074
0.6943849885844748
0.712810359589041
0.7436501141552512
0.7312357305936074
0.7436501141552512
0.666220747716895
0.6943849885844748
0.6943849885844748
0.712810359589041
0.6861800799086758
0.666220747716895
Stopping search: maximum iterations reached --> 1
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5, min_samples_split=5, random_state=42)
```

```python
# Evaluate the model's performance on a test dataset (not shown in this example)
# Make predictions on the test data with the final model
y_pred = final_classifier.predict(X_test)

# Evaluate the final model's performance
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with Best Hyperparameters: {accuracy}")
```

```
Accuracy with Best Hyperparameters: 0.666220747716895
```

```python
#confusion Matrix
matrix =confusion_matrix(y_test, y_pred)
class_names=[1,2,3,4]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="crest" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
#Classification Report
print(classification_report(y_test, y_pred))
```




```
              precision    recall  f1-score   support

           0       0.68      1.00      0.81     33183
           1       0.43      0.12      0.19      5108
           2       1.00      1.00      1.00      2296
           3       0.00      0.00      0.00      4435
           4       0.41      0.30      0.35      4146
           5       0.28      0.03      0.06      3355
           6       0.00      0.00      0.00      3541

    accuracy                           0.67     56064
   macro avg       0.40      0.35      0.34     56064
weighted avg       0.53      0.67      0.56     56064

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score ar
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score ar
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score ar
  _warn_prf(average, modifier, msg_start, len(result))
```

Fig 9. Construction and assessment of Decision Tree with PSO

- <u>CNN with Particle Swarm Optimization</u>

```
y_train
```

```
0
39011     3
519452    0
101445    0
490860    0
458002    0
         ..
457631    0
220511    5
530557    5
543537    0
512190    1
Name: theft, Length: 504576, dtype: int64
```

```
# CNN with PSO
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(504576, 11) (56064, 11) (504576,) (56064,)
```

```
X_train
```

:Electricity [kW](Hourly)	Cooling:Electricity [kW](Hourly)	Heating:Electricity [kW](Hourly)	InteriorLights:Electricity [kW](Hourly)	InteriorEquipment:Electricity [kW](Hourly)	Gas:Facility [kW](Hourly)	Heating:Gas [kW] (Hourly)	InteriorEquipment:Gas [kW](Hourly)	Water Heater:WaterSystems:Gas [kW](Hourly)	Class
6.890159	0.000000	0.000000	80.527626	73.092045	124.774046	118.679618	3.20568	2.888748	7
87.622200	698.472327	0.000000	220.879908	53.238744	31.264788	0.000000	4.83426	26.430528	9
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	11
25.398563	0.695908	0.000000	6.987380	14.149973	45.195101	41.672601	3.50250	0.020000	14
7.629663	20.296357	0.000000	32.837857	8.089532	0.000000	0.000000	0.00000	0.000000	13
...	...	...	...	...	...	...	...	...	...
37.135666	89.221392	0.000000	32.180462	77.539730	216.379229	5.416123	49.98180	160.981306	2
0.591778	2.380178	0.000000	3.827051	7.574053	0.792556	0.000000	0.00000	0.792556	5
70.235790	0.000000	0.000000	220.879908	137.835547	482.271324	413.630131	4.83426	63.806933	9
4.088366	9.606964	0.000000	9.179851	19.424500	19.011903	0.000000	16.69940	2.312503	0
13.106755	0.000000	0.188151	34.382155	10.771038	11.841134	11.841134	0.00000	0.000000	12

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from pyswarm import pso
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense

def fitness_function(params):
    learning_rate, num_filters, kernel_size, num_neurons = params
    print(learning_rate, num_filters, kernel_size, num_neurons)
    # Reshape the data to be suitable for a CNN
    X_train_reshaped = X_train.values.reshape(-1, X_train.shape[1], 1, 1)
    X_test_reshaped = X_test.values.reshape(-1, X_test.shape[1], 1, 1)

    # Define the model
    model = Sequential()
    model.add(Conv2D(num_filters, (int(kernel_size),1), activation='relu', input_shape=(X_train.shape[1], 1, 1), padding='same'))
    model.add(MaxPooling2D(pool_size=(2, 1)))
    model.add(Flatten())
    model.add(Dense(num_neurons, activation='relu'))
    model.add(Dense(7, activation='softmax'))

    # Compile the model
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Train the model
    model.fit(X_train_reshaped, y_train, epochs=2, verbose=0)

    # Evaluate the model
    _, accuracy = model.evaluate(X_test_reshaped, y_test, verbose=0)

    return -accuracy  # Negative accuracy for maximization

lb = [1, 8,3,8]  # Lower bounds for max_depth and min_samples_split
ub = [2, 216,9,64]  # Upper bounds for max_depth and min_samples_split
best_params, _ = pso(fitness_function, lb, ub, swarmsize=10, maxiter=1)

# Extract the best hyperparameters
best_learning_rate, best_num_filters, best_kernel_size, best_num_neurons = best_params
```

```python
# Build and train the CNN with the best hyperparameters
best_model = Sequential()
best_model.add(Conv2D(int(best_num_filters), (int(best_kernel_size), 1), activation='relu', input_shape=(X_train.shape[1], 1, 1), padding='same'))
best_model.add(MaxPooling2D(pool_size=(2, 1)))
best_model.add(Flatten())
best_model.add(Dense(int(best_num_neurons), activation='relu'))
best_model.add(Dense(7, activation='softmax'))

best_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=best_learning_rate),
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])
```

```
Stopping search: maximum iterations reached --> 1
```

```
# X_train
```

```
X_train_reshaped = X_train.values.reshape(-1, X_train.shape[1], 1, 1)
X_test_reshaped = X_test.values.reshape(-1, X_test.shape[1], 1, 1)
best_model.fit(X_train_reshaped, y_train, epochs=2)

# Evaluate the best model on the test set
test_loss, test_accuracy = best_model.evaluate(X_test_reshaped, y_test)
print("Test Accuracy:", test_accuracy)
```

```
Epoch 1/2
15768/15768 [==============================] - 59s 4ms/step - loss: 14.9641 - accuracy: 0.5914
Epoch 2/2
15768/15768 [==============================] - 56s 4ms/step - loss: 1.4714 - accuracy: 0.5916
1752/1752 [==============================] - 4s 2ms/step - loss: 1.5069 - accuracy: 0.5919
Test Accuracy: 0.5918771624565125
```

```
y_train.unique()
```

```
array([0, 2, 3, 5, 6, 4, 1])
```

```
y_pred = best_model.predict(X_test_reshaped)
y_pred
```

```
1752/1752 [==============================] - 8s 4ms/step
array([[0.47123596, 0.27073756, 0.02106509, ..., 0.07435344, 0.0423843 ,
        0.06758527],
       [0.47123596, 0.27073756, 0.02106509, ..., 0.07435344, 0.0423843 ,
        0.06758527],
       [0.47123596, 0.27073756, 0.02106509, ..., 0.07435344, 0.0423843 ,
        0.06758527],
       ...,
       [0.47123596, 0.27073756, 0.02106509, ..., 0.07435344, 0.0423843 ,
        0.06758527],
       [0.47123596, 0.27073756, 0.02106509, ..., 0.07435344, 0.0423843 ,
        0.06758527],
       [0.47123596, 0.27073756, 0.02106509, ..., 0.07435344, 0.0423843 ,
        0.06758527]], dtype=float32)
```

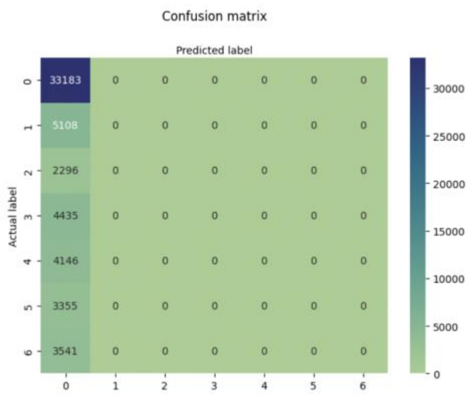
```
y_pred = np.argmax(y_pred,axis=1)
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
y_pred
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
#confusion Matrix
matrix =confusion_matrix(y_test, y_pred)
class_names=[1,2,3,4]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="crest" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()
#Classification Report
print(classification_report(y_test, y_pred))
```




```
              precision    recall  f1-score   support

           0       0.59      1.00      0.74     33183
           1       0.00      0.00      0.00      5108
           2       0.00      0.00      0.00      2296
           3       0.00      0.00      0.00      4435
           4       0.00      0.00      0.00      4146
           5       0.00      0.00      0.00      3355
           6       0.00      0.00      0.00      3541

    accuracy                           0.59     56064
   macro avg       0.08      0.14      0.11     56064
weighted avg       0.35      0.59      0.44     56064

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted
  _warn_prf(average, modifier, msg_start, len(result))
```

Fig 10. Construction and assessment of CNN

- <u>LSTM with Particle Swarm Optimization</u>

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(504576, 11) (56064, 11) (504576,) (56064,)
```

```
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
y_test_new = np.argmax(y_test,axis=1)
X_train1 = np.expand_dims(X_train,axis=2)
X_test1 = np.expand_dims(X_test,axis=2)
X_test1.shape
```

```
(56064, 11, 1)
```

```
import pyswarms
import keras
from keras.models import Sequential
from keras.layers import Activation, LSTM, Dense, Flatten, Dropout
import numpy
from pyswarms.utils.plotters import plot_cost_history, plot_contour, plot_surface
from pyswarms.utils.plotters.formatters import Mesher, Animator
from pyswarms.utils.plotters.formatters import Designer
import matplotlib.pyplot as plt
from IPython.display import Image
```

```python
#import config

def plotCostHistory(optimizer):


    try:

        plot_cost_history(cost_history=optimizer.cost_history)

        plt.show()
    except:
        raise

def plotPositionHistory(optimizer, xLimits, yLimits, filename, xLabel, yLabel):


    try:

        d = Designer(limits=[xLimits, yLimits], label=[xLabel, yLabel])
        animation = plot_contour(pos_history=optimizer.pos_history,
                    designer=d)

        animation.save(filename, writer='ffmpeg', fps=30)
        Image(url=filename)

        plt.show()
    except:
        raise

def plot3D(optimizer, xValues, yValues, zValues):

    try:

        #Obtain a position-fitness matrix using the Mesher.compute_history_3d() method.
        positionHistory_3d = Mesher.compute_history_3d(optimizer.pos_history)

        d = Designer(limits=[xValues, yValues, zValues], label=['x-axis', 'y-axis', 'z-axis'])

        plot3d = plot_surface(pos_history=positionHistory_3d,
                              mesher=Mesher, designer=d,
                              mark=(1,1,zValues[0])) #BEST POSSIBLE POSITION MARK (* --> IN GRAPHIC)

        return plot3d

    except:
        raise
```

```python
def plotTrainValAcc(history):

    try:

        #Train and validation accuracy
        plt.plot(history.history['accuracy'], 'b', label='Training accurarcy')
        plt.plot(history.history['val_accuracy'], 'r', label='Validation accurarcy')
        plt.title('Training and Validation accurarcy')
        plt.legend()
        plt.figure()

    except:
        raise

def plotTrainValLoss(history):

    try:

        #Train and validation loss
        plt.plot(history.history['loss'], 'b', label='Training loss')
        plt.plot(history.history['val_loss'], 'r', label='Validation loss')
        plt.title('Training and Validation loss')
        plt.legend()
        plt.show()


    except:
        raise

def confclassif(y_test_new, y_pred):
    try:
      #confusion Matrix
      matrix =confusion_matrix(y_test_new, y_pred)
      fig, ax = plt.subplots()
      sns.heatmap(pd.DataFrame(matrix), annot=True, cmap="crest" ,fmt='g')
      ax.xaxis.set_label_position("top")
      plt.tight_layout()
      plt.title('Confusion matrix', y=1.1)
      plt.ylabel('Actual label')
      plt.xlabel('Predicted label')
      plt.show()

      #Classification Report
      print("Classification Report : ")
      print(classification_report(y_test_new, y_pred))

    except:
        raise
```

```python
def lstm(x_train, x_test, y_train, y_test, neurons, batch_size, epochs):


    try:
        model=Sequential()
        model.add(LSTM(256, return_sequences=False, input_shape=(int(11),1)))
        model.add(Dense(units=7))
        model.add(Activation('sigmoid'))
        model.compile(loss='categorical_crossentropy',optimizer=Adam(),metrics=['accuracy'])  # CROSSENTROPY BECAUSE IT'S MORE ADEQUATED TO MULTI-CLASS PROBLEMS
        model.summary()
        #FIT MODEL
        historyOfTraining = model.fit(
            x=x_train,
            y=y_train,
            batch_size=batch_size,
            validation_data=(x_test, y_test),
            epochs=5,
            shuffle=False #IF I USE STATEFUL MODE, THIS PARAMETER NEEDS TO BE EQUALS TO FALSE
        )

        plotTrainValAcc(historyOfTraining)
        plotTrainValLoss(historyOfTraining)

        y_pred = model.predict(x_test)
        y_pred = np.argmax(y_pred,axis=1)
        y_test_new = np.argmax(y_test,axis=1)
        print ("LSTM PSO:Accuracy : ", accuracy_score(y_test_new,y_pred)*100)
        confclassif(y_test_new, y_pred)

        predict = model.predict(x=x_test, batch_size=batch_size)
        print(predict)
        print(y_test)

        predict = (predict == predict.max(axis=1)[:, None]).astype(int)
        print(predict)

        numberRights = 0
        for i in range(len(y_test)):
            indexMaxValue = numpy.argmax(predict[i], axis=0)
            if indexMaxValue == numpy.argmax(y_test[i],
                                             axis=0):  # COMPARE INDEX OF MAJOR CLASS PREDICTED AND REAL CLASS
                numberRights = numberRights + 1

        hitRate = numberRights / len(y_test)  # HIT PERCENTAGE OF CORRECT PREVISIONS

        return hitRate

    except:
        raise
```

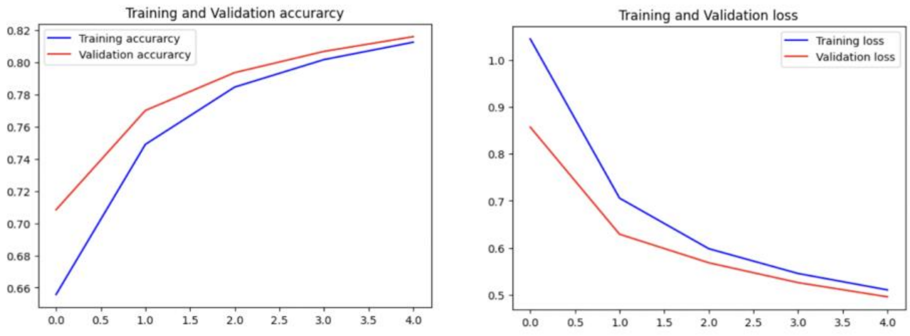
```python
TYPE = 'type'
OPTIONS = 'options'
GLOBAL_BEST = 'G'
LOCAL_BEST = 'L'
OPTIONS = 'options'
GLOBAL_BEST = 'G'
LOCAL_BEST = 'L'
C1 = 'c1'
C2 = 'c2'
INERTIA = 'w'
NUMBER_NEIGHBORS = 'k'
MINKOWSKI_RULE = 'p'
TYPE = 'type'
OPTIONS = 'options'
X_LABEL_FILTERS = 'n_filtros'
X_LABEL_NEURONS = 'n_neuronios'
Y_LABEL_EPOCHS = 'n_epochs'

SIGMOID = 'sigmoid'
TANH = 'tanh'
RELU = 'relu'
```

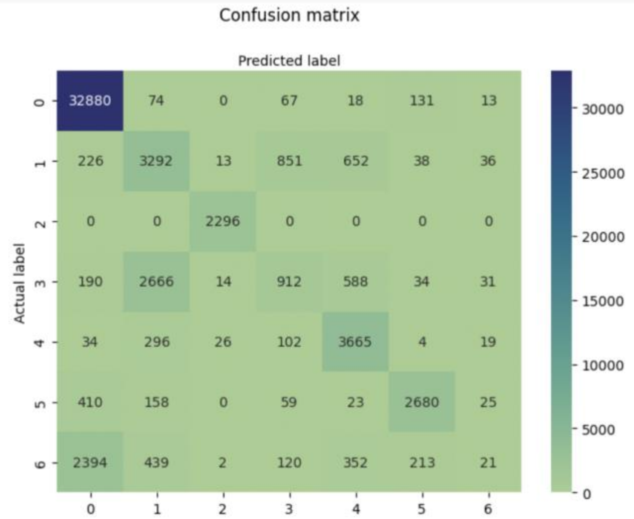
```python
def lostFunction(particleDimension, x_train, x_test, y_train, y_test, batch_size):


    try:

        #RETRIEVE DIMENSIONS VALUES, AND I NEED TO CONVERT FLOAT VALUES (CONTINUOUS) TO INT
        neurons = int(particleDimension[0])
        epochs = int(particleDimension[1])

        #CALL LSTM_MODEL function
        accuracy = lstm(x_train=x_train, x_test=x_test, y_train=y_train, y_test=y_test,
                        neurons=neurons, batch_size=batch_size, epochs=epochs)

        #APPLY COST FUNCTION --> THIS FUNCTION IS EQUALS TO CNN COST FUNCTION
        loss = 1.5 * ((1.0 - (1.0/neurons)) + (1.0 - (1.0/epochs))) + 2.0 * (1.0 - accuracy)
        print(accuracy)
        return loss

    except:
        raise

def particlesLoop(particles, x_train, x_test, y_train, y_test, batch_size):
```

```python
    try:

        numberParticles = particles.shape[0] #NUMBER OF PARTICLES

        allLosses = [lostFunction(particleDimension=particles[i], x_train=x_train, x_test=x_test,
                     y_train=y_train, y_test=y_test, batch_size=batch_size) for i in range(1)]

        return allLosses #NEED TO RETURN THIS PYSWARMS NEED THIS

    except:
        raise

def applyLSTM_PSO(x_train, x_test, y_train, y_test, batch_size, numberParticles, iterations, dimensions, bounds, **kwargs):


    try:

        #GET KWARG type ARGUMENT
        topology = kwargs.get(TYPE)

        #INITIALIZATION OF PSO --> CONSIDERING TWO POSSIBLE TOPOLOGIES gbest AND lbest
        optimizer = None
        if topology == GLOBAL_BEST:
            optimizer = pyswarms.single.GlobalBestPSO(n_particles=numberParticles, dimensions=dimensions,
                                                      options=kwargs.get(OPTIONS), bounds=bounds)
        elif topology == LOCAL_BEST:
            optimizer = pyswarms.single.LocalBestPSO(n_particles=numberParticles, dimensions=dimensions,
                                                      options=kwargs.get(OPTIONS), bounds=bounds)
        else:
            raise AttributeError

        #PSO OPTIMIZATION PASSING LOOP PARTICLES ITERATION FUNCTION particlesLoop, applying lstm for all particle in all iterations
        cost, pos = optimizer.optimize(particlesLoop, x_train=x_train, x_test=x_test, y_train=y_train,
                                       y_test=y_test, batch_size=batch_size,iters=iterations)

        return cost, pos, optimizer

    except:
        raise
```

```python
#LSTM WITH PSO

# #DEFINITION OF LSTM PARAMETERS, EPOCHS AND NEURONS ARE DEFINED BY PSO
batch_size = 32
#
# #DEFINITION OF PSO PARAMETERS
numberParticles = 10
iterations = 1
dimensions = 2 # [0] --> NEURONS , [1] --> EPOCHS
#
#DEFINITION OF DIMENSIONS BOUNDS, X AXIS --> NEURONS and Y AXIS --> EPOCHS
minBounds = numpy.ones(2)
maxBounds = numpy.ones(2)
maxBounds[0] = 251 #I REDUCE THIS DIMENSIONS, IN ORDER TO MAKE OPTIMIZATION MORE QUICKLY
maxBounds[1] = 201
bounds = (minBounds, maxBounds)
#
#DEFINITION OF DIFFERENT TOPOLOGIES OPTIONS
lbest_options = {C1 : 0.3, C2 : 0.2, INERTIA : 0.9, NUMBER_NEIGHBORS : 4, MINKOWSKI_RULE : 2}
lbest_kwargs = {TYPE : LOCAL_BEST, OPTIONS : lbest_options}
gbest_options = {C1 : 0.3, C2 : 0.2, INERTIA : 0.9}
gbest_kwargs = {TYPE : GLOBAL_BEST, OPTIONS : gbest_options}
#
#PASSING ALL THIS OPTIONS TO LSTM_PSO applyLSTM_PSO FUNCTION
cost, pos, optimizer = applyLSTM_PSO(x_train=X_train1, x_test=X_test1, y_train=y_train, y_test=y_test, batch_size=batch_size,
                                     numberParticles=numberParticles, iterations=iterations, dimensions=dimensions,
                                     bounds=bounds, **lbest_kwargs)
print(cost)
print(pos)
#
#PLOT GRAPHICS ILLUSTRATING THE COST VARIATION AND PARTICLES MOVEMENT AND CONVERGENCE

#plotCostHistory(optimizer=optimizer)
plotPositionHistory(optimizer=optimizer, xLimits=(minBounds[0], maxBounds[0]),
                    yLimits=(minBounds[1], maxBounds[1]), filename='lstmParticlesPosConvergence.mp4',
                    xLabel=X_LABEL_NEURONS, yLabel=Y_LABEL_EPOCHS)
```

```
2023-11-01 16:30:44,389 - pyswarms.single.local_best - INFO - Optimize for 1 iters with {'c1': 0.3, 'c2': 0.2, 'w': 0.9, 'k': 4, 'p': 2}
pyswarms.single.local_best:   0%|          |0/1Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 256)               264192

 dense (Dense)               (None, 7)                 1799

 activation (Activation)     (None, 7)                 0

=================================================================
Total params: 265991 (1.01 MB)
Trainable params: 265991 (1.01 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/5
15768/15768 [==============================] - 86s 5ms/step - loss: 1.0443 - accuracy: 0.6557 - val_loss: 0.8568 - val_accuracy: 0.7084
Epoch 2/5
15768/15768 [==============================] - 91s 6ms/step - loss: 0.7058 - accuracy: 0.7490 - val_loss: 0.6291 - val_accuracy: 0.7701
Epoch 3/5
15768/15768 [==============================] - 87s 6ms/step - loss: 0.5983 - accuracy: 0.7846 - val_loss: 0.5684 - val_accuracy: 0.7936
Epoch 4/5
15768/15768 [==============================] - 102s 6ms/step - loss: 0.5456 - accuracy: 0.8017 - val_loss: 0.5262 - val_accuracy: 0.8068
Epoch 5/5
15768/15768 [==============================] - 105s 7ms/step - loss: 0.5107 - accuracy: 0.8125 - val_loss: 0.4960 - val_accuracy: 0.8160
```






```
Classification Report :
              precision    recall  f1-score   support

           0       0.91      0.99      0.95     33183
           1       0.48      0.64      0.55      5108
           2       0.98      1.00      0.99      2296
           3       0.43      0.21      0.28      4435
           4       0.69      0.88      0.78      4146
           5       0.86      0.80      0.83      3355
           6       0.14      0.01      0.01      3541

    accuracy                           0.82     56064
   macro avg       0.64      0.65      0.63     56064
weighted avg       0.77      0.82      0.78     56064

1752/1752 [==============================] - 4s 3ms/step
[[9.9768102e-01 7.7652000e-02 1.7950815e-06 ... 1.8762577e-04
```

Fig 11. Construction and assessment of LSTM

Figure 11 demonstrates that the LSTM with PSO model has an accuracy score of 82%.

After implementation of all the models, it is evident now that XGBoost with PSO is the best performing model.

6 Conclusion

The Configuration Manual contains a thorough and detailed overview, presented in a sequential manner, of the whole procedure involved in implementing the research project. The necessary prerequisites, data preparation, exploratory data analysis, model deployment and assessment, are all demonstrated here through the use of snapshots. A comprehensive and sequential elucidation of each segment in the paper has been included to assist the reader in recreating the procedure.