**pseudo codes**

1. **Features selection pseudo code**

**Start**

Function feature\_extraction\_selection\_boruta(dataframe):

# Separate features and target variable

X = dataframe excluding 'Label' column

y = 'Label' column from dataframe

# Initialize LightGBM classifier with specific parameters

Initialize LightGBM classifier with:

boosting\_type = 'gbdt'

objective = 'binary'

metric = 'binary\_logloss'

class\_weight = 'balanced'

learning\_rate = 0.5

max\_depth = 10

num\_leaves = 50

min\_child\_samples = 20

subsample = 0.8

colsample\_bytree = 0.8

reg\_alpha = 0.5

reg\_lambda = 0.5

n\_estimators = 100

# Initialize Boruta feature selector with specific parameters

Initialize Boruta feature selector with:

estimator = LightGBM classifier

n\_estimators = 'auto'

max\_iter = 50

perc = 100

alpha = 0.5

two\_step = True

random\_state = 42

verbose = 2

# Fit Boruta feature selector to data

Fit Boruta selector to X and y

# Get selected features from Boruta

selected\_features = Features in X where Boruta selector supports them

# Create a new dataframe with only selected features

dataframe\_selected = dataframe with only selected features

Return dataframe\_selected

# Apply feature extraction and selection

df\_selected\_features\_boruta = feature\_extraction\_selection\_boruta(preprocessed dataframe)

# Print selected features

Print "Selected Features using BorutaPy with LightGBM:", columns of df\_selected\_features\_boruta

**End**

1. **Pseudo Code of proposed model**

**Start**

# Define an attention layer

Class Attention:

Method \_\_init\_\_():

Initialize Attention layer

Method build(input\_shape):

Initialize weights W, b, u

Method call(inputs):

Compute score using W and b

Compute attention\_weights using score and u

Compute context\_vector as weighted sum of inputs

Return context\_vector

# Define the enhanced TCN model

Function build\_enhanced\_tcn\_model(input\_shape, num\_classes):

Initialize inputs with shape input\_shape

Apply Conv1D layers with ReLU activation and MaxPooling1D

Apply Attention layer to the output of last Conv1D layer

Flatten the output of Attention layer

Apply Dropout and Dense layers

Initialize outputs with softmax activation

Create and return Model with inputs and outputs

# Initialize enhanced TCN model

Initialize tcn\_model using build\_enhanced\_tcn\_model with input\_shape (9, 1) and num\_classes 2

Compile tcn\_model with Adam optimizer and sparse\_categorical\_crossentropy loss

# Reshape training and test data for TCN input

Reshape X\_train and X\_test to shape (-1, 9, 1)

# Define early stopping callback

Initialize early\_stopping with patience of 3

# Train enhanced TCN model

Train tcn\_model with X\_train\_tcn, y\_train, using validation data (X\_test\_tcn, y\_test) and early\_stopping callback

# Evaluate enhanced TCN model

Evaluate tcn\_model with X\_test\_tcn and y\_test

Print validation loss and accuracy

# Use TCN model predictions as features

Predict X\_train\_tcn and X\_test\_tcn using tcn\_model

# Initialize Logistic Regression and other classifiers

Initialize Logistic Regression and Stacking Classifier

# Define hyperparameters for tuning

Define param\_grid with hyperparameters for Logistic Regression

# Initialize GridSearchCV for hyperparameter tuning

Initialize GridSearchCV with stacking model and param\_grid

# Fit GridSearchCV

Fit grid\_search with X\_train\_tcn\_pred and y\_train

# Get the best estimator from grid search

Get best stacking model from grid\_search

# Make predictions on test set using best model

Predict y\_test using best\_stacking\_model

# Evaluate accuracy

Print best stacking model parameters

Print accuracy of the enhanced hybrid model

**End**