National College of Ireland

Configuration Manual for Evaluating the Impact of Feature Engineering Techniques on Supervised Machine Learning Algorithms for IoT Intrusion Detection Systems: A case Study on Mirai Botnet Attacks

MSc Academic Research Project

Cybersecurity

Dina Allam

Student ID: x23291583

School of Computing

National College of Ireland

Supervisor: Mustafa UI Raza

MSc Project Submission Sheet

School of Computing

Student Name:	……Dina Allam……
Student ID:	……X23291583……
Programme:	…… Cybersecurity **Year:** …2025….
Module:	…… Msc Research Project Configuration Manual
Lecturer:	……Mustafa UI Raza ……
Submission Due Date:	……11 / 12/ 2025 ……
Project Title:	Evaluating the Impact of Feature Engineering Techniques on Supervised Machine Learning Algorithms for IoT Intrusion Detection Systems: A case Study on Mirai Botnet Attacks
Word Count:	…… **Page Count:** ……..……..

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:	……Dina Allam……
Date:	……11/12/2025……

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 for Evaluating the Impact of Feature Engineering Techniques on Supervised Machine Learning Algorithms for IoT Intrusion

Detection Systems: A case Study on Mirai Botnet Attacks

Dina Allam

Student ID: x23291583

Introduction

Network anomaly detection has become increasingly critical in modern cybersecurity landscapes. As network infrastructures grow more complex and cyber threats become more sophisticated, the ability to automatically identify unusual patterns that may indicate security breaches, system failures, or malicious activities is paramount. Traditional rule-based approaches often fall short when dealing with novel attack patterns or subtle anomalies that deviate from normal behaviour in unexpected ways.

Machine learning offers a promising alternative by learning patterns from historical data and generalizing to detect previously unseen anomalies. However, the effectiveness of machine learning models depends heavily on feature representation and model selection. This study explores these dimensions systematically, comparing traditional classification approaches with advanced feature engineering techniques to determine optimal strategies for anomaly detection.

Project Overview

This report presents a systematic investigation into anomaly detection using machine learning techniques on network traffic data. The primary objective was to identify the most effective combination of feature engineering approaches and classification algorithms for detecting anomalous behavior in network systems. Through rigorous experimentation with multiple feature extraction methods and five different classification models

Hardware/Software Implementation

To implement this project the following configuration used

Hardware tools:

- Operating System: Windows 11 (64-bit)
- Processor: Intel® Core™ Ultra 9 (Intel Eco Edition)
- Storage (Hard drive): 1 TB SSD
- RAM: 32 GB

Software tools:

- Google Colab
- Python 3
- Hardware accelerator: CPU

Data Collection

The data used to evaluate the proposed model is collected from secondary resource "Canadian Institute for Cybersecurity" website as an opensource Dataset named "CIC IoT-DIAD 2024" . The dataset includes two subsets each for different purpose, one includes preextracted features using CICFlowMeter named as "AD_Flow-based-features"and the other includes features extracted from Packet-per-Packet analysis named as "DI_AD_Packet-based-features".

AD_Flow-based-features subset is utilised in this study for couple of reasons including the bidirectional communication in each record that is helpful in identifying any abnormal activity and its compatibility with the supervised learning models

Used Libraries

As shown in the following Figure 1 number of libraries imported to carry on the study including:

Numpy: to handle mathematical calculations that were used in data preprocessing, Autoencoders and PCA.

Pandas: to handle tabular data .csv dataset files, and in preprocessing dataset.

Seaborn and matplotlib: mainly used for its visualisation in confusion matrix and PCA variance graph

Moreover, other libraries were imported to carry on the machine learning algorithm and specific preprocessing techniques such as class weighting, encoding labels and other techniques.

Importing the Packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_predict
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.metrics import classification_report, accuracy_score, f1_score, confusion_matrix, recall_score, precision_score
from sklearn.utils.class_weight import compute_class_weight
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import requests
from bs4 import BeautifulSoup
from io import StringIO
from tqdm import tqdm
import os
import shutil
import warnings
warnings.filterwarnings('ignore')
```

Figure 1

Data Preprocessing

A rigorous approach used on the dataset to ensure data quality. In this study sequential checks and techniques were conducted to prepare the data for the feature extraction stage and model evaluation.

These techniques were implemented to handle inconsistencies, remove noise, resolve null and infinite values, remove nonnumeric features and deal with imbalanced classes.

After loading the data, adding label to the Merai with “Merai” and to the benign with “Benign”, merging them into one data frame. A series of checks done and the following steps are conducted:

- Remove any duplicated columns as shown in Figure 2.
- Remove nonnumeric columns that are not contributing to classification process (Flow ID, Src IP, Dst IP, Timestamp) ending with 79 features.
- Any Null or infinite values replaced with median value to ensure robustness for skewed data and don’t influence by outliners while reducing the variance.

```
Data shape before dropping duplicates : (572918, 85)
Data shape after dropping duplicates : (572701, 85)
```

Figure 2

After cleaning data, the summary of the data-frame as shown in Figure 3.

```
ORIGINAL FEATURES (BEFORE CLEANING):
  Total columns: 83

  Non-numeric columns (will be removed): 4
    - Flow ID
    - Src IP
    - Dst IP
    - Timestamp

  Numeric features retained: 79

  Sample feature names (first 10):
     1. Src Port
     2. Dst Port
     3. Protocol
     4. Flow Duration
     5. Total Fwd Packet
     6. Total Bwd packets
     7. Total Length of Fwd Packet
     8. Total Length of Bwd Packet
     9. Fwd Packet Length Max
    10. Fwd Packet Length Min
    ... and 69 more features

  Missing values found: 70
  ✓ Missing values filled with median
  Infinite values found: 334
  ✓ Infinite values replaced with median

✓ Final feature shape after cleaning: (572701, 79)
  Features: 79, Samples: 572701
```

Figure 3

Moreover, A critical challenge encountered during class distribution check, was class imbalance in the dataset, as shown in the original class distribution visualization as shown in Figure 4:

```
ORIGINAL CLASS DISTRIBUTION (BEFORE IMBALANCE HANDLING):
======================================================================
  benign                         : 398314 samples (69.55%)
  mirai                          : 174387 samples (30.45%)

  Total samples: 572701
  Number of classes: 2
  Imbalance ratio (max/min): 2.28:1
======================================================================
```

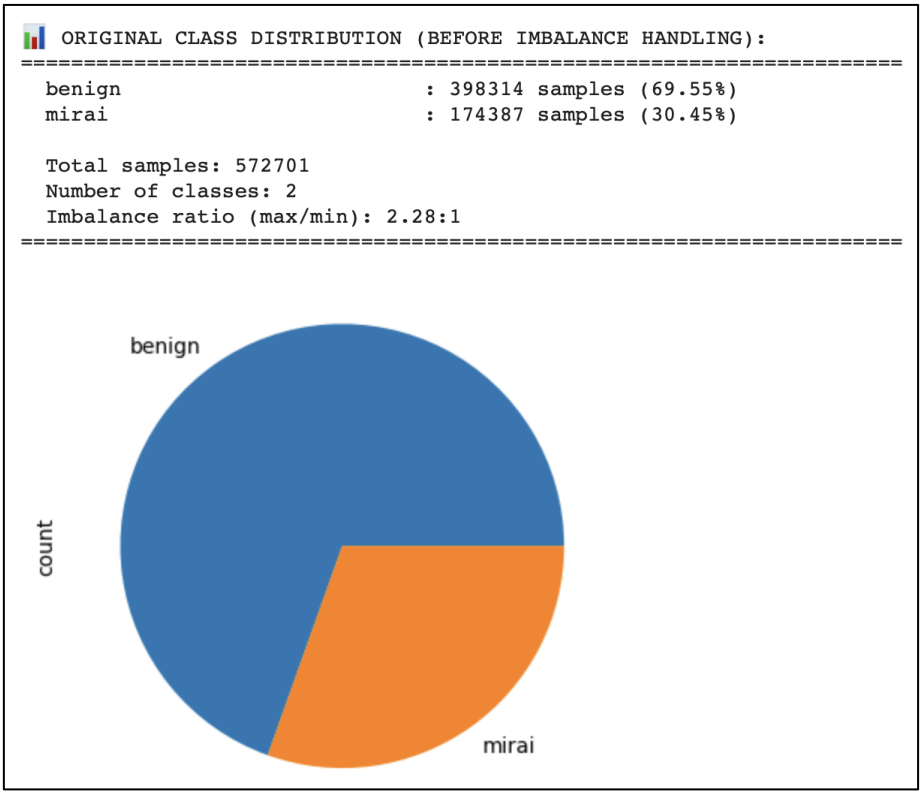


Figure 4

To address this imbalance, we implemented class weights in our models, which penalize misclassifications of the minority class more heavily as shown in Figure 5. This approach ensures that models don't simply learn to predict the majority class to achieve high accuracy but instead develop genuine discriminative capabilities.

```
⚖️ HANDLING CLASS IMBALANCE WITH CLASS WEIGHTS:
======================================================================

📊 Train Set Distribution (Natural - BEFORE weighting):
  benign                         : 278819 samples (69.55%)
  mirai                          : 122071 samples (30.45%)

⚖️ Computed Class Weights (inverse frequency):
  Formula: n_samples / (n_classes * class_count)

  benign                         : weight =   0.7189 (count: 278819)
  mirai                          : weight =   1.6420 (count: 122071)

💡 Impact of Class Weights:
  - Minority classes get HIGHER weights (more penalty for misclassification)
  - Majority classes get LOWER weights (less penalty)
  - This balances the learning process without changing the data
  - Test set remains with NATURAL distribution for realistic evaluation

📈 Effective Sample Distribution (AFTER applying weights):
  benign                         : 200445.0 effective samples
  mirai                          : 200445.0 effective samples

✓ Class weights computed for 2 classes
======================================================================
```

Figure 5

Feature Engineering Strategies

In this project two feature extraction techniques used on different tuning to evaluate its performance against the original dataset features.

Original Features:

pre-processed original features without any dimensionality reduction or transformation.

Autoencoder-Based Features:

Autoencoders was the first stage trained with varying bottleneck dimensions (16, 32, and 64 dimensions). Autoencoders learn compressed representations of data by training neural networks to reconstruct their inputs through a narrow bottleneck layer.

During the initial tuning using ‘sigmoid’ in the decoding output layer, found that the (mean = 6.13, std = 4.61, max = 525.10) as shown in Figure 6. this problem encountered because of

the StandardScaler Standardization technique used that gives positive and negative values while sigmoid restrict output from 0 to 1 range.

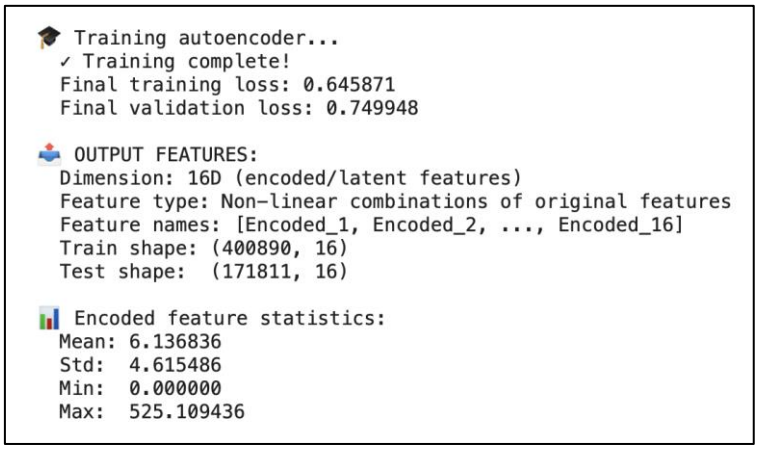


Figure 6

To address this problem the decoder activation replaced with 'linear' allowing Autoencoders to reconstruct data in positive and negative ranges. After replacing the activation mode, the training loss, validation loss, Mean and Std dropped and the representation quality enhanced as shown in Figure 7.

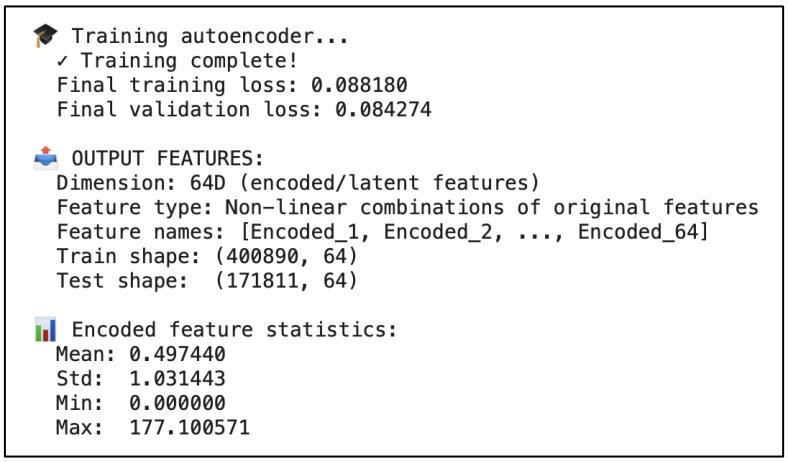


Figure 7

The features set of each dimension stored to be assessed on the models.

Also, it is worth to mention that the other Autoencoder parameters tuned as following:

Number of layers in encoder and decoder are six symmetric layers, three for the encoder and 3 for the decoder as shown in figure 8.

```python
def build_autoencoder(input_dim, encoding_dim):
    """Build autoencoder with specified encoding dimension"""
    # Encoder
    encoder_input = keras.Input(shape=(input_dim,))
    encoded = layers.Dense(128, activation='relu')(encoder_input)
    encoded = layers.Dense(64, activation='relu')(encoded)
    encoded = layers.Dense(encoding_dim, activation='relu')(encoded)

    # Decoder
    decoded = layers.Dense(64, activation='relu')(encoded)
    decoded = layers.Dense(128, activation='relu')(decoded)
    decoded = layers.Dense(input_dim, activation='linear')(decoded)

    # Full autoencoder
    autoencoder = keras.Model(encoder_input, decoded)
    encoder = keras.Model(encoder_input, encoded)

    return autoencoder, encoder

# Train autoencoders with different dimensions
encoding_dims = [16, 32, 64]
encoders = {}
encoded_features = {}
```

Figure 8

Batch size set to 256 to maintain acceptable training time, this done for 20 epochs and validation split on 10% of the data as shown in Figure 9.

```python
# Train
print(f"\n🎓 Training autoencoder...")
history = autoencoder.fit(
    X_train_scaled, X_train_scaled,
    epochs=20,
    batch_size=256,
    validation_split=0.1,
    verbose=0
)
```

Figure 9

PCA-Enhanced Autoencoder Features:

Building on the Autoencoder features for dimensionally reduction as shown on Figure 10

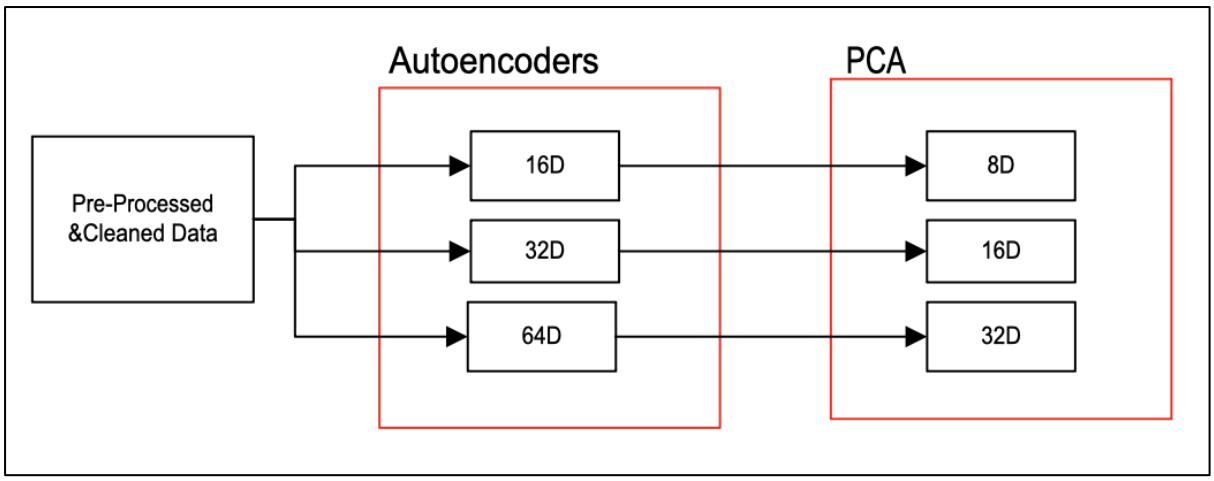


Figure 10

After applying PCA found that the results of the variance of each dimension reduction as shown in the following Table1

Process	*No. of Features Before PCA*	*No. of Features After PCA*	*Variance*
AE16→ PCA8	16	8	91.65%
AE32→ PCA8	32	8	83.93%
AE32→ PCA16	32	16	94.90%
AE64 → PCA8	64	8	79.53%
AE64 → PCA16	64	16	91.28%
AE64 → PCA32	64	32	98.75%

Table 1

The table shows that the various decreases when applying aggressive reduction (more than 50% features reduction). And among all tuning's dimensional reduction from AE64 features to 32 features retained over 98% of information and gives minimal information loss.

Machine learning models

Each of the following models were trained on every feature set configuration, resulting in a comprehensive comparison matrix that reveals how different algorithms respond to various feature representations.

The feature set were:

- Original features after preprocessing
- Extracted features after Autoencoder:
  - 64 features
  - 32 features
  - 16 features
- Extracted features after Autoencoder+ PCA features:

- AE 64 -> PCA 32
- AE 64 -> PCA 16
- AE 64 -> PCA 8
- AE 32 -> PCA 16
- AE 32 -> PCA 8
- AE 16 -> PCA 8

Logistic regression

Logistic regression tuned as following figure 11,12.

```
if model_name == 'LogisticRegression':
    model = LogisticRegression(max_iter=2000, class_weight='balanced', random_state=42)
```

Figure 11

```
results['LogisticRegression'] = train_and_evaluate_model_with_metrics(
    'LogisticRegression', feature_sets, y_train, y_test, class_weight_dict, le
)
```

Figure 12

And shows confusion matrix heatmap results as following Figures 13-22:

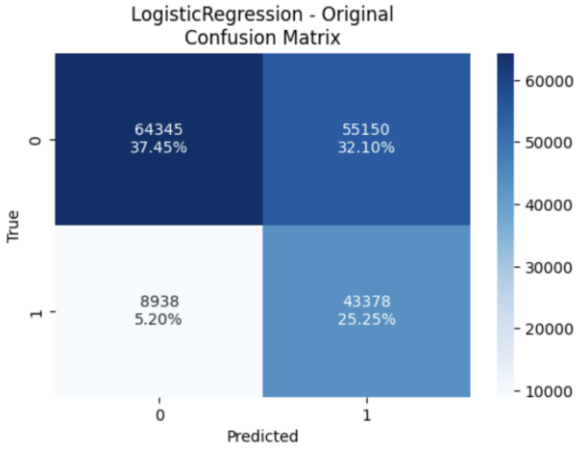


Figure 13- Original features

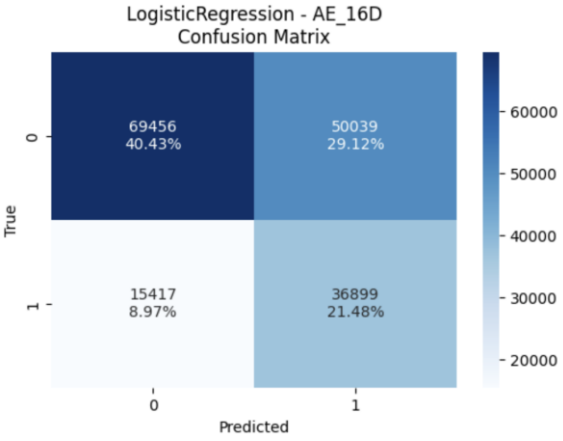


Figure 14- 16 features extracted from Autoencoders

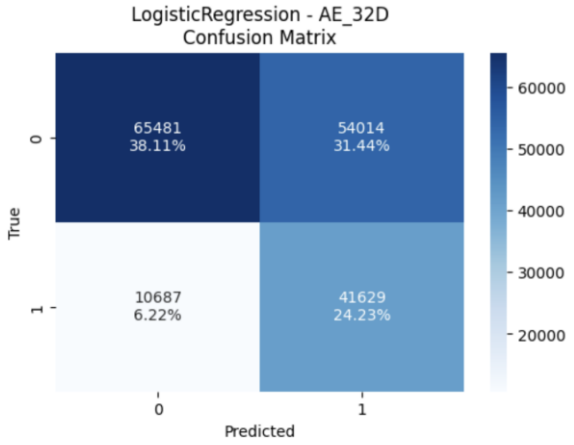


Figure 15- 32 features extracted from Autoencoders

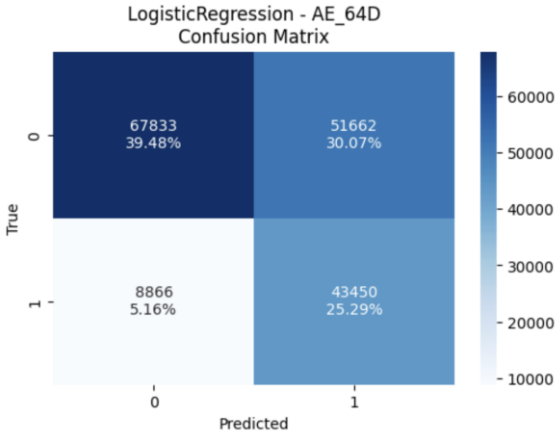


Figure 16- 64 features extracted from Autoencoders

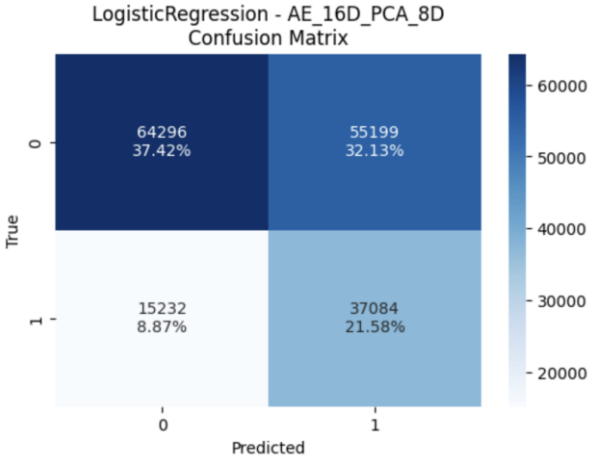


Figure 17- 8 features extracted from AE 16 to PCA 8

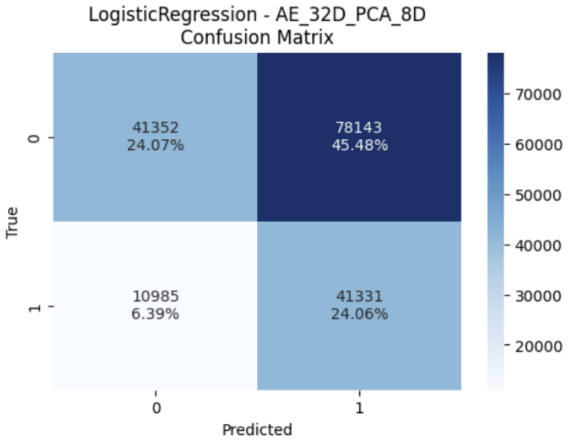


Figure 18- 8 features extracted from AE 32 to PCA 8

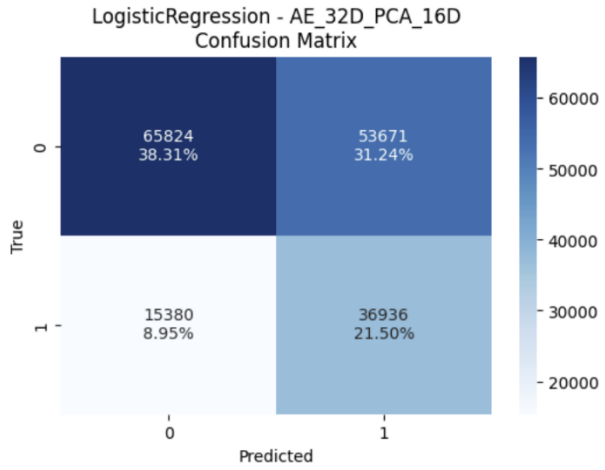


Figure 19- 16 features extracted from AE 32 to PCA 16

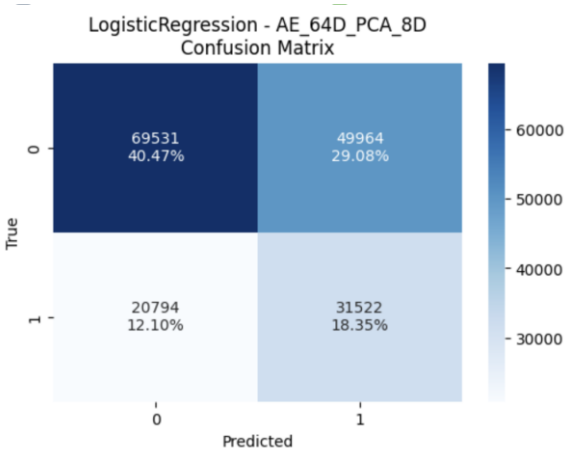
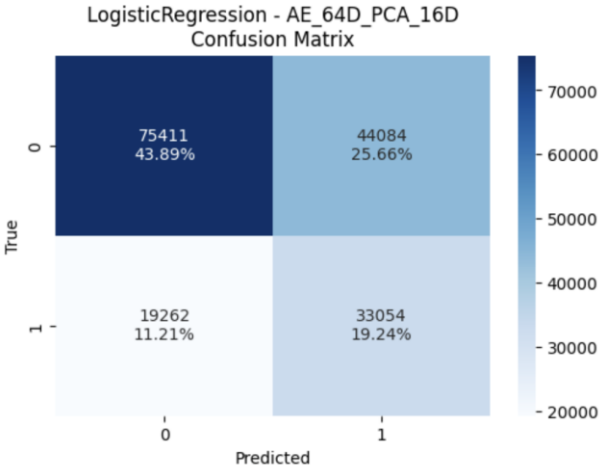


Figure 20 -8 features extracted from AE 64 to PCA 8



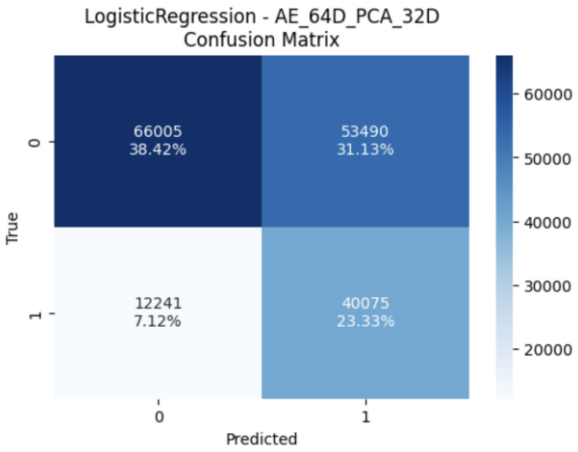


Figure 21- 16 features extracted from AE 64 to PCA 16 Figure 22 -32 features extracted from AE 64 to PCA 32

And logistic regression Accuracy and F1 scores are shown in the following Table 2.

logistic regression			
	feature set	Accuracy	F1
	original	62.70%	63.94%
Autoencoders	AE_16D	61.90%	63.41%
	AE_32D	62.34%	63.69%
	AE_64D	64.77%	66.04%
Autoencoders + PCA	AE_16D_PCA_8D	59.01%	60.56%
	AE_32D_PCA_8D	56.30%	57.91%
	AE_32D_PCA_16D	60.44%	61.99%
	AE_64D_PCA_8D	58.82%	60.44%
	AE_64D_PCA_16D	63.13%	64.53%
	AE_64D_PCA_32D	61.74%	63.16%

Table 2

Random Forest

Logistic regression tuned as following Figure 23,24.

```
elif model_name == 'RandomForest':
    model = RandomForestClassifier(n_estimators=100, class_weight='balanced',
                                   random_state=42, n_jobs=-1)
```

Figure 23

```
results['RandomForest'] = train_and_evaluate_model_with_metrics(
    'RandomForest', feature_sets, y_train, y_test, class_weight_dict, le
)
```

Figure 24

And shows confusion matrix heatmap results as following Figures 25-34 :

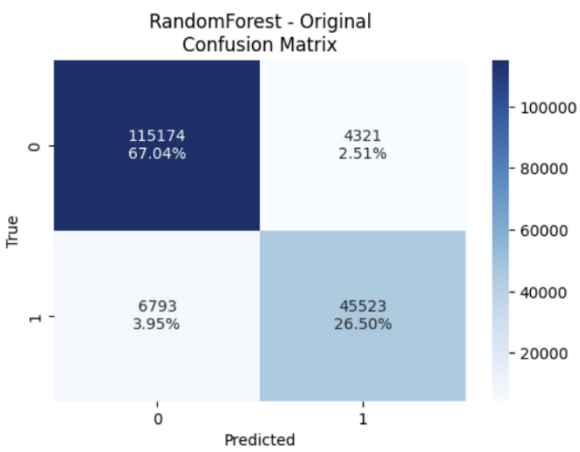


Figure 25 - Original features

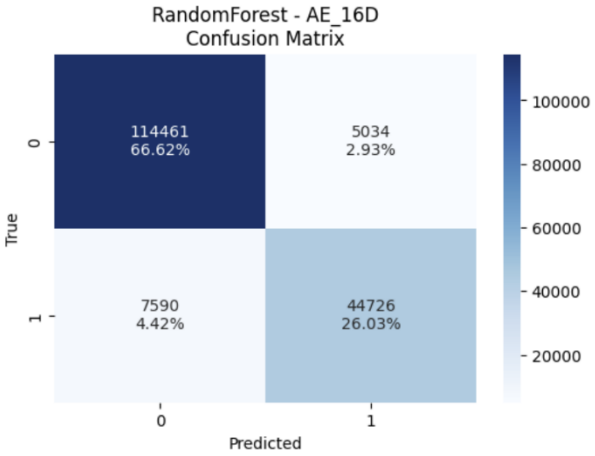


Figure 26-16 features extracted from Autoencoders

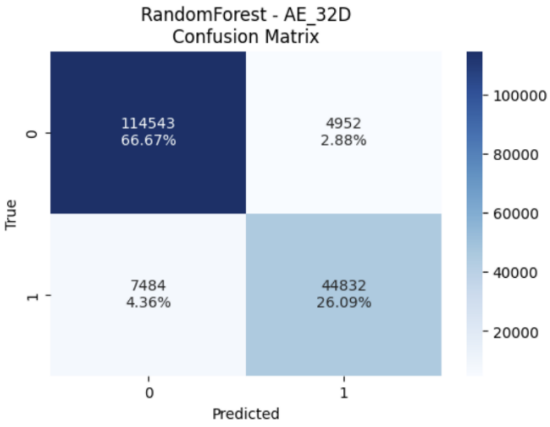


Figure 27- 32 features extracted from Autoencoder

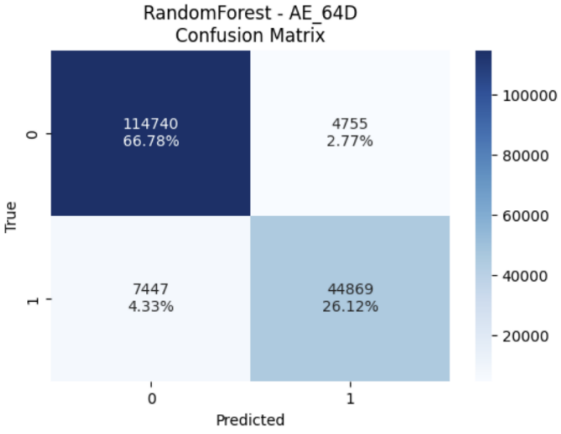


Figure 28- 64 features extracted from Autoencoders

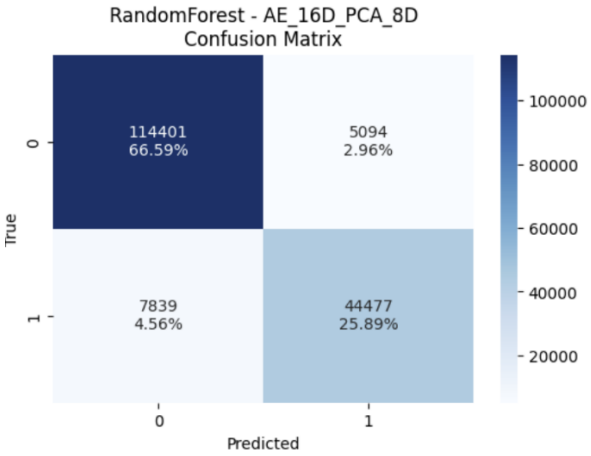


Figure 29- 8 features AE_16D_PCA_8D

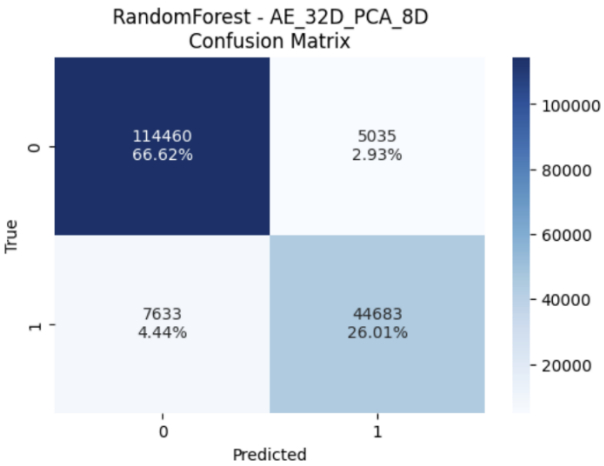


Figure 30- 8 features AE_32D_PCA_8D

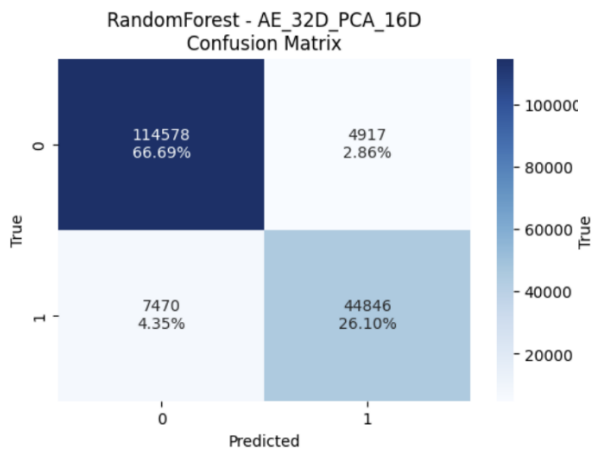


Figure 31-16 features AE_32D_PCA_16D

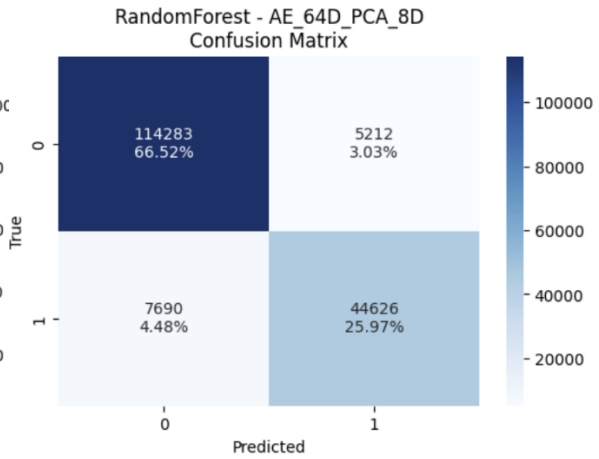


Figure 32-8 features AE_64D_PCA_8D

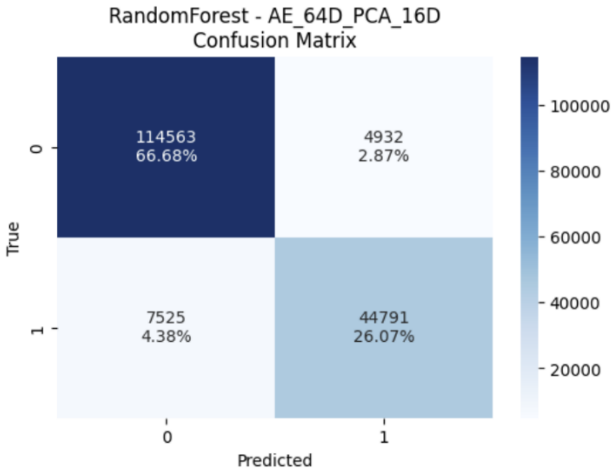


Figure 33- 16 features AE_64D_PCA_16D

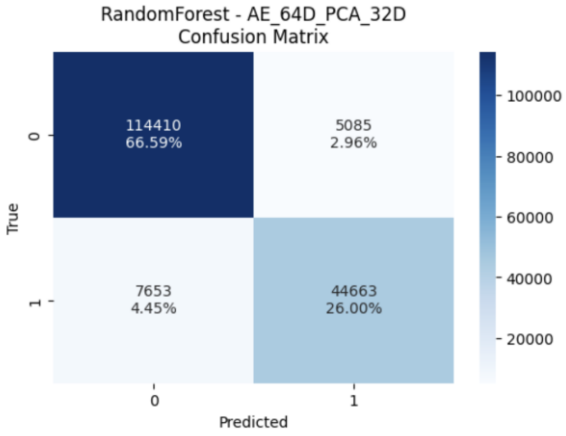


Figure 34-32 features AE_64_PCA_32D

And Random Forest Accuracy and F1 scores are shown in the following Table 3

Random Forest

	feature set	Accuracy	F1
	original	93.53%	93.49%
Autoencoders	AE_16D	92.59%	92.54%
	AE_32D	92.81%	92.76%
	AE_64D	92.76%	92.71%
Autoencoders + PCA	AE_16D_PCA_8D	92.18%	92.13%
	AE_32D_PCA_8D	92.27%	92.22%
	AE_32D_PCA_16D	92.60%	92.55%
	AE_64D_PCA_8D	92.17%	92.11%
	AE_64D_PCA_16D	92.49%	92.44%
	AE_64D_PCA_32D	92.59%	92.53%

Table 3

XGBoost

XGBoost tuned as following Figure 35,36.

```
elif model_name == 'XGBoost':
    model = XGBClassifier(n_estimators=100, random_state=42, n_jobs=-1,
                          eval_metric='mlogloss')
```

Figure 35

```
  results['XGBoost'] = train_and_evaluate_model_with_metrics(
      'XGBoost', feature_sets, y_train, y_test, class_weight_dict, le
  )
```

Figure 36

And shows confusion matrix heatmap results as following Figures 37-44

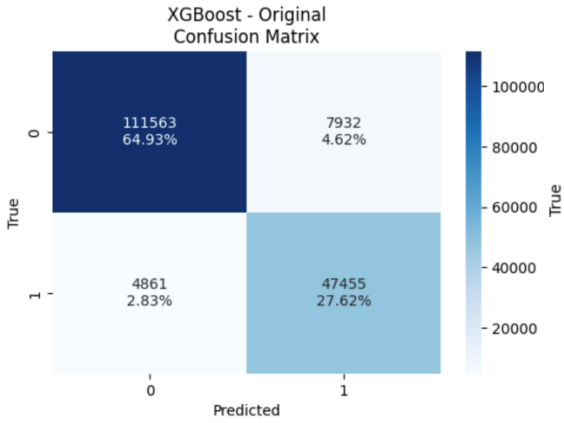


Figure 35

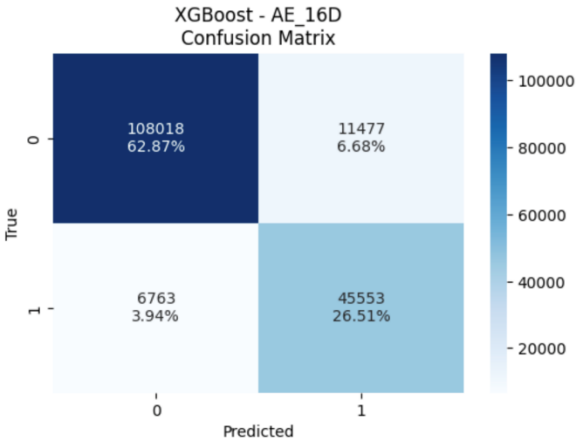
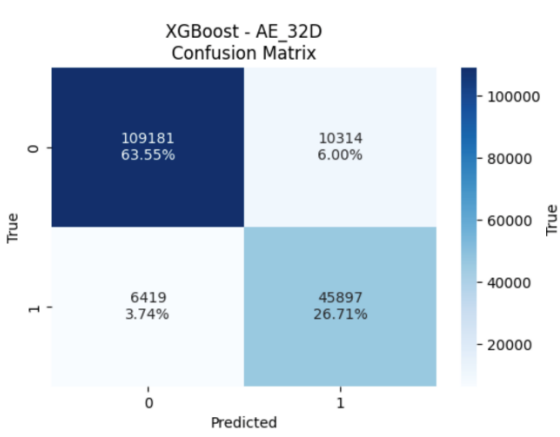


Figure 36



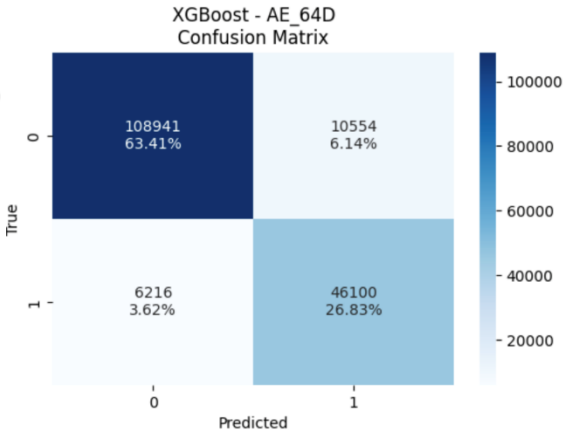
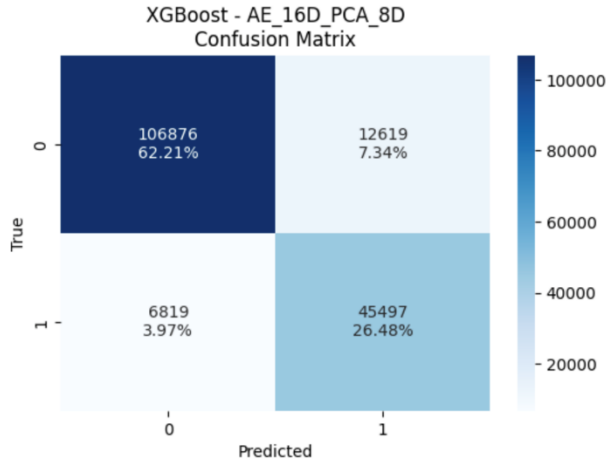


Figure 37

Figure 38



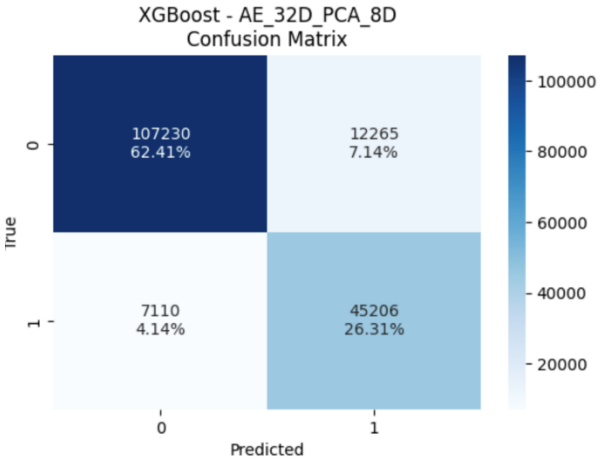
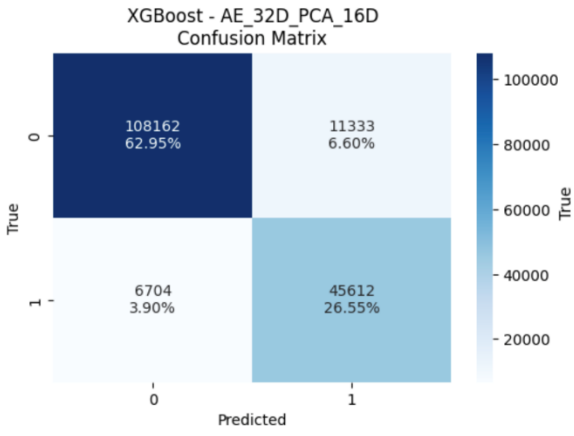


Figure 39

Figure 40



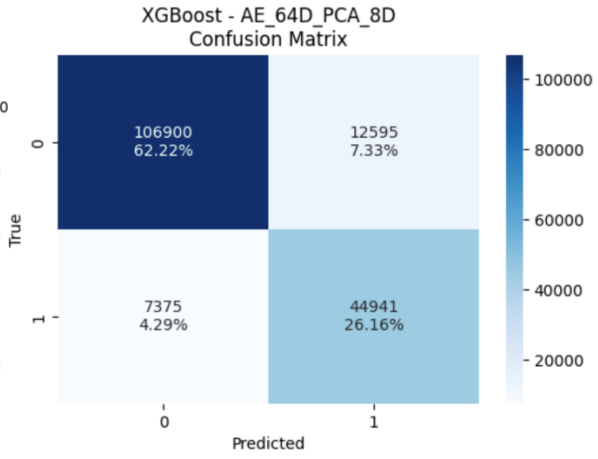


Figure 41

Figure 42

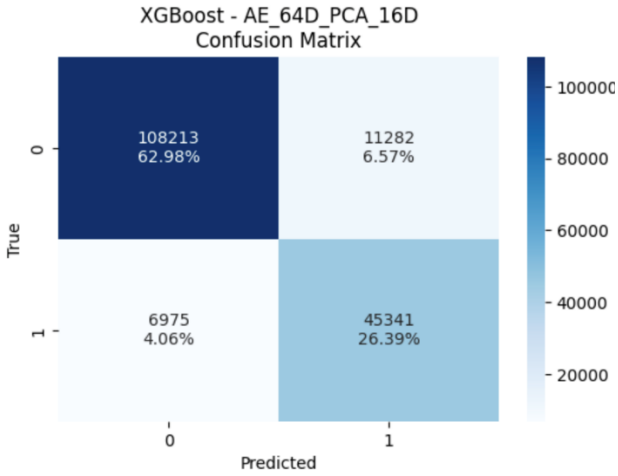


Figure 43

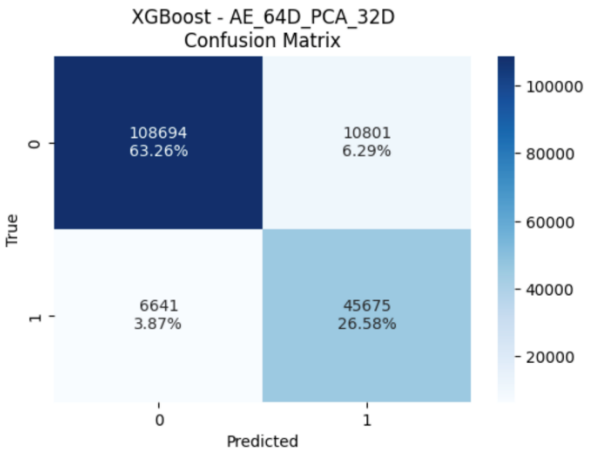


Figure 44

And XGBoost Accuracy and F1 scores are shown in the following Table 4

XGBoost

	feature set	Accuracy	F1
	original	92.55%	92.61%
Autoencoders	AE_16D	89.38%	89.51%
	AE_32D	90.26%	90.35%
	AE_64D	90.24%	90.34%
Autoencoders + PCA	AE_16D_PCA_8D	88.69%	88.84%
	AE_32D_PCA_8D	88.72%	88.86%
	AE_32D_PCA_16D	89.50%	89.62%
	AE_64D_PCA_8D	88.38%	88.52%
	AE_64D_PCA_16D	89.37%	89.49%
	AE_64D_PCA_32D	89.85%	89.95%

LightGBM

tuned as following Figure 45,46.

```
results['LightGBM'] = train_and_evaluate_model_with_metrics(
    'LightGBM', feature_sets, y_train, y_test, class_weight_dict, le
)
```

Figure 45

```
elif model_name == 'LightGBM':
    model = LGBMClassifier(n_estimators=500, max_depth=15, class_weight='balanced',
                           random_state=42, verbose=-1)
else:
```

Figure 46

And shows confusion matrix heatmap results as following Figures 47-

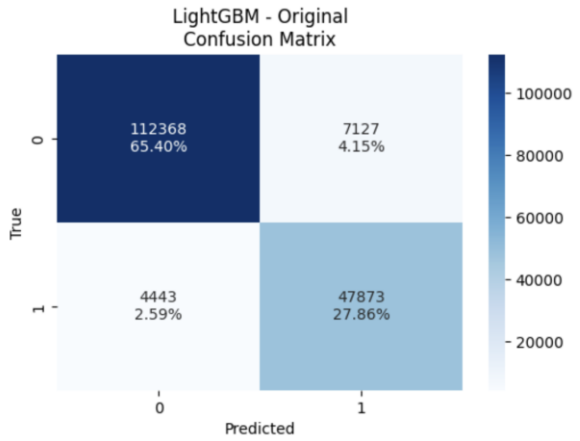


Figure 47

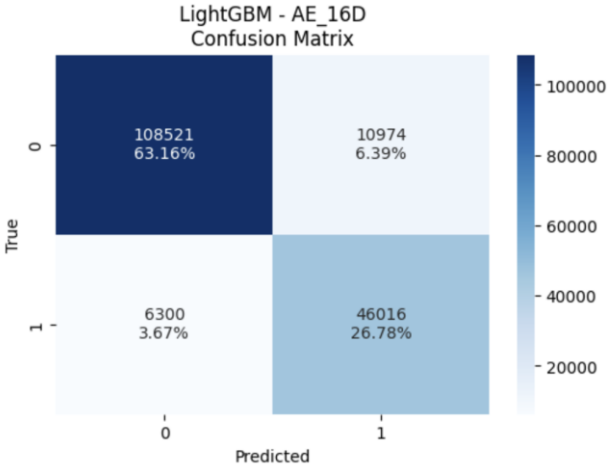


Figure 48

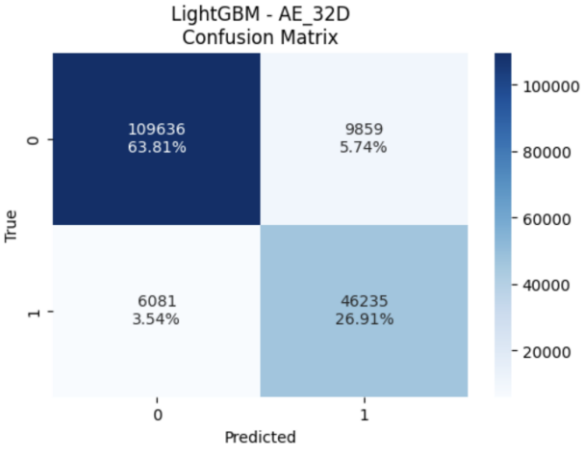


Figure 49

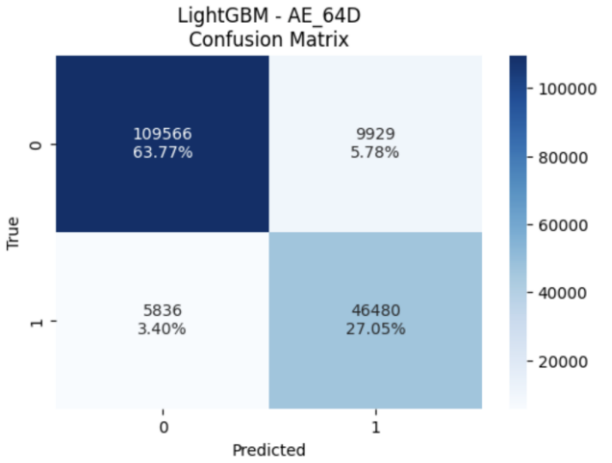


Figure 50

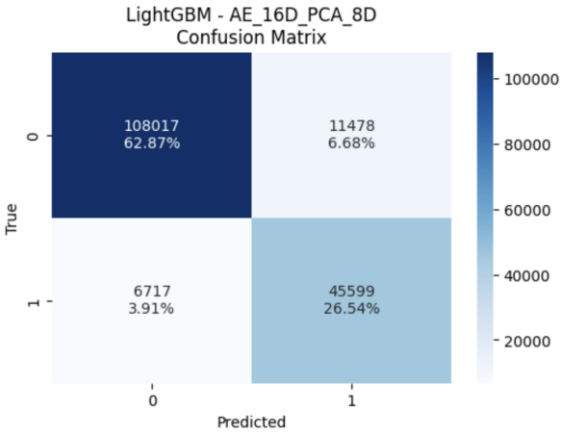


Figure 51

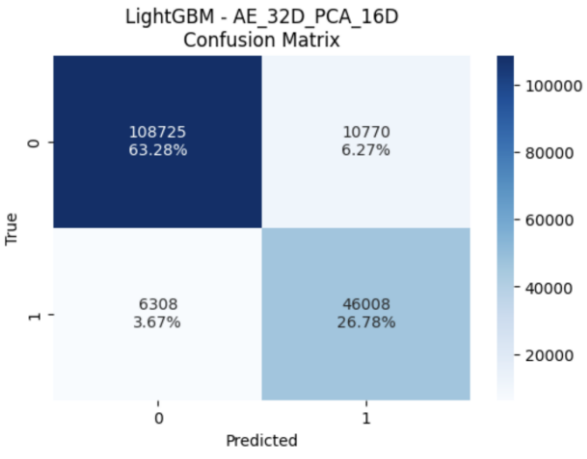


Figure 52

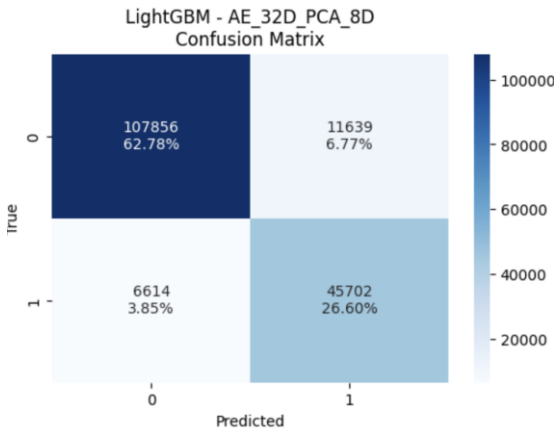


Figure 53

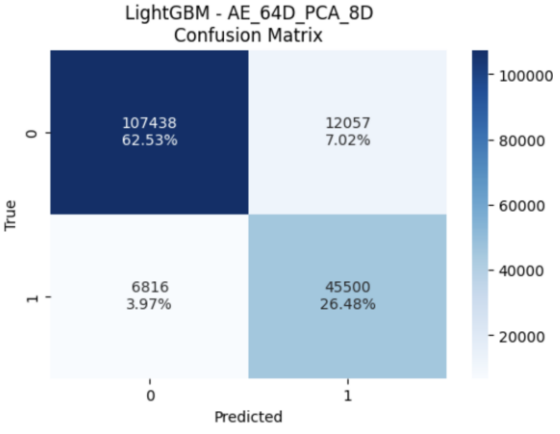


Figure 54

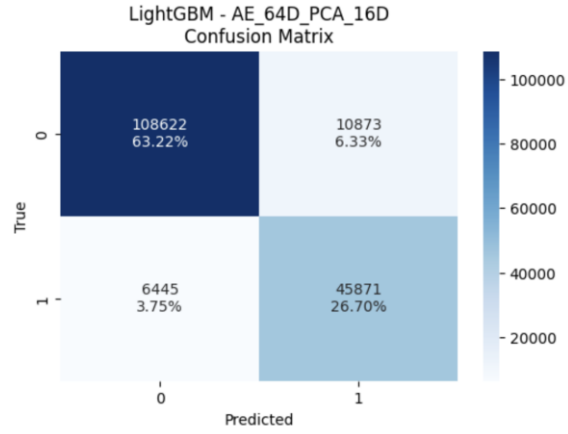


Figure 55

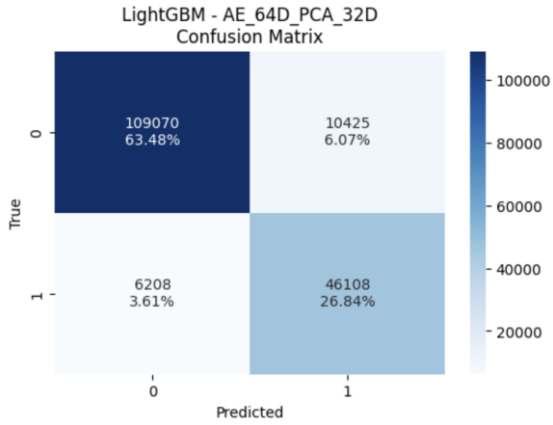


Figure 56

And LightGBM Accuracy and F1 scores are shown in the following Table 5

LightGBM			
	feature set	Accuracy	F1
	original	93.27%	93.31%
Autoencoders	AE_16D	89.95%	90.06%
	AE_32D	90.72%	90.81%
	AE_64D	90.82%	90.92%
Autoencoders + PCA	AE_16D_PCA_8D	89.41%	89.53%
	AE_32D_PCA_8D	89.38%	89.51%
	AE_32D_PCA_16D	90.06%	90.17%
	AE_64D_PCA_8D	89.02%	89.15%
	AE_64D_PCA_16D	89.92%	90.03%
	AE_64D_PCA_32D	90.32%	90.42%

Table 5