National College of Ireland

Configuration Manual

MSc Research Project
Cybersecurity

Sailee Wakhare
Student ID: 23100508

School of Computing
National College of Ireland

Supervisor: Niall Heffernan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sailee Sanjeev Wakhare		
Student ID:	23100508		
Programme:	MSc Cybersecurity	**Year:**	2024-25
Module:	Practicum Part 2		
Lecturer:	Niall Heffernan		
Submission Due Date:	12/12/2024		
Project Title:	Securing Finance System Using Deep Learning Techniques: Credit Card Fraud Detection		
Word Count:	1034	**Page Count: 6**	

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: Sailee Sanjeev Wakhare

Date: 12/12/2024

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

Sailee Wakhare
23100508:

1 Configuration Manual

The set of all software and hardware tools which was used to carry out this research are highlighted in below sections. It has steps to create the environment required to implement this research.

2 Software Requirements

For this thesis, below software tools were used:

Computing platform: Jupyter Notebook version 7.2.1

Programming Language: Python version 3.12.2

Dataset collection: Kaggle

Tools for making diagrams and tables: Draw.io**,** Excel

3 Hardware Requirements

The hardware requirements are as follows:

Device: HP Pavilion Laptop

OS: Windows 11 Home Single Language Version 23H2

RAM: 16 GB

4 Creating Virtual Environment

In Jupyter, virtual environment is created to avoid library dependencies and to switch faster between programs. Below are the steps for the same:

1) Make sure that Python is installed. Install the virtualenv module using the pip install virtualenv command.
2) Go to the directory which is going to be used and create virtual environment using command: python -m venv environmentname, here environmentname is the placeholder for environment name of your choice.
3) Once done, activate the virtual environment using command: environmentname \Scripts\activate

Then add this to Jupyter notebook using command: python -m ipykernel install --user --name= environmentname --display-name "Python (environmentname)"

5 Implementation

Now that the setup is done, the implementation takes place through following steps.

1) Data Preprocessing – Data cleaning is carried out by:
   a) Handling the missing values and null values
   b) Eliminating the duplicate values
   c) Shuffling dataset for bias reduction

```
# Data Cleaning
# Filling missing values with column mean
data.fillna(data.mean(numeric_only=True), inplace=True)

# Removing duplicate rows
data.drop_duplicates(inplace=True)

# Shufflling the dataset to reduce bias
data = shuffle(data, random_state=42)
```

Figure 1. Data Cleaning

2) The outliers are removed using the IQR method. In this method, for the numerical columns in data, IQR (Interquartile range) is calculated, and upper, lower bounds are selected and these are used to filter the dataset.

```
# Step 2: Remove outliers using IQR for numeric columns
numeric_cols = data.select_dtypes(include=[float, int]).columns

def remove_outliers_iqr(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]

for col in numeric_cols:
    data = remove_outliers_iqr(data, col)
```

Figure 2. Removing Outliers

3) Feature Scaling is carried out to stabilize the data and Label encoding is performed to handle categorical data on the dataset.

```
# Step 3: Feature Scaling
scaler = StandardScaler()
data[numeric_cols] = scaler.fit_transform(data[numeric_cols])

# Step 4: Label Encoding for the target variable
label_encoder = LabelEncoder()
data['Outcome'] = label_encoder.fit_transform(data['Outcome'])
```

Figure 3. Feature Scaling

4) The data imbalance is handled using the SMOTE technique where synthetic cases for the minority class is generated. In below snippet, the data is prepared for SMOTE and SMOTE is applied on it.

```
# Step 5: Prepare Features for SMOTE
X = data.drop(columns=['Outcome'])
y = data['Outcome']

# Handle categorical variables
X = pd.get_dummies(X, drop_first=True)

# Apply SMOTE to handle imbalanced data
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

Figure 4. SMOTE for data balancing

5) The data is split into the data splitting phase into 80% for training set and 20% for testing set.

```
# Step 6: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

Figure 5. Data Splitting

6) The hyperparameter tuning is performed using the RandomSearchCV.

```
random_search = RandomizedSearchCV(
    MLPClassifier(solver='adam', max_iter=3000, random_state=42),
    param_distributions,
    n_iter=10,
    n_jobs=-1,
    cv=3,
    scoring='accuracy',
    random_state=42
)

random_search.fit(X_train, y_train)
best_params = random_search.best_params_
```

Figure 6. Hyperparameter Tuning

Implementing the algorithms:

Multilayer Perceptron: The best parameters are considered and MLP is applied with 50 epochs for iteration. The warm_start method is used to keep the model training continuously occurring. The training and validationaccuracy is captured with each epoch passing. Multiple iterations allows the model to learn from data until it reaches stability. Once the model is trained, it is tested and evaluation metrices are calculated and printed for results.

```
# Step 8: Training MLP model with best parameters (without early_stopping)
mlp_best = MLPClassifier(**best_params, solver='adam', max_iter=1, warm_start=True, random_state=42

# Track training progress for loss and accuracy
train_accuracies = []
val_accuracies = []
train_losses = []
val_losses = []

# Incremental training loop
epochs = 50  # Number of epochs
for epoch in range(epochs):
    mlp_best.fit(X_train, y_train)  # Incremental training with warm_start
    train_accuracies.append(mlp_best.score(X_train, y_train))  # Calculate training accuracy
    val_accuracies.append(mlp_best.score(X_test, y_test))  # Calculate validation accuracy
    if hasattr(mlp_best, "loss_"):
        train_losses.append(mlp_best.loss_)  # Store the current training loss
    val_predictions = mlp_best.predict_proba(X_test)
    val_loss = -np.mean(np.log(val_predictions[range(len(y_test)), y_test]))
    val_losses.append(val_loss)
```

Figure 7. MLP

Long Short-Term Memory: The LSTM model is trained for sequential data, where three layers are considered with unit 128, 64, 32. For thr purpose of achieving stability, to avoid overfitting challenges, the batch normalization and dropout layers are used. The outout from initial layers goes through the dense layers. Adam optimizer is used here for compiling the model. Also, overfitting is avoided using early stopping. 50 epochs are used to train the model. The classification report and other evaluation metrices are calculated.

```
# Build the LSTM model with enhancements
lstm_model = Sequential([
    LSTM(128, input_shape=(X_train_lstm.shape[1], 1), return_sequences=True),
    BatchNormalization(),
    Dropout(0.5),
    LSTM(64, return_sequences=True),
    BatchNormalization(),
    Dropout(0.5),
    LSTM(32),
    Dense(50, activation='relu'),
    Dense(y_train_lstm.shape[1], activation='softmax')
])
# Compiling the model
lstm_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
# Learning Rate Scheduler
def lr_schedule(epoch, lr):
    if epoch > 20:
        return lr * 0.5
    return lr
lr_scheduler = LearningRateScheduler(lr_schedule)
# Train the model with callbacks
history = lstm_model.fit(
    X_train_lstm, y_train_lstm,
    epochs=50,
    batch_size=32,
```

Figure 8. LSTM

Convolution Neural Network: The data that is split is used and it is reshaped. It is necessary for the data to fit into format of CNN. The CNN model is build using first the convolutional layer which is followed by he max-pooling layer and dropout, and then for the classification, it goes through dense layers. The Adam optimizer is used and this model is trained for over 100 epochs. The evaluation metrices are calculated and analyzed.

```
# Step 6: Preparing Data for CNN
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Reshape the data for the CNN model
X_train_cnn = X_train.values.reshape((X_train.shape[0], X_train.shape[1], 1)).astype('float32')
X_test_cnn = X_test.values.reshape((X_test.shape[0], X_test.shape[1], 1)).astype('float32')

# Convert the target variable to categorical
y_train_cnn = to_categorical(y_train).astype('float32')
y_test_cnn = to_categorical(y_test).astype('float32')

# Build the CNN model
cnn_model = Sequential([
    Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(X_train_cnn.shape[1], 1)),
    MaxPooling1D(pool_size=2),
    Dropout(0.5),
    Flatten(),
    Dense(50, activation='relu'),
    Dense(y_train_cnn.shape[1], activation='softmax')
])

# Compile the model
cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model and capture the history
history = cnn_model.fit(X_train_cnn, y_train_cnn, epochs=100, batch_size=32, validation_data=(X_test_cnn, y_test_cnn))
```

Figure 9. CNN

K-Nearest Neighbour: The KNN algorithm is applied by on the dataset. The resampled data is used for this by considering k=5 which means that there are five neighbours with equal weightage. The testing data is used to make predictions. The evaluation metrices like accuracy, confusion matrix and classification report containing precision, recall, F1-score is calculated.

```
# Step 7: Applying K-Nearest Neighbors (KNN)
knn = KNeighborsClassifier(n_neighbors=5, weights='uniform')
knn.fit(X_train_resampled, y_train_resampled)

# Step 8: Making Predictions
y_pred = knn.predict(X_test)

# Step 9: Evaluating the Model
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nAccuracy Score:")
accuracy = accuracy_score(y_test, y_pred) * 100
print(f"Accuracy: {accuracy:.2f}%")
```

Figure 10. KNN

Implementing SHAP for feature importance with XGBoost: The XGBoost model is used to train the data and after that, SHapely Additive exPlanations is used to carry out feature importance. The shap explainer is used to caculate how much each feature contributes and what is the impact. The absolute mean of these values is considered and aparticluar percentage is allocated to each feature. This percentage signifies the feature's contribution to results and to understand the decision-making of the model.

```python
# Explain the model's predictions using SHAP
explainer = shap.Explainer(model, X_train_smote)
shap_values = explainer(X_train_smote)

# Calculate percentage contribution for feature importance
feature_importance = np.abs(shap_values.values).mean(axis=0)
features = X_train_smote.columns
percentage_contribution = (feature_importance / feature_importance.sum()) * 100

# Create a DataFrame for feature importance with percentage contribution
feature_importance_df = pd.DataFrame({'Feature': features, 'Importance': feature_importance, 'Percentage': percentage_contribution})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Display feature importance table
print(feature_importance_df)

# Custom bar chart for SHAP feature importance with percentage contribution
plt.figure(figsize=(16, 10))
plt.bar(feature_importance_df['Feature'], feature_importance_df['Percentage'], color='lightcoral')
plt.ylabel('Percentage Contribution (%)', fontsize=14)
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.xlabel('Features', fontsize=14)
plt.title('Feature Importance based on SHAP values (Percentage Contribution)', fontsize=16)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

Figure 11. SHAP