Configuration Manual

MSc Research Project
MSC CYBER SECURITY

SUNIL KUMAR PRATURI
Student ID: X23242558

School of Computing
National College of Ireland

Supervisor: **NIALL HEFFERNAN**

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	SUNIL KUMAR PRATURI
Student ID:	X23242558
Programme:	MSC CYBER SECURITY **Year:** 2024-2025
Module:	MSC PRACTICUM
Lecturer:	NIALL HEFFERNAN
Submission Due Date:	12-12-2024
Project Title:	Evaluating the Effectiveness of Machine Learning Algorithms in Detecting Phishing Attacks
Word Count:	383 WORDS **Page Count:** 06

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: SUNIL KUMAR PRATURI

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

SUNIL KUMAR PRATURI
Student ID: X23242558

1. Introduction
This manual provides the configuration details and steps for running the phishing dataset analysis. The dataset consists of features for phishing detection, and this analysis will include data preprocessing, feature engineering, model selection, evaluation, and hyperparameter tuning.

Minimum System Requirements

Hardware Requirements:

- **Operating System**: Windows 10/11, macOS 10.15 or higher, or any Linux distribution (Ubuntu 18.04 or higher).
- **Processor**: Intel Core i5 or equivalent AMD processor, at least 4 cores.
- **RAM**: Minimum 8 GB (16 GB recommended for smoother operations).
- **Graphics Processing Unit (GPU)**: Optional, but recommended if using machine learning for optimization.
- **Storage**: At least 10 GB of free disk space for storing images, models, and results.

Software Requirements

- Python: Version 3.8 or higher
- Jupyter Notebook: For running the code and visualizations.

2. Environment Setup
Ensure that the following Python packages are installed to run the analysis:

Pandas: For data handling and preprocessing
NumPy: For numerical operations
Matplotlib: For plotting and visualization
Seaborn: For advanced visualizations (e.g., heatmaps, box plots)
Scikit-learn: For machine learning models, preprocessing, and evaluation
You can install the required dependencies using pip:

pip install pandas numpy matplotlib seaborn scikit-learn

3. Dataset Configuration
The analysis uses two CSV files for training and testing:

Training Dataset: phishing_dataset_train.csv
Testing Dataset: phishing_dataset_test.csv
These datasets must be placed in the working directory or specified in the script for the pd.read_csv() function to load them.

Dataset Structure
Target Column: class
This is the binary target variable indicating phishing (1) or not (0).
Feature Columns: All columns except class, including numerical and categorical features.

Data Exploration and Preprocessing

Load the Datasets

```
import pandas as pd
import numpy as np


# Load the training and testing datasets
train_df = pd.read_csv('phishing_dataset_train.csv')
test_df = pd.read_csv('phishing_dataset_test.csv')

# Display the first few rows of the training dataset
print("Training Dataset Preview:")
print(train_df.head())

# Display the first few rows of the testing dataset
print("\nTesting Dataset Preview:")
print(test_df.head())
```

```
Training Dataset Preview:
   domain_similarity  url_length  http_protocol  num_dot  num_slash  \
0               0.89          29              1        1          0
1               0.88          28              1        1          0
```

4. Data Preprocessing Configuration

Handling Missing Values
The code includes checks for missing values in both training and testing datasets:

Feature Scaling
Standard scaling is applied to numerical features to normalize them between zero and one:

Identify Target Variables and Feature Columns

```
# List all columns in the training dataset
print("Columns in Training Dataset:")
print(train_df.columns.tolist())

# Define the target column
target_column = 'class'  # Confirm if 'class' is indeed your target

# Features are all columns except the target column
feature_columns = [col for col in train_df.columns if col != target_column]

print(f"\nTarget Column: {target_column}")
print(f"Number of Feature Columns: {len(feature_columns)}")
print("Feature Columns:")
print(feature_columns)
```

```
Columns in Training Dataset:
['domain_similarity', 'url_length', 'http_protocol', 'num_dot', 'num_slash', 'num_double_slash', 'num_h
thesis', 'num_curly_bracket', 'num_square_bracket', 'num_less_and_greater', 'num_tilde', 'num_asterisk'
se_history', 'redirect', 'num_a_href', 'num_input', 'num_button', 'num_link_href', 'num_iframe', 'class

Target Column: class
Number of Feature Columns: 25
Feature Columns:
['domain_similarity', 'url_length', 'http_protocol', 'num_dot', 'num_slash', 'num_double_slash', 'num_h
```

Feature Engineering

You can add new features or modify existing ones to improve model performance. For example, the interaction between num_dot and num_slash is created:

Feature Engineering

```
[13]: # Identify categorical features
categorical_features = train_df.select_dtypes(include=['object', 'category']).columns.tolist()
print(f"\nCategorical Features: {categorical_features}")
```

```
Categorical Features: []
```

- Feature Scaling

```
[7]: # Define the target column
target_column = 'class'

# Identify numerical features (excluding the target)
numerical_features = train_df.select_dtypes(include=[np.number]).columns.tolist()
numerical_features = [col for col in numerical_features if col != target_column]

print(f"Numerical Features ({len(numerical_features)}):")
print(numerical_features)
```

```
Numerical Features (25):
['domain_similarity', 'url_length', 'http_protocol', 'num_dot', 'num_slash', 'num_double_slash', 'num_hyphen', 'num_underscore', 'num_equal', 'num_paran
thesis', 'num_curly_bracket', 'num_square_bracket', 'num_less_and_greater', 'num_tilde', 'num_asterisk', 'num_plus', 'url_inc_at', 'url_inc_ip', 'respon
se_history', 'redirect', 'num_a_href', 'num_input', 'num_button', 'num_link_href', 'num_iframe']
```

5. Model Configuration

Model Selection

The following models are included in the analysis:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, confusion_matrix, classification_report
from sklearn.model_selection import cross_val_score

import matplotlib.pyplot as plt
import seaborn as sns

# Initialize the models with default parameters
rf = RandomForestClassifier(random_state=42)
svm = SVC(probability=True, random_state=42)
knn = KNeighborsClassifier()
nb = GaussianNB()
dt = DecisionTreeClassifier(random_state=42)

# List of models for iteration
models = {
    'Random Forest': rf,
    'SVM': svm,
    'k-NN': knn,
    'Naïve Bayes': nb,
    'Decision Tree': dt
}
```

Model Training

Each model is trained using the fit() method, and predictions are made on the test set:

```python
for model_name, model in models.items():
    print(f"\nTraining and evaluating {model_name}...")

    # Train the model
    model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else model.decision_function(X_test)

    # Calculate performance metrics
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_proba)

    # Store the metrics
    performance_metrics[model_name] = {
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-Score': f1,
        'ROC-AUC': roc_auc
    }

    # Print classification report
    print(f"Classification Report for {model_name}:")
```

6. Hyperparameter Tuning Configuration

Random Forest Hyperparameter Tuning

Use GridSearchCV to tune the hyperparameters of the Random Forest model:

[21]:

```python
from sklearn.model_selection import GridSearchCV

# Define parameter grid for Random Forest
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize GridSearchCV
grid_search_rf = GridSearchCV(
    estimator=rf,
    param_grid=param_grid_rf,
    cv=5,
    scoring='f1',
    n_jobs=-1,
    verbose=2
)

# Perform Grid Search
grid_search_rf.fit(X_train, y_train)

# Best parameters and score
print(f"\nBest Parameters for Random Forest: {grid_search_rf.best_params_}")
print(f"Best F1-Score: {grid_search_rf.best_score_:.4f}")
```

Svm Hyperparameter tuning

- Support Vector Machines (SVM) Hyperparameter Tuning

[22]:

```python
from sklearn.model_selection import GridSearchCV

# Define parameter grid for SVM
param_grid_svm = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto']
}

# Initialize GridSearchCV
grid_search_svm = GridSearchCV(
    estimator=svm,
    param_grid=param_grid_svm,
    cv=5,
    scoring='f1',
    n_jobs=-1,
    verbose=2
)

# Perform Grid Search
grid_search_svm.fit(X_train, y_train)

# Best parameters and score
print(f"\nBest Parameters for SVM: {grid_search_svm.best_params_}")
print(f"Best F1-Score: {grid_search_svm.best_score_:.4f}")
```

7. Model Evaluation Configuration

Cross-Validation

Perform k-fold cross-validation to evaluate the stability of the models:

[26]:
```python
from sklearn.model_selection import cross_val_score

# Number of folds
k = 5

# Iterate through the models and perform cross-validation
for model_name, model in models.items():
    print(f"\nPerforming {k}-Fold Cross-Validation for {model_name}...")
    cv_scores = cross_val_score(model, X_train, y_train, cv=k, scoring='f1')
    print(f"F1-Score CV Mean: {cv_scores.mean():.4f} | CV Std: {cv_scores.std():.4f}")
```

```
Performing 5-Fold Cross-Validation for Random Forest...
F1-Score CV Mean: 0.9756 | CV Std: 0.0088

Performing 5-Fold Cross-Validation for SVM...
F1-Score CV Mean: 0.9614 | CV Std: 0.0273

Performing 5-Fold Cross-Validation for k-NN...
F1-Score CV Mean: 0.9619 | CV Std: 0.0180

Performing 5-Fold Cross-Validation for Naïve Bayes...
F1-Score CV Mean: 0.3310 | CV Std: 0.0343
```