Configuration Manual

MSc Research Project
Cyber security

MAHESH KONI
Student ID:23146931

School of Computing
National College of Ireland

Supervisor: Khadija Hafeez

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Mahesh koni

Student ID: 23146931

Programme: Cyber security **Year:** 2024

Module: MSc Research project

Lecturer: Khadija Hafeez

Submission Due Date: 12/12/2024

Project Title: Anomaly Detection-Based Approach for Identifying Domain Generation Algorithm (DGA) Domain in cybersecurity

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: MAHESH KONI

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

MAHESH KONI
23146931

1 Introduction

This project focuses on detecting Domain Generation Algorithm (DGA) domains in cybersecurity using anomaly detection and machine learning. The workflow includes data preprocessing, feature extraction, and model training, leveraging advanced classification techniques

System Requirements

- Operating System: Windows 10, macOS, or Linux
- Google Colab
- Programming Language: Python 3.8 or higher

Libraries and Frameworks:

- csv
- gensim
- pandas
- tensorflow
- numpy
- matplotlib
- nltk
- seaborn
- scikit-learn
- hdbscan
- DBSCAN

Installation Instructions

Install the required libraries on google colab using the !pip install command

```python
import csv
from gensim.models import Word2Vec
import pandas as pd
import os
import tensorflow as tf
import numpy as np
from tensorflow.keras.layers import Input, Embedding, Conv1D, MaxPooling1D, Bidirectional, LSTM, Dense, Dropout, ThresholdedReLU, Flatten
from keras.models import Sequential
from tensorflow.keras import Model
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import nltk
from nltk.util import ngrams
import seaborn as sns
import re
import math
from scipy.spatial.distance import euclidean, pdist, squareform
from sklearn.metrics.pairwise import pairwise_distances
!pip install hdbscan
import hdbscan
from sklearn.cluster import DBSCAN

from sklearn.datasets import make_blobs
```

Dataset Details

Dataset Name: dataset_DGA.csv
Description: Contains labeled domain data for clustering and classification.
Key Columns:

Label: Indicates domain type (legitimate/DGA).
Domain: Domain name for analysis.

```python
print("\nDomain counts:")
print(domain_counts)
```

```
Label counts:
Label
dga      10144
legit    10056
Name: count, dtype: int64

DGA Type counts:
DGA_Type
alexa          10056
emotet           451
fobber           447
tinba            432
simda            427
pykspa           427
kraken           425
padcrypt         424
matsnu           423
symmi            421
pushdo           420
suppobox         418
ramdo            411
ramnit           407
cryptolocker     404
```

2 Data Preprocessing

Feature Engineering Steps:

1. **Number of Characters (num_char):** The length of each domain name.
2. **Unique Character Rate (unique_char_ratio):** The ratio of unique characters to total characters in each domain.
3. **Number of Vowels and Consonants:** Counts of vowels (`vowels`) and consonants (`consonants`) in each domain name.
4. **Percentage of Numeric Characters:** Calculates the percentage of numeric characters in each domain name.

Code + Text

```
import pandas as pd
import math
import re
from collections import Counter
from nltk.util import ngrams
from sklearn.preprocessing import LabelEncoder

def data_preprocessing(data):
    # Num of characters
    data["num_char"] = data["Domain"].apply(len)

    # Unique character rate
    def calculate_unique_characters_ratio(domain):
        return len(set(domain)) / len(domain)
    data["Unique characters ratio"] = data["Domain"].apply(calculate_unique_characters_ratio)

    # Number of consonant
    data['Vowels'] = data['Domain'].str.lower().str.count(r'[aeiou]')
    data['Consonant'] = data.Domain.str.lower().str.count(r'[a-z]') - data['Vowels']

    # Percentage of numeric characters
    def percentage_of_numerical_chars(data):
        alpha = []
        numeric = []
        for i in data["Domain"]:
```

5. **Entropy:** Measures the randomness or unpredictability of each domain name.
6. **N-grams and Similarity:**

   Calculates the Jaccard similarity between each domain name and a list of legitimate domain name 3-grams (`d_3gram`) and 4-grams (`d_4gram`).

```
data = percentage_of_numerical_chars(data)

# Entropy
def entropy(s):
    p, lns = Counter(s), float(len(s))
    return -sum(count / lns * math.log(count / lns, 2) for count in p.values())

data['Entropy'] = data['Domain'].apply(lambda x: entropy(x))

# N-grams and similarity
def create_ngrams(data_list, n):
    n_grams = []
    for i in data_list:
        l1 = list(ngrams(str(i), n))
        for gram in l1:
            n_grams.append("".join(gram))
    return n_grams

def jaccard_similarity(str1, d, n):
    str1_ngrams = create_ngrams([str1], n)
    intersection = len(set(str1_ngrams).intersection(set(d)))
    union = len(set(str1_ngrams).union(set(d)))
    return float(intersection) / union

list1 = data.loc[data["Label"] == "legit", "Domain"].tolist()
```

7. **Number of Dots (ndots):** Counts the number of dots in each domain name.

8. **Number of Consecutive Vowels:** Calculates the number of consecutive vowels in each domain name.

```
    union = len(set(str1_ngrams).union(set(d)))
    return float(intersection) / union

list1 = data.loc[data["Label"] == "legit", "Domain"].tolist()
d_3gram = create_ngrams(list1, 3)
d_4gram = create_ngrams(list1, 4)

data["3_grams"] = data["Domain"].apply(lambda x: jaccard_similarity(x, d_3gram, 3))
data["4_grams"] = data["Domain"].apply(lambda x: jaccard_similarity(x, d_4gram, 4))

# Number of dots
data['ndots'] = data['Domain'].str.count("\.")

# Number of consecutive vowels
def count_vowels(word):
    return sum(1 for i in range(len(word) - 1) if word[i] in 'aeiou' and word[i + 1] in 'aeiou')

data["Number of consecutive vowels"] = data["Domain"].apply(count_vowels)

# Longest sequence of consonants
def find_consonants(string):
    q = re.findall(r'[^aeiou]+', string)
    return len(max(q, key=len)) if q else 0

data["Longest consonant sequence"] = data["Domain"].apply(find_consonants)
```

9. **Longest Sequence of Consonants:** Finds the longest sequence of consecutive consonants in each domain name.
10. **Label Encoding:** Encodes the `Label` and DGA_Type columns using `LabelEncoder` from scikit-learn.

```
    # Label encoding
    label_encoder = LabelEncoder()
    data["Label"] = label_encoder.fit_transform(data["Label"])
    if 'DGA_Type' in data.columns:
        data['DGA_Type'] = label_encoder.fit_transform(data['DGA_Type'])

    return data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20200 entries, 0 to 20199
Data columns (total 14 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Label                          20200 non-null  int64
 1   DGA_Type                       20200 non-null  int64
 2   Domain                         20200 non-null  object
 3   num_char                       20200 non-null  int64
 4   Unique characters ratio        20200 non-null  float64
 5   Vowels                         20200 non-null  int64
 6   Consonant                      20200 non-null  int64
 7   percentage of numeric characters  20200 non-null  float64
 8   Entropy                        20200 non-null  float64
 9   3_grams                        20200 non-null  float64
 10  4_grams                        20200 non-null  float64
 11  ndots                          20200 non-null  int64
 12  Number of consecutive vowels   20200 non-null  int64
 13  Longest consonant sequence     20200 non-null  int64
dtypes: float64(5), int64(8), object(1)
memory usage: 2.2+ MB
```

Exploratory Data Analysis

Counts and distributions of DGA types and labels were analyzed to understand the dataset's composition. Bar charts were plotted to visualize the frequency of different DGA types.

+ Code + Text Connect ✦ Ge

```
0
```

```
# Get unique DGA types and their counts
dga_type_counts = data['DGA_Type'].value_counts()
dga_type_counts
```

DGA_Type	count
0	10056
5	451
6	447
24	432
21	427
15	427
8	425
13	424
9	423

Feature and Target Variable Splitting

The dataset was preprocessed to extract features and target variables:

- **Features (X):** All columns except `Label`, `Domain`, and DGA_Type.
- **Target (y):** The `Label` column, which indicates whether a domain is legitimate or generated by a DGA.

The data was split into training and testing subsets using an 80-20 split:

```
from sklearn.model_selection import train_test_split
X = data.drop(['Label', 'Domain','DGA_Type'], axis=1)
y = data['Label']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=2)
```

3 Model Training and Testing

Random Forest Classifier

After comparing models, the Random Forest Classifier was selected for hyperparameter tuning and final evaluation. The following steps were implemented:

Hyperparameter Tuning:

- GridSearchCV was used to tune parameters such as:

- `n_estimators` (number of trees)
- `max_features` (features considered for split)
- `max_depth` (tree depth)
- `max_leaf_nodes` (maximum leaf nodes)

Model Training:

The classifier was trained on the training set using the best hyperparameters identified:

Random Forest Classifier

```
[ ] from sklearn.model_selection import train_test_split, GridSearchCV
    from sklearn.ensemble import RandomForestClassifier

    # Define the hyperparameters grid
    param_grid = {
        'n_estimators': [25, 50, 100, 150],
        'max_features': ['sqrt', 'log2', None],
        'max_depth': [3, 6, 9],
        'max_leaf_nodes': [3, 6, 9],
    }

    # Initialize the RandomForestClassifier
    rf_classifier = RandomForestClassifier(random_state=42)

    # Initialize GridSearchCV
    grid_search = GridSearchCV(rf_classifier, param_grid, cv=5, n_jobs=-1)

    # Fit the model on the training data
    grid_search.fit(X_train, y_train)

    # Get the best parameters
```

```
# Fit the model on the training data
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Get the best model
best_model = grid_search.best_estimator_

# Evaluate the model on the test data
accuracy = best_model.score(X_test, y_test)
print("Accuracy:", accuracy)
```

```
Best Parameters: {'max_depth': 6, 'max_features': None, 'max_leaf_nodes': 9, 'n_estimators': 25}
Accuracy: 0.9522277227722772
```

```
[ ] from sklearn.metrics import classification_report, confusion_matrix

    # Make predictions
    y_pred = best_model.predict(X_test)

    # Classification report
```

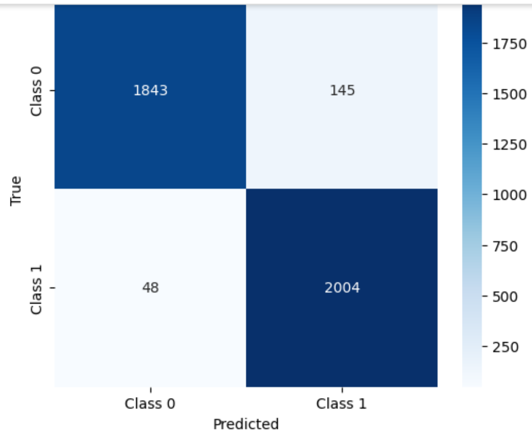
Evaluation:

1. Accuracy: Assessed on the testing set.
2. Confusion Matrix: To analyze prediction performance.
3. Classification Report: Provided precision, recall, and F1-score

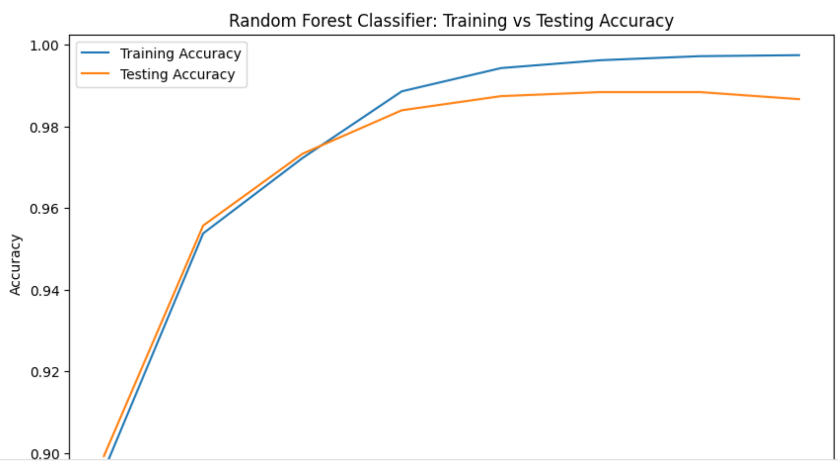
```python
from sklearn.metrics import classification_report, confusion_matrix

# Make predictions
y_pred = best_model.predict(X_test)

# Classification report
print(classification_report(y_test, y_pred))

# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
              precision    recall  f1-score   support

           0       0.97      0.93      0.95      1988
           1       0.93      0.98      0.95      2052

    accuracy                           0.95      4040
   macro avg       0.95      0.95      0.95      4040
weighted avg       0.95      0.95      0.95      4040

Confusion Matrix:
 [[1843  145]
```

Visualize the Results




```python
import matplotlib.pyplot as plt
```



References

Nassar, N., Jafar, A. and Rahhal, Y., 2020. A novel deep multi-criteria collaborative filtering model for recommendation system. *Knowledge-Based Systems*, 187, p.104811.

Beheshti, A., Yakhchi, S., Mousaeirad, S., Ghafari, S.M., Goluguri, S.R. and Edrisi, M.A., 2020. Towards cognitive recommender systems. *Algorithms*, 13(8), p.176.

Sharma, S., Rana, V. and Malhotra, M., 2021. Automatic recommendation system based on hybrid filtering algorithm. *Education and Information Technologies*, 27, pp.1-16.

Reddy, S.R.S., Nalluri, S., Kunisetti, S., Ashok, S. and Venkatesh, B., 2019. Content-based movie recommendation system using genre correlation. In *Smart Intelligent Computing and Applications* (pp. 391-397). Springer, Singapore.

Yasen, M. and Tedmori, S., 2019. Movies reviews sentiment analysis and classification. In *Proceedings of the IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, pp. 860-865.

Rajput, N. and Chauhan, S., 2019. Analysis of various sentiment analysis techniques. *International Journal of Computer Science and Mobile Computing*, 8(2), pp.75-79.

Shaukat, Z., Zulfiqar, A.A., Xiao, C., Azeem, M. and Mahmood, T., 2020. Sentiment analysis on IMDB using lexicon and neural networks. *SN Applied Sciences*, 2(2), pp.1-10.

Widiyaningtyas, T., Hidayah, I. and Adji, T.B., 2021. User profile correlation-based similarity (UPCSim) algorithm in movie recommendation system. *Journal of Big Data*, 8, p.52.

Singh, R.H., Maurya, S., Tripathi, T., Narula, T. and Srivastav, G., 2020. Movie recommendation system using cosine similarity and KNN. *International Journal of Engineering and Advanced Technology (IJEAT)*, 9(5), pp.2-3.

Kumar, S., De, K. and Roy, P.P., 2020. Movie recommendation system using sentiment analysis from microblogging data. *IEEE Transactions on Computational Social Systems*, 7(4), pp.915-923.

Rahman, A. and Hossen, M.S., 2019. Sentiment analysis on movie review data using machine learning approach. In *Proceedings of the International Conference on Bangla Speech and Language Processing (ICBSLP)*, IEEE, pp. 1-4.

Uddin, S., Khan, A., Hossain, M.E. and Moni, M.A., 2019. Comparing different supervised machine learning algorithms for disease prediction. *BMC Medical Informatics and Decision Making*, 19(1), pp.1-16.

Ghosh, S., Dasgupta, A. and Swetapadma, A., 2019. A study on support vector machine based linear and non-linear pattern classification. In *Proceedings of the International Conference on Intelligent Sustainable Systems (ICISS)*, IEEE, pp. 24-28.

Dashtipour, K., Gogate, M., Adeel, A., Larijani, H. and Hussain, A., 2021. Sentiment analysis of Persian movie reviews using deep learning. *Entropy*, 23(5), p.596.

Soubraylu, S. and Rajalakshmi, R., 2021. Hybrid convolutional bidirectional recurrent neural network based sentiment analysis on movie reviews. *Computational Intelligence*, 37(2), pp.735-757.