National College of Ireland

Configuration Manual

MSc Research Project
Cybersecurity

Saiprasad Salian
Student ID: x22183761

School of Computing
National College of Ireland

Supervisor: Michael Prior

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Saiprasad Salian
Student ID:	x22183761
Programme:	Cybersecurity
Year:	2024
Module:	MSc Research Project
Supervisor:	Michael Prior
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	785
Page Count:	8

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:	Sai Salian
Date:	11th August 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

Saiprasad Salian
x22183761

1 Introduction

The study proposed an approach for Phishing Detection using DistilBERT for sentiment analysis with SVM as a classifier. This documnet provides the different software and hardware requirements required for the setup of the proposed design. We have included the implementation as per the flow required below. The primary goal of the research is to find out phishing emails from a dataset containing spam and real emails. All while using the power of sentiment analysis and classification. The research proposes and implements the methodology for a system that detects phishing emails. There are several key steps to this implementation which we have covered below.

2 System Requirements

Given below are the hardware and software requirements to replicate the proposed approach.

2.1 Software Requirements

- Python v3
- Google Colab
- Google Drive

2.2 Hardware Configuration of Google Colab

- CPU RAM of 12.7 GB
- GPU RAM of 15 GB
- Disk Space of 78.2 GB

2.3 Base System Hardware configuration

- Processor -12th Gen Intel(R) Core(TM) i5-1235U 1.30 GHz
- RAM - 16.0 GB (15.7 GB usable)
- Hard Disk - 500 GB SSD
- OS - Windows 11 64 bit

2.4 Libraries

- Pandas
- NumPy
- NLTK
- Torch
- Joblib
- Scikit-learn
- Transformers
- Matplotlib
- Google Colab‘ (for cloud-based implementation)

3 Implementation

3.1 Preparing the Data

3.1.1 Mounting Google Drive

We start off by mounting the google drive so that we can access the dataset that we've stored in the drive.

```
from google.colab import drive
drive.mount('/content/drive')
```

Figure 1: Mounting Google Drive

3.1.2 Importing Libraries:

Importing essential libraries is the next step to avoid any errors later. Libraries such as NumPy, Transfromers, sklearn, torch and so on are imported.

```
import pandas as pd
import numpy as np
import nltk
import torch
import joblib
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from transformers import DistilBertTokenizer, DistilBertModel
```

Figure 2: Importing all the libraries

3.1.3 Downloading NLTK Data:

Using the NLTK we download stopwords which will be further used for preprocessing of the data.

```
nltk.download('stopwords')
```

Figure 3: Downloading Stopwords

3.1.4 Loading the Dataset:

Next step is to load the dataset from the google drive and just to check is the dataset is loaded we print the first few rows from the dataset.

```
# Loading the dataset
df = pd.read_csv('/content/drive/My Drive/Thesis - Phishing Detection/Datasets/phishing_email.csv')

# Displaying the first few rows
df.head()
```

Figure 4: Loading the Dataset

3.1.5 Keyword Frequency Analysis:

This analysis looks for and counts the instances of particular keywords , such "money," "offer," and "urgent," that are frequently linked to phishing attempts in a dataset of emails. When these keywords are compared to their frequency in phishing and non-phishimg emails, trends that highlight common phishing strategies are revealed. A bar chart is used to illustrate the findings and display the frequency of these keywords in each category.

```python
# @title Frequency of Specific Keywords in Spam vs Non-Spam Emails

import matplotlib.pyplot as plt

keywords = ['money', 'free', 'offer', 'win', 'prize', 'urgent', 'click']

spam_counts = [df[df['label'] == 1]['text_combined'].str.lower().str.count(keyword).sum() for keyword in keywords]
non_spam_counts = [df[df['label'] == 0]['text_combined'].str.lower().str.count(keyword).sum() for keyword in keywords]

# Creating the bar chart
plt.figure(figsize=(10, 6))
plt.bar(keywords, spam_counts, label='Spam', alpha=0.7)
plt.bar(keywords, non_spam_counts, bottom=spam_counts, label='Non-Spam', alpha=0.7)

plt.xlabel('Keywords')
plt.ylabel('Frequency')
plt.title('Frequency of Specific Keywords in Spam vs Non-Spam Emails')
_ = plt.legend()
```

Figure 5: Creating a bar chart

3.1.6 Text Preprocessing:

This part is very important in order to clean the raw text data from the email dataset. All the test is first converted into lower case, then the punctuation marks are removed, and common stopwords such as 'and', 'the', and 'is' are filtered out to reduce the noise. When we make sure that the data going as input to the models is clean it helps improve the overall efficiency.

```python
import string

def preprocess_text(text_combined):
    stop_words = set(stopwords.words('english'))
    text = text_combined.lower()
    text = ''.join([char for char in text if char not in string.punctuation])
    text = ' '.join([word for word in text.split() if word not in stop_words])
    return text

df['text_combined'] = df['text_combined'].apply(preprocess_text)

# Spliting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df['text_combined'], df['label'], test_size=0.2, random_state=42)
```

Figure 6: Preprocessing the text

3.1.7 Data Splitting:

Let's split the datasets into training and testing sets. For this we use `'train_test_split'` from the module `'sklearn.model_selection'`. Using this makes sure that data has been split randomly without any bias.

```python
# Spliting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df['text_combined'], df['label'], test_size=0.2, random_state=42)
```

Figure 7: Splitting the data

3.2 Model Training

In this section we will explain the steps required for embedding generation using the DistilBERT model and training the SVM classifier.

3.2.1 Loading the Tokenizer and Model:

It starts off with loading the pre-trained DistilBERT tokenizer and model.

```
# Loading the saved tokenizer and DistilBERT model
tokenizer = DistilBertTokenizer.from_pretrained('/content/drive/My Drive/Thesis - Phishing Detection/Models/tokenizer')
model = DistilBertModel.from_pretrained('/content/drive/My Drive/Thesis - Phishing Detection/Models/distilbert_model')
```

Figure 8: Loading the tokenizer and model

3.2.2 Setting Up the Device:

Here we are using the GPU as it'll be faster to run the DistilBERT model.

```
# Checking if GPU is available, if yes then connect to it
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
model.to(device)
```

Figure 9: Using the GPU

3.2.3 Generating Embeddings:

Text data are transformed into numerical vectors that represent the context and meaning of the words. Since DistilBERT can not be feed with regular data we first need to tokenized them. The output of this model is then feed to the classifier for traininga nd testing purpose.

```
def get_embeddings(texts, tokenizer, model, device, batch_size=32):
    """Function to get DistilBERT embeddings for a list of texts."""
    embeddings = []
    model.eval()

    with torch.no_grad():
        for i in range(0, len(texts), batch_size):
            batch_texts = texts[i:i + batch_size]
            inputs = tokenizer(batch_texts, return_tensors='pt', truncation=True, padding=True, max_length=512).to(device)
            outputs = model(**inputs)
            batch_embeddings = outputs.last_hidden_state.mean(dim=1).cpu().numpy()
            embeddings.extend(batch_embeddings)

    return np.array(embeddings)

# Getting embeddings for training and testing sets
X_train_embeddings = get_embeddings(X_train.tolist(), tokenizer, model, device)
X_test_embeddings = get_embeddings(X_test.tolist(), tokenizer, model, device)
```

Figure 10: Generating Embeddings

3.2.4 Converting Embeddings:

SVM doesn't take in the embeddings as it is for input, so we need to convert them into real number format and then use for classification.

```
X_train_embeddings = X_train_embeddings.reshape(X_train_embeddings.shape[0], -1)
X_test_embeddings = X_test_embeddings.reshape(X_test_embeddings.shape[0], -1)
```

Figure 11: Converting the embeddings

3.2.5 Training the SVM Classifier:

The numeral real number representation produced in the previous step is then feed to the SVM classifier for the purpose of training. We are using the linear kernel SVM as it helps maintain efficiency.

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Training the SVM classifier
clf = SVC(kernel='linear', random_state=42)
clf.fit(X_train_embeddings, y_train)
```

Figure 12: Training the SVM

3.3 Testing

Once the model is trained it's time to test the model with the unseen data left after the splitting.

3.3.1 Making Predictions:

Now, we predict the labels using the trained SVM model.

```
# Making predictions on the test set
y_pred = clf.predict(X_test_embeddings)
```

Figure 13: Making Prediction

3.3.2 Evaluating the Model:

After the prediction is done we evaluate the model by checking the metrics and confusion matrix.

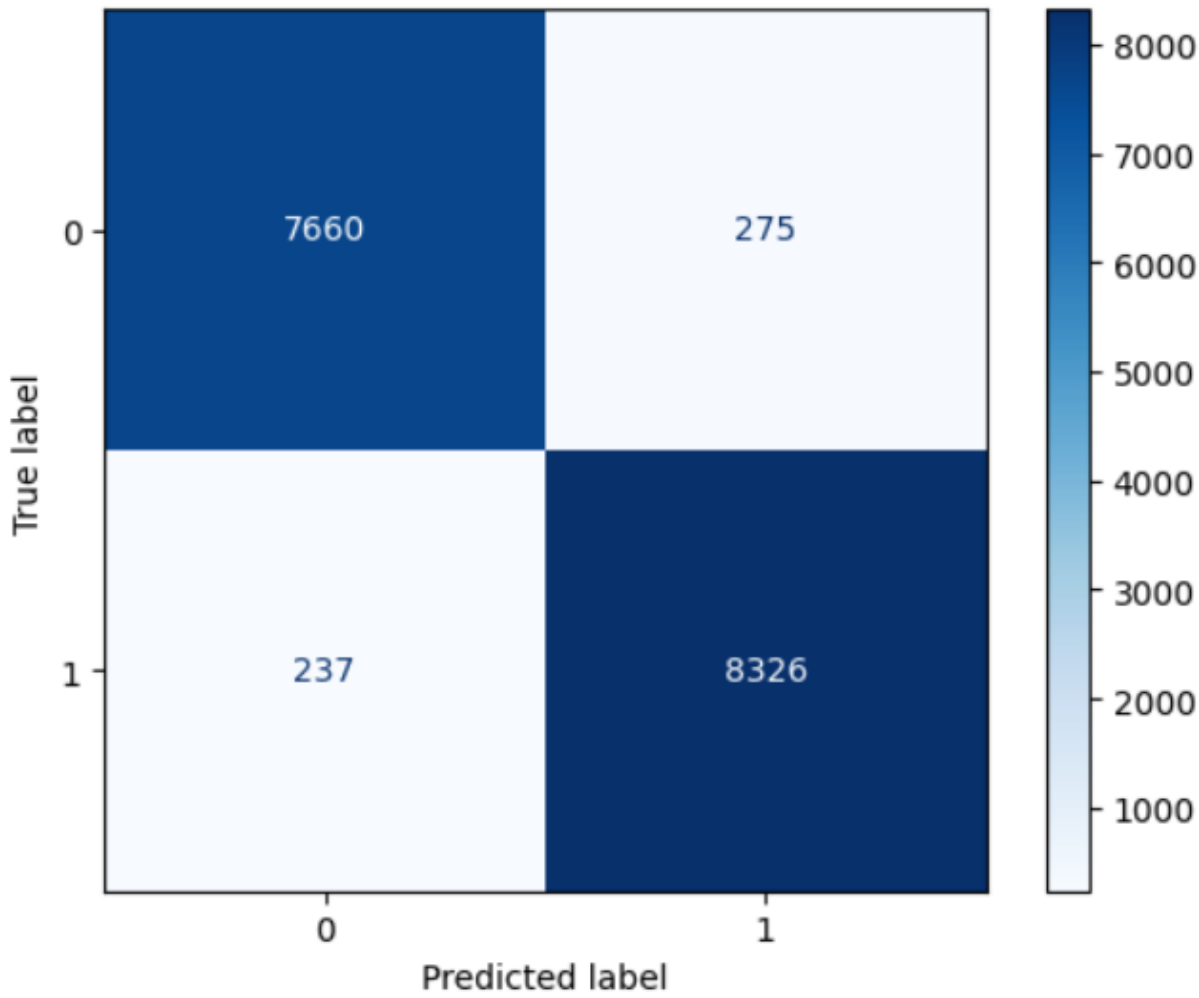

Figure 14: Confusion Matrix

	precision	recall	f1-score	support
0	0.97	0.97	0.97	7935
1	0.97	0.97	0.97	8563
accuracy			0.97	16498
macro avg	0.97	0.97	0.97	16498
weighted avg	0.97	0.97	0.97	16498

Figure 15: Evaluation metrics

3.3.3 Saving and Loading the Model:

As the last step we saved the model using 'joblib' library to be used in future without having to retrain the model.

```
# Saving the model
joblib.dump(clf, '/content/drive/My Drive/Thesis - Phishing Detection/Models/svm_model.pkl')
```

Figure 16: Saving the model