National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Monika Rana
Student ID: x21204497

School of Computing
National College of Ireland

Supervisor: Abubakr Siddig

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Monika Rana.

Student ID: x21204497

Programme: MSc in Data Analytics **Year:** 2022 – 2023

Module: Research Project

Lecturer: Abubakr Siddig

Submission Due Date: 14th August 2023

Project Title: Drift Phenomenon based Email Spam Prediction with LSTM and GRU Approach

Word Count: 468 **Page Count:** 9

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: Monika Rana

Date: 14th August 2023

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

Monika Rana
X21204497

1 Introduction

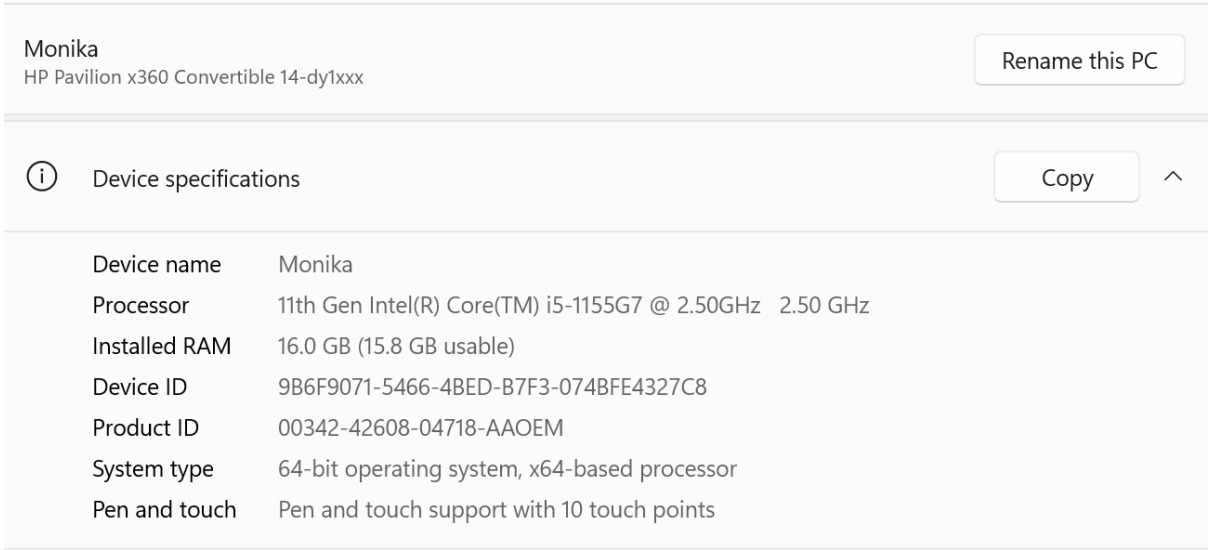
A configuration manual contains comprehensive information on how to configure a system or device. The goal of the manual is to completely describe how to carry out the research study. It also details the machine setup necessary to create and run the models. The procedures involve setting up both the minimal configuration necessary for a project's success and the necessary programmes and packages.

2 Project Files Detail

In this project, data preparation, exploration, modelling, and assessment are all done using Google Collab and Jupyter notebook.

3 System Specification

A system specification provides details the technical specifications and prerequisites of a system. It includes details on the components, operations, layout, and other technical features of the system.



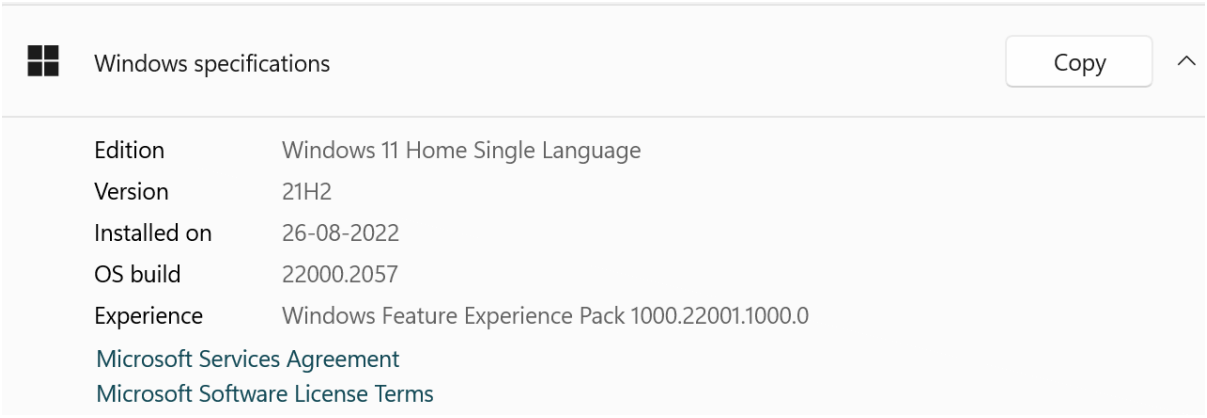


Figure 7: System Specification

4 Software Used

• Google Collab, jupyter Notebook: Used for Exploration and processing, modelling and evaluation

5 Importing Library

```
# Installing all the nessesary Libraries:
!pip install contractions
!pip install seaborn
!pip install matplotlib
!pip install nltk
!pip install keras
!pip install pickle-mixin
```

Figure 7: Procedure to Install all necessary libraries

```
# Importing all the nessessary Libraries:
import numpy as np
import pandas as pd
import string

import re
import collections
import contractions
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('dark_background')
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.simplefilter(action='ignore', category=Warning)
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import keras
```

```
from keras.layers import Dense, Embedding, LSTM, Dropout, GRU
from keras.models import Sequential
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
import pickle
from sklearn.metrics import confusion_matrix
```

Figure 7: Procedure to import all necessary libraries

6 Importing Dataset

```
# Importing the training dataset:
spam_dataframe = pd.read_csv("./dataset_orig_without_300_spam.csv")
display(spam_dataframe[['text','target']])
```

```
drift_300_dataframe = pd.read_csv("./300_drift_spam.csv")
```

```
display(drift_300_dataframe[['text','target']])
```

	text	target
0	From rssfeeds@jmason.org Fri Oct 4 11:02:10 20...	1
1	From rssfeeds@jmason.org Wed Oct 9 10:52:41 20...	1
2	From rssfeeds@jmason.org Sat Oct 5 10:37:26 20...	1
3	From rssfeeds@jmason.org Mon Sep 30 13:43:58 2...	1
4	From rssfeeds@jmason.org Wed Oct 2 11:44:35 20...	1

Figure 7: Procedure to Fetch the dataset in Jupyter Notebook

7 Preprocessing

```
# Removing rows from dataframe, where there is no label avilable or the label is not 0/1:
# This is needed as these rows don't have any valid label.
problem_idx = []
for idx in range(len(spam_dataframe['target'])):

  try:
    a = int(spam_dataframe['target'][idx])
  except:
    print(spam_dataframe['target'][idx])
    problem_idx.append(idx)
    continue

  if int(spam_dataframe['target'][idx]) != 0 and int(spam_dataframe['target'][idx]) != 1:
    problem_idx.append(idx)
print(problem_idx)

print(len(spam_dataframe))
spam_dataframe.drop(problem_idx, axis=0, inplace=True)
print(len(spam_dataframe))
```

```
# Converting all float values in label to int 0/1 values:
# Some labels in the dataset are 1/0, but are present as float values, so we convert them to int values.

label_list = []
for val in spam_dataframe['target']:
  label_list.append(int(val))


spam_dataframe['target'] = label_list

for val in spam_dataframe['target']:
  if str(val) != '0' and str(val) != '1':
    print(str(val))
```

```
# Dropping all the unnessesary columns from the dataset, keeping only the (text, label):
spam_dataframe.drop(spam_dataframe.iloc[:, 2:], inplace=True, axis=1)
```

```
# This function acts as the main preprocessing pipeline:
def Preprocessing(spam_dataframe):

    #(1) Removal of Punctuations:
    spam_dataframe['text'] = spam_dataframe['text'].apply(lambda x: remove_punctuation(x))

    ###############################################################################################
    #(2) Removal of Stop words:

    import nltk
    nltk.download('stopwords')
    from nltk.corpus import stopwords
    stops = set(stopwords.words('english'))
    print(stops)

    STOPWORDS = set(stopwords.words('english'))

    spam_dataframe['text'] = spam_dataframe['text'].apply(lambda x: remove_stepwords(x,STOPWORDS))
###############################################################################################
#(3) Removing Special Character:

spam_dataframe['text'] = spam_dataframe['text'].apply(lambda x: rem_spc_char(x))


###############################################################################################
#(4) Stemming:

from nltk.stem.porter import PorterStemmer
ps = PorterStemmer()

spam_dataframe['stemmed_text'] = spam_dataframe['text'].apply(lambda x: stem_words(x, ps))

###############################################################################################
#(5) Removing URL:

spam_dataframe['text'] = spam_dataframe['text'].apply(lambda x: remove_url(x))
```

```
#########################################################################################################
#(6) Removing html and lowering case:

spam_dataframe['text'] = spam_dataframe['text'].apply(lambda x: remove_html_And_LowerCase(x))


# Returning the Preprocessed dataframe:
return spam_dataframe
```

```
# In this section, we are encoding the labels(0/1) and the Input-texts(String) to a suitable mathematical
# representation, which the Deel Learning models can work with:

#########################################################################################################
# Encoding the Labels:
def LabelEncoding(df):
    lb_enc = LabelEncoder()
    y = lb_enc.fit_transform(df["target"])
    print(type(y))
    print(y)
    return y
```

```
#########################################################################################################

# Encoding the Inputs:
def InputEncoder(df):
    X = df["text"]
    tokenizer = Tokenizer() #initializing the tokenizer
    tokenizer.fit_on_texts(X)# fitting on the text data
    text_to_sequence = tokenizer.texts_to_sequences(X) # creating the numerical sequence

    # Printing the encoding results:
    for i in range(5):
           print("Text               : ",X[i] )
           print("Numerical Sequence : ", text_to_sequence[i])

    return text_to_sequence, tokenizer
#########################################################################################################
```

```
# Preprocessing, Encoding and Padding Pipeline:
spam_dataframe = Preprocessing(spam_dataframe)
y = LabelEncoding(spam_dataframe)
text_to_sequence, tokenizer = InputEncoder(spam_dataframe)
padded_sequence, max_length_sequence = Padding(text_to_sequence)
```

```
# In this section we are performing the padding(pre-padding), to ensure that all the sequences have
# the same length before they are being fed into the models:

# Padding the Sequences:
def Padding(text_to_sequence):
    max_length_sequence = max([len(i) for i in text_to_sequence])
    # finding the length of largest sequence
    padded_sequence = pad_sequences(text_to_sequence, maxlen=max_length_sequence, padding = "pre")
    return padded_sequence, max_length_sequence
```

```
# Preprocessing, Encoding and Padding Pipeline for drift_300_dataset:
drift_300_dataframe = Preprocessing(drift_300_dataframe)
drift_300_y = LabelEncoding(drift_300_dataframe) # Test Labels
drift_300_text_to_sequence, drift_300_tokenizer = InputEncoder(drift_300_dataframe)
drift_300_padded_sequence, drift_300_max_length_sequence = Padding(drift_300_text_to_sequence)
drift_300_test = drift_300_padded_sequence # Test Texts
```

8 Modelling

```
# LSTM Model Creation:

TOT_SIZE = len(tokenizer.word_index)+1
def create_model():

      lstm_model = Sequential()
      lstm_model.add(Embedding(TOT_SIZE, 32, input_length=max_length_sequence))
      lstm_model.add(LSTM(50))
      lstm_model.add(Dropout(0.4))
      lstm_model.add(Dense(20, activation="relu"))
      lstm_model.add(Dropout(0.3))
      lstm_model.add(Dense(1, activation = "sigmoid"))
      return lstm_model

lstm_model = create_model()
lstm_model.compile(loss = "binary_crossentropy", optimizer = "adam", metrics = ["accuracy"])
lstm_model.summary()
```

```
# GRU Model Creation:

TOT_SIZE = len(tokenizer.word_index)+1
def create_model_GRU():
  model = Sequential()
  model.add(Embedding(TOT_SIZE, 50, input_length=max_length_sequence))
  model.add(GRU(50))
  model.add(Dropout(0.4))
  model.add(Dense(20, activation="relu"))
  model.add(Dropout(0.3))
  model.add(Dense(1, activation = "sigmoid"))
  return model

gru_model = create_model_GRU()
gru_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
gru_model.summary()
```

9 Result

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

# Create a sample confusion matrix:
lstm_conf_matrix = confusion_matrix(pred_y, y_test) # True Negatives (TN), False Positives (FP)
                                                    # False Negatives (FN), True Positives (TP)

# Calculate metrics
tn, fp, fn, tp = lstm_conf_matrix[0][0], lstm_conf_matrix[0][1], lstm_conf_matrix[1][0], lstm_conf_matrix[1][1]

lstm_accuracy = (tp + tn) / (tp + tn + fp + fn)
lstm_precision = tp / (tp + fp)
lstm_recall = tp / (tp + fn)
lstm_f1_score = 2 * (lstm_precision * lstm_recall) / (lstm_precision + lstm_recall)

# Following are the metrics on which we will evaluate the model:
print("LSTM Accuracy: ", lstm_accuracy)
print("LSTM Precesion: ", lstm_precision)
print("LSTM Recall: ", lstm_recall)
print("LSTM F1 Score: ", lstm_f1_score)
```

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

# Create a sample confusion matrix:
gru_conf_matrix = confusion_matrix(pred_y, y_test) # True Negatives (TN), False Positives (FP)
                                                   # False Negatives (FN), True Positives (TP)

# Calculate metrics
tn, fp, fn, tp = gru_conf_matrix[0][0], gru_conf_matrix[0][1], gru_conf_matrix[1][0], gru_conf_matrix[1][1]

gru_accuracy = (tp + tn) / (tp + tn + fp + fn)
gru_precesion = tp / (tp + fp)
gru_recall = tp / (tp + fn)
gru_f1_score = 2 * (gru_precesion * gru_recall) / (gru_precesion + gru_recall)

print("GRU Accuracy: ", gru_accuracy)
print("GRU Precesion: ", gru_precesion)
print("GRU Recall: ", gru_recall)
print("GRU F1 Score: ", gru_f1_score)
```

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

# Create a sample confusion matrix:
drift_300_conf_matrix = confusion_matrix(drift_300_pred_y, drift_300_y) # True Negatives (TN), False Positives (FP)
                                                    # False Negatives (FN), True Positives (TP)

# Calculate metrics
tn, fp, fn, tp = drift_300_conf_matrix[0][0], drift_300_conf_matrix[0][1], drift_300_conf_matrix[1][0], drift_300_conf_

drift_300_accuracy = (tp + tn) / (tp + tn + fp + fn)
drift_300_precesion = tp / (tp + fp)
drift_300_recall = tp / (tp + fn)
drift_300_f1_score = 2 * (drift_300_precesion * drift_300_recall) / (drift_300_precesion + drift_300_recall)

print("LSTM - Drift 300 Accuracy: ", drift_300_accuracy)
print("LSTM - Drift 300 Precesion: ", drift_300_precesion)
print("LSTM - Drift 300 Recall: ", drift_300_recall)
print("LSTM - Drift 300 F1 Score: ", drift_300_f1_score)
```

```
# LSTM Model Evaluation on Drift 300 Dataset:
#############################################################################################

drift_300_pred = lstm_model.predict(drift_300_test)

threshold = 0.5
drift_300_pred_y = np.where(drift_300_pred > threshold, 1,0)

print(confusion_matrix(drift_300_pred_y, drift_300_y))

#############################################################################################
```

```
from sklearn.metrics import confusion_matrix, classification_report

# Create a sample confusion matrix:
drift_300_conf_matrix = confusion_matrix(drift_300_pred_y, drift_300_y) # True Negatives (TN), False Positives (FP)
                                                                        # False Negatives (FN), True Positives (TP)

# Calculate metrics
tn, fp, fn, tp = drift_300_conf_matrix[0][0], drift_300_conf_matrix[0][1], drift_300_conf_matrix[1][0], drift_300_conf_

drift_300_accuracy = (tp + tn) / (tp + tn + fp + fn)
drift_300_precesion = tp / (tp + fp)
drift_300_recall = tp / (tp + fn)
drift_300_f1_score = 2 * (drift_300_precesion * drift_300_recall) / (drift_300_precesion + drift_300_recall)

print("GRU - Drift 300 Accuracy: ", drift_300_accuracy)
print("GRU - Drift 300 Precesion: ", drift_300_precesion)
print("GRU - Drift 300 Recall: ", drift_300_recall)
print("GRU - Drift 300 F1 Score: ", drift_300_f1_score)
```

```
# GRU Model Evaluation on Drift 300 Dataset:
#############################################################################################

drift_300_pred = gru_model.predict(drift_300_test)

threshold = 0.5
drift_300_pred_y = np.where(drift_300_pred > threshold, 1,0)

print(confusion_matrix(drift_300_pred_y, drift_300_y))

#############################################################################################
```