School of Computer Science and Engineering

J Component Project Report

Programme : B. Tech (CSE – AI & ML)

Course Title : Information Security Audit and Analysis

Course Code : CSE3501

Slot : F1

Title: Malicious URL & Phishing URL Detection

Team Members:

Subritta Chatterjee | 20BAI1026

Jayanand Jayan | 20BAI1085

Rohita Chakraborty | 20BAI1213

Faculty: Rukmani P

(Assistant Professor ,VIT, Chennai)

Date: 18.11.2022

ABSTRACT

Attackers often try to change one or more components of the URL's structure to deceive users for spreading their malicious URL. Malicious URLs are known as links that adversely affect users. These URLs will redirect users to resources or pages on which attackers can execute codes on users' computers, redirect users to unwanted sites, malicious websites, or other phishing sites, or malware downloads. Malicious URLs can also be hidden in download links that are deemed safe and can spread quickly through file and message sharing in shared networks.

A phishing website is a common social engineering method that mimics trustful uniform resource locators (URLs) and webpages. Through phishing attacks, the phisher targets naïve online users by tricking them into revealing confidential information, with the purpose of using it fraudulently. Main aim of the attacker is to steal banks account credentials. In United States businesses, there is a loss of US$2billion per year because their clients become victims of phishing.

TABLE OF CONTENTS

INTRODUCTION

Uniform Resource Locator (URL) is used to refer to resources on the Internet. The characteristics and two basic components of the URL are presented as: protocol identifier, which indicates what protocol to use, and resource name, which specifies the IP address or the domain name where the resource is located. It can be seen that each URL has a specific structure and format. Attackers often try to change one or more components of the URL's structure to deceive users for spreading their malicious URL. Malicious URLs are known as links that adversely affect users. These URLs will redirect users to resources or pages on which attackers can execute codes on users' computers, redirect users to unwanted sites, malicious websites, or other phishing sites, or malware download. Malicious URLs can also be hidden in download links that are deemed safe and can spread quickly through file and message sharing in shared networks. Some attack techniques that use malicious URLs include: Drive-by Download, Phishing and Social Engineering, and Spam.

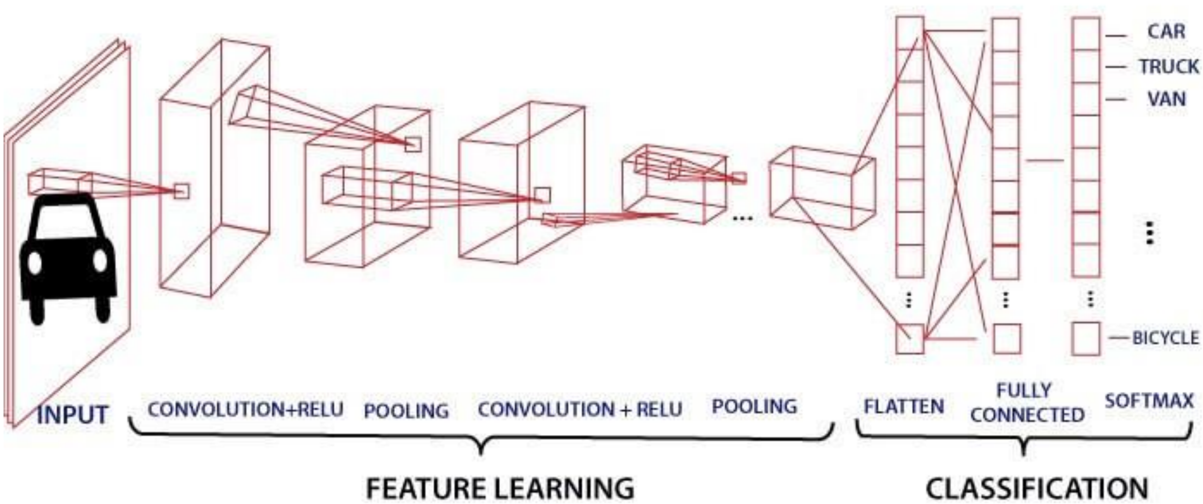
Regarding the problem of detecting malicious URLs, there are two main trends at present as malicious URL detection based on signs or sets of rules, and malicious URL detection based on behavior analysis techniques. The method of detecting malicious URLs based on a set of markers or rules can quickly and accurately detect malicious URLs. However, this method is not capable of detecting new malicious URLs that are not in the set of predefined signs or rules. The method of detecting malicious URLs based on behavior analysis techniques adopt machine learning or deep learning algorithms to classify URLs based on their behaviors.

The Internet has become an indispensable part of our life, However, It also has provided opportunities to anonymously perform malicious activities like Phishing. Phishers try to deceive their victims by social engineering or creating mockup websites to steal information such as account ID, username, password from individuals and organizations. Although many methods have been proposed to detect phishing websites, Phishers have evolved their methods to escape from these detection methods. One of the most successful methods for detecting these malicious activities is Machine Learning. This is because most Phishing attacks have some common characteristics which can be identified by machine learning methods.

PROPOSED METHODOLOGY

Convolution Neural Network

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.



Convolutional Neural Network Design :

The construction of a convolutional neural network is a multi-layered feed-forward neural network, made by assembling many unseen layers on top of each other in a particular order. It is the sequential design that gives permission to CNN to learn hierarchical attributes.

In CNN, some of them are followed by grouping layers and hidden layers are typically convolutional layers followed by activation layers.

The pre-processing needed in a ConvNet is kindred to that of the related pattern of neurons in the human brain and was motivated by the organization of the Visual Cortex.

Layers of CNN model:

- Embedding layer – Keras offers an Embedding layer that can be used for neural networks on text data. It requires that the input data be integer encoded, so that each word is represented by a unique integer. This data preparation step can be performed using the Tokenizer API also provided with Keras. The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset. It is a flexible layer that can be used in a variety of ways, such as:
  - It can be used alone to learn a word embedding that can be saved and used in another model later.
  - It can be used as part of a deep learning model where the embedding is learned along with the model itself.
  - It can be used to load a pre-trained word embedding model, a type of transfer learning.

- Conv1D - This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. If use_bias is True, a bias vector is created and added to the outputs. Finally, if activation is not None, it is applied to the outputs as well.

- Dense - Dense layer is the regular deeply connected neural network layer. It is the most common and frequently used layer. The output shape of the Dense layer will be affected by the number of neurons / units specified in the Dense layer. For example, if the input shape is (8,) and the number of units is 16, then the output shape is (16,).

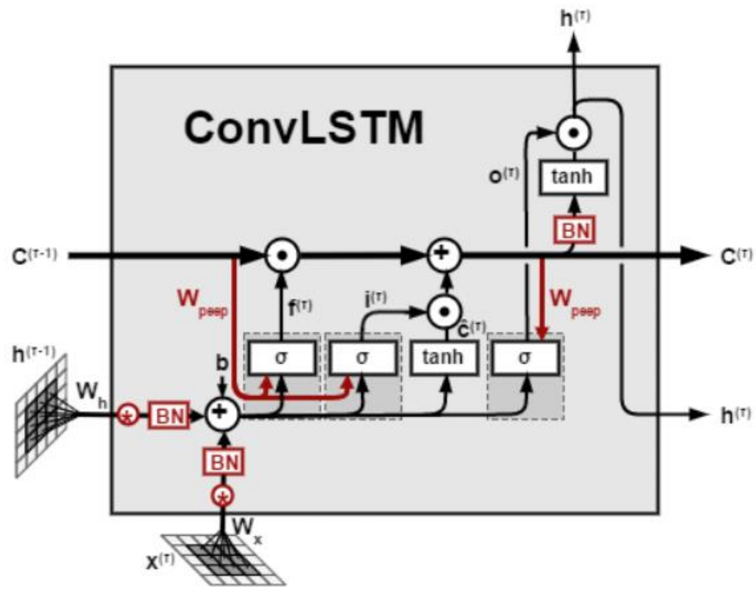
ELU activation function - Exponential Linear Unit or its widely known name ELU is a function that tend to converge cost to zero faster and produce more accurate results. Different to other activation functions, ELU has a extra alpha constant which should be positive number. ELU is very similiar to RELU except negative inputs. They are both in identity function form for non-negative inputs. On the other hand, ELU becomes smooth slowly until its output equal to -α whereas RELU sharply smoothes.

Adam Optimizer - The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing.

Convolution LSTM

Data collected over successive periods of time are characterised as a Time Series. In such cases, an interesting approach is to use a model based on LSTM (Long Short Term Memory), a Recurrent Neural Network architecture. In this kind of architecture, the model passes the previous hidden state to the next step of the sequence. Therefore holding information on previous data the network has seen before and using it to make decisions. In other words, the data order is extremely important.

In our case, sequencial images, one approach is using ConvLSTM layers. It is a Recurrent layer, just like the LSTM, but internal matrix multiplications are exchanged with convolution operations. As a result, the data that flows through the ConvLSTM cells keeps the input dimension (3D in our case) instead of being just a 1D vector with features.



A ConvLSTM cell

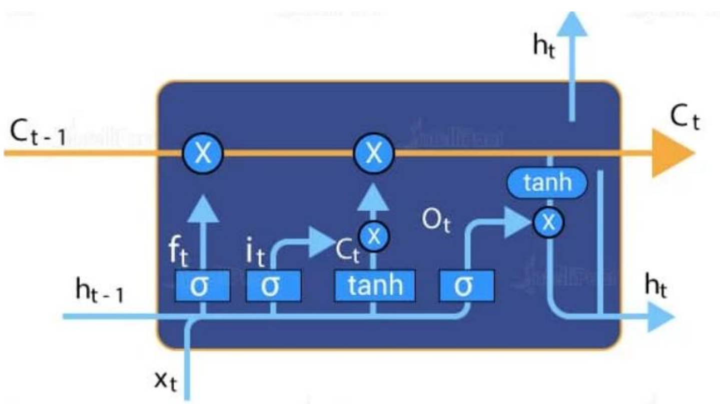
A different approach of a ConvLSTM is a Convolutional-LSTM model, in which the image passes through the convolutions layers and its result is a set flattened to a 1D array with the obtained features. When repeating this process to all images in the time set, the result is a set of features over time, and this is the LSTM layer input.

Our model of hybrid of convolution and LSTM (ConvLSTM) starts off with an Embedding layer which treats text inputs as numerical values because LSTMs and CNNs work on numerical values. Then, we add a 1-Dimensional Convolutional Layer with 256 filters of size 5*5, following same value of padding and an ELU (Exponential Linear Unit) activation function. We apply Max Pooling with a pool size of 4. Then to avoid overfitting of data, we use a dropout with a rate of 0.5. Now, we insert an LSTM layer of output size 32. We again apply dropout with the same rate to avoid overfitting. Finally, we add a fully connected layer of 1 node for output purposes and this layer uses an activation function of Sigmoid nature. We compile the model with a Binary Cross Entropy loss function, an Adam optimizer and Accuracy as metric of assessment. Finally, we fit the training set into the model over 10 epochs with a batch size of 32 per epoch.

Simple LSTM

LSTM stands for long short-term memory networks, used in the field of Deep Learning. It is a variety of recurrent neural networks (RNNs) that are capable of learning long-term dependencies, especially in sequence prediction problems. LSTM has feedback connections, i.e., it is capable of processing the entire sequence of data, apart from single data points such as images.

The central role of an LSTM model is held by a memory cell known as a 'cell state' that maintains its state over time. The cell state is the horizontal line that runs through the top of the below diagram. It can be visualized as a conveyor belt through which information just flows, unchanged.



Information can be added to or removed from the cell state in LSTM and is regulated by gates. These gates optionally let the information flow in and out of the cell. It contains a pointwise multiplication operation and a sigmoid neural net layer that assist the mechanism.

Our LSTM model first contains an Embeddings layer which treats text input as numerical values because LSTMs work on numerical values. Then there is an LSTM layer which has a dimension of 32. Finally, we use a fully connected layer with a sigmoid activation function. We compile the model with a Binary Cross Entropy loss function and an Adam optimizer and use the Accuracy as a metric. We then finally fit the training set to the model over 10 epochs.

IMPLEMENTATION

Malicious URL Detection

In [16]:
```
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split
from string import printable
from keras.utils import pad_sequences
import tensorflow as tf
import json
from tensorflow.keras.models import model_from_json
from pathlib import Path
import tensorflow as tf
from keras.models import Model
from keras import regularizers, Sequential
from keras.layers import Dense, Dropout, Activation, Lambda, Flatten, Input, ELU, LSTM, Embedding, BatchNormalization, Conv1D, co
from keras.preprocessing import sequence
from keras.optimizers import Adam
from keras.utils import np_utils
from keras import backend as K
from sklearn.preprocessing import LabelEncoder
```

In [3]:
```
def read_data():
  df = pd.read_csv("malicious_phish.csv")
  url_int_tokens = [
      [printable.index(x) + 1 for x in url if x in printable] for url in df.iloc[:, 0]
  ]

  max_len = 75
  X = pad_sequences(url_int_tokens, maxlen=max_len)
  le1 = LabelEncoder()

  df['type'] = le1.fit_transform(df['type'])
  target = np.array(df['type'])
  x_train, x_test, target_train, target_test = train_test_split(X, target, test_size=0.25, random_state=42)

  return x_train, x_test, target_train, target_test
```

In [6]:
```
x_train, x_test, target_train, target_test = read_data()
```

Out[6]:
```
array([3, 0, 0, ..., 1, 0, 0])
```

In [5]:
```
max_len = 75
emb_dim = 32
max_vocab_len = 101
lstm_output_size = 32
W_reg = regularizers.l2(1e-4)
epochs_num = 10
batch_size = 32
```

In [ ]:
```
def save_model(model, fileModelJSON, fileWeights):
  if Path(fileModelJSON).is_file():
        os.remove(fileModelJSON)
  json_string = model.to_json()
  with open(fileModelJSON, 'w') as f:
      json.dump(json_string, f)

  if Path(fileWeights).is_file():
      os.remove(fileWeights)
  model.save_weights(fileWeights)


def load_model(fileModelJSON, fileWeights):
    with open(fileModelJSON, 'r') as f:
        model_json = json.load(f)
        model = model_from_json(model_json)

    model.load_weights(fileWeights)
    return model
```

CNN

In [15]:
```
model1 = Sequential()
model1.add(Embedding(input_dim=max_vocab_len, output_dim=emb_dim, input_length=max_len, embeddings_regularizer=W_reg))
model1.add(Conv1D(kernel_size=2, filters=256, padding='same', activation='elu'))
model1.add(Conv1D(kernel_size=3, filters=256, padding='same', activation='elu'))
model1.add(Conv1D(kernel_size=4, filters=256, padding='same', activation='elu'))
model1.add(Conv1D(kernel_size=5, filters=256, padding='same', activation='elu'))
model1.add(Dense(1024))
model1.add(ELU())
model1.add(BatchNormalization())
model1.add(Dense(1024))
model1.add(ELU())
model1.add(BatchNormalization())
model1.add(Dense(1, activation='sigmoid'))

model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
y_train = np.asarray(target_train).astype('float32').reshape((-1,1))
y_test = np.asarray(target_test).astype('float32').reshape((-1, 1))
model1.fit(x_train, y_train, epochs=epochs_num, batch_size=batch_size)
```

```
Epoch 1/10
15263/15263 [==============================] - 294s 19ms/step - loss: -0.4483 - accuracy: 0.4130
Epoch 2/10
15263/15263 [==============================] - 261s 17ms/step - loss: -0.8669 - accuracy: 0.4794
Epoch 3/10
15263/15263 [==============================] - 258s 17ms/step - loss: -1.0284 - accuracy: 0.4931
Epoch 4/10
15263/15263 [==============================] - 261s 17ms/step - loss: -1.1228 - accuracy: 0.4973
Epoch 5/10
15263/15263 [==============================] - 262s 17ms/step - loss: -1.2106 - accuracy: 0.5066
Epoch 6/10
15263/15263 [==============================] - 257s 17ms/step - loss: -1.2736 - accuracy: 0.5120
Epoch 7/10
15263/15263 [==============================] - 248s 16ms/step - loss: -1.3224 - accuracy: 0.5168
Epoch 8/10
15263/15263 [==============================] - 248s 16ms/step - loss: -1.3636 - accuracy: 0.5201
Epoch 9/10
15263/15263 [==============================] - 249s 16ms/step - loss: -1.4059 - accuracy: 0.5235
Epoch 10/10
15263/15263 [==============================] - 247s 16ms/step - loss: -1.4365 - accuracy: 0.5261
```

Out[15]: <keras.callbacks.History at 0x7f7e0a613d10>

In [17]:
```
loss, accuracy = model1.evaluate(x_test, y_test, verbose=0)
print("Final cross validation accuracy =", accuracy)
```

```
Final cross validation accuracy = 0.4807218313217163
```

Convolutional LSTM

In [18]:
```
model2 = Sequential()
model2.add(Embedding(input_dim=max_vocab_len, output_dim=emb_dim, input_length=max_len, embeddings_regularizer=W_reg))
model2.add(Conv1D(kernel_size=5, filters=256, padding='same', activation='elu'))
model2.add(MaxPooling1D(pool_size=4))
model2.add(Dropout(0.5))
model2.add(LSTM(lstm_output_size))
model2.add(Dropout(0.5))
model2.add(Dense(1, activation='sigmoid'))

model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model2.fit(x_train, target_train, epochs=epochs_num, batch_size=batch_size)
```

```
Epoch 1/10
15263/15263 [==============================] - 89s 6ms/step - loss: -47.5746 - accuracy: 0.6221
Epoch 2/10
15263/15263 [==============================] - 85s 6ms/step - loss: -148.1568 - accuracy: 0.6199
Epoch 3/10
15263/15263 [==============================] - 85s 6ms/step - loss: -251.0025 - accuracy: 0.6222
Epoch 4/10
15263/15263 [==============================] - 86s 6ms/step - loss: -354.5500 - accuracy: 0.6232
Epoch 5/10
15263/15263 [==============================] - 86s 6ms/step - loss: -459.5931 - accuracy: 0.6241
Epoch 6/10
15263/15263 [==============================] - 93s 6ms/step - loss: -559.8657 - accuracy: 0.6254
Epoch 7/10
15263/15263 [==============================] - 102s 7ms/step - loss: -667.5125 - accuracy: 0.6270
Epoch 8/10
15263/15263 [==============================] - 98s 6ms/step - loss: -770.2989 - accuracy: 0.6283
Epoch 9/10
15263/15263 [==============================] - 97s 6ms/step - loss: -873.0135 - accuracy: 0.6269
Epoch 10/10
15263/15263 [==============================] - 95s 6ms/step - loss: -971.2258 - accuracy: 0.6212
```

Out[18]: <keras.callbacks.History at 0x7f7e0a0d8f90>

In [19]:
```
loss, accuracy = model2.evaluate(x_test, y_test, verbose=0)
print("Final cross validation accuracy =", accuracy)
```

```
Final cross validation accuracy = 0.6768940687179565
```

Simple LSTM

In [ ]:
```
model3 = Sequential()
model3.add(Embedding(input_dim=max_vocab_len, output_dim=emb_dim, input_length=max_len, embeddings_regularizer=W_reg))
model3.add(LSTM(lstm_output_size))
model3.add(Dense(1, activation='sigmoid'))

print(model3.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 75, 32)            3232

 lstm_2 (LSTM)               (None, 32)                8320

 dense (Dense)               (None, 1)                 33

=================================================================
Total params: 11,585
Trainable params: 11,585
Non-trainable params: 0
_________________________________________________________________
None
```

In [ ]:
```
model3.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [ ]:
```
x_valid, y_valid = x_train[:batch_size], target_train[:batch_size]
x_train2, y_train2 = x_train[batch_size:], target_train[batch_size:]
model3.fit(x_train2, y_train2, validation_data=(x_valid, y_valid), batch_size=batch_size, epochs=epochs_num)
```

```
Epoch 1/10
15262/15262 [==============================] - 95s 6ms/step - loss: -15.7385 - accuracy: 0.5644 - val_loss: -43.6021 - val_accu
racy: 0.6562
Epoch 2/10
15262/15262 [==============================] - 93s 6ms/step - loss: -97.6877 - accuracy: 0.6403 - val_loss: -132.8496 - val_acc
uracy: 0.5938
Epoch 3/10
15262/15262 [==============================] - 94s 6ms/step - loss: -205.5326 - accuracy: 0.6586 - val_loss: -193.5416 - val_ac
curacy: 0.6562
Epoch 4/10
15262/15262 [==============================] - 93s 6ms/step - loss: -309.2408 - accuracy: 0.6837 - val_loss: -298.3667 - val_ac
curacy: 0.6875
Epoch 5/10
15262/15262 [==============================] - 92s 6ms/step - loss: -411.1554 - accuracy: 0.6835 - val_loss: -382.7057 - val_ac
curacy: 0.7812
Epoch 6/10
15262/15262 [==============================] - 93s 6ms/step - loss: -501.7339 - accuracy: 0.6663 - val_loss: -467.4328 - val_ac
curacy: 0.7188
Epoch 7/10
15262/15262 [==============================] - 96s 6ms/step - loss: -622.1306 - accuracy: 0.6710 - val_loss: -487.7188 - val_ac
curacy: 0.7188
Epoch 8/10
15262/15262 [==============================] - 92s 6ms/step - loss: -737.4807 - accuracy: 0.6771 - val_loss: -823.3483 - val_ac
curacy: 0.6562
Epoch 9/10
15262/15262 [==============================] - 113s 7ms/step - loss: -813.1235 - accuracy: 0.6572 - val_loss: -691.8895 - val_a
ccuracy: 0.7188
Epoch 10/10
15262/15262 [==============================] - 101s 7ms/step - loss: -940.8181 - accuracy: 0.6544 - val_loss: -737.4906 - val_a
ccuracy: 0.7188
```

Out[16]: <keras.callbacks.History at 0x7fe3b2d9c0d0>

In [ ]:
```
loss, accuracy = model1.evaluate(x_test, target_test, verbose=0)
print("Final cross validation accuracy =", accuracy)
```

```
Final cross validation accuracy = 0.7038599848747253
```

Phishing URL Detection

In [1]:
```
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split
from string import printable
from keras.utils import pad_sequences
import tensorflow as tf
import json
from tensorflow.keras.models import model_from_json
from pathlib import Path
import tensorflow as tf
from keras.models import Model
from keras import regularizers, Sequential
from keras.layers import Dense, Dropout, Activation, Lambda, Flatten, Input, ELU, LSTM, Embedding, BatchNormalization, Conv1D, cc
from keras.preprocessing import sequence
from keras.optimizers import Adam
from keras.utils import np_utils
from keras import backend as K
from sklearn.preprocessing import LabelEncoder
```

In [5]:
```
def read_data():
  df = pd.read_csv("Phishing_dataset.csv")
  df.drop(df.columns[0], axis=1, inplace=True)
  url_int_tokens = [
      [printable.index(x) + 1 for x in url if x in printable] for url in df.iloc[:, 0]
  ]

  max_len = 75
  X = pad_sequences(url_int_tokens, maxlen=max_len)
  le1 = LabelEncoder()

  df['Label'] = le1.fit_transform(df['Label'])
  target = np.array(df['Label'])
  x_train, x_test, target_train, target_test = train_test_split(X, target, test_size=0.25, random_state=42)

  return x_train, x_test, target_train, target_test
```

In [6]:
```
x_train, x_test, target_train, target_test = read_data()
```

In [8]:
```
max_len = 75
emb_dim = 32
max_vocab_len = 101
lstm_output_size = 32
W_reg = regularizers.l2(1e-4)
epochs_num = 10
batch_size = 32
```

In [9]:
```
def save_model(model, fileModelJSON, fileWeights):
  if Path(fileModelJSON).is_file():
        os.remove(fileModelJSON)
  json_string = model.to_json()
  with open(fileModelJSON, 'w') as f:
      json.dump(json_string, f)

  if Path(fileWeights).is_file():
      os.remove(fileWeights)
  model.save_weights(fileWeights)


def load_model(fileModelJSON, fileWeights):
    with open(fileModelJSON, 'r') as f:
        model_json = json.load(f)
        model = model_from_json(model_json)

    model.load_weights(fileWeights)
    return model
```

CNN

In [10]:
```
model1 = Sequential()
model1.add(Embedding(input_dim=max_vocab_len, output_dim=emb_dim, input_length=max_len, embeddings_regularizer=W_reg))
model1.add(Conv1D(kernel_size=2, filters=256, padding='same', activation='elu'))
model1.add(Conv1D(kernel_size=3, filters=256, padding='same', activation='elu'))
model1.add(Conv1D(kernel_size=4, filters=256, padding='same', activation='elu'))
model1.add(Conv1D(kernel_size=5, filters=256, padding='same', activation='elu'))
model1.add(Dense(1024))
model1.add(ELU())
model1.add(BatchNormalization())
model1.add(Dense(1024))
model1.add(ELU())
model1.add(BatchNormalization())
model1.add(Dense(1, activation='sigmoid'))

model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
y_train = np.asarray(target_train).astype('float32').reshape((-1,1))
y_test = np.asarray(target_test).astype('float32').reshape((-1, 1))
model1.fit(x_train, y_train, epochs=epochs_num, batch_size=batch_size)
```

```
Epoch 1/10
235/235 [==============================] - 184s 773ms/step - loss: 0.7481 - accuracy: 0.5539
Epoch 2/10
235/235 [==============================] - 177s 752ms/step - loss: 0.6616 - accuracy: 0.5636
Epoch 3/10
235/235 [==============================] - 178s 758ms/step - loss: 0.6561 - accuracy: 0.5748
Epoch 4/10
235/235 [==============================] - 190s 808ms/step - loss: 0.6542 - accuracy: 0.5714
Epoch 5/10
235/235 [==============================] - 178s 756ms/step - loss: 0.6524 - accuracy: 0.5804
Epoch 6/10
235/235 [==============================] - 177s 752ms/step - loss: 0.6516 - accuracy: 0.5707
Epoch 7/10
235/235 [==============================] - 177s 753ms/step - loss: 0.6465 - accuracy: 0.5763
Epoch 8/10
235/235 [==============================] - 177s 755ms/step - loss: 0.6444 - accuracy: 0.5747
Epoch 9/10
235/235 [==============================] - 178s 756ms/step - loss: 0.6415 - accuracy: 0.5842
Epoch 10/10
235/235 [==============================] - 177s 754ms/step - loss: 0.6340 - accuracy: 0.5869
```

Out[10]: `<keras.callbacks.History at 0x7fdddf351b90>`

In [11]:
```
loss, accuracy = model1.evaluate(x_test, y_test, verbose=0)
print("Final cross validation accuracy =", accuracy)
```

```
Final cross validation accuracy = 0.6055307388305664
```

Convolutional LSTM

In [12]:
```
model2 = Sequential()
model2.add(Embedding(input_dim=max_vocab_len, output_dim=emb_dim, input_length=max_len, embeddings_regularizer=W_reg))
model2.add(Conv1D(kernel_size=5, filters=256, padding='same', activation='elu'))
model2.add(MaxPooling1D(pool_size=4))
model2.add(Dropout(0.5))
model2.add(LSTM(lstm_output_size))
model2.add(Dropout(0.5))
model2.add(Dense(1, activation='sigmoid'))

model2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model2.fit(x_train, target_train, epochs=epochs_num, batch_size=batch_size)
```

```
Epoch 1/10
235/235 [==============================] - 13s 42ms/step - loss: 0.4720 - accuracy: 0.7788
Epoch 2/10
235/235 [==============================] - 10s 42ms/step - loss: 0.3059 - accuracy: 0.8699
Epoch 3/10
235/235 [==============================] - 11s 45ms/step - loss: 0.1856 - accuracy: 0.9275
Epoch 4/10
235/235 [==============================] - 10s 44ms/step - loss: 0.1343 - accuracy: 0.9523
Epoch 5/10
235/235 [==============================] - 10s 43ms/step - loss: 0.0986 - accuracy: 0.9675
Epoch 6/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0756 - accuracy: 0.9741
Epoch 7/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0591 - accuracy: 0.9800
Epoch 8/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0596 - accuracy: 0.9796
Epoch 9/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0529 - accuracy: 0.9815
Epoch 10/10
235/235 [==============================] - 10s 42ms/step - loss: 0.0451 - accuracy: 0.9856
```

Out[12]: `<keras.callbacks.History at 0x7fdddf06d1d0>`

In [13]:
```
loss, accuracy = model2.evaluate(x_test, y_test, verbose=0)
print("Final cross validation accuracy =", accuracy)
```

```
Final cross validation accuracy = 0.9819999933242798
```

Simple LSTM

In [14]:
```
model3 = Sequential()
model3.add(Embedding(input_dim=max_vocab_len, output_dim=emb_dim, input_length=max_len, embeddings_regularizer=W_reg))
model3.add(LSTM(lstm_output_size))
model3.add(Dense(1, activation='sigmoid'))

print(model3.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 75, 32)            3232

 lstm_1 (LSTM)               (None, 32)                8320

 dense_4 (Dense)             (None, 1)                 33

=================================================================
Total params: 11,585
Trainable params: 11,585
Non-trainable params: 0
_________________________________________________________________
None
```

In [15]:
```
model3.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
x_valid, y_valid = x_train[:batch_size], target_train[:batch_size]
x_train2, y_train2 = x_train[batch_size:], target_train[batch_size:]
model3.fit(x_train2, y_train2, validation_data=(x_valid, y_valid), batch_size=batch_size, epochs=epochs_num)
```

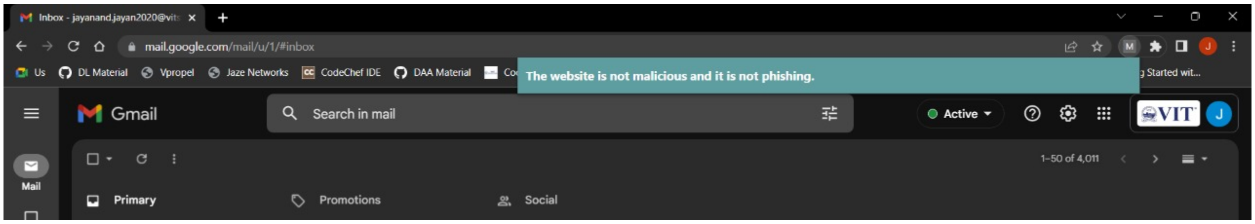
```
Epoch 1/10
234/234 [==============================] - 12s 37ms/step - loss: 0.5001 - accuracy: 0.7491 - val_loss: 0.3636 - val_accuracy:
0.8438
Epoch 2/10
234/234 [==============================] - 9s 39ms/step - loss: 0.4209 - accuracy: 0.7939 - val_loss: 0.3380 - val_accuracy: 0.
8750
Epoch 3/10
234/234 [==============================] - 11s 48ms/step - loss: 0.3957 - accuracy: 0.8148 - val_loss: 0.3029 - val_accuracy:
0.8750
Epoch 4/10
234/234 [==============================] - 8s 34ms/step - loss: 0.3751 - accuracy: 0.8352 - val_loss: 0.3071 - val_accuracy: 0.
9062
Epoch 5/10
234/234 [==============================] - 8s 34ms/step - loss: 0.3503 - accuracy: 0.8531 - val_loss: 0.2348 - val_accuracy: 0.
9375
Epoch 6/10
234/234 [==============================] - 8s 34ms/step - loss: 0.3356 - accuracy: 0.8610 - val_loss: 0.2376 - val_accuracy: 0.
9375
Epoch 7/10
234/234 [==============================] - 8s 34ms/step - loss: 0.3291 - accuracy: 0.8634 - val_loss: 0.2279 - val_accuracy: 0.
9375
Epoch 8/10
234/234 [==============================] - 8s 34ms/step - loss: 0.3131 - accuracy: 0.8725 - val_loss: 0.2245 - val_accuracy: 0.
9375
Epoch 9/10
234/234 [==============================] - 8s 34ms/step - loss: 0.3067 - accuracy: 0.8763 - val_loss: 0.2373 - val_accuracy: 0.
9375
Epoch 10/10
234/234 [==============================] - 8s 34ms/step - loss: 0.2992 - accuracy: 0.8838 - val_loss: 0.2260 - val_accuracy: 0.
9375
```

Out[15]:
```
<keras.callbacks.History at 0x7fddda8ca8d0>
```

In [16]:
```
loss, accuracy = model3.evaluate(x_test, target_test, verbose=0)
print("Final cross validation accuracy =", accuracy)
```

```
Final cross validation accuracy = 0.885200023651123
```

Working of the Chrome Extension



RESULT AND DISCUSSIONS

In this work, we have described how a chrome extension can classify the URLs based upon the given feature set. We use a dataset of more than 6,40,000 URLs. We read this dataset and store it as a Pandas DataFrame. We then split each URL according to the symbols (characters other than the alphabets) so that we can do the analysis on the different parts of the URL (like the base address, query string parameters etc.). For uniformity, we fix the length of all the URLs at a length of 75 characters. Finally, we split the dataset into a training and testing set.

The Future work is to fine tuning the machine learning algorithm that will produce the better result by utilizing the given feature set. Adding to that the open question is how we can handle the huge number of URLs whose features will evolve over time. Certain efforts have to be made in that direction so as to come up with a more robust feature set which can change with respect to the evolving changes.

REFERENCES

- https://www.ijitee.org/wp-content/uploads/papers/v8i4s2/D1S0085028419.pdf
- https://towardsdatascience.com/phishing-domain-detection-with-ml-5be9c99293e5
- https://ieeexplore.ieee.org/document/8494159
- https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0258361