MALICIOUS URL & PHISHING URL DETECTION

INFORMATION SECURITY AUDIT AND ANALYSIS - CSE 3501

TEAM MEMBERS:

20BAI1026 - SUBRITTA CHATTERJEE

20BAI1085 - JAYANAND JAYAN

20BAI1213 - ROHITA CHAKRABORTY

GUIDE NAME - RUKMANI P

Model description

Data Preprocessing

We obtained two different datasets - a dataset of more than 6,40,000 URLs for Malicious URL Detection and another dataset of 10,000 domains for Phishing URL detection. We read the 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.

CNN model

Our CNN model consists of four different 1-Dimensional Convolutional Layers. Each layer uses 256 filters, with a padding method of same value and an activation function of ELU (Exponential Linear Unit). The first convolutional layer uses filters of size 3*3, second uses filters of size 4*4, third uses filters of size 5*5 and the final fourth convolution layer uses filters of size 6*6. We then connect a fully connected layer of 1024 nodes. Then we pass an activation function of ELU and then apply Batch Normalization to the model, and then repeat the process of connecting a fully connected layer, applying the activation function and then applying batch normalization. Finally, one more fully connected layer is added with only 1 node, for the output purposes. This final layer uses an activation function of Sigmoid nature. We compile the model with Adam for optimizers and a loss of Binary Cross Entropy with the metric for assessment used as Accuracy. We finally fit the training set into the model over a total of 10 epochs with a batch size of 32 per epoch.

Convolution LSTM

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

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 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 Screenshots

Malicious URL Detection

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
```

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 [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
```

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
```

Results

Malicious URL Detection

Model	Accuracy
CNN	48%
Convolutional LSTM	67.6%
Simple LSTM	**70.38%**

Phishing URL Detection

Model	Accuracy
CNN	60%
Convolutional LSTM	**98.2%**
Simple LSTM	88.5%

Thank you!