The Great Host/Lexical URL Reputation Bake-off ™

This assignment has us approach classifying URLs based only on the URL and other statistics about it without crawling, a method employed by companies to save time and money.

Requirements

Having the following packages installed is essential to run the script

1. Python 3
2. Keras
3. Tensorflow
4. Numpy
5. Scikit learn

To run the model, type in the following command to the command line:

```
python3 model.py train.json classify.json
```

Solutions

I played around with multiple machine learning algorithms. I currently get the best results with a 3 layer neural network using these six features 'alexa_rank', 'default_port', 'domain_age_days', 'port', 'num_domain_tokens', 'num_path_tokens' and 'malicious_url' for the label. I chose these parameters as analyzing the dataset suggested strong correlation between a malicious URL and the first four features. I added the last two features as these were easy to add to the classifiers and seemed to be important features. These features all in all were sufficient enough for the neural network to get a 100% training, validation and testing accuracy on from the labeled data set. Apart from using a neural network, I used Logistic Regression and Naive Bayes for seeing how these algorithms perform on the same task. Logistic Regression gave a 99% accuracy and Naive Bayes gave a slightly lower accuracy at 96% with some modifications to the dataset (removing negative values for domain age in the dataset). I created a test set which was the last 200 elements in the labeled dataset. The neural network had a validation test set which was 25% of the dataset - the last 200 elements. Running the neural network for a hundred epochs (few seconds) leads the neural network to convergence in most cases and if not close to 98%. Running the network for 200 epochs has always led the algorithm to reach convergence in my test cases. Logistic Regression had a test set accuracy of 99.5% and naive bayes with a test accuracy of 96%. I did not integrate any features that would require me to use sentences as that would require to form a mapping (not the hard part) and then adding an embedding, LSTM layer and concatenating the layers. Concatenating the layers for my current neural network and the LSTM layer would change the model type from a sequential to a non sequential which is generally more time consuming in my experience and with the

perfect accuracy being obtained with the current configuration did not require me to further pursue the idea. One of the errors I suspect for the Naive Bayes implementation is that the port number category was not split into multiple columns to account for them not being ordinal categorical variables. This definitely affects the probability calculation and may make the model behave in an erroneous manner. Using a Gaussian Naive Bayes distribution accounts for the frequency values for the alexa rank. I also had to set None values for the Alexa rank to the max alexa rank plus one for naive bayes and zero for others. Zero gives a perfect accuracy for the neural network and a high one for logistic regression but a fifty six percent accuracy for naive bayes. One of the possible reasons is that it considers the frequency to be 0 for not having an alexa rank (which is true but does not focus on the attribute) and devalues the importance of the feature which the other models learn. Another feature engineering done was setting the domain days to 0 for negative values for domain days in the dataset which increased the naive bayes accuracy by 8%. The most obvious improvement to improve accuracy would be to add another attribute to check if domains return an IP Address and encoding file extensions returned by web pages. Adding the tokenized URL names would also help but will be challenging to think about ways for inserting varying size URLs (LSTMs with a fixed size, 0's for URL positions that finish early, Columns with the same logic (URL_col_1, URL_col_2, URL_col_3, ….) and 0's for short domains that don't fill the number of attributes). The final model for testing the classification accuracy uses the neural network as it has the best performance out of all the ones tested.

Snippets for the accuracy:

Logistic Regression:

Training Accuracy:

```
In [114]: reg.score(data[:-200], labels[:-200])
          # Note this is the test set, the last 200 elements on which I did not train the dataset
          # reg.score(data[-200:], labels[-200:])

Out[114]: 0.9889258028792912
```

Testing Accuracy:

```
In [107]: # reg.score(data[:-200], labels[:-200])
          # Note this is the test set, the last 200 elements on which I did not train the dataset
          reg.score(data[-200:], labels[-200:])

Out[107]: 0.995
```

Neural Network:

Training and Validation Accuracy:

```
1354/1354 [==============================] - 0s 87us/step - loss: 0.0160 - acc: 1.0000 - val_loss: 0.0176 - val_acc: 1.0000
Epoch 100/100
1354/1354 [==============================] - 0s 83us/step - loss: 0.0156 - acc: 1.0000 - val_loss: 0.0172 - val_acc: 1.0000
```

Testing Accuracy:

In [67]:
```
# Cell used for calculating the test set score
count = 0
ans = model.predict_classes(data[-200:])
for i in range(len(ans)):
    if ans[i] == labels[len(labels)-200+i]:
        count += 1
print(count/len(ans))
```

```
1.0
```

Gaussian Naive Bayes:

Training Accuracy:

In [96]:
```
# gnb.score(data[:-200], labels[:-200])
gnb.score(data[-200:], labels[-200:])
```

Out[96]:
```
0.96
```

Testing Accuracy:

In [115]:
```
gnb.score(data[:-200], labels[:-200])
# gnb.score(data[-200:], labels[-200:])
```

Out[115]:
```
0.9689922480620154
```

Code Snippets

Feature Engineering:

In [203]:
```
max_alexa_rank = 0
for i in range(len(corpus)):
    max_alexa_rank = max(int(corpus[i]['alexa_rank']) if corpus[i]['alexa_rank'] is not None else 0, x)
max_alexa_rank += 1
```

In [204]:
```
import numpy as np
import sys

data = []
labels = []
for i in range(len(corpus)):
    data.append([0 if corpus[i]['alexa_rank'] == None else int(corpus[i]['alexa_rank']), int(corpus[i]['default_port']),
                 int(corpus[i]['domain_age_days']), int(corpus[i]['port']), int(corpus[i]['num_domain_tokens']),
                 int(corpus[i]['num_path_tokens'])])
    labels.append(int(corpus[i]['malicious_url']))
data = np.asarray(data)
labels = np.asarray(labels)
```

In [217]:
```
for i in range(len(data)):
    for j in range(len(data[i])):
        if (data[i][j] < 0):
#             print(i)
            data[i][j] = 0
```

Logistic Regression:

In [205]:
```
import sklearn
from sklearn.linear_model import LogisticRegression

reg = LogisticRegression().fit(data[:-200], labels[:-200])
```

In [206]:
```
print(reg.score(data[:-200], labels[:-200])
# Note this is the test set, the last 200 elements on which I did not train the dataset
# reg.score(data[-200:], labels[-200:])
```

Out[206]: 0.9889258028792912

Neural Network:

In [208]:
```
from keras import regularizers
from keras.models import Sequential
from keras.layers import Dense, BatchNormalization
from keras.wrappers.scikit_learn import KerasRegressor
```

In [209]:
```
# A simple four layer neural network that again only trains on all examples but the last 200
model = Sequential()
model.add(Dense(100, input_shape=(6,), kernel_initializer='normal', activation='relu'))
model.add(Dense(200, kernel_initializer='normal', activation='relu'))
model.add(Dense(100, kernel_initializer='normal', activation='relu',kernel_regularizer=regularizers.l2(0.01)))
# model.add(BatchNormalization( momentum=0.99, epsilon=0.001))
model.add(Dense(2, activation='softmax', kernel_initializer='normal'))
# Compile model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['acc'])
print(model.summary())
```

Model Summary for neural Network:

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_44 (Dense)             (None, 100)               700
_________________________________________________________________
dense_45 (Dense)             (None, 200)               20200
_________________________________________________________________
dense_46 (Dense)             (None, 100)               20100
_________________________________________________________________
dense_47 (Dense)             (None, 2)                 202
=================================================================
Total params: 41,202
Trainable params: 41,202
Non-trainable params: 0
_________________________________________________________________
```

Naive Bayes on next page

Naive Bayes:

In [224]:

```
# A naive Bayes model for the same dataset
# Note the model performs poorly if the alexa rank is set to 0 for examples without an alexa rank
# Also the naive bayes approach here is flawed as this model requires a frequency table (columns denoting port numbers) instead o
from sklearn.naive_bayes import GaussianNB, MultinomialNB

for i in range(len(data)):
    data[i][0] = max_alexa_rank if data[i][0] == 0 else data[i][0]

gnb = GaussianNB()
y_pred = gnb.fit(data[:-200], labels[:-200]).predict(data[-200:])
```