I was given the task of creating a model that could classify whether a given URL is malicious or not.

Looking for data to train on, I came across [this dataset](https://www.kaggle.com/sid321axn/malicious-urls-dataset), which seemed big enough to get solid results. The data is distributed in the following manner:  
benign 428103

defacement 96457

phishing 94111

malware 32520

This distribution is a bit skewed, but I decided not to go into that at the moment.

Upon my research I’ve learned that the “traditional” way of doing so is using a lot of preprocessing in order to extract many lexical features from the URL, such as counting the number of special characters, the length of the URL, the length of different portions of the URL, such as the top-level domain, etc.

After extracting features, I have tested their correlation with the target Y value:  
A screenshot of a computer

Description automatically generated with medium confidence

We can see the www-count, Top-Level Domain length and the HTTP count are the most correlative to Y.

I have chosen XGBoost and Random Forest as algorithms wo provide with a baseline performance and given the extracted features they have shown the following result.

Random Forest:  
 precision recall f1-score support

0 0.97 0.98 0.98 85621

1 0.96 0.95 0.95 44618

accuracy 0.97 130239

macro avg 0.97 0.96 0.97 130239

weighted avg 0.97 0.97 0.97 130239

accuracy: 0.969

XGBoost:  
precision recall f1-score support

0 0.95 0.98 0.97 85621

1 0.96 0.91 0.93 44618

accuracy 0.96 130239

macro avg 0.96 0.95 0.95 130239

weighted avg 0.96 0.96 0.96 130239

accuracy: 0.956

In an analysis of the most important features for each model, we saw that the same features that have the highest correlation are the most important ones.

After a baseline result has been set, I tried a basic fully connected deep neural network, with the same lexical features used. It displayed inferior performance compared to the previous models.

After that, I’ve made s quick research of more advanced architectures for URL classification, and found a research paper name [URLNet](https://arxiv.org/pdf/1802.03162.pdf), which suggests using convolutional layers over embedding representation of characters and words within the URL.

Diagram

Description automatically generatedI’ve implemented a simplistic version of the architecture suggested in this paper, which relies only on embedding of characters:

This network has significantly more parameters, but is presenting superior performance:  
 precision recall f1-score support

0 0.99 0.99 0.99 85621

1 0.98 0.98 0.98 44618

accuracy 0.98 130239

macro avg 0.98 0.98 0.98 130239

weighted avg 0.98 0.98 0.98 130239

accuracy: 0.983

Since Convolutional layers take advantage of spatial information, I thought RNN cells like LSTM could draw even more information from historical data and may perhaps outperform the previous architecture. This, unfortunately, was not the case. The model presented similar performance, while taking a lot longer to both train and predict.

I’ve used the given data from phishtank as my test data, and the CNN model showed the following:  
 precision recall f1-score support

0 0.00 0.00 0.00 0

1 1.00 0.98 0.99 13116

accuracy 0.98 13116

macro avg 0.50 0.49 0.50 13116

weighted avg 1.00 0.98 0.99 13116

accuracy: 0.985

The entire test set is composed of malicious (as opposed to the training set), but still the model presented a similar performance.

Future work:

1. Further implement URLNet architecture (use words embeddings as well).
2. Incorporate the lexical extracted features into a model along with the CNN over embeddings
3. Stack XGBoost, Random Forest and the CNN model.
4. Try using XGBoost over the embbedings, in connjunction with the existing lexical features.