Detecting Malicious URLs Using Machine Learning: Documentation Report

Authors - Group 6

Yuxuan Liu	Y.Liu188@newcastle.ac.uk
Zhijie Huang	Z.Huang33@newcastle.ac.uk
Arman Jalilian	A.Jalilian@newcastle.ac.uk
Huihao Yang	H.Yang27@newcastle.ac.uk
Muhammad Moaz Adnan	M.M.Adnan2@newcastle.ac.uk
Joseph Straw	J.Straw2@newcastle.ac.uk

Detecting Malicious URLs using Machine Learning

Abstract

In this report, we have documented our efforts towards incorporating Machine Learning for detecting malicious URLs. We begin with a brief description of the problem statement and its challenges. We then go over our data acquisition and feature engineering methods for training and testing the models. Finally, we conclude with a comparison between our chosen classifiers while also discussing their learning parameters.

1 Introduction

The Uniform Resource Locator (URL) is an integral part of the Internet. It is the address that allows users to browse a plethora of interconnected content with just a click. However, this convenience can be easily exploited by bad actors as an attack vector for their criminal pursuits, which can be anything from spamming to malware. Given the fickle nature of URLs, traditional detection mechanisms such as blacklists are frequently outstripped by the number of newer entries due to their over-reliance on supervised feedback. Therefore, in order to meet the time-sensitive demands of identification, we turn to Machine Learning for a set of algorithms which can grow and improve from experience.

1.1 The Challenge

Machine Learning, typically, has the following steps for integration: feature engineering, model selection, and model training/testing. For us, feature engineering proved to be the most difficult part. We scoured the internet for resources about what constituted as an important attribute. To that end, we found two papers ([1], [2]) discussing lexical and web-based analysis of URLs for feature extraction. We started with 60 features which were deemed effective for detecting armed URLs; however, that number quickly dwindled as our models failed to establish any relevance between some of the features and our dataset. We observed that attributes were only applicable if the collected data had enough URLs which exhibited them to create a meaningful boundary, regardless of their merit in literature.

1.2 Model Choices

For URL detection, the problem of discovery can be boiled down to whether one is malicious or benign. From our literature review, we identified four machine learning models for this type of binary classification: (1) Random forest, (2) Decision Tree, (3) Logistic Regression, and (4) Multilayer Perceptron. The first three were implemented using Sklearn, while the last one was constructed via TenserFlow. Each model was then evaluated according to 5 metrics with an emphasis on the number of false negatives:

Metric	Description
False Negatives	Number of dangerous URLs that were mislabeled
Precision	How often the model's positive prediction is correct
Recall	Probability of detection (sensitivity)
F-Score	The harmonic mean of precision and recall
Accuracy	Percentage of correct predictions

2 Methods

2.1 Data Collection

In order for a model to work well with a wide range of URLs, it needs to be trained on a balanced and diverse dataset:

Balanced – Refers to the quantity of representation for both groups, i.e., an equal presence of malicious and benign URLs.

Diversity - Is the breadth of data available for training. The dataset should be able to provide a comprehensive range of URL formations.

The following table gives examples for both types:

Type	Diversity Examples
Benign	www.google.com https://www.google.com.pk/?client=safari https://en.wikipedia.org/wiki https://www.ncl.ac.uk
Malicious	https://smilesvoegol.servebbs.org/voegol.php https://mmilobin/webscr/cmd=_registration-ru https://www.ita-crick.com/

While searching for pre-labelled lists, we noticed that many of them only used the domain name to represent benign URLs, whereas there was no such bias with their malicious counterparts. As a result, early testing showed that our models would frequently misclassify benign URLs if they had a path component. So to get around this, we leveraged three sources to create our training data:

The Aalto University Dataset (2014) [3] – Contains 96,018 samples of malicious and benign URLs. Although much older than the other sources, it has by far the most disparate entries for the benign category. It is also well-balanced with a 50-50 split of either type.

Openphish Dataset - A regularly-updated database of malicious URLs. We use its free-tier feed to retrieve about 2200 samples (the maximum).

Majestic Million Dataset – A list of 1 million benign URLs which can be downloaded as a csv file. We only use 10,000 samples from this list.

The majestic million and openphish entries makeup the dynamic portion of our data. Their inclusion exposes the models to newer forms of URLs which might not be present in the older source. And by keeping their quantity to 12% of the Aalto dataset, we're able to avail the latter's variety without affecting its balance. As for testing, we used openphish after all previous samples had been replaced (about 1 day), and any 10,000 values from the majestic million that were beyond what was used in the training set.

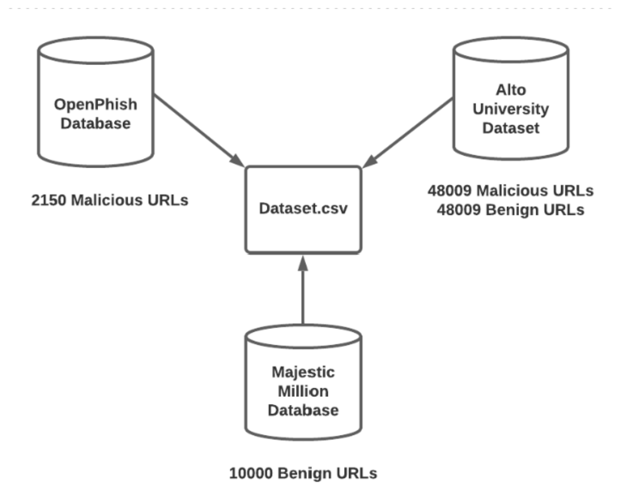


Figure 1: Training Data

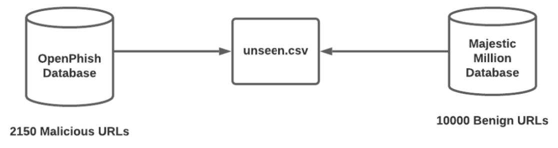


Figure 2: Testing Data

2.2 Data Visualisation

To begin this project, we first had to analyse the collected data and understand which features are most correlated with malicious and benign URLs. This would also give us a better insight into which features are more suitable to use in the model before having to run the model.

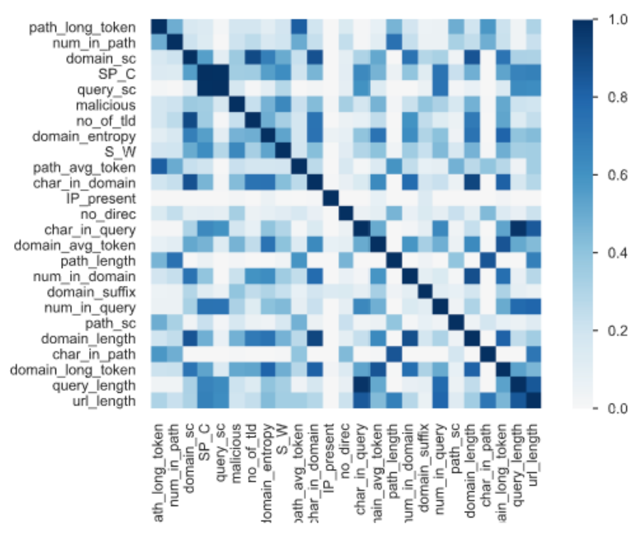


Figure 3: Correlation between features using Phi_K coefficient correlation

Using Phi co-efficient correlation, we can see certain features that are either correlated heavily or not correlated at all with the malicious feature. We chose the Phi correlation as it is proven to work consistently between categorical, ordinal and interval variables [4] which the dataset initially had a variation of. Looking at this heat map, we can also create new features based strongly correlated ones. For example, seeing that special characters (SP_C) had a considerable amount of correlation to malicious URLs. From this finding, we took it upon ourselves to create a new feature seeking for a variety of special elements such as words. This led us to create said feature (S_W) and this had the most correlation with malicious. From this, we can understand that a large range of malicious URLs consist of special words such as "login" and "password". We also established that the IP present feature had little to no correlation with malicious URLs which gave us the notion of dropping said feature before any modelling took place.

2.3 Data Wrangling

Data wrangling is the pre-processing of a dataset before it is it used for any machine learning. In this section, we will talk about how the raw data was cleaned and manipulated to suit the models that we would like to use it on.

We used Python's 'urllib' toolset to tokenize the target addresses in preparation for feature extraction. At a minimum, it expected URLs to have a protocol section in order to deconstruct them properly.

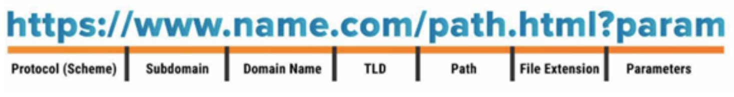
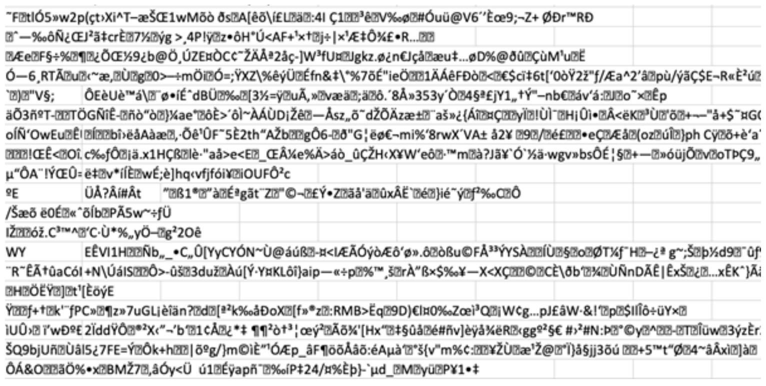


Figure 4: URL structure [5]

A lot of our data values did not come with the scheme. There were some which were so malformed that the tokenizer would throw 'invalid URL' exceptions (refer to figure – 4 for the correct format). Since we were deriving our own attributes from scratch, we could ignore any pre-computed feature columns, and therefore any null values, that came with the data. To tackle the first issue, we added a code snippet which would prepend an http protocol string if one was not provided, even for an empty literal. For the second problem, we wrote a small script which would iterate over the URL column and remove the offending values. We encapsulated this code in a try-except block to avoid crashing the program every time an exception occurred.



```
˜F�tlÓ5»w2p(çt›Xi^T–æŠŒ1wMõò ðs�A[êõ\í£L�ä�:4l Ç1��³ê�V%‰ø�#Óuü@V6´´Èœ9;¬Z+ ØÐr™RÐ
�ˆ—%‰ôÑ¿ŒJ²ã‡crÈ�7½�ýg >‚4P!ÿ�z•ôH"Ú<AF+¹×†�j÷|×¹Æ‡Ô¾£•R…��
�Æe�F§÷%�¶�¿ÕŒ½9¿b@Ö¸ÚZE¤ÒC¢˜ŽÄÅª2âç-]W³fU¤�Jgkz.ø¿n€Jçå�æu‡…øD%@ðû�ÇùM¹u�Ë
Ó—6¸RTÃ�u�‹~æ‚�Ù�g�0>—÷mÖi�Ó=;ŸXZ\%êýÜ�Éfn&‡\°%7õÉ"ieÖ��1ÄÁêFÐò�<�€$cï‡6t[‘0òŸ2ž"f/Æa^2’â�pù/ýãÇ$E¬R«È²ú�
`�)�"V§;      ÔEèUè™á\�¨ø•íÉˆdBÜ�%‰�[3½=ÿ�uÃ‚»�væä�;ä�ô.´8Å»353y´Ò�4§ª£jY1„†Ý"–nb€�áv‘á:�J�o˜×�Êp
äÕ3ñºT-��TÖGÑîÊ-�ñò“ò�}¼ae°�ôÈ>´ôÌ~ÀÁÙD¡Žê�—Åsz„õ˜dŽÕÄzæ±�¨aš»¿{Áî�¤Ç��yÏ�!ÙÌ¯�H¡Ûì•�Â<ëK�³Ù�'õ�+-–"å+$˜¤G
oÍÑ‘OwEu�Ê!�Í��bî›ëåAàæ�‚·Õê¹ÛF˜5È2th“AŽb��gÔ6-�ð"G¦ëø€¬mi%‘8rwX´VA± å2¥ �9�/�é£��•eÇ�Æå�(oz�úÎ�}ph Cÿ�õ+è‘a
���!ŒÊ<�Oî. c‰ƒÔ�¡ä.x1HÇß�lè·"aå>e<E�_ŒÂ¼e%Ä>áò_ûÇŽH‹X¥W‘eô�·™m�à?Jã¥`Ó`½ä·wgv»bsÔÉ¦§�+—�»óüjÕ�v�oTÞÇ9„
µ"ÔA¨!ÝŒÛ= ë‡�v*íÍÈ�wÉ;è]hq‹vfjfói¥�iOUFÔ²c
ºE            ÜÅ?Âí#Ât      ”�ß1®�"à�Éªgãt¨Z�"©¬�£Ý•Z�ãå'ä�ûxÂË`�é�}ié˜ý�f²‰C�Ô
/Šæõ ë0É�«ˆõÍb�PÃ5w~÷fÜ
IŽ��óž.C³™^�‘C·Ù*%„yÖ–�g²2Oê
WY            EÊVl1H��Ñb„_•C„Û[YyCYÓN~Ù@áúß�-¤<IÆÃÓýòÆô‘ø».ô�òßu©FÅ³³ÝYSÀ��ÍÙ�§�o�ØT¼ƒ¯H�–¿ª g~;Š�þ½d9�¨ûf
¨R˜ÊÃ†ûaCól +N\ÚáIS��Ô>-ûš�3dužÀú[Ý·Y¤KLôî}aip—«÷p�%™‚š�rÀ"ß×$%‰¥—X<XÇ��©�CÈ\ðb‘�¾�ÙÑnDÃÊ|ÊxŠ�¿�…xÊK^}Ã
�H�ÖËŸ�]�t¹[ÈöýE
Ÿ��f+†�k'¨ƒPC»�¶z»7uGLj¡èîän?�d�[ª²k%‰åÐoX�[f»®z�:RMB>Ëq�9D)€l¤0%‰Zœì³Q�¡W¢g…pJ£âW·&!‘�p�$IlÎô÷üY×�
ìUÛ›� ï'wÐºE 2ÏddŸÔ�®²X‹"¬‘b‘�1¢Å�¿*‡ ¶¶²ò†³¦œý²�Ãõ¾'[Hx"�‡§ûå�é#ñv]èÿå¾ëR�‹ggº²§€ #›²#N:Þ�°©y�^��-�T�Îûw�3ýzÈr
ŠQ9bjUñ�Ùâl5¿7FE=Ý�Ôk+h��|õºg/}m©iÈ"¹ÓÆp_âF¶öõÅâõ:éAµà‘�°š{v"m%¢:��¥ŽÙ�æ¹Ž@�°Ï}å§jj3õú ��+5™t"Ø�4~âÂxì�]à�
ÔÁ&O��ãÖ%•x�BMŽ7�‚âÓy<Ü  ú1�Éÿapñ¨�%‰íP‡24/¤%Èþ}-`µd_�M�yü�P¥1•‡
```

Figure 5: Corrupted URLs in the Aalto dataset

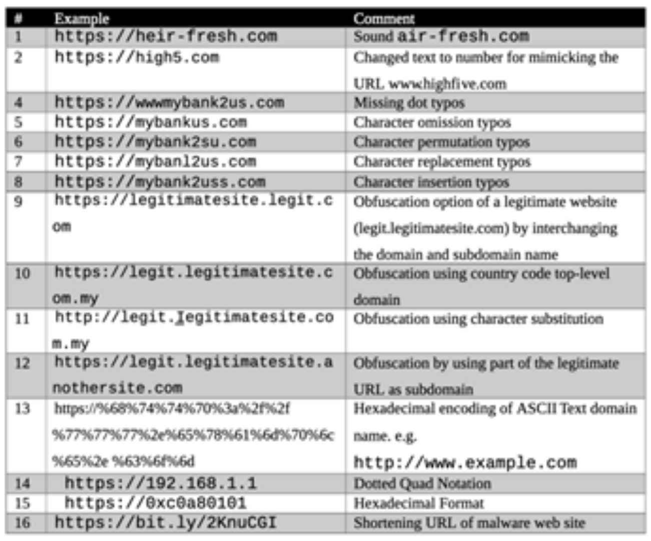
2.4 Feature Engineering

Feature engineering is the process of converting raw data into meaningful traits that better represent the problem to the models, leading to an improved accuracy on unseen values [6]. We opted for lexical and web-based analysis to create our features.

2.4.1 Lexical features

The fastest way of acquiring discernable attributes is from the text itself. Attackers use different obfuscation techniques to trick users by mimicking benign URLs. In doing so, they leave behind traces of randomness, unusual character placements, and many other anomalies. We consulted Mamun's paper [1] as a starting point for our lexical features, and found the following to be the most relevant:

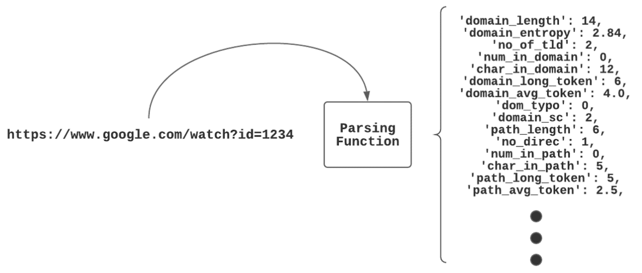
Feature	**Description**
Entropy	Provides the randomness associated with a string; a reliable way of detecting the use of Domain Generation Algorithms.
URL Length	Malicious URLs tend to be longer.
Number of Top Level Domains	Malicious URLs employ multiple TLDs.
Number of Characters	Character usage in different parts of a URL.
Number of Digits	Number usage in different parts of a URL.
Average Token Length	Malicious URLs have a higher average word size.
Number of Special Characters	Malicious URLs incorporate a multitude of Special Characters for obfuscation.
Special Words	Malicious URLs favor using words like login or password.



#	Example	Comment
1	https://heir-fresh.com	Sound air-fresh.com
2	https://high5.com	Changed text to number for mimicking the URL www.highfive.com
4	https://wwwmybank2us.com	Missing dot typos
5	https://mybankus.com	Character omission typos
6	https://mybank2su.com	Character permutation typos
7	https://mybanl2us.com	Character replacement typos
8	https://mybank2uss.com	Character insertion typos
9	https://legitimatesite.legit.com	Obfuscation option of a legitimate website (legit.legitimatesite.com) by interchanging the domain and subdomain name
10	https://legit.legitimatesite.com.my	Obfuscation using country code top-level domain
11	http://legit.Iegitimatesite.com.my	Obfuscation using character substitution
12	https://legit.legitimatesite.anothersite.com	Obfuscation by using part of the legitimate URL as subdomain
13	https://%68%74%74%70%3a%2f%2f%77%77%77%2e%65%78%61%6d%70%6c%65%2e %63%6f%6d	Hexadecimal encoding of ASCII Text domain name. e.g. http://www.example.com
14	https://192.168.1.1	Dotted Quad Notation
15	https://0xc0a80101	Hexadecimal Format
16	https://bit.ly/2KnuCGI	Shortening URL of malware web site

Figure 6: Obfuscation techniques [7]

Our parsing program leveraged Python's 'urllib' library to separate the resource addresses into three parts: the domain name, the path, and the query. Each component was analyzed for a subset of the aforementioned features based on obfuscation patterns. This was done because our models ranked some attributes higher when computed for a particular section, such as the domain's entropy, the path's special characters, etc. We used this information to remove redundant traits and fine-tune our feature profile to match our dataset.



```
'domain_length': 14,
'domain_entropy': 2.84,
'no_of_tld': 2,
'num_in_domain': 0,
'char_in_domain': 12,
'domain_long_token': 6,
'domain_avg_token': 4.0,
'dom_typo': 0,
'domain_sc': 2,
'path_length': 6,
'no_direc': 1,
'num_in_path': 0,
'char_in_path': 5,
'path_long_token': 5,
'path_avg_token': 2.5,
```

Figure 7: Parsing function

2.4.2 Web-based features

For a malicious website, in addition to the URL part, response header and HTML file are very important components. Many attack methods are based on these two parts. Common methods include the control of cookies, the use of iframe tags to embed illegal pages, the use of onerror events to conduct cross-site scripting attacks, or phishing attacks that pretend to be well-known websites, and so on. The extraction and processing of this information can logically obtain some features that are very relevant to this problem, and then help us better solve the problem of malicious link classification. We built on Kumi's [2] recommendations and crawled the resulting web pages of our dataset for information. We found 32 features to analyze, but some of the more important are as follows:

Feature	**Description**
Content Length	The number of characters that the page contains from the user's point of view which means it excludes all labels.
Number of Labels	A malicious website may be relatively rudimentary and therefore affect the values of these two parameters.
Maximum Characters in Line	The number of characters in the line containing the most characters on the page.
Number of Meta Tags	Meta tags are located in the head of the document and define the name/value pairs associated with the document
Average Length of Cookie	Cookies are usually small text files that the server embeds into the user's computer. This can be altered to be a malicious text file.
Document Call Count	The number of times the HTML page attempts to call the HTML document. By using the API provided by the DOM, the node can be dynamically and maliciously modified.
Number of iframe Tags	Malicious websites often include parts of normal websites in their malicious websites using iframe tags, in order to deceive visitors into thinking that this is a normal website.

We had to create a separate web crawler program to retrieve these features. We found Python's 'BeautifulSoup' library as an effective tool for our application. It provides some simple, python-style functions to handle navigation, searching, modifying analysis and more. It provides users with the data they need to crawl by parsing documents. Beautiful Soup automatically converts input documents to Unicode and output documents to utf-8 [7]. It converts a complex HTML document into a complex tree structure, where each node is a python object, and all objects can be grouped into 4 categories: Tag, NavigableString, BeautifulSoup and Comment.

The process of jumping to a URL's webpage is a lot slower than simply assessing its text. However, the bigger problem was that some of them were no longer active for a visit. This made it difficult to gauge exactly how much the web features contributed in the decision-making for dead samples.

3 Results

3.1 Logistic Regression (LR)

Background - For this use case, Binary Logistic Regression was used to classify URL's as either malicious or not. This model uses a sigmoid function which forms an "S Curve". This allows for the plotting of probabilities to gain a more definitive binary output. Such a model requires that the independent variable (is malicious) is binary. This is why this model was selected to be used [9][10].

Model - The model was built using the sklearn *LogisticRegression* class. This class has multiple tuneable parameters which were tested to gain the best results with the model. The solver type was compared between a few options, with the final model giving the best results using *newton-cg*. The solver method uses the Newton quadratic approximation method to find the quadratic function minimisation. It does this for both first and second partial derivatives. Due to the use of the Hessian Matrix for second partial derivatives this solver can be quite computationally expensive. Despite this, it did give the best results and was therefore used.

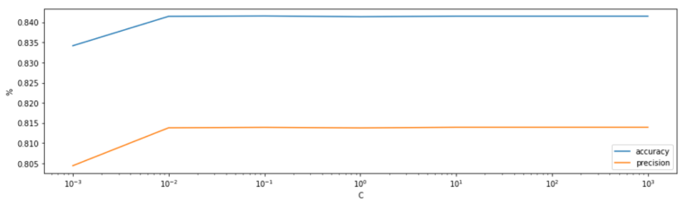


Figure 8: Logistic regression C value

C (inverse of regularisation strength) was another tuned parameter. A range of values was selected using a logarithmic scale. This was done to test C over a wider range with fewer tests. The results showed that there was minimal impact on the model as long as C was at a value higher than 10^-1. At very high values of C overfitting may become an issue for the model. A value of 10^3 was selected to ensure the model was fitting data but was not over-fitting.

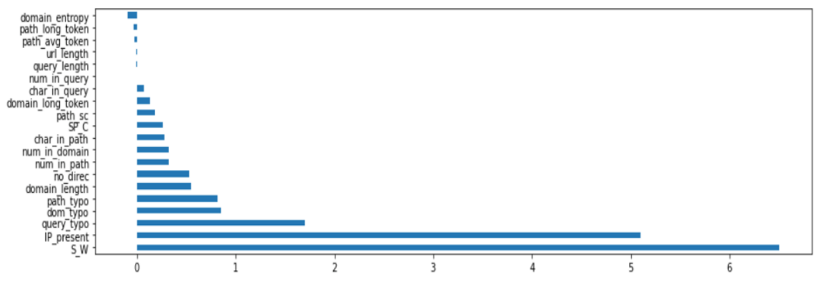


Figure 9: Logistic regression feature importance

Feature Importance - The resultant model gave unique values for feature relevance: S_W and IP_present gave by far the most value to the models' predictions.

Model Results - The end model results on the seen data gave an accuracy value of 83.9%. The confusion matrix does show that the model does give a large number of FN (False Negative) and FP (False Positive) results. In the case of the unseen data, the end accuracy rate was far higher at 92.4%, however, there was still a large number of FN and FP occurrences.

3.2 Decision Tree (DT)

Background - A decision tree is a tree-like model of decisions and possible consequences. In the case of Decision tree learning the decision tree structure may be used to form a predictive model. This is done by representing observations of an item as the branches and the conclusions as to the leaves [11].

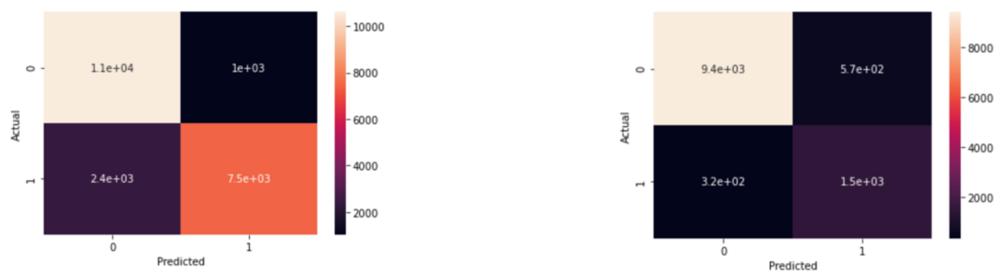


Figure 10: Logistic regression confusion matrix results on seen data (left) and unseen data (right)

Model - For this specific use case of detecting malicious URL's the target variable has a discrete set of values it can be, True or False. Due to this the decision tree being used would fall under a classification tree.

The model implemented was the sklearn *DecisionTreeClassifier* class. This class allows for multiple tuning parameters and some were tested to tune the model. The criterion parameter used was the gini impurity index. This is defined as so: $Gini = 1 - \sum_j (p_j)^2$. Gini impurity works on the principle of measuring the frequency at which any element of the dataset will be mislabelled [12]. From this, the index will try to "purify" the nodes and once the index reaches zero, a node will be "pure". At this point each element in the node is unique and the node will not be split again. Gini was selected int his case as it gave superior results to general entropy. Another good reason for the use of gini is the lower computational cost.

The *max_depth* parameter was also tested. Setting this value too low would result in some nodes remaining impure and would result in a worse result. Too high and there is the potential for the algorithm to try and purify every node to a point with minimal impact on the model. A max depth of 20 was found to give a good balance and so was selected.

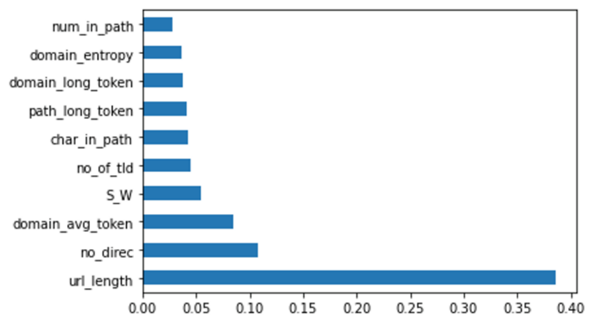


Figure 11: Decision tree feature importance

Feature Importance - The end model gave a quite broad range of feature weightings. URL length by far was the most important but the importance roughly correlates with a logarithmic curve, which may be expected. Due to this most features did impact the model but there was not much difference after the top three.

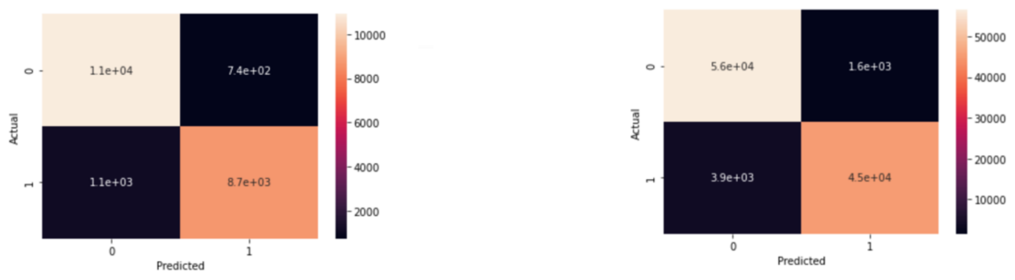


Figure 12: Decision tree confusion matrix results on seen data (left) and unseen data (right)

Model Results - The end model results on the seen data gave an accuracy value of 91.2%. The confusion matrix does show that the model does give a lower number of FN (False Negative) and FP (False Positive) results than logistic regression but still a significant number. In the case of the unseen data was higher at 94.9%. The number of FP and FN cases also dropped proportionally giving evidence of this being a better prediction model in this case.

3.3 Random Forest (RF)

Background - Random forest may be seen as an extension of or follow on from decision trees. This is since random forest is the collection of a large number of decision trees. These models are randomly generated and each spot out a class prediction, the class prediction with the most votes is then selected to become the models' prediction [13]. This leverages the fact that "A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models". In short, this allows for the average result to be more accurate more of the time while using the same architecture as a decision tree.

Model - The model implemented was the sklearn *RandomForestClassifier* class. This class allows for multiple tuning parameters, similar to the DecisionTreeClassifier class. This is due to being based on the same architecture as mentioned. Similarly to the decision tree, the gini index was found to give the best results with the resultant model. The max depth was also tested similarly and a depth of 30 was found to give the best results without giving up too much computing time. For the case of the impurity index, gini and entropy criterions were tested and graphed against one another.

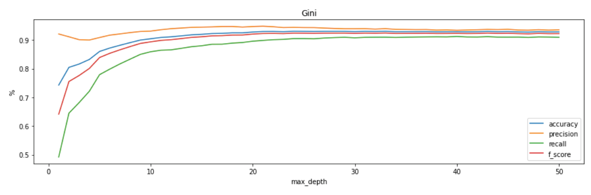


Figure 13: Model with gini criterion

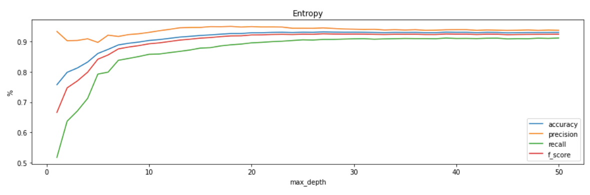


Figure 14: Model with entropy criterion

The resultant graphs showed that both impurity criteria correlated with the max depth the same way however, gini gave overall higher accuracy, precision, and recall values. It is also noticeable in these graphs that with increases to max depth there may be some model over-fitting so capping this to 30 was the best compromise.

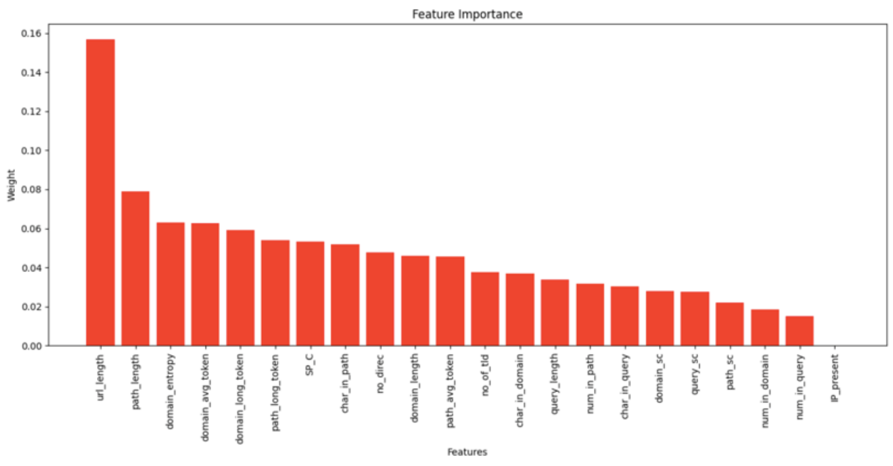


Figure 15: Random forest feature importance

Feature Importance - Here, the relation between features and end model results are similar but still distinct from the Decision tree. In both cases, URL length was the most important feature. The drop-off after the most important feature is more gradual however and this is probably due to the nature of random forest. Due to this more features are considered and this may allow for a more reliable and accurate predictive model.

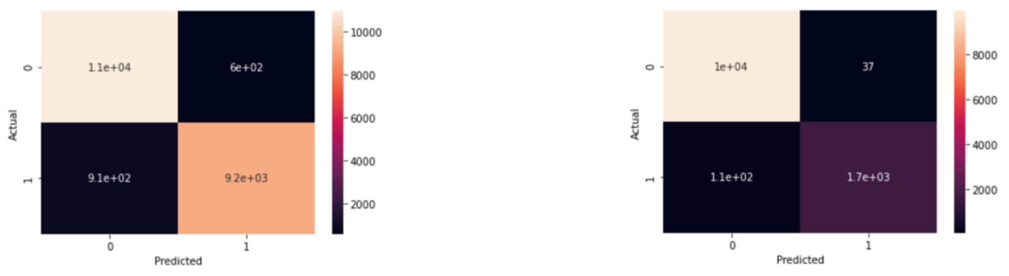


Figure 16: Random forest confusion matrix results on seen data (left) and unseen data (right)

Model Results - The end model results on the seen data gave an accuracy value of 93.0%. The confusion matrix shows that this model gave the fewest number of FN and FP results when compared to logistic regression and decision trees. The unseen data gave an accuracy result of 98.7%. This incredibly high accuracy rate along with its low FP and FN rate shows that this model is apt in the detection of malicious URL's lexically.

3.4 Multilayer Perceptron (MLP)

Background - This experiment uses the *tensorflow* framework to build this fully connective model. The model uses an 8-layer hidden layer model. The number of neurons in each layer is in the form of an integer power of two. And using a layer-by-layer decreasing method for sorting.

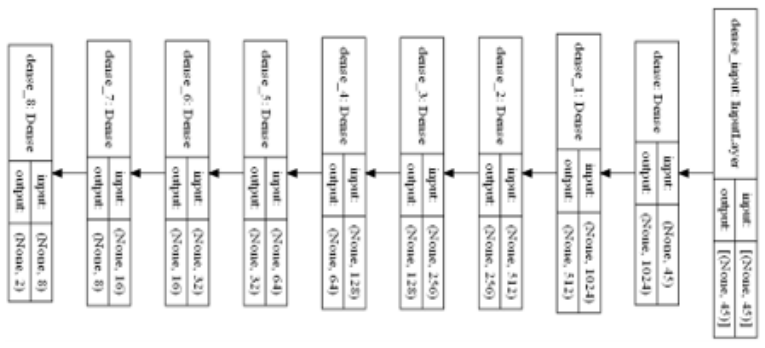


Figure 17: MLP Structure

Model - As the most important part of the model, activation function almost directly affects the result of model training. In the first seven hidden layers, the activation function of each layer is elu function. When training the model, three activation functions are tried to build the model. They are the sigmoid function inherited from logistic regression, the relu function, and the last is the elu function upgraded from relu.

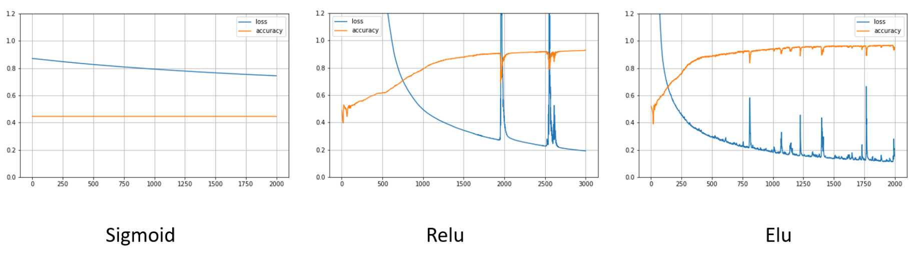


Figure 18: MLP activation hyperparameter

From the figure above, it can be clearly seen that although the loss value of the sigmoid function is decreasing, the accuracy rate has hardly improved. The training effect of relu is already very good, but the convergence of the function is still relatively slow. The elu function has a relatively large improvement in training efficiency due to its fast function convergence characteristics, and a slight improvement in the accuracy rate.

Just like the activation function, both batch size and learning rate play a significant role in tuning the model. Too small a batch size will also make the gradient descent oscillations in the training process serious, which is not conducive to convergence. But too large batch set will make each iteration take a long time. In the case of limited computing power, we chose one-fourth of the training set size as the final batch size to train our model.

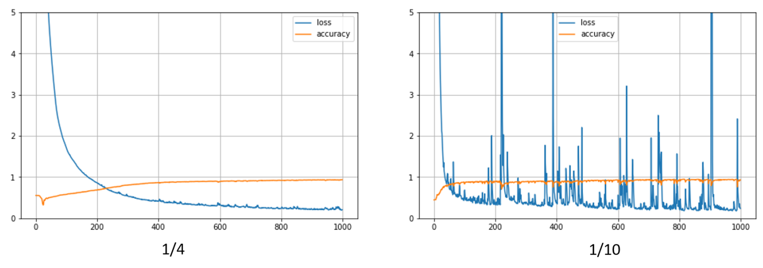


Figure 19: MLP batch-size hyperparameter

Conversely, the model converges faster when the learning rate is large, but the accuracy and loss fluctuate vary seriously, and even affect the accuracy. Reducing the learning rate can make the training of the model smoother. But too small a learning rate will make the convergence speed too slow, making the training model too long. The appropriate learning rate should consider these two factors [14]. Finally, we chose $1 * 10^{-6}$ as the final learning rate.

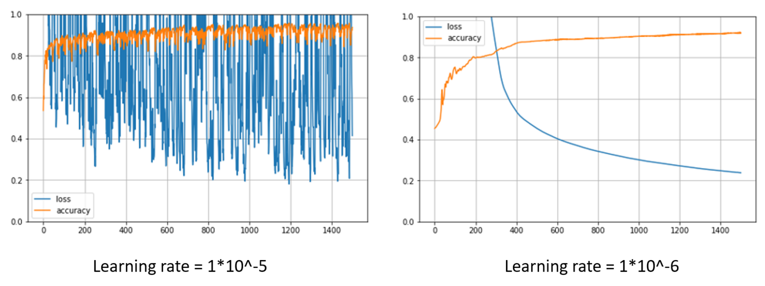


Figure 20: MLP learning rate hyperparameter

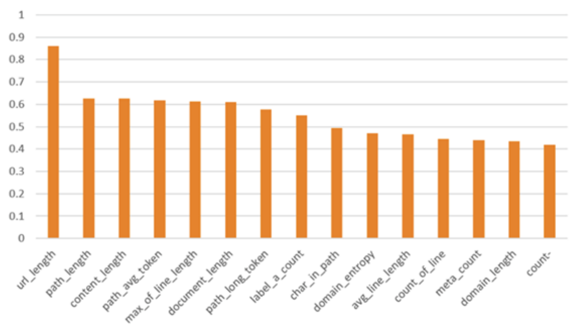


Figure 21: MLP feature importance with maximal information coefficient

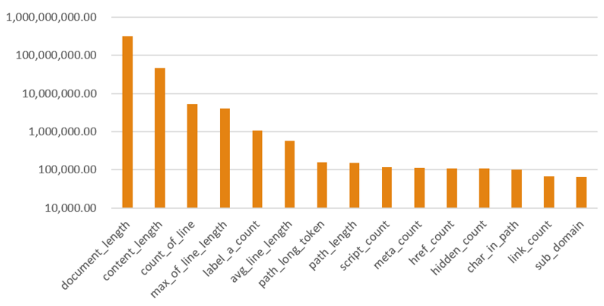
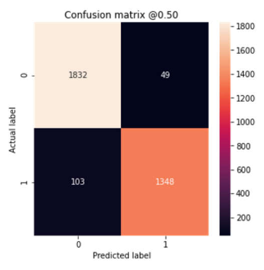


Figure 22: MLP feature importance with chi-squared test

Feature Importance - We have selected two algorithms: Chi-square detection and maximum information coefficient, both of which belong to the filtering method. Chi-square test is an algorithm based on the chi-square distribution, and the maximum information coefficient is based on mutual information theory. Simply put, through these two algorithms, we can get the correlation between each feature and the judgment of whether the URL is malicious in the form of numbers. From the results, The URL, path, and content lengths were valued the most.



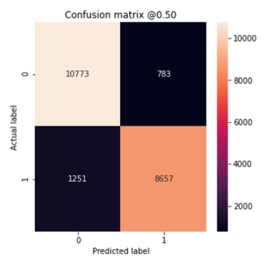


Figure 23: MLP feature importance

Model Results - The end model results on the seen data gave an accuracy value of 95.5%. The unseen data gave an accuracy result of 90.7%. The confusion matrix on both results show that contrary to other models, MLP performed better on the seen data in terms of accuracy and false negatives. This result may be as the web-based features work much better on an MLP model.

3.5 Summary

Here, we will be comparing all models based on the data provided and concluding which one performed the best in all metrics mentioned in 1.2. Below, we show the results in a tabular format:

Comparison of results on seen data						
Model	**FN**	**FP**	**Precision**	**Recall**	**F-Score**	**Accuracy**
RF	914	602	94%	91%	92%	93%
DT	1149	683	93%	88%	91%	91%
LG	2414	1037	88%	76%	81%	84%
MLP	103	49	96%	93%	95%	95%

Comparison of results on unseen data						
Model	**FN**	**FP**	**Precision**	**Recall**	**F-Score**	**Accuracy**
RF	111	37	98%	94%	96%	99%
DT	166	56	98%	91%	94%	98%
LG	323	573	72%	82%	77%	92%
MLP	1251	783	92%	87%	89%	91%

As shown in the tables, among the four models chosen, RF performed the best. This has been concluded by still considering all metrics, however, focusing on false negative and false positive rates where this model is considerably lower than others. MLP was the most computationally expensive out of all the models with the highest number of false negatives in the unseen data, indicating that it had overfitted.

From these experiments and while looking at the feature importance's, we can see how differently each one behaves when modelling. While all other models prioritise URL length, logistic regression prioritises the special words and IP present features. This leads us to believe that there are vast variations of finding how to improve a models outcome. This is due to us initially focusing on feature correlation as IP present had very little correlation with malicious URLs.

4 Discussion

In our attempts at tackling this problem, we struggled with finding a balance between data representation and feature selection. We tried to compile a dataset which could accurately capture a wide range of URLs whilst having an equal number of malicious and benign examples. The term 'wide' is very subjective given how disposable they are. We understood this limitation and decided to include a regularly-updated feed of more recent samples. Our motivation was that a new supply would keep the models informed of the changing formats. However, a consequence of this revolving door approach was that our initial feature set would sometimes perform better or worse depending on what test data we used, indicating that the attributes became more outdated/relevant with the newer values.

If we were to attempt this project again, a proposed extension would be to add online learning where the models are able to update their features and training data after a set interval. We would also like to delve into each feature of a dataset to see if we could possibly generate features that ensured a URL was definitely malicious. A wide range of classifiers could also be used as we would like to experiment with feature importance as logistic regression went a completely different route to the other models.

From this group task, we found it pleasant to work on a project with so many intricacies in as each respective team member would be tackling a completely different aspect of machine learning at any time. We would then bring our findings to meetings to discuss and integrate into our research and code. The challenges that were presented to an individual in the group would be brought up in meetings as another team member would have either had practice prior to the challenge or would come up with creative and logical solutions to the problem. This made working in given time-constraints very efficient.

References

[1] Mamun, M. and Rathore, M. and Habibi L. A. and Stakhanova, N. and Ghorbani, A., 2016, *Detecting Malicious URLs Using Lexical Analysis.* vol 9955. 467-482.
`https://doi.org/10.1007/978-3-319-46298-1_30`

[2] Kumi, S. and Lim, C. and Lee, S., 2021, *Malicious URL Detection Based on Associative Classification. Entropy.*
`https://doi.org/10.3390/e23020182`

[3] Marchal, S., 2014, *PhishStorm - phishing / legitimate URL dataset.*
`https://research.aalto.fi/en/datasets/phishstorm-phishing-legitimate-url-dataset`

[4] Baak, M. and Koopman, R. and Snoek, H. and Klous, S., 2020, *A new correlation coefficient between categorical, ordinal and interval variables with Pearson characteristics.*
`https://www.sciencedirect.com/science/article/abs/pii/S0167947320301341?via\%3Dihub`
Computational Statistics; Data Analysis, North-Holland

[5] Zvelo, Z., 2021, *Base Domain URL vs. Full Path URL. What's the Difference?.*
`https://zvelo.com/base-domain-url-vs-full-path-url-whats-the-difference/`

[6] Brownlee, J., 2021, *Discover Feature Engineering, How to Engineer Features and How to Get Good at It.*
`https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it`

[7] Iliyas, S., 2018, *Detecting malicious URLs using lexical features and machine learning techniques.*
`https://www.researchgate.net/publication/330000314_Detecting_malicious_URLs_using_lexical_features_and_machine_learning_techniques`

[8] Hajba, G., 2018, *Website Scraping with Python.*
`https://bmansoori.ir/book/Website\%20Scraping\%20with\%20Python.pdf`

[9] Peng, J. and Lida L. K. and Ingersoll G., 2002, *An Introduction to Logistic Regression Analysis and Reporting.*
`https://www.researchgate.net/publication/242579096_An_Introduction_to-_Logistic_Regression_Analysis_and_Reporting`

[10] Anandkumar, V., 2019, *Malicious-URL Detection using Logistic Regression Technique.*
`https://www.researchgate.net/publication/338766287_Malicious-URL_Detection_using_Logistic_Regression_Technique`

[11] Web Focus Info Center, 2021, *Explanation of the Decision Tree Model.*
`https://webfocusinfocenter.informationbuilders.com/wfappent/TLs/-TLrstat/source/DecisionTree47.htm`

[12] Aznar, P., 2020, *Decision Trees: Gini vs Entropy.*
`https://quantdare.com/decision-trees-gini-vs-entropy/`

[13] Yiu, T., 2019, *Understanding Random Forest.*
`https://towardsdatascience.com/understanding-random-forest-58381e0602d2`

[14] Brownlee, J., 2019, *Understand the Impact of Learning Rate on Neural Network Performance.*
`https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/`

[15] Gaillard, F., 2020, *Batch size (machine learning).*
`https://radiopaedia.org/articles/batch-size-machine-learning`

[16] Clevert, DA. and Unterthiner, T. and Hochreiter, S., 2015, *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).*
`https://arxiv.org/abs/1511.07289`