Final Project - Malicious URL detection

1. Introduction

Cybercriminals will use any means necessary to comprise individuals' or organization's networks to collect sensitive information. Malicious URLs can be used to extract information such as login information or credit card details. Some of the information collected can then be leveraged to further comprise networks which can then cause further damage to an organization or individual. The impact of this can vary from monitoring computer systems on the network to deploying ransomware within the network using comprised login information.

Blacklisting services have been developed to help detect harmful websites. A blacklist is a database containing a list of all URLs that are known to be malicious. However, the blacklist likely needs to be manually maintained. Changes to one or more components of the URL string may result in the URL not being part of the blacklist and thus many malicious sites are not blacklisted. This can be due to the site being too new, never assessed or assessed incorrectly.

The goal of this project is to determine if an AI solution is deployed to recognize malicious URLs, how well would it detect a malicious URL when trained on a dataset containing an equal amount of malicious and benign URLs.

2. Data

The Malicious URL detection dataset we are using for this project comes from Kaggle, located here[1].

The dataset contains two .csv files, one train.csv (containing 6,728,848 rows) and one test.csv (containing 1,682,213 rows). Both files have the same 60 columns with the key columns to highlight at this stage being "url", "label" and "source".

Our target is the "label" feature with a value of 0 indicating a benign URL and 1 indicating a malicious URL. The "url" is the unprocessed URL that is to be classified. The "source" column is a reference to where the URL was gathered from.

The URLs in the training dataset are sourced from: 'phishtank', 'majestic_million', 'data_clean_test_mendel', 'dmoz_harvard', 'data_clean_train_mendel', 'alexatop1m', 'domcop', 'ALL-phishing-domains', 'tranco_K2K4W', 'ALL-phishing-links', 'manual', 'aa419', 'openphish', 'top500Domains'

The URLs in the test dataset are sourced from: 'dmoz_harvard', 'majestic_million', 'alexatop1m', 'data_clean_train_mendel', 'ALL-phishing-domains', 'domcop', 'ALL-phishing-links', 'phishtank', 'tranco_K2K4W', 'data_clean_test_mendel', 'aa419', 'manual', 'openphish'

The rest of the 57 features break the URLs down into each of their features.

The first set of features are lexical features. These are the elements of the URL string itself. Lexical features include metrics like the URL length, the number of letters, special characters or digits in the URL. The second set of features are network, DNS and host features. These include the subdomain length, information about the top-level domain or if the URL contains an IP address [2].

3. Importing

The two datasets were imported using pandas .read_csv() method. This reads the contents of the file into a dataframe for the path given to it. The path to each file was created by getting the current working directory using Python’s OS library’s .getcwd() method and appending the string “\data\\” to it, to have the path to the folder containing the .csv files. Two variables were then created, one for the path to the data folder plus the “train_dataset.csv” filename and the same for the "test_dataset.csv" filename. The paths were then passed to the read_csv() method individually to create dataframes for both “test_dataset.csv” and “train_dataset.csv”.

4. Data Preparation

With the dataset we selected, we have a good starting point with relatively clean data as it has the maximum useability score of 10 on Kaggle. Our target variable is the ‘label’ column in our dataset with all others being potential features to train from.

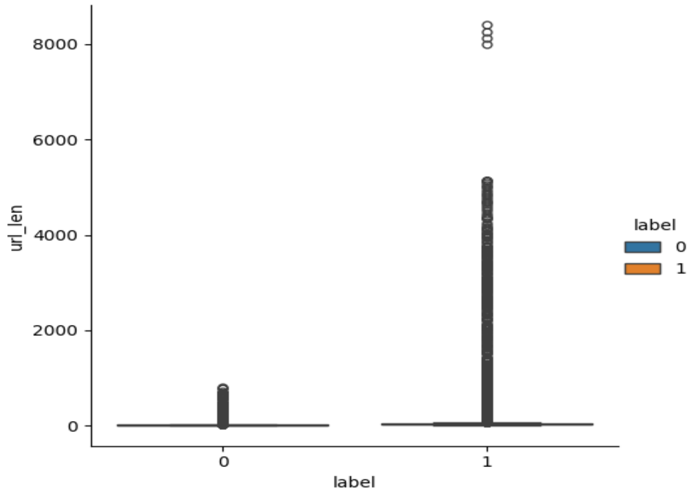
The first portion of our data preparation involved removing any null or NaN values from the dataframes. To do this both dataframes were investigated using our find_null_values function. This takes a dataframe as an argument and returns the sum of each column’s null values. No null values were found in the data and therefore no records needed to be removed.

Next, we analyzed duplicate values in two parts, first in the individual dataframes and then between the two dataframes. For the individual dataframes we defined the function find_duplicates function. This takes a dataframe, column name and returns a count of the duplicates present in the pandas series. We then ran this method for both datasets and no duplicates were found in the individual datasets.

To analyze the duplicates between the two datasets we need to concatenate the train_df and test_df into a new dataframe using the pandas .concat method. In this new dataframe, we add a column called ‘Duplicated’ by using the .duplicated() method, passing keep=False to this method will add a value of “True” to this column for any duplicate URLs found. No duplicate values were found between the two dataframes.

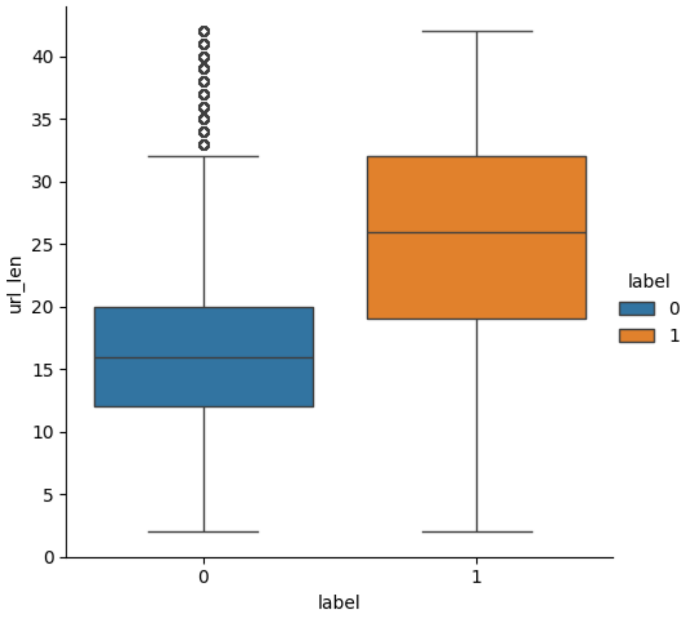
We next focused our attention on the outliers in the data using the merged dataframe (df_merge). To get a further understanding of distribution of the data we plotted the “url_len” column using a box plot. The box plot provides a visual representation of the data’s distribution by showing the median and interquartile range (IQR). Based on the median and IQR being squashed we have confirmed that outliers exist in the data for the “url_len” feature.

Fig. 1: “url_len” grouped by target with outliers



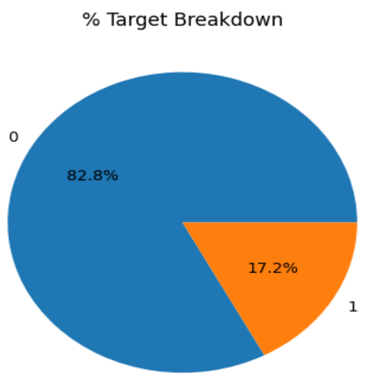
To remove the outliers, we used the IQR method. This method involves getting 25th percentile and subtracting 1.5 times the IQR to create the lower bounds of our data. For the upper bounds we add 1.5 times the IQR to the 75th percentile. Any datapoints with a “url_len” smaller than the lower bounds or higher than the upper bounds are excluded from the dataset.

Fig 2. “url_len” grouped by target with outliers removed



Next, we looked at our target variable in more detail, finding that the dataset is imbalanced with 82.8% of the dataset having a “label” value of 0 (Fig.3). This meant the benign URLs were over-represented in our dataset and would lead to us needing to either oversample the malicious URLs or undersample the benign URLs.

Fig 3. Percentage breakdown of the target



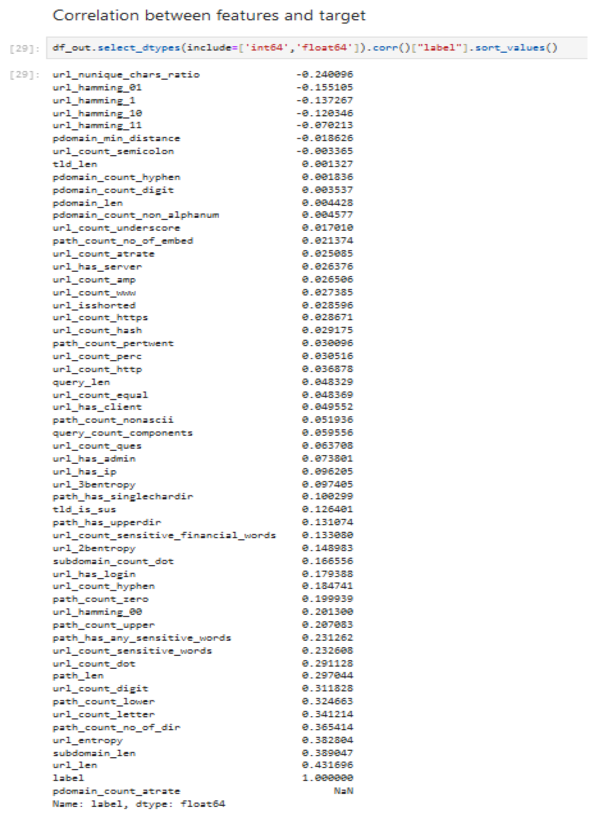
We then looked at the correlation of the numeric features to the target. 48 of the features in the dataset have a correlation coefficient between -0.3 and 0.3. This indicates that these features have weak linear relationships with the target (Fig 4).

7 features have a moderate positive relationship with target and one feature ("pdomain_count_atrate") returned a NaN value (Fig 4). Further investigation of this column found that the only value present in the records was 0. As there is no change in this feature throughout the dataset, we removed it using pandas .drop() method.

As mentioned previously our dataset is imbalanced and to handle this, we chose to randomly undersample the benign URLs. This method was chosen as the dataset is very large and the compute resources available to us would not allow for the oversampled dataset to be processed by our model. This was done using the RandomUnderSampler from the imblearn library.

The final part of our data preparation was to scale the data. First, we split our resampled dataset into X_train, X_test, y_train and y_test using sklearn's train_test_split function. Then, using sklearn's StandardScaler we fitted X_train data and transformed X_test data. The StandardScaler scales our dataset so that the distribution of our data will have a mean value of 0 and standard deviation of 1.

Fig 4. Correlation between features and target



Correlation between features and target

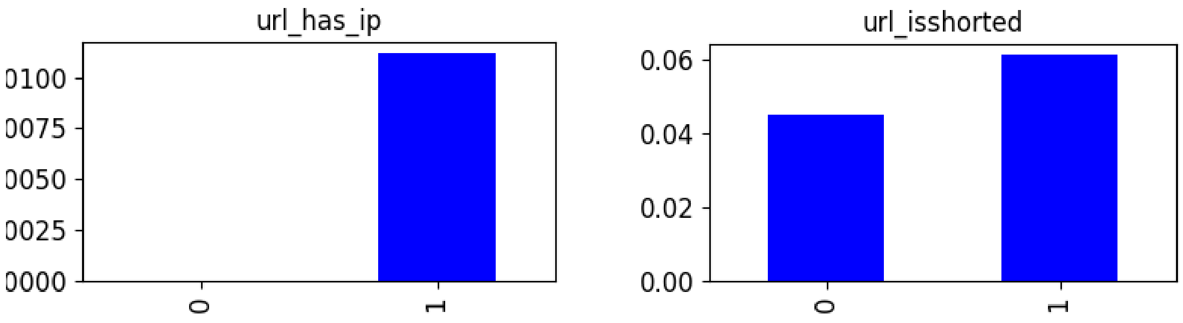
```
[29]: df_out.select_dtypes(include=['int64','float64']).corr()["label"].sort_values()
```

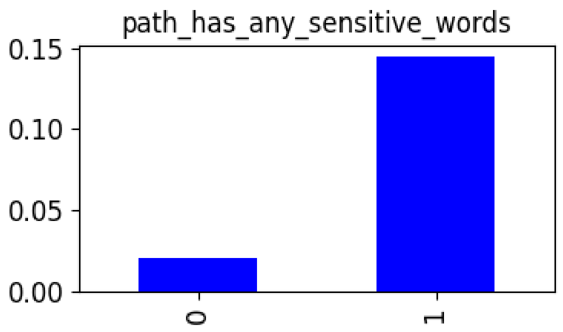
```
[29]: url_nunique_chars_ratio                 -0.240096
      url_hamming_01                          -0.155105
      url_hamming_1                           -0.137267
      url_hamming_10                          -0.120346
      url_hamming_11                          -0.070213
      pdomain_min_distance                    -0.018626
      url_count_semicolon                     -0.003365
      tld_len                                  0.001327
      pdomain_count_hyphen                     0.001836
      pdomain_count_digit                      0.003537
      pdomain_len                              0.004428
      pdomain_count_non_alphanum               0.004577
      url_count_underscore                     0.017010
      path_count_no_of_embed                   0.021374
      url_count_atrate                         0.025085
      url_has_server                           0.026376
      url_count_amp                            0.026506
      url_count_www                            0.027385
      url_isshorted                            0.028596
      url_count_https                          0.028671
      url_count_hash                           0.029175
      path_count_pertwent                      0.030096
      url_count_perc                           0.030516
      url_count_http                           0.036878
      query_len                                0.048329
      url_count_equal                          0.048369
      url_has_client                           0.049552
      path_count_nonascii                      0.051936
      query_count_components                   0.059556
      url_count_ques                           0.063708
      url_has_admin                            0.073801
      url_has_ip                               0.096205
      url_3bentropy                            0.097405
      path_has_singlechardir                   0.100299
      tld_is_sus                               0.126401
      path_has_upperdir                        0.131074
      url_count_sensitive_financial_words      0.133080
      url_2bentropy                            0.148983
      subdomain_count_dot                      0.166556
      url_has_login                            0.179388
      url_count_hyphen                         0.184741
      path_count_zero                          0.199939
      url_hamming_00                           0.201300
      path_count_upper                         0.207083
      path_has_any_sensitive_words             0.231262
      url_count_sensitive_words                0.232608
      url_count_dot                            0.291128
      path_len                                 0.297044
      url_count_digit                          0.311828
      path_count_lower                         0.324663
      url_count_letter                         0.341214
      path_count_no_of_dir                     0.365414
      url_entropy                              0.382804
      subdomain_len                            0.389047
      url_len                                  0.431696
      label                                    1.000000
      pdomain_count_atrate                          NaN
      Name: label, dtype: float64
```

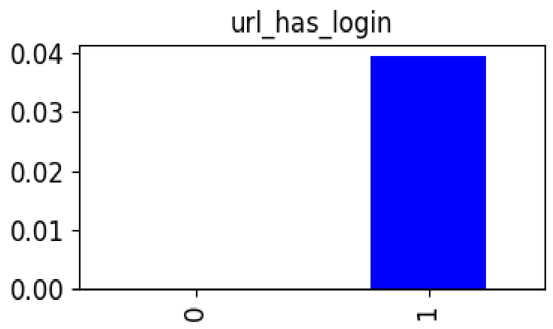
5. Data Visualization

Generate at least four charts using the Matplotlib library:

Fig 5. Boolean Features' mean value grouped by target

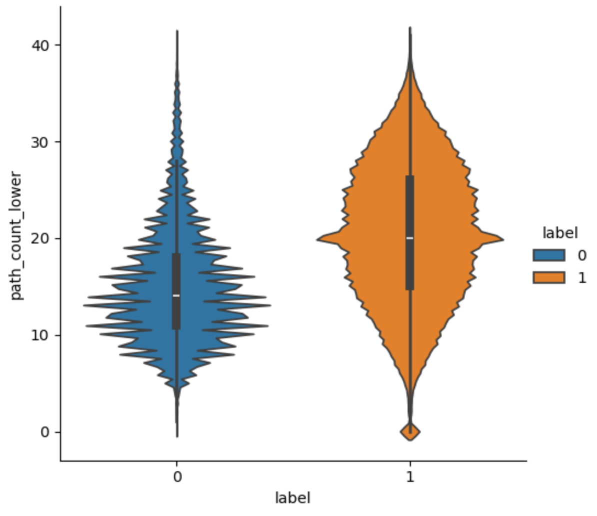






Generate at least four charts using the Seaborn library:

Fig 6. Violin plots of the moderately correlated features grouped by the target



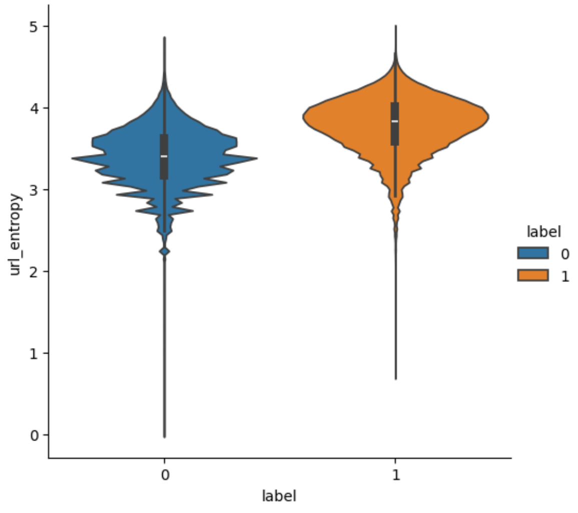
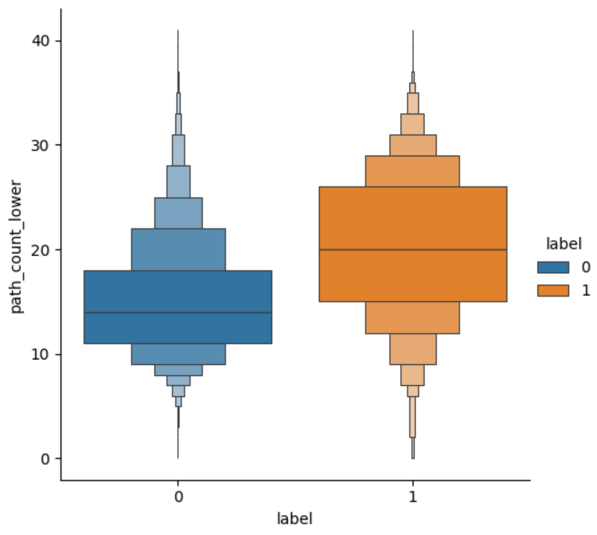


Fig 7. Boxen plots of the count and len features grouped by the target



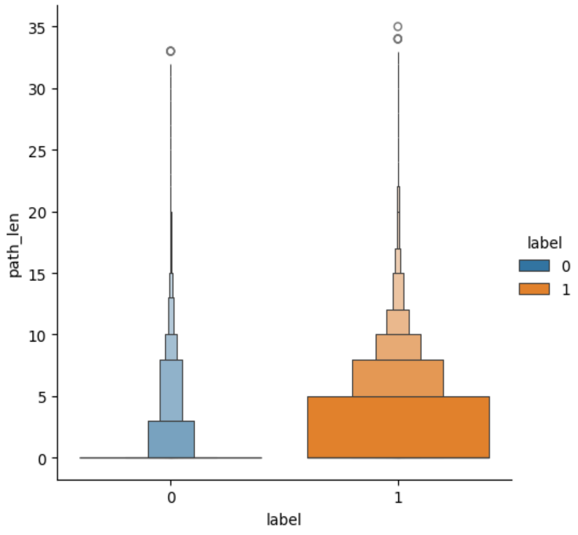


Fig 8. Correlation heatmap of the len features

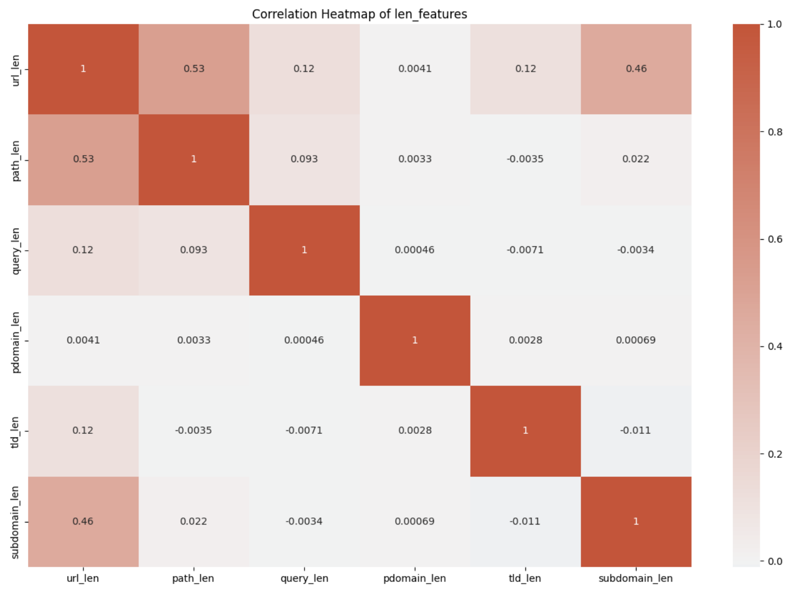
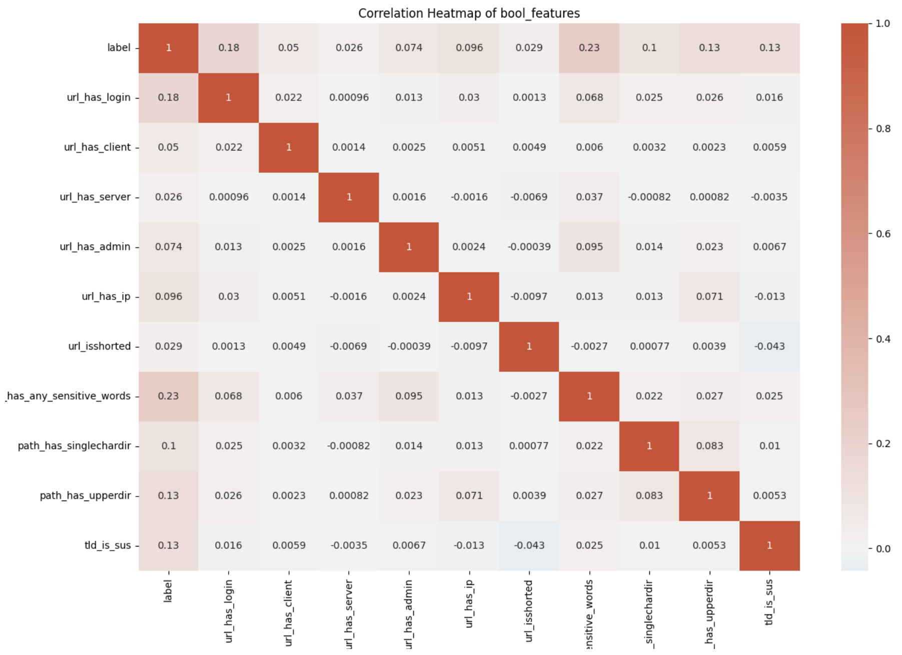


Fig 9. Correlation heatmap of the Boolean features



More charts can be found in the plots directory with the code submission

6. Insights

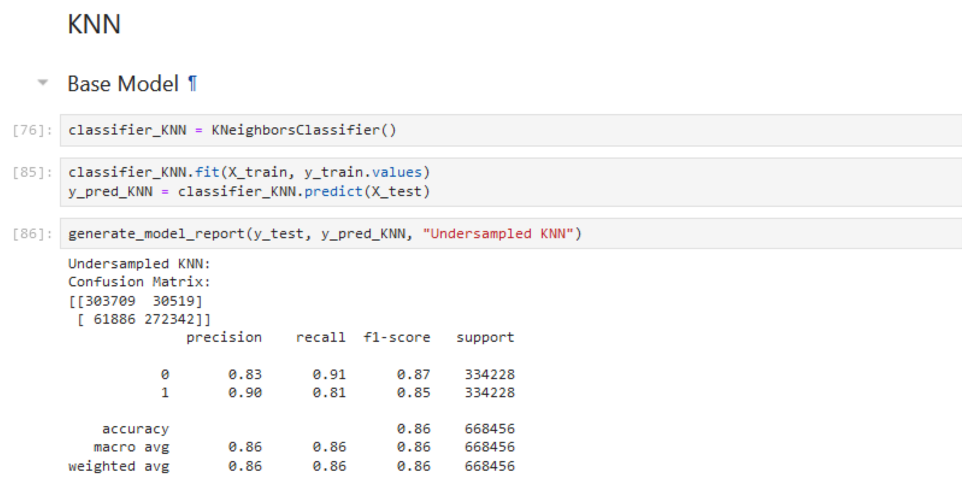
1. The number of benign and malicious URLs in our dataset is imbalanced and will require over/under sampling (Fig 3)
2. There was a significant number of outliers present in the dataset (Fig 1 & Fig 2)
3. The Boolean features in the dataset have a higher mean value for malicious URLs than benign URLs (Fig 5)
4. Malicious URLs have a higher median length than benign URLs (Fig 2)
5. No Boolean or count features are strongly correlated with one another (Fig 9).
6. The moderately correlated features are all based on length (Fig 9). This makes sense as the longer the URL the longer the expected sub domain for example.

7. Machine Learning

Classification methods were chosen for our project as our problem is a binary classification problem where our target variables are two discrete values 0 (benign) and 1 (malicious).

With this known we made the decision to initially test two algorithms for binary classification, Logistic Regression and the K Nearest Neighbor (KNN) classifier. In our initial testing KNN had a higher accuracy score of 0.85 (Fig 11) in comparison to the Logistic Regression's accuracy of 0.82 (Fig 12). Based on the higher accuracy value, we chose KNN to conduct further hyperparameter tuning.

Fig 10. KNN model metrics



KNN

Base Model ¶

```
[76]: classifier_KNN = KNeighborsClassifier()
```

```
[85]: classifier_KNN.fit(X_train, y_train.values)
      y_pred_KNN = classifier_KNN.predict(X_test)
```

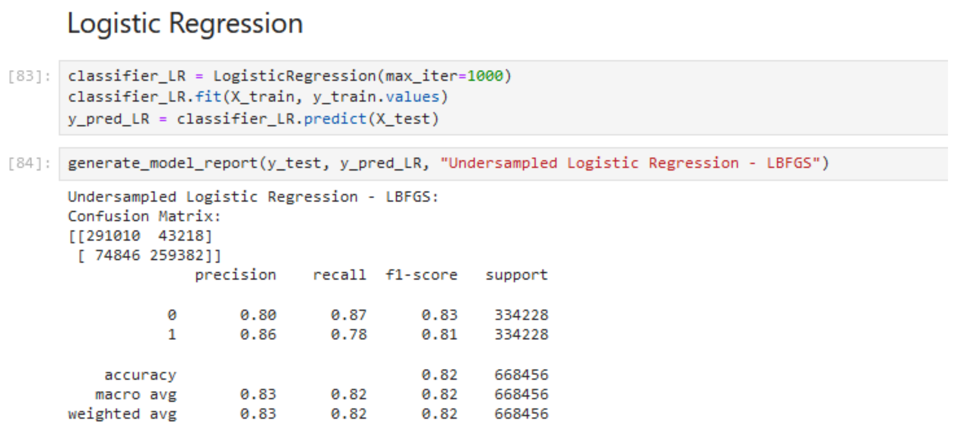
```
[86]: generate_model_report(y_test, y_pred_KNN, "Undersampled KNN")
```

```
Undersampled KNN:
Confusion Matrix:
[[303709  30519]
 [ 61886 272342]]
              precision    recall  f1-score   support

           0       0.83      0.91      0.87    334228
           1       0.90      0.81      0.85    334228

    accuracy                           0.86    668456
   macro avg       0.86      0.86      0.86    668456
weighted avg       0.86      0.86      0.86    668456
```

Fig 11. Logistic Regression model metrics


Logistic Regression

```
[83]: classifier_LR = LogisticRegression(max_iter=1000)
      classifier_LR.fit(X_train, y_train.values)
      y_pred_LR = classifier_LR.predict(X_test)
```

```
[84]: generate_model_report(y_test, y_pred_LR, "Undersampled Logistic Regression - LBFGS")
```

```
Undersampled Logistic Regression - LBFGS:
Confusion Matrix:
[[291010  43218]
 [ 74846 259382]]
              precision    recall  f1-score   support

           0       0.80      0.87      0.83    334228
           1       0.86      0.78      0.81    334228

    accuracy                           0.82    668456
   macro avg       0.83      0.82      0.82    668456
weighted avg       0.83      0.82      0.82    668456
```

For our hyperparameter tuning we used GridSearch to test two hyperparameters, the number of neighbors and the method of distance calculation. This resulted in our best combination of from number of neighbors (5,7,9 or 11) and the distance calculation methods ('minkowski', 'euclidean' or 'manhattan'), being tied between {'metric': 'euclidean', 'n_neighbors': 11} and {'metric': 'minkowski', 'n_neighbors': 11} with a mean accuracy score of 0.8641895153. The results of the GridSearch can be seen in the “KNN_cv_results.csv” file included with the code submission.

8. Conclusion

Domain knowledge plays a key part in understanding the dataset and in the selection of features to train the model on. With better knowledge of URLs and the features included in this dataset, we believe better feature selection could be done. The selected features could then lead to improved model predictions.

To conduct a full tuning of hyperparameters a high level of compute and length of time is needed. Combining the cross validation from grid search and the number of combinations available with the hyperparameters leads to a large number of jobs. We believe that further improvements can be made to the model by tuning the hyperparameters further.

With more resources, additional values for the number of neighbors could have been tested and a higher number of cross validations could have been included. This may have resulted in further model improvements.

Additional hyperparameters that could be investigated could include the ‘weight’ function which is used in the prediction the function or the ‘leaf_size’ which could improve the time taken for the model to complete and the memory required during its runtime.

Overall, we feel that there are more improvements to be made to the model at this stage. For potential next steps with the project, we feel improved feature selection, bagging and ensemble methods would be best explored.

9. References

[1] - www.kaggle.com. (n.d.). *Tabular dataset ready for malicious url detection*. [online] Available at: https://www.kaggle.com/datasets/pilarpieiro/tabular-dataset-ready-for-malicious-url-detection/data?select=train_dataset.csv [Accessed 20 Mar. 2024].

[2] - Aljabri, M., Altamimi, H.S., Albelali, S.A., Al-Harbi, M., Alhuraib, H.T., Alotaibi, N.K., Alahmadi, A.A., Alhaidari, F., Mohammad, R.M.A. and Salah, K. (2022). Detecting Malicious URLs Using Machine Learning Techniques: Review and Research Directions. IEEE Access, 10, pp.121395–121417. doi:https://doi.org/10.1109/access.2022.3222307.