Malicious URL detection using Ensemble Methods

Chouliaras Andreas ACHOULIARAS@INF.UTH.GR
Pappas Apostolos APOPAPPAS@INF.UTH.GR

Abstract

With the continuous expansion of the Internet and its users, more and more security issues come to light. As much as security measures increase to protect systems that make up the Internet, the user remains the most vulnerable part of it. Malicious URLs leading to malicious websites, are a common and serious threat to cybersecurity. Malicious URLs host unsolicited content (spam, phishing, drive-by exploits, etc.) and lure unsuspecting users to become victims of scams (monetary loss, theft of private information, and malware installation), causing losses of billions of dollars every year. The most classical way of dealing with this threat is the usage of blacklists. Blacklists tend to lack the scalability and the overall ability to to detect newly generated malicious URLs. In this report, we examine the effectiveness of several Machine Learning algorithms on detecting such Malicious URLs taking training time into account as well. We also examine ensemble methods, in order to further improve our model's accuracy.

1. Introduction

The importance and the effect that the World Wide Web has, is continuously increasing. Unfortunately, the technological advancements come coupled with new sophisticated techniques to attack and scam users. Scam attacks include a certain type of websites that perform fraud by tricking users into revealing sensitive information which eventually lead to theft of money, identity, or even installing malware in the user's system. There are various ways of implementing an attack. Some of them are:

- SQL injections
- Explicit hacking attempts
- Phishing and Social Engineering
- Drive-by-Download
- Denial of Service(DoS)
- Distributed Denial of Service(DDoS)

It is common that the mentioned attacks are realized through spreading compromised URLs. These compromised URLs are called *Malicious*. Research has shown that almost one third of all websites(thus, their URLs as well) are malicious in nature. Unsuspecting users visit such web sites and become victims of various types of scams, including monetary loss, theft of private information, and malware installation (Hong, 2012). The most traditional way of dealing with such threats is using blacklists.

Blacklists are essentially databases that contain several URLs marked as malicious. However, new URLs are created everyday, making these databases somewhat obsolete. In particular, there are numerous cases where the attackers create and modify URLs that seem legitimate, fooling the unsuspecting user into the attack. The main problem with blacklists is their incapability in making predictions for new URLs other than the ones already in their database (Sinha et al., 2008).

In order to overcome this barrier, we can use and apply several Machine Learning techniques that fit on the training data, in this case several URLs, and make predictions about newly seen URLs (Sahoo et al., 2017).

Having the training dataset, we first need to preprocess it in order for our algorithm to be able to build the classifier model. There is a wide variety of Machine Learning algorithms we can choose from, like Logistic Regression, K-Nearest-Neighbors, Neural Networks, Support Vector Machines, Decision Trees and others. After selecting the appropriate algorithm, the next step is to train our model. This will fit our model on the training data and will be the foundation of each prediction in the future.

When predicting and classifying a URL, there are two types of misclassification that can be made: False Positives and False Negatives. A False Positive (or FP) is just a "false alarm". In our case, the model classifies a URL as malicious, while in reality it is not. A False Negative (or FN) in

University of Thessaly, ECE417 Course Project.

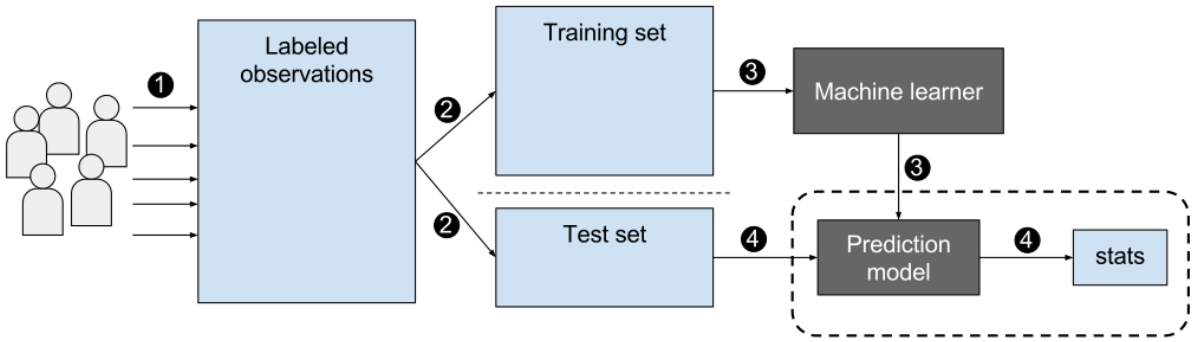

Figure 1. The procedure of evaluation for every machine learning algorithm.

our problem, is the misclassification of a malicious URL as non-malicious. One can understand the negative impact of the second kind of misclassifications. Therefore our aim is to keep FNs as low as possible.

1.1. Dataset and Preprocessing

The dataset used for our experimentations, can be found at www.kaggle.com. It contains a total of 420,464 URLs, 75,643 of which are malicious and 344,821 are safe. Each URL contains only the host name and the path, excluding the HTTP Protocol at the beginning, if it had any.

Table 1. Quantitative description of the dataset.

DATASET	URLS
TOTAL URLS	420,464
MALICIOUS URLS	75,643
SAFE URLS	344,821
% OF MALICIOUS	18
% OF SAFE	82

As it is, the dataset cannot be directly used by Machine Learning algorithms. Thus, there is the need to transform it into vectors and matrices, easily manipulated by such algorithms.

The algorithm we used for this purpose is *tf-idf*, short for ”term frequency-inverse document frequency”. It is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. Term frequency (tf) is essentially the output of the Bag-of-Words model. For a specific document, it determines how important a word is by looking at how frequently it appears in it. For a word to be considered a signature word of a document, it shouldn't appear that often in the other documents. Thus, a signature words document frequency must be low, meaning its inverse document frequency must be high. The idf is usually calculated as:

$$idf(W) = log \frac{\#(documents)}{\#(documents\ containing\ word\ W)}$$

The *tf-idf* is the product of these two frequencies. For a word to have high *tf-idf* in a document, it must appear a lot of times in said document and must be absent in the other documents. It must be a signature word of the document.

In Python, the *tf-idf* algorithm is implemented using the *TfidfVectorizer* class. After the tokenization we have a vocabulary of about 420,000 tokens. To use it for the models, we use 80% of the dataset as our training set and the rest 20% as our test set.

1.2. Non-Ensemble Models Evaluated

The non-ensemble algorithms used in this report were:

- Logistic Regression
- Naive Bayes (Bernoulli)
- Naive Bayes (Multinomial)
- K-Nearest-Neighbors
- Multilayer Perceptron Classifier

Each of these algorithms is supported and implemented through *sklearn* package in Python.

1.3. Ensemble Models Evaluated

To further improve the classification results, we also evaluate the following ensemble methods. Firstly, we examine the effectiveness of the **Random Forest** algorithm on our data. Then, we constructed a multistage ensemble model of the best non-ensemble algorithms and a **Multilayer Perceptron** model to perform the ensemble using Tensorflow and Keras (Yang and Browne, 2004).

2. The Evaluation Metrics

Before presenting the results of our experimentation, we should first take a look at the metrics on the basis of which we compare the different Machine Learning algorithms we mentioned above.

Firstly, we use Accuracy, Precision and Recall to evaluate our models. Accuracy is, in its core, the ratio of number of correct predictions to the total number of input samples.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ input\ samples}$$

Precision refers to the proportion of positive identifications that was actually correct and is defined as the number of true positives divided by the number of true positives plus the number of false positives.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

On the other hand, Recall refers to the proportion of actual positives that was identified correctly and is defined as the number of true positives divided by the number of true positives plus the number of false negatives.

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Although Accuracy seems like a good and comprehensible way to evaluate our model, it may very well lead us to misinterpreting the results of our algorithm. We should have in mind that we examine a problem where False Positives and False Negatives have a big impact. Thus, we use the F_1 score as another way to evaluate our models. F_1 score is the harmonic mean of Precision and Recall and it is suited to problems where the cost of False Positives and False Negatives are very different. Therefore, this score takes both false positives and false negatives into account.

$$F_1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Moreover, we use the Area Under Curve, also known as AUC, a metric which is connected to the Receiver Operating Characteristic Curve (ROC curve). AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability.

The ROC curve is plotted with True Positive Rate against the False Positive Rate as can be seen at *Figure 2*.

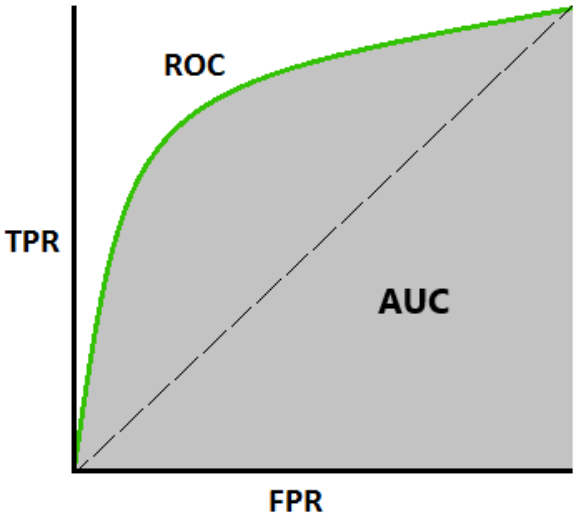

Figure 2. ROC curves and AUC.

So we know that the higher the AUC metric, the more successful a model is at a classification problem.

Last but not least we take into consideration the time needed to train a model and make predictions with it. It is very important to produce results as fast as possible in addition to good classification results, because training a slow model might make the predictions obsolete.

3. The Non-Ensemble Approach

In this section, we examine the results of the algorithms mentioned in subsection 1.2. We present the corresponding metrics' values and ROC curves of each algorithm implemented.

3.1. Logistic Regression

In this case, we implement a simple binary Logistic Regression model. Logistic Regression, in its nature, is a statistical procedure that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. The logistic function $\sigma(z) = \frac{1}{1+exp(-z)}$ is extensively used in order for the decision boundary to be modelled. Finally, the decision boundary is given by $\beta_0 + x^T\beta = 0$, where x is the vector of our predictors and β_i the parameters of the model.

Using the *sklearn* package, we considered a tolerance level of $1e-4$ and chose the *SAGA* solver given 100 iterations to converge. SAGA (Defazio et al., 2014), a modification of SAG (Stochastic Average Gradient), is also recommended by *Scikit* documentation as the best solver we can choose for very large datasets.

Indeed, we noticed that the model was trained within seconds. Testing the model gave us an accuracy of 96.3%, a Precision and a Recall of 97.3% and 81.6% respectively. The F_1 score lies at 88.8% while the Area Under Curve is at 0.905 as seen in *Figure 3*.

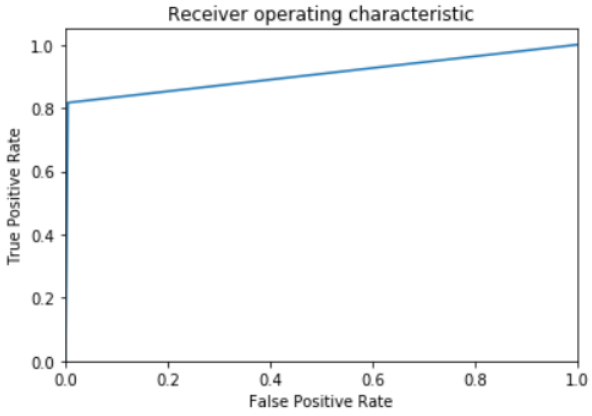

Figure 3. ROC curve for the Logistic Regression model. The AUC is 0.905

3.2. Naive Bayes

Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features. It was introduced (though not under that name) into the text retrieval community in the early 1960s (Maron, 1961), and remains a popular (baseline) method for text categorization. With appropriate pre-processing, it is competitive in this domain with more the advanced methods.

With the use of Maximum A Posteriori estimation, the corresponding classifier, a Bayes classifier, is the function that assigns a class label $\hat{y} = C_k$ for some k as follows:

$$\hat{y} = \underset{k \in \{1,\dots,K\}}{\operatorname{argmax}} \; p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

The assumptions on distributions of features are called the event model of the Naive Bayes classifier. For discrete features like the ones encountered in document classification, multinomial and Bernoulli distributions are popular.

3.2.1. Bernoulli Naive Bayes

For the Bernoulli Naive Bayes, the likelihood function is given by:

$$p(\mathbf{x} \mid C_k) = \prod_{i=1}^{n} p_{k_i}^{x_i} (1 - p_{k_i})^{(1-x_i)}$$

We trained the Bernoulli Naive Bayes model using the *sklearn* package. Setting every parameter to default, our training was completed in a couple of seconds.

Testing the model gave us an accuracy of 95.12%, a Precision and a Recall of 97.9% and 74.5% respectively. The F_1 score lies at 84.6% while the Area Under Curve is at 0.871 as seen in *Figure 4*. We can see this modes isn't so successful at reducing the False Negatives.

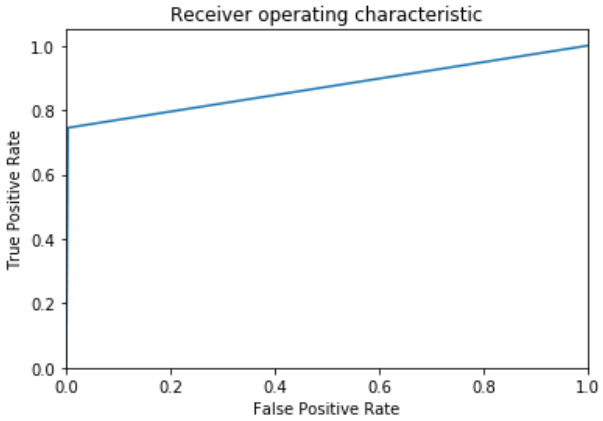

Figure 4. ROC curve for the Bernoulli Naive Bayes model. The AUC is 0.871

3.2.2. Multinomial Naive Bayes

For the Multinomial Naive Bayes, the likelihood function is given by:

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i {p_{ki}}^{x_i}$$

The classifier though becomes a linear classifier when expressed in log-space:

$$\begin{aligned} p(C_k \mid \mathbf{x}) &\propto \log \left(p(C_k) \prod_{i=1}^{n} {p_{ki}}^{x_i} \right) \\ &= \log p(C_k) + \sum_{i=1}^{n} x_i \cdot \log p_{ki} \\ &= b + \mathbf{w}_k^\top \mathbf{x} \end{aligned}$$

where $b = \log p(C_k)$ and $w_{ki} = \log p_{ki}$

We trained the Multinomial Naive Bayes model again using the *sklearn* package. Setting every parameter to default, our training was completed again in a couple of seconds.

Testing the model gave us an accuracy of 96.32%, a Precision and a Recall of 99.3% and 80.1% respectively. The F_1 score lies at 88.7% while the Area Under Curve is at 0.899

as seen in *Figure 5*. We can see this modes is less successful at reducing the False Negatives but does an almost perfect job at reducing the False Negatives.

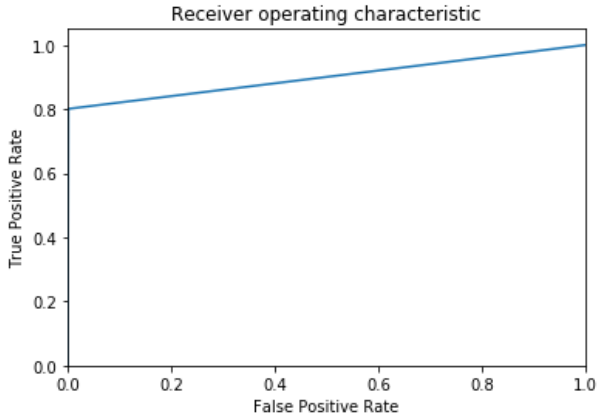

Figure 5. ROC curve for the Multinomial Naive Bayes model. The AUC is 0.899

3.3. Multilayer Perceptron Classifier

A Multilayer Perceptron (MLP) is a class of feedforward artificial neural network. A MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. As mentioned in the *sklearn* documentation, the model trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters. It can also have a regularization term added to the loss function that shrinks model parameters to prevent overfitting.

For the sake of experimentation, we first used the *MLP-Classifier*. The model consisted of only one hidden layer with ten perceptrons. The tolerance was set to default and the model had a maximum of 10 iterations to converge. Although the results were impressive, the training process was too slow. Thus, we thought that a reduction in the number of features would have a positive impact on the training time at least.

For this purpose, we used TruncatedSVD, which implements a variant of singular value decomposition (SVD) that only computes the k largest singular values, where k is a user-specified parameter. Mathematically, truncated SVD applied to training samples X produces a low-rank approximation X:

$$X \approx X_k = U_k \Sigma_k V_k^\top$$

After this operation, $U_k \Sigma_k^\top$ is the transformed training set with k features. TruncatedSVD is very similar to PCA, but differs in that it works on sample matrices X directly instead of their covariance matrices. When the column-wise (per-feature) means of X are subtracted from the feature values, truncated SVD on the resulting matrix is equivalent to PCA. According, once again, to the *sklearn* documentation while the TruncatedSVD transformer works with any feature matrix, using it on tfidf matrices is recommended. Thus, we considered it our best choice in feature reduction.

We trained our model using the TruncatedSVD's results alongside the same parameters as before. Although the model's training time reduced substantially, its accuracy dropped by 5.5%. Recall had a major 18.5% drop followed by a 9% drop in AUC and a 7.73% decrease in Precision. The F_1 score also dropped by 13.8%.

We present the original MLP model's metrics as they were a lot better. Both model's results are included in *Table 2*. The accuracy of the model was 98.03%. We achieved a precision of 97%, a recall of 82.6% and a F_1 score of 89.2%. The final Area Under Curve was 0.905 as seen in *Figure 6*.

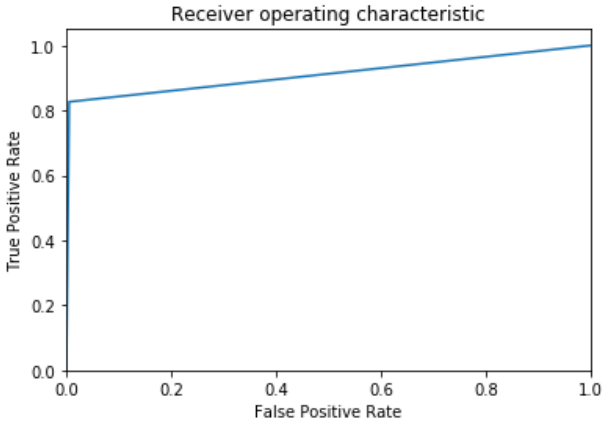

Figure 6. ROC curve for the MLP model. The AUC is 0.905

3.4. k-Nearest-Neighbors

The k-Nearest Neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. KNN does not construct a strict model but is based on the training examples which are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples.

In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point. The most commonly used distance metric, and also the one that we use in our implementation, is the Euclidean distance:

$$d(p,q) = d(q,p) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

Because of the fact that k-NN is not based on a strict model, it is suffering when applied to large datasets. Our dataset was not an exception. So, as we did in the previous subsection where we examined the Multi-Layer Perceptron, we tried feature reduction using TruncatedSVD. Testing the model again, this time with the feature reduction, the accuracy decreased by 1%, precision saw a major drop by almost 10%, while recall increased by 3% which is good. F_1 score dropped by approximately 3.3% as well as AUC by 0.9%. It seems that the advantage we had in execution time did not translate to better performance.

So this model's accuracy was 95.58%, the precision and recall were 89,2% and 82,5% respectively, the F_1 score lies at 85.7% and the AUC was 0.901 as seen in in *Figure 7*

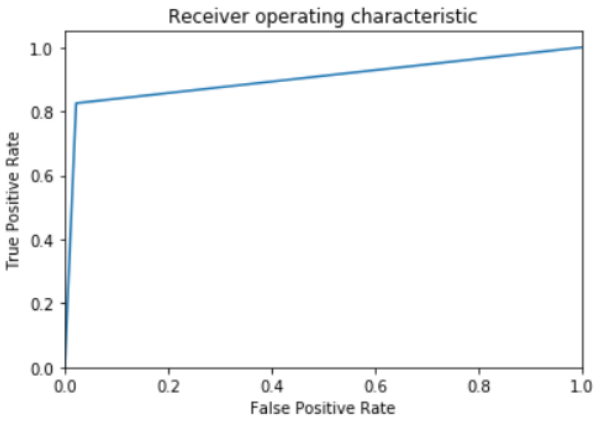

Figure 7. ROC curve for the k-Nearest-Neighbors (SVD) model. The AUC is 0.901

4. The Ensemble Approach

The general idea is that with ensemble methods we try using multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone.

4.1. Random Forest

In this part we first examine the Random Forest algorithm. In its core it creates multiple random Decision Trees and merges them into one model in order to get a more accurate and stable prediction with less overfitting. After some hyperparameter adjustment we decide to use ten estimators as a good compromise between training time and accuracy. But with a dataset so large as this, the time needed for training is substantially large. As previously mentioned we tried feature reduction here as well using the TruncatedSVD. The results were impressive. The accuracy dropped less than 1%, the precision dropped less than 5% and the $F1$ score dropped by less than 0.5%. But on the other hand the recall improved by 2% which is good and the AUC increased by about 1%. Overall it seems that the predictive power of the model this time using the TruncatedSVD improved, if only slightly, as it managed to reduce the False Negatives. The greatest profit was that the training time was reduced by 97%.

At this point we tried using more estimators but there weren't any observable changes in the results. So we achieved for the SVD version of the Random Forest 96.8% accuracy, 94.4% and 87.4% precision and recall respectively. The F_1 score lies at 90.8% while the AUC is at 0.931 as seen in *Figure 8*

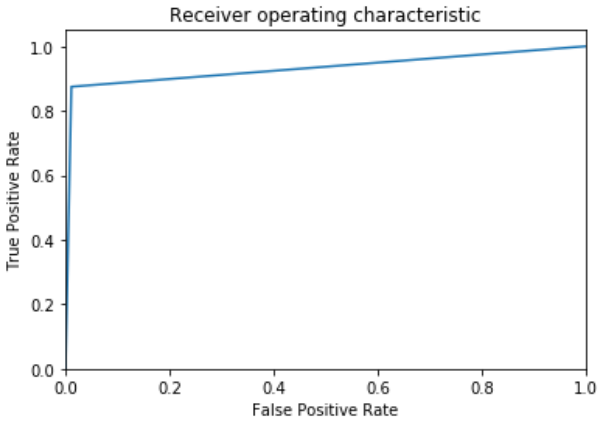

Figure 8. ROC curve for the Random Forest (SVD) model. The AUC is 0.931

4.2. Multistage ensemble model using MLP

As we previously stated, the idea here is to exploit the best of our models and use their predictions to further improve the results. There are multiple ways to ensemble models. One way is to take the average of the corresponding models' predictions as the final prediction of the model. Another way is to use the weighted average of the corresponding models' predictions. There are other methods like bagging, boosting etc. (Yang and Browne, 2004).

4.2.1. Our approach

We are particularly interested in the weighted average method. The idea behind this method is that, of all the models we have, some of them might be better at recognizing some patterns and others might excel at recognizing different ones. So, it is a good idea to assign weights at each

model that describes how good is the particular model. We can take this one step further by assigning weights to every prediction a model gives that describes now how good is the particular model's prediction for a specific input instance. We used a Multilayer Perceptron to perform this job. So we have a first stage that consists of the best algorithms we already described and a second stage with a MLP classifier that takes the predictions from the previous stage and outputs the final prediction as seen in *Figure 9*

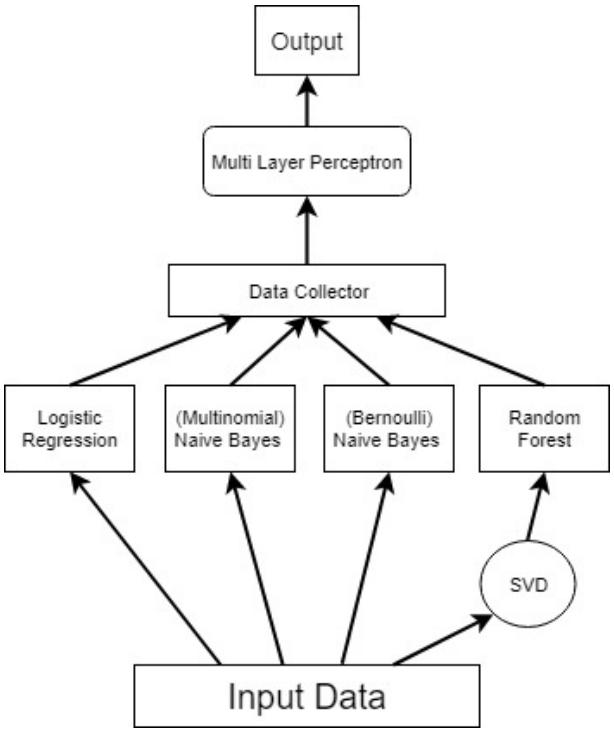

Figure 9. The Custom Ensemble Model Flow Chart

4.2.2. Model Selection

The selection of the models for the first stage is no simple task. We need to consider not only the metrics results but also the execution times because we want the overall model to be relatively fast to train while also achieving better results than any of the constituent models. Furthermore as we already stated we want to decrease the False Negatives as much as possible so the recall metric and the ROC curves play major role. As a result, we decided to use Logistic Regression, Multinomial Naive Bayes, Bernoulli Naive Bayes and Random Forest with SVD feature reduction. Those models, while they are the fastest, they have rather different confusion matrices, with some of them keeping False Negatives low and others keeping False Positives low, something that may help the ensemble model.

4.2.3. The MLP model

To provide the data the MLP needs at the second stage, we took the prediction output vectors from the first stage and stacked them into a matrix with number of rows equal to the number of instances and number of columns equal to the number of the models used. This was the "dataset" for the MLP. The MLP consists of one input layer, two hidden layers with 50 and 10 neurons respectively and one final layer to produce the result. We used Tensorflow and Keras to implement the MLP model.

4.2.4. Training and Predicting

As the first stage models were already trained, we just needed to train the second stage MLP model. To do that we make predictions with the first layer using the training set as input. The resulting predictions are connected together in a matrix and are forwarded in the second layer. Here we use this stage-1 dataset to train the MLP as normal using 20% of it for validation. After the training is completed the predictions are produced by feeding the stage one models with the test set. then the output predictions are forwarded to the second layer to make the stage two "test set" which is fed to the MLP model. The resulting output vector of the MLP is the total model's predictions.

4.2.5. Results

The results were better than initially expected. We achieved the best accuracy among the constituent models at 97.61% falling short behind the sklearn MLP model. It has lower precision than the constituent models at 94.3% but has by far the best recall at 92.2% and the best F_1 score at 93.3&. Finally it has the best AUC at 0.955 as seen in *Figure 10*

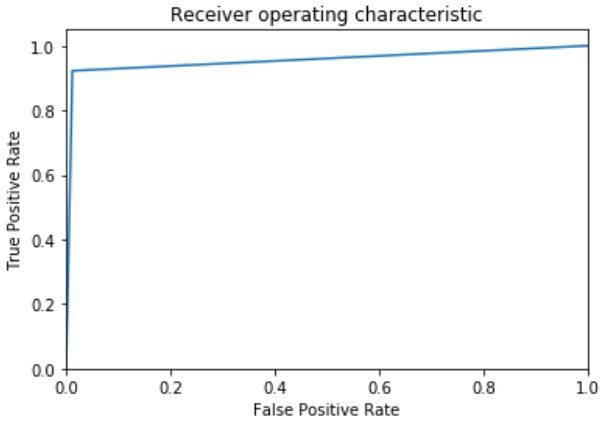

Figure 10. ROC curve for Ensemble model. The AUC is 0.955

We provide a comparative chart with the metrics for all the models we used in *Table 2*

Table 2. Comparative Table of ML Models.

ALGORITHM	PRECISION	RECALL	F1 SCORE	AUC	ACCURACY(%)
MLP (SVD)	0.895	0.674	0.769	0.828	92.7
NB BERNOULLI	0.979	0.745	0.846	0.871	95.12
NB MULTINOMIAL	0.993	0.801	0.887	0.899	96.32
KNN (SVD)	0.892	0.825	0.857	0.901	95.58
LOGISTIC REGRESSION	0.973	0.816	0.888	0.905	96.3
KNN	0.993	0.801	0.886	0.909	96.12
MLP	0.970	0.826	0.892	0.910	98.0
RANDOM FOREST	0.979	0.856	0.914	0.926	97.1
RANDOM FOREST (SVD)	0.944	0.874	0.908	0.931	96.8
CUSTOM ENSEMBLE MODEL	0.943	0.922	0.933	0.955	97.61

Concluding Remarks and Future Directions

Malicious URLs are a serious threat to cybersecurity. The detection of such URLs plays a critical role for many cybersecurity applications, and clearly machine learning approaches are a promising direction. We tested a lot of machine learning algorithms, more than the included, like SVM which we excluded because it was so slow that we never actually managed to finish the training. We saw how interesting and effective ensemble methods can be and even a custom-made ensemble model that consists of fast and diverse models can produce impressive results.

A lot more work could be done, testing even more algorithms and trying grid-search for the faster ones or even an optimized version of it. Working with ensemble methods never ends because we can try more and more complex models and methods. A good evolution to all this is AutoML with capabilities to do most of the work automatically like data preprocessing, model selection, hyperparameter optimization etc.

Acknowledgements

We would like to give our sincere appreciation to the ECE 417: Machine Learning for Data Science and Analytics class and to our professor Elias Houstis for his guidance throughout the entire semester. We were able to overcome ourselves and push though what we thought was our limits.

References

A. Defazio, F. Bach, and S. Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in Neural Information Processing Systems*, 2, 07 2014.

J. Hong. The state of phishing attacks. *Commun. ACM*, 55: 74–81, 01 2012. doi: 10.1145/2063176.2063197.

M. E. Maron. Automatic indexing: an experimental inquiry. *Journal of the ACM (JACM)*, 8(3):404–417, 1961.

D. Sahoo, C. Liu, and S. C. Hoi. Malicious url detection using machine learning: a survey. *arXiv preprint arXiv:1701.07179*, 2017.

S. Sinha, M. Bailey, and F. Jahanian. Shades of grey: On the effectiveness of reputation-based blacklists. In *2008 3rd International Conference on Malicious and Unwanted Software (MALWARE)*, pages 57–64, Oct 2008. doi: 10.1109/MALWARE.2008.4690858.

S. Yang and A. Browne. Neural network ensembles: Combining multiple models for enhanced performance using a multistage approach. *Expert Systems*, 21:279 – 288, 11 2004. doi: 10.1111/j.1468-0394.2004.00285.x.

The dataset can be found at:
https://www.kaggle.com/antonyj453/urldataset