PhishScan

Group 10

Jack Hsieh
yhsieh37@asu.edu

Siddharth Ranjan
sranja18@asu.edu

Ian Oxley
ioxley@asu.edu

Naveen Kumar
nmanokar@asu.edu

Willem Grier
Wgrier@asu.edu

1 Introduction

1.1 Motivation

With the growing use of internet, scams have been more prominent than ever. The FBI's Internet Crime Complaint Center published in their latest report [5] that victims suffered over $4.2 billion in losses. While the common assumption is that elderly citizens are the most vulnerable group, the report shows that people over 60 only account for 23% of victims. In fact, business email account compromise, the most prevalent form of phishing attack, affects younger employees five times more often than older employees [8]. Notably, APWG's Phishing Activity Trends Report for 2023 [1] shows that the majority of attacks are perpetrated through systems provided by reputable corporations such as Google and Microsoft. Additionally, financial institutions are the most targeted sector, accounting for 23% of all attacks [1].

Human error appears to be the root cause of successful phishing attacks. Distraction, fatigue, and pressure to work quickly are all factors that contribute to humans making mistakes and becoming victim to phishing attacks while on the job [8]. While educating people on how to identify phishing attacks is important, the improvement of phishing tactics is rapid and may outpace for the rate at which humans can be trained. Therefore, a robust automated solution is required.

Just as AI has become an effective tool for writing emails and documents for personal and business use, cybercriminals have also begun utilizing AI *en force*. *WormGPT*, *FraudGPT*, *DarkBART*, and *DarkBERT* are chatbots with custom LLMs designed to aid cybercriminals develop phishing attacks [7]. People face challenges in identifying AI-generated versus human-generated content. These challenges have severe social and economic consequences when applied to cybersecurity. Consequently, an AI-driven approach is prudent in tackling this rapidly growing use of LLMs in cybercrime.

1.2 Problem Statement

The evident efficacy of certain machine learning models for classification tasks lends itself to addressing the problem of automated phishing identification. Phishing websites remain the most effective and common method of phishing [1]. We propose that machine learning models can be used to classify websites as phishing or legitimate.

The *Web Page Phishing Detection Dataset* [3] contains a balanced set of 11430 websites, exactly half of which are verified phishing sites. The 88 extracted features contain metadata about the URL structure and syntax, content of the web pages, and web traffic.

We propose studying the impact of embedding the URL into models. It is important because feature extraction and selection can take a long time and sometimes requires extensive knowledge of the domain, if utilizing transformers can yield similar or even better results, it means we can formulate a good model within a shorter time frame which might reduce the over all cost of a machine project in the text classification use-case.

We first obtain baseline results from a variety of classifiers. Additionally, the BERT transformer [2] is used to compare a multi-modal approach that tokenizes the web page URL. A comparative analysis highlights the need for improvements in methodology and the choice of classification algorithm for this task and dataset. From this insight, an integrated classifier combining the top two performing models is proposed, designed, and implemented.

1.3 Related Work

[6] discusses a deep learning approach to detect phishing web pages. The authors propose a framework using a multilayer perceptron (MLP), a type of feed-forward neural network, to classify web pages as phishing, suspicious, or legitimate. The dataset includes ten features for each web page. The model achieved 95% training accuracy and 93% test accuracy. The paper also suggests adding more layers to the neural network and using more advanced techniques like back-propagation for improvement. We add to this paper by exploring a wider range of classification methods and using a richer dataset of 88 features.

1.4 Technical Background

The baseline results are collected for the following classifiers:

- Logistic Regression
- Support Vector Machines
- Decision Tree
- Recurrent Neural Network
- Graph Neural Network
- Random Forest
- XGBoost
- KNN

1.4.1 Logistic Regression

Logistic regression is a statistical model used for binary classification tasks. It's commonly used binary classification (e.g., success/failure, yes/no, 0/1) based on one or more independent variables (features). Logistic regression is particularly useful when the dependent variable is categorical. The output is typically passed through a Sigmoid function and labels are determined by a set threshold.

1.4.2 Support Vector Machines

Support Vector Machine (SVM) is a supervised machine learning algorithm commonly used for classification and regression tasks. SVM works by finding the hyperplane that best separates the data points into two classes, maximizing the margin between the closest points of each class, known as support vectors. The decision boundary is determined by these support vectors, making SVM robust to outliers.

1.4.3 Decision Tree

A Decision Tree is a flowchart-like tree structure where each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label (decision). It is a simple yet effective tool for classification and regression tasks. Decision trees split the data into subsets based on the values of input features, recursively partitioning the dataset until each leaf node corresponds to a class label.

1.4.4 Recurrent Neural Network

Recurrent Neural Network (RNN) is a derived feed-forward approach which can also retain information from previous inputs by including a memory component. This memory component is referred to as long-short-term-memory (LTSM) and is what's responsible for making RNN different from other neural networks. RNN is commonly used for handling data in which order of the data matters, meaning it can effect later entries. This approach suits our project because phishing emails have correlated elements. For example, each character in the URL string can effect later characters and the overall semantics of the string.

1.4.5 Graph Neural Network

Graph Neural Network is a deep learning approach that takes graphs as input. This method captures relation between each connected nodes in the graphs. This model is very useful for geometric data such as the friends network system used on social media platforms. We decided to try this approach in our subject because it would be interesting to see how different graph formulation of the extracted features affects the result. [4] proposed a hierarchical GNN for text classification. In this milestone, we will try using a simpler GNN model as a baseline so we can see the impact of incorporating the URL embeddings.

1.4.6 Random Forest

Random Forest is an ensemble learning method used for classification and regression tasks. It operates by constructing multiple decision trees during training and outputting the class that is the mode of the classes (in the case of classification) or the mean prediction (in the case of regression) of the individual trees. Random Forest combines the simplicity of decision trees with the flexibility of ensemble methods, resulting in a model that is both accurate and robust to overfitting.

1.4.7 XGBoost

XGBoost is an advanced implementation of gradient boosting algorithm. It is a highly efficient and scalable algorithm that is widely used in machine learning competitions and real-world applications. XGBoost works by sequentially building an ensemble of weak decision tree models, where each new tree attempts to correct the errors made by the previous ones. It employs sophisticated regularization techniques to control overfitting, making it robust and accurate.

1.4.8 KNN

K-Nearest Neighbors (KNN) is a simple, non-parametric, and lazy learning algorithm used for classification and regression tasks. It is based on the principle of proximity, where the class of a data point is determined by the majority class among its k-nearest neighbors. The value of k is a user-defined parameter, and the distance between data points is typically measured using metrics such as Euclidean, Manhattan, or Minkowski distance.

2 Contributions

Jack Hsieh -	Complete experiments and evaluations on Graph Neural Network and the corresponding sections in this report.
Ian Oxley -	Responsible for planning, building, evaluating, and reporting the Recurrent Neural Network approach.
Willem Grier -	Organized and analyzed results for all methods. Implemented experiments for the Random Forest Classifier.
Siddharth Ranjan -	Analyzed the results and used Logistic Regression with Transformers for obtaining the results.
Naveen Kumar -	Analyzed and evaluated results for XGBoost, SVM, KNN and Decision Tree models. Formulated and recorded the results in the other models section of the Report.
All -	Discussion and Formulating of problem, experiment planning, discussion of results, drawing conclusions and completing the final report.

3 Experiments

In this section we highlight the details of each approach. While the overall results are shown in the Results section, we include approach-specific results here. For each of the examined methods, we evaluate the models both with and without using Boruta feature selection. Additionally, we evaluate models that have been trained both with and without using the URL feature as an input. For all methods, URL text features were extracted using Bert-Tiny. For evaluation, we utilize cross-validation and measure the f1 score. Since the dataset is balanced, there are less worry about bias of the f1 score.

3.1 Recurrent Neural Network

3.1.1 RNN Definition

The RNN structure is designed to fulfil a sequence of binary classification tasks. It's architecture comprises of an LTSM layer followed by two fully connected layers. This data structure implements the properties discussed in 1.4.4, allowing for a steady flow of the input and a single output result.

The model utilizes the following hyper-parameters:

- Input feature size: features in each input sequence, determining the size of the LTSM layer.
- Hidden size: number of hidden units in the LTSM layer, controlling model complexity.
- Number of Layers: number of stacked LTSM layers, making for a deeper model.
- Learning Rate: controls the rate at which the model optimizes during the training process.
- Epochs: How many times the entire training process is run.

The hyper parameters control various aspects of the neural network's training process allowing for better fine tuning and accuracy.

3.1.2 Training

The training occurs over a number of epochs, with each epoch iterating over a batch of data. For each batch, the model first utilizes Adam optimizer followed by prediction computation. After, it utilizes MSEloss to compute loss in comparison to the ground truth and then back-propagation process to update the model parameters based on the loss. We also accumulate loss throughout this process, allowing us to monitor and understand the training process for hyper parameter tuning.

3.1.3 Evaluation

The evaluation occurs by utilizing the same process described in the previous section except utilizing a test dataset and without back propagation since we don't want to update the model. Once the entire dataset is processed, we analyze the sensitivity, specificity, accuracy, precision, and F1 score of the produced data. These metrics provide insight into the model's performance, allowing for ease of tuning the hyperparameters.

Scenario 1 **All features except the URL**
For this network, we include all the features except URL since URL is a text feature. As stated in 1.4.4, we expect including URL to benefit RNN, thus this is more utilized as a baseline and quick training process.

Scenario 2 **Only the URL embeddings**
For this network, we only include the features after the bert tokenizer on URL has ran, resulting in a total of 128 features. We found that this scenario was unable to reach the same accuracy as the previous scenario, believing this is due to the LTSM properties RNN has benefiting from more types of data.

Scenario 3 **All features including the URL**
For this network, before training, we utilize Bert tokenizer mentioned in 3.1 to tokenize the url text feature into a format the model can read. This results in an extra 128 features compared to the original 89, resulting in an input shape for LTSM of 215. As predicted in the milestone report, including URL has better results than not including URL

We also tested both scenarios above with and without Boruta feature selection and found scenario three with Boruta had the best performance. We believe this is due to RNN benefiting from the input being simplified while still having access to the url feature whose important with LTSM was discussed in 1.4.4. On the other hand, neural networks have built in feature selection making these results confusing. From what we can gather, we believe this phenomenon will only be reflected on RNN due to the LTSM layer. As a result, we did not include Bourta in GNN discussed later.

3.1.4 End Results

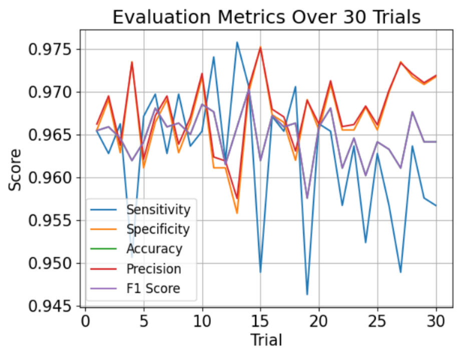

Figure 1: RNN Evaluation Scores

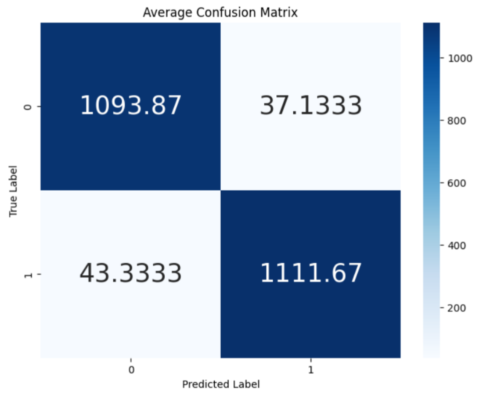

Figure 2: RNN Confusion Matrix Average

After spending ample time tuning the hyper-parameters, we found that the best results were produced when we reduced to a single layer with a hidden size of 256 and batch size of 64. We would train with a learning rate of 0.005 over 10 epochs. This would result in an RNN model complex enough to train with all the features without resulting in over fitting. With this approach, we were able to get an average F1 score of 0.9628 with a 0.003 error between trials. Figure 1 shows the evaluation scores mentioned in 3.2.3 across 30 trials and Figure 2 shows the average confusion matrix across those thirty trials.

3.2 Graph Neural Network

3.2.1 Creating the Graph

Before we can train our GNN, we need to process our input into a graph. For this, we assign an arbitrary ID (an unique integer) to each data point and connect the feature nodes to it depending on the scenario.

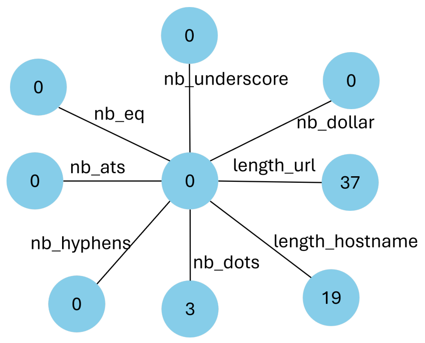

Figure 3: Input graph example of GNN

Scenario 1 **All features except the URL**
For this graph, we simply make every feature other than the URL text feature into separate nodes and form undirected edge indexes between the features and the supernode.

Figure 3 shows a truncated example of the graph, in the actual graph that we were using, there were 87 feature nodes.

Scenario 2 **Only the URL embeddings as single node**
For this graph, we first tokenize and and transform the url text into embeddings by using the Bert-tiny model, which gives an output of size 128, this size allows the model to run in a more manageable time. Then we have to also increase the size of the supernode to 128 to match it. The values of the supernode's vector were filled in by the same integer or ID the data point was assigned to. Now that we have an input of shape (2,128), simply connect the two with an undirected edge index will form our desired graph.

Scenario 3 **Only the URL embeddings as separate nodes**
For this Scenario, we still decided to try and make the embeddings into different nodes and connect them to the supernode. Since GNNs learns the relation and draws information from connected nodes, by making the embeddings into different nodes, it is possible that this will also work as well as scenario 2.

3.2.2 Training and tuning the GNN

Training is done simply by feeding the established graph of each datapoint into the GNN and optimzied using the Adam optimizer. The GNN consists on a graph convolutional layer with ReLu activation, and a linear output layer that was applied by Sigmoid activation function. Before and after the convolutional layer are batch normalization to mitigate some impact of some larger range features such as "length_url" and "length_hostname". We apply a dropout rate after the convolutional layer and the output layer to prevent from overfitting. We tried BCE loss and MSE loss for optimization and MSE gave on average 20% better scores.

Hyper parameters for tuning includes hidden dimension, dropout rate, and learning rate. Parameters that we've experimented with are shown in Table 1. Each parameter combinations ran through 20 epochs for each of the 5 folds. After tuning, 256 hidden layer dimension, 0.2 dropout rate and 1×10^{-4} seems to give the best results for each scenario.

Table 1: Hyper-parameters of tuning GNN

Hidden layer dimension	Dropout Rate	Learning Rate
128	0.2	1×10^{-4}
256	0.25	5×10^{-4}
	0.3	1×10^{-3}

3.2.3 Evaluation and Results

For evaluation, we ran five-fold cross-validation, tuned each model to our best effort and got the best average f1 score for each scenario. Table 2 shows the best average f1 score for each scenario where we can see that by only using the embedding, we were able to get better results than by using the other manually extracted feature.

One difference of the GNN structure between the scenarios is that in scenario 2 and 3, where we only used the URL embeddings, we dropped the batch normalization before the first graph convolutional layer because this gave better results.

Table 2: Best average f1 score of GNN model

Scenario 1	Scenario 2	Scenario 3
0.6334	0.6666	0.6459

3.3 Logistic Regression

3.3.1 Overview

We are using the Logistic Regression model from the Scikit-Learn library in this experiment. For the following scenarios, we've evaluated our model by doing a 75:25 split to the dataset. We've also done a five-fold cross validation to confirm results.

Scenario 1 **All features except the URL**
For this case we dropped the URL feature from the dataset and performed data cleaning on the remaining dataset. After this we performed a train test split in ratio of 75:25 and obtained a F1 score of 0.9439 by using Logistic Regression.

Scenario 2 **All features except URL using Feature Selection**
For this case we dropped the URL column from the dataset and performed feature selection using Boruta. Boruta selected 25 features which were passed to Logistic Regression model. The obtained F1 score was significantly less than the other two cases. To confirm that this is the case, we've done cross validation for this scenario and got a score of 0.9327.

Scenario 3 **All features including URL**
For this case we utilized the URL column by using Transformers(BERT). After utilizing the transformers and obtaining the embeddings of the URL column. We concatenated it with rest of the columns and performed a train test split in ratio of 75:25 and obtained a F1 score of 0.9639. Including URL gives better result as URL helps the model to better understand texts content and meaning. By adding URL the model becomes more accurate at classifying texts.

The overall result comparison between the scenarios are similar to that of other methods.

3.4 Random Forest

We examine the accuracy of the Random Forest classifier for cases with and without feature selection as well as with and without using the URL as a feature input. For the case with feature selection and URL utilization, the process is the following:

1. Extract URL text features using Bert-Tiny (128 length)
2. Append URL text features to the original dataset
3. Use Boruta Random Forest to select all significant features.
4. Perform hyper-parameter training with 5-Fold Cross Validation to attain the best RFC model
5. Evaluate model on the 80/20 test set
6. Repeat for 10 random seeds

Table 3: Hyperparameter tuning results

	No URL		URL	
	No Boruta	Boruta	No Boruta	Boruta
Features	87	39	215	171
Tree Depth	15	16	15	15
Estimators	319	293	319	278

Feature Selection We find in Table 3 that feature selection significantly reduces the complexity of the random forest model, reducing the number of estimators by approximately 10%. The maximum depth of the trees is consistent across all configurations.

URL Utilization We find in Table 4 that the Random Forest approach yields better accuracy without utilizing the URL text features.

3.5 Other Models

First the min max scaler was used to normalize the numbers for all the numerical columns. A bert based tokenizer was used to tokenize the url column which was text. The accuracy for different models lie XGBoost, Decision Tree, SVM, KNN were calculated without the url columns using multiple feature selection methods like Boruta and scikit-learn feature selection out of which the Boruta feature selector was finalized for feature selection since it gave comparatively better results. Then the accuracy for the model for different scenarios were calculated and the scenarios giving better results were finalised for model selection. 5 fold cross validation was used to obtain the best model parameters and a 75:25 train test split was utilised to get the model accuracy for all the scenarios listed below. The different scenarios used are explained in brief below

Scenario 1 **All features except the URL** The accuracy for all the models was first calculated without the url column and without using any feature selection methods. Surprisingly, just doing min max scaler for the numerical columns were sufficient enough to obtain above average results for all the models.

Scenario 2 **All features except URL and using Feature Selection** The accuracy for all the models were then calculated for the finalized feature selector Boruta. The Boruta feature selector was used with 5 fold cross validated XGBoost model to finalize 26 features out of the 87 feature list. Though the accuracy for the models when using Boruta dropped the drop was very less and non noticeable.

Scenario 3 **All features including URL** The accuracy was calculated for the models including the url column by embedding the url column using Bert-Tiny(128 length). The url columns addition proved to be extremely game changing in identifying the phishing links since most of the models accuracy increased with SVM even getting an accuracy of about 0.9702 from the previous 0.9576 with an increase of over ± 0.002

4 Results

Table 4: F1 scores for each approach

Model	No URL		URL	
	No Boruta	Boruta	No Boruta	Boruta
Logistic Regression	0.9439 ± 0.003	0.9327 ± 0.003	0.9639 ± 0.003	-
SVM	0.9576 ± 0.003	0.9536 ± 0.003	0.9702 ± 0.003	-
Decision Tree	0.9384 ± 0.003	0.9387 ± 0.003	0.9402 ± 0.003	-
RNN	0.9614 ± 0.003	0.9621 ± 0.002	0.9613 ± 0.004	0.9628 ± 0.003
GNN	0.6334 ± 0.003	-	0.6666 ± 0.008	-
Random Forest	0.9638 ± 0.004	0.9646 ± 0.008	0.9624 ± 0.004	0.9621 ± 0.003
XGBoost	$\mathbf{0.9713 \pm 0.003}$	$\mathbf{0.9702 \pm 0.003}$	$\mathbf{0.9737 \pm 0.003}$	-
KNN	0.9499 ± 0.003	0.9383 ± 0.003	0.9408 ± 0.003	-

XGBoost is the best-performing classifier overall. However, our research is more interested in the influence of the URL text features on performance. We see in Figure 4 an increase in the overall average F1 score from 91.33% to 92.12% when the URL is used as a feature. 6 of 8 classification methods improve from the URL text features.

However, Boruta feature selection did not prove to be as useful. We see in Figure 5 a decrease in the overall F1 from 91.66% to 91.57%. Only 3 of 8 methods improve overall when using feature selection. RNN benefits from both Boruta and URL utilization. This is probably due to the model becoming less complex as the number of features is reduced while still taking full advantage of LTSM with the URL feature.

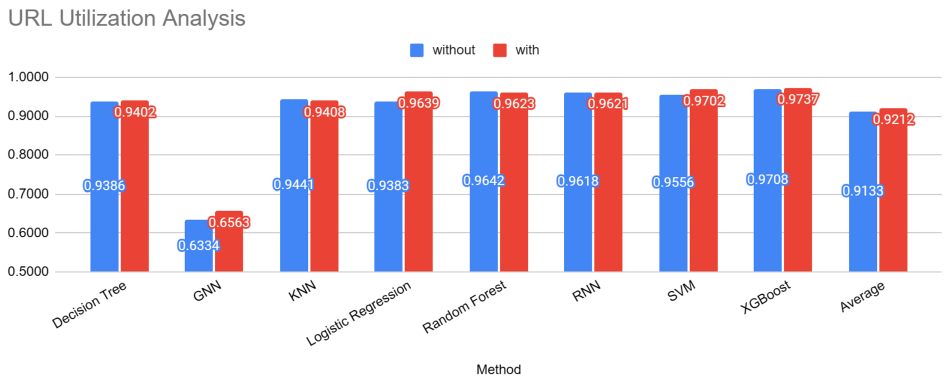

Figure 4: Performance Impact of Using URL Features

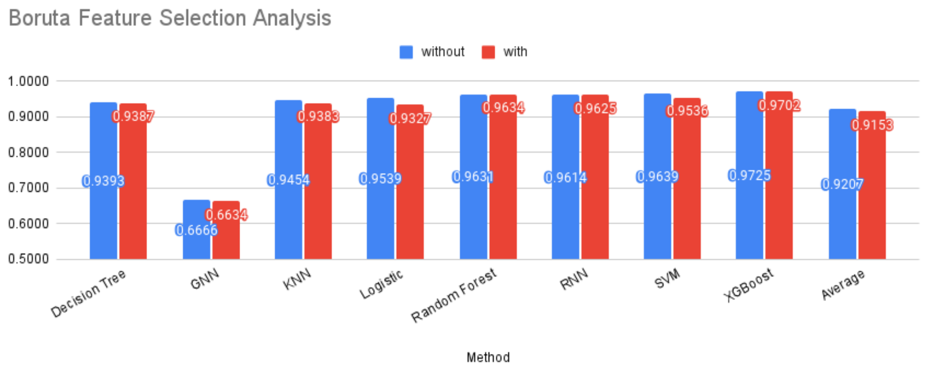

Figure 5: Performance Impact of Using Boruta feature selection

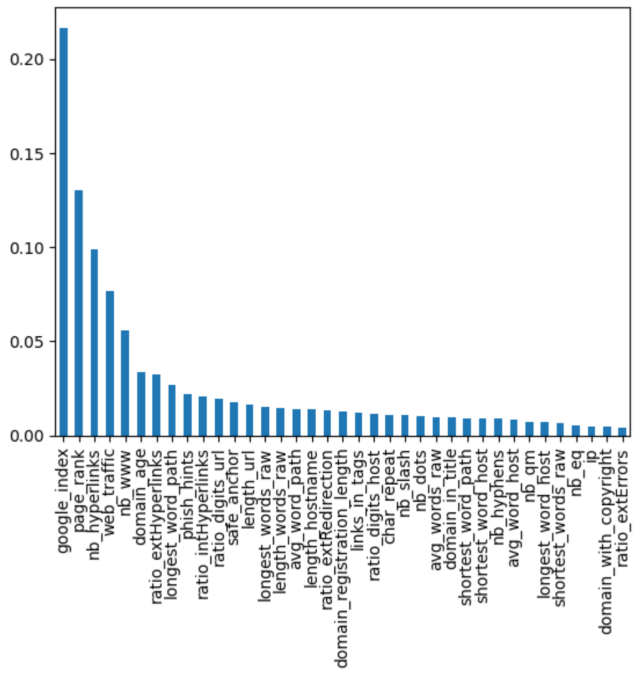

Figure 6: Significance of the 39 features selected by Boruta. 48 features were discarded.

5 Conclusion

We've achieved our goal proposed in our milestone, which is to study the impact of text embeddings to different methods. We've also combined Random Forest (Boruta for feature selection) with most other methods to compare the results.

Though the accuracy for the models when using Boruta dropped the drop was not significant. Feature Selection might be very useful for cases in which there are many hundreds of features since it helps in achieving comparatively similar accuracy with around quarter of the original features which reduces the computational effort required to train the model. We see this benefit in the Random Forest classifier, where using Boruta reduced the forest size by over 10%.

We can see in Figure 4 that the accuracy by using the URL embeddings increased for most models where GNN, KNN and SVM benefited the most. This shows that instead of having to spend time on extracting and selecting features in a text binary classification use-case, using tokenizers and transformers to embed it into models might be a good alternative.

References

[1] APWG (2023). Phishing activity trends report, 2nd quarter 2023.

[2] Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

[3] Hannousse, A. (2021). Web page phishing detection.

[4] Hua, S., Li, X., Jing, Y., and Liu, Q. (2022). A semantic hierarchical graph neural network for text classification.

[5] IC3, F. (2021). 2020 internet crime report.

[6] Saha, I., Sarma, D., Chakma, R. J., Alam, M. N., Sultana, A., and Hossain, S. (2020). Phishing attacks detection using deep learning approach. In *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 1180–1185.

[7] SlashNext (2023). The state of phishing 2023.

[8] Tessian (2023). Psychology of human error 2022: Research report.