SMS Smishing Detection

Jorge Pais
KU Leuven Group T
Programme for exchange students
jorgemiguel.lopespais@student.kuleuven.be

Senne De Greef
KU Leuven Group T
Dutch Programme
senne.degreef@student.kuleuven.be

Abstract

This paper presents a comprehensive study on the detection of SMS smishing using novel machine learning techniques. We employed a dataset comprising various SMS messages, categorized as ham, spam, or smishing, and explored multiple machine learning models including Support Vector Machines (SVM), Random Forest Classifier (RFC), Naive Bayes, and neural networks. Our study involved extensive preprocessing of the SMS text, including tokenization, removal of stopwords, and lemmatization, to prepare the data for model training. We evaluated these models based on accuracy and the Receiver Operating Characteristic Area Under the Curve (ROC AUC) metrics. Our findings indicate that while most models achieved high accuracy, the Support Vector Machine with an RBF kernel and the Random Forest Classifier exhibited superior performance in SMS message classification.

I. Introduction

The exponential growth of the internet and communication technologies in recent years has brought many benefits like, revolutionized communication, commerce, and information exchange. However, this progress has also given rise to scams and threats. Among these threats, email phishing and SMS smishing attacks have emerged as a pervasive and sophisticated method to compromise sensitive information or spread malware. The increasing complexity of these attacks poses a significant challenge to traditional security measures. That is why we think there is an urgent need for more accurate models to identify these attacks, and in this way prevent that such email and SMS messages show up in user inboxes mitigating this type of attack vectors.

We found a dataset that contained multiple SMS messages. The input of our algorithms took the raw body text of those SMS message, whether the SMS contained any URL, e-mail or phone number. Based on this we trained many different models like SVM, RFC and Naive Bayes. The models were trained to classify the SMS message between ham, spam or smishing.

II. Related Work

Our model implementations were guided by insights learned from different research papers ([1], [2], [3], [4]) that conducted an examination of diverse classifiers, encompassing Naive Bayes, Random Forest, SVM, and logistic regression on a dataset that contained multiple phishing e-mails. The authors systematically evaluated these classifiers with a focus on metrics such as accuracy, recall, precision and F1-scores. Most papers found superior overall performance of Support Vector Machines (SVM) and Random Forest, outshining Naive Bayes and logistic regression, which exhibited slightly diminished accuracy rates.

In our research we want to implement these models, and some extra, to confirm their results. And take a deeper look in the optimal options to classify SMS messages on ham, spam or smishing. Because our dataset is similar as the one used in research paper ([1]), we hope to find some similar results in the evaluation of our implemented models.

III. Dataset and Features

The dataset utilised is titled ”SMS Phishing Dataset for Machine Learning and Pattern Recognition” [5], which consists on a collection of labelled text messages, aimed at being used for Smishing and Phishing text detection, in this cased more focused for SMS smishing detection models. The dataset has a total of

5971 SMS messages, which have been labeled under the following 3 categories: Ham/legitimate (4844), Spam (489) and Smishing (638). As for features, the dataset features following:

- `TEXT` The raw text content of each message
- `URL` Indicating whether the SMS contained any URL (boolean)
- `EMAIL` Indicating whether the SMS contained any email address (boolean)
- `PHONE` Indicating whether the SMS contained any phone number or not (boolean)

IV. Methods

Because our dataset was unbalanced, we started by doing some oversampling on the minority classes before splitting the training and testing data. After this, we did the necessary data pre-processing. Then we trained different models and compared them by different performance metrics.

A. Text Pre-processing

Before building utilizing any machine learning models, some preprocessing of the data is required. The primary objective of this `TextPreprocessing` class is to prepare textual data for analysis by cleaning and transforming it. The process begins with the removal of numerical values and punctuation from the text, followed by converting the entire text to lowercase to ensure uniformity. The text is then tokenized into individual words, effectively converting it into an array of discrete elements. To refine the dataset further, common English stopwords, which contribute little semantic meaning, are excluded from the text body. Finally, the lemmatization process is applied to the remaining words, ensuring that different grammatical forms of a word are reduced to their root form.

For all the other columns that weren't the text body, ordinal encoding was utilised in order to convert the string values into a numerical representation which can be interpreted directly.

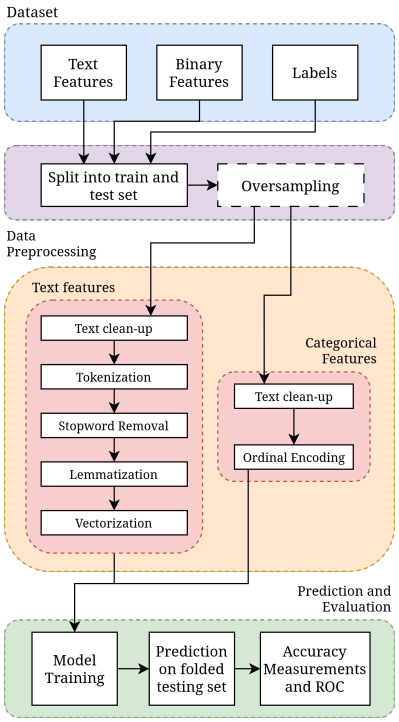

Fig. 1. Model Structure

B. Models

1) Logistic Regression: A Logistic Classifier, is a foundational algorithm utilized for binary classification tasks. The logistic classifier is represented as $f_{\mathbf{w},b}(x) = g(\mathbf{w}\cdot\mathbf{x}+b)$ which uses the sigmoid function, denoted as $g(z) = \frac{1}{1+e^{-z}}$. This function serves to map real-valued inputs z to the range (0, 1), providing a probabilistic interpretation of the output. While training/fitting the model a gradient descent algorithm is used to iteratively optimize the model parameters, namely the weight vector w and bias term b, and thus minimizing loss function. The regularization term is governed by the hyperparameter λ.

For the purpose of this work, the `LogisticClassifier` class was implemented in order to use the logistic regression code developed in the assignments within our *SKlearn* [6] pipeline. Since the memory performance of our code is not optimal, especially considering the number of features after tokenization and vectorization, *SKlearn's* `LogisticRegression` classifier was utilised along with `OneVsRestClassifier` in order to build 3 binary classifier models, one for each class.

2) Naive Bayes Classifier: The Naive Bayes is a probabilistic classification algorithm that makes the assumption that dataset features, follow a Gaussian (normal) distribution. The fundamental principle behind Naive Bayes is Bayes' theorem, which relates the posterior probability $p(\mathcal{C}_k|x_1, ..., x_n)$ of a class given the features to the prior probability $p(\mathcal{C}_k)$ of the class and the likelihood of observing those features given the class $p(x_1, ..., x_n|\mathcal{C}_k)$ in the following way:

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathcal{C}_k)p(\mathbf{x}|\mathcal{C}_k)}{p(\mathbf{x})}$$

The name naive comes from the (indeed naive) assumption that the dataset features are conditionally independent from each other. In this case the main problem is that of determining the prior probabilities, since the class posteriors are simple to infer. The case of Gaussian Naive Bayes, the priors are calculated the following matter:

$$p(x = v|\mathcal{C}_K) = \frac{1}{\sqrt{2pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Multinomial Naive Bayes is designed for discrete data, which is better for text classification. It assumes that the features are generated from a multinomial distribution. The maximum likelihood estimates for the prior probabilities in this case are:

$$p(\mathbf{x}|\mathcal{C}_k) = \frac{\#D\{x_i = x_{ij} \wedge y = \mathcal{C}_k\}}{\#D\{y = \mathcal{C}_k\}}$$

For the naive bayes classifier, the `GaussianNB` and `MultinomialNB` classes were utilised, using the same one-vs-all strategy as before. A `FunctionTransformer` is also required when using these models, as these do not accept *NumPy* sparse arrays as input.

3) Random Forest Classifier: Random Forest is a ensemble machine learning method, meaning that it makes use of multiple machine learning algorithms. It relies on a multitude of Decision Trees, in which each tree is trained on a random subset of the training data which is typically referred to as bootstrapping or bagging. This increases the performance of what a single decision tree is able to offer, since these tend to overfit as they grow deeper.

The key concept is that Random Forests introduce extra randomness when building the decision trees, where instead of searching for the most optimal feature whilst splitting a none, it search for a that feature from a random subset of all features. For classification tasks, each decision tree in the forest votes for a class, and the class scores are averaged out from the number of trees that decided on it.

For this case the `RandomForestClassifier` was utilised, with the number of estimators set to 1000. With more estimators in the forest, it became noticeable that the training and inference performance decreased.

4) Support Vector Machine: The main idea behind Support Vector Machines is to find an optimal hyperplane in a N-dimensional space that is capable of separate data points into different classes. The hyperplane can be written a the set of points $\mathbf{x}$ satisfying $\mathbf{w^T x} - b = 0$, where $\mathbf{w}$ and b are the parameters to optimize. In the case where the input data is linearly separable (and the dataset is normalized), it is possible to define two hyperplanes $\mathbf{w^T x} - b = 1$ and $\mathbf{w^T x} - b = -1$, such that anything above the first is off one class, and everything below the latter is off another class. In this case, the hyperplane in between these two is referred to as the maximum-margin. All data points are treated as vectors, but only the data points which are outliers are used to calculate the hyperplanes.

For most problems though usually data classes are not linearly separable, and thus some transformation need to be performed. To tackle this, kernel functions can be utilized to transform the data boundaries in order to better fit the nature of the data:

- *Linear kernel:* $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)$
- *Polynomial (d degree):* $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d$
- *Polynomial (up to d degree):* $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d$
- *Gaussian radial basic function:* $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d$
- *Sigmoid Kernel:* $k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\eta x_i \cdot x_j + v)$

For our models, the `SVC` support vector classifier from *SKlearn* was utilised, which allows us to set specific kernels and indicate whether or not to take class weight/size into consideration, eliminating the need for over/undersampling the data and possibly loose out on performance.

5) Neural Network: The concept of a neural network includes various types of networks. However, one of the most basic types is a feedforward network, which consists of an input layer, one or more hidden layers, and an output layer. Each layer is made up of nodes, or "neurons," with each node in a layer connected to every node in the next layer. Each neuron, receives input from some other nodes and computes an output. Each unit's input has an associated weight (w), which is assigned based on its relative importance to other inputs. The node applies a activation function f (defined in advance) to the weighted sum of its inputs.

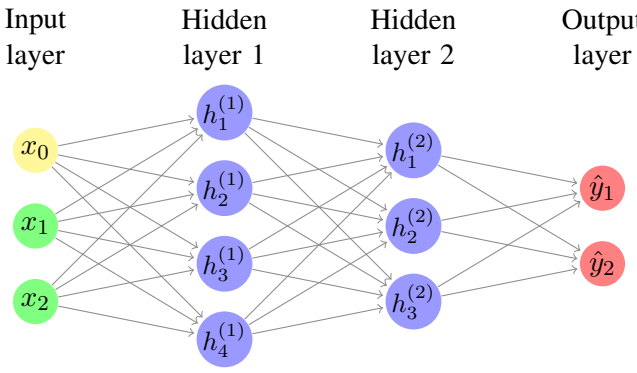

Fig. 2. Example of a artificial neural network's structure

For our model, the *Keras* framework [7] was utilized to develop a neural network based on the code from the assignments. The network developed consisted on a embedding layer, which converts the categorical data that results from the text preprocessing into continuous data, which then the model learns to map the words appropriately. Then this is followed by a 15 unit fully connected layer and finally the 3 output nodes.

C. Performance Metrics

The models underwent assessment based on two key performance metrics. Initially, accuracy was computed to measure the overall effectiveness of each model in performing a specific task, utilizing the formula:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100\%$$

While accuracy is widely used, it's suitability diminishes in cases of imbalanced datasets, where one class prevails significantly. Consequently, the evaluation extended to the Receiver Operating Characteristic Area Under the Curve (ROC AUC) metric. This metric, used in classification models captures the model's proficiency in distinguishing between true and false positives, graphically represented by the ROC curve. The Area Under the Curve (AUC), a concise metric ranging from 0 to 1, encapsulates this performance, with higher values indicating superior discrimination.

V. Results/Discussion

In general, our models achieved consistently high accuracies. However, indications of potential overfitting in our trained models should warrant further examination. In this section, a concise evaluation of each model's performance for text-based SMS classification is provided.

We started our analysis, depicted in figures 3 and 4, with the Gaussian Naive Bayes model which exhibited the lowest performance. This outcome aligns with expectations, as the model assumes a Gaussian distribution for feature data. However, our text messages were vectorized using TF-IDF, resulting in a highly complex non-normally distributed dataset. Additionally, binary features (URL, email, and phone) are not optimal for this model, which is better suited for continuous features.

The Multinomial Naive Bayes model yielded superior results, excelling in classifying the discrete data. This model constructs histograms of common words in ham, spam, or smishing messages. By incorporating Laplace smoothing ($\alpha = 1$), the model addresses issues arising from words present in only one class, preventing biased classification.

Following Naive Bayes, we employed the Random Forest Classifier, generating multiple decision trees on bootstrapped datasets. While increasing the number of trees (n) improves generalization, computational demands must be considered. The Random Forest Classifier with n = 1000 demonstrated diminishing returns in performance relative to increased computational requirements. Overall, this model proved to be among the best-performing options for SMS message classification.

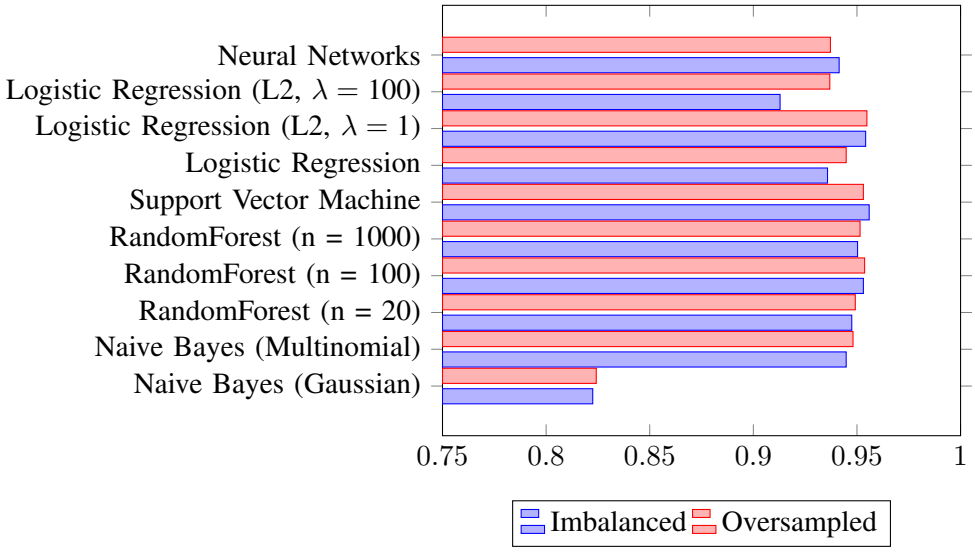

Fig. 3. Accuracy score across classifiers

The Logistic Regression model was also employed, initially without regularization, resulting in potential overfitting. Introducing L2 regularization addressed this issue, with λ values of 1 and 100. While a lower λ yielded a flexible model prone to overfitting, a higher λ enhanced resistance to overfitting at the expense of predictive power.

The Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel exhibited the most favorable performance metrics. The RBF kernel effectively captured complex, non-linear relationships inherent in textual data. Using linear or poly models exhibited very poor performance, so they were disregarded from this analysis.

Finally, a neural network was implemented, featuring a 4-layer sequential model with an embedding layer, flatten layer, a dense layer with 15 units and ReLU activation for handling non-linear relationships, and a final dense layer with 5 units using the softmax function for class probability distribution. Training over 100 epochs aimed to mitigate overfitting and enhance model performance.

In summary, our comparative analysis suggests that the Support Vector Machine with an RBF kernel and the Random Forest Classifier are particularly robust choices for SMS message classification, while the Multinomial Naive Bayes model also presents competitive performance. Gaussian Naive Bayes is the worst performing model and should in general be avoided to use in text classifying problems.

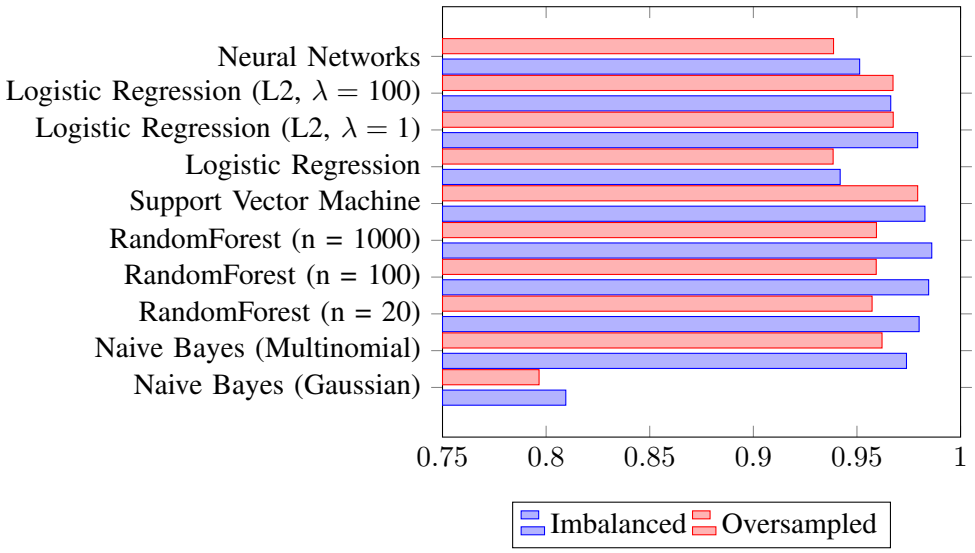

Fig. 4. ROC AUC score across different Classifiers

VI. Conclusion

From the work developed and the results obtained, some conclusions could be made. Firstly, that the unbalanced nature of the dataset utilised lead to some possibly misleading results, as the performance of most models employed was generally very high, indicating that there might be some overfitting occurring in some of them. Oversampling data points sort of helped, but it is not a perfect solution.

Secondly, even though most models achieved high accuracy values on the folded dataset, the difference in ROC score displays them different ability between them in distinguishing between classes. As for possible future work, this could include gathering (or even generating) a bigger dataset for which larger models such as deep neural networks models can be trained against. BERT models are also a compelling option to explore, as they are one of the ubiquitous baselines for natural language processing.

It can overall be said that all objectives set out for this project were met with great success, and that the work developed here helped the students to deepen their knowledge of different machine learning models, including some beyond the ones that were taught during the lectures, and also to gain some insight into the textual data processing.

VII. Contributions

Throughout our project, we both actively engaged in various aspects, sharing the workload and responsibilities. Jorge took the lead in handling text preprocessing and setting up the *SKlearn* pipeline that was needed for training the different models, including Naive Bayes, Logistic Regression, Support Vector Machines (SVM), and Random Forest Classifier (RFC). Senne focused on researching related works and developed the neural network. He also visualised the metrics data of all the models.

Our communications were clear and we made a conscious effort to meet every two weeks, where we collaborated on writing the reports, implementing and testing code, and creating our project poster. In general we can say that our teamwork was successful.

REFERENCES

[1] S. Rawal, B. Rawal, A. Shaheen, and S. Malik, "phishing detection in e-mails using machine learning - researchgate," Oct 2017. [Online]. Available: https://www.researchgate.net/publication/320257918_Phishing_Detection_in_E-mails_using_Machine_Learning

[2] H. E. Karhani, R. A. Jamal, Y. B. Samra, I. H. Elhajj, and A. Kayssi, "Phishing and smishing detection using machine learning," in *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2023, pp. 206–211.

[3] C. Balim and E. S. Gunal, "Automatic detection of smishing attacks by machine learning methods," in *2019 1st International Informatics and Software Engineering Conference (UBMYK)*, 2019, pp. 1–3.

[4] T. Choudhary, S. Mhapankar, R. Bhddha, A. Kharuk, and D. R. Patil, "A machine learning approach for phishing attack detection," *Journal of Artificial Intelligence and Technology*, vol. 3, no. 3, p. 108–113, May 2023. [Online]. Available: https://ojs.istp-press.com/jait/article/view/197

[5] S. Mishra and D. Soni, "Sms phishing dataset for machine learning and pattern recognition," Jun 2022. [Online]. Available: https://data.mendeley.com/datasets/f45bkkt8pr/1

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[7] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.