ECE 597 Capstone Project: Machine Learning for Phishing Detection

Date: August 6th, 2024

Payton Murdoch, V00904677 (MTIS)

Dhruvit Boricha, V01047102 (MTIS)

Raunak Ghetla, V01047108 (MTIS)

Oche Eko, V01033252 (MTIS)

Joe Nikesh Puthota John, V01033708 (MADS)

Koushik Sivarama Krishnan, V01031395 (MADS)

Swathi Gnanasekar, V01033644 (MADS)

Table of Contents

Table of Contents....1
Abstract....2
Introduction....2
Literature Review for ML Phishing Detection....3
Experiment....4
  Preprocessing....5
    Text Normalization....5
    Lemmatization....5
    Merging of the datasets....5
    Merging of subject and body columns....5
    Setting Target Column....5
  Feature Engineering....6
    Bag-of-Words (BoW)....6
    Term Frequency-Inverse Document Frequency (TF-IDF)....6
    Word Embeddings (Word2Vec)....7
  Model Construction and Tuning....8
Results....9
  Table of Results....9
    Bag of words....9
    TF-IDF....10
    Embeddings....11
  F1 vs AUC score....11
Discussion....12
Complications/Future Works....13
Conclusion....14
References....16

Abstract

As social engineering attacks rely on exploiting people rather than the technology they utilize, it's no surprise that fraudulent and phishing emails are an ever-persistent threat to the business landscape. Therefore, it is imperative to employ automated methodologies to detect and block phishing-type emails by scanning and extracting subtle contextual information to discern it from real traffic. This brings us to Machine Learning models, which can be trained based on specific features ranging from word counters to statistical prediction of neighbouring words. Utilizing phishing data provided by the UVic Systems team, we have been able to examine and evaluate the performance of four models, Logistic Regression, Naive-Bayes, Random Forest and Neural Networks, through three different features so that we can recommend the best-performing model for this task.

Introduction

In businesses, email is the primary means of communication. While it is an efficient contact method, attackers have become increasingly persistent in developing strategies to exploit the users through deception. This brings us to phishing, a social engineering method that takes advantage of people by tricking them into sharing sensitive information through email, websites or even phone calls that can appear to be legitimate.[1] In Canada alone, since 2021, there have been over 150,000 reported instances of fraud by these means, resulting in approximately $600 million.[2] Email phishing alone is an increasingly frequent threat to the business landscape. In 2023, a survey conducted with select organizations reported that 94% of them reported at least one incident where an employee fell victim to a phishing attack and that they had received approximately 21 million phishing-related emails.[3] This is, of course, just a fraction of phishing and fraudulent emails circulating in today's landscape; Microsoft alone in 2022 blocked over 70 billion fraudulent emails, and within those emails, there were approximately 531,000 phishing URLs taken down by the company.[4] Furthermore, Google also reports that its email service, Gmail, blocks over 100 million phishing-related emails daily.[5]

With such a persistent threat, we must understand how businesses employ defensive measures to combat such a problem. Through multiple layers of defence, from multi-factor authentication to proper cybersecurity training, an organization hopes to minimize the efforts of attackers utilizing phishing attempts.[6] The most important defence mechanisms to explore are email scanners and filters, which parse email content and discern whether or not malicious content may be present and block them accordingly. Accomplishing this is a challenging task. However, businesses have taken advantage of advanced AI methods to automate this process and provide reliable, albeit not impenetrable, defence.[7] This report will explore one such application of AI detection methods, Machine Learning. The following sections will review select literature on machine learning for phishing detection to establish a series of proven, reliable models. Furthermore, we will describe our approach for the experiment and the aggregated results, which will be discussed and compared. Lastly, we will discuss the complications of our approach and possible project extensions.

Literature Review for ML Phishing Detection

Phishing is a social engineering attack that can not exploit computer vulnerabilities but targets computer users to be successful. Hence, phishing detection would be deployed efficiently using machine learning algorithms that learn trends and patterns in phishing emails and thus distinguish between these phishing emails and real emails.
Recent technology developments have leveraged various machine learning algorithms for improved phishing email detection. The focus algorithms for this project are:

1. Logistics Regression Model
2. Naive-Bayes Model
3. Neural Model
4. Random Forest Classifier

Logistic Regression is a simple-to-use linear model for binary classification. It works especially well with smaller datasets and in cases with a roughly linear relationship between the features and the target variable. [8],[9] These reports demonstrate its effectiveness and usefulness in phishing detection, making it a popular option for baseline models and scenarios with constrained computational resources.
The Naive Bayes classifier is a probability-related algorithm that relies on the Bayes theorem and assumes feature independence. Because of its simplicity, Naive Bayes is computationally efficient and especially well-suited for problems involving text classification, which are prevalent in phishing detection (e.g., content analysis of emails). [10],[11] demonstrates how successful Naive Bayes is in detecting phishing attempts even though it relies on the feature independence assumption, which is not always true and can occasionally affect accuracy.
Deep learning models, in particular, have demonstrated remarkable effectiveness in several categories, including phishing detection using Neural Networks. These models may learn complicated representations and patterns from data, frequently outperforming conventional approaches in difficult tasks. [12],[13] elaborates on how neural networks outperform other methods for phishing detection, highlighting how they may adjust and get better with more data. However, the main disadvantages include the possibility of overfitting, the need for significant computer resources, and a big training dataset.
Using the Random Forest ensemble learning technique, many decision trees are constructed during training, producing a class that is the mean of the courses of the individual trees. This technique is especially good at handling big datasets with plenty of features since it can catch intricate patterns in the data and is resistant to overfitting. [14],[15] These investigations have shown how reliable and accurate it is at detecting phishing attempts. However, the drawbacks of Random Forest include the difficulty in understanding the intricate model and its high computational demands.

The various machine learning techniques captured have pros and cons for application and use cases. While Neural Networks and Random Forests may capture complicated data patterns and deliver great accuracy, they, in turn, demand a large amount of computational power and resources. Conversely, effective and understandable models are offered by Naive Bayes and Logistic Regression, making them appropriate for scenarios requiring minimal processing power and real-time applications. The particular needs of the phishing detection system, such as the ideal ratio of interpretability, accuracy, and resource availability, frequently determine the best model selection for any application case.

Experiment

Given that we have developed a firm understanding of the Machine Learning models we hope to employ for this task, we will discuss the experiment's setup. Before beginning this experiment, we were given access to two datasets. The UVic Systems staff provided the first dataset, which was composed solely of phishing emails. To ensure the privacy of the data we have been given access to, the emails were sanitized so that we only got the subject and body content of the emails and no other network-related data. The second dataset is composed solely of real emails and contains additional network information that would need to be parsed so that it can match the phishing set data. This section is divided into three parts to describe the experimental procedure. The first section discusses the data preprocessing to fit it for the next part, feature engineering. Following the features, we discuss the construction and tuning of models, as well as their parameters and threshold probabilities, to improve the overall performance of the models. In general, the process for our experimentation can be summarized in the Figure below.

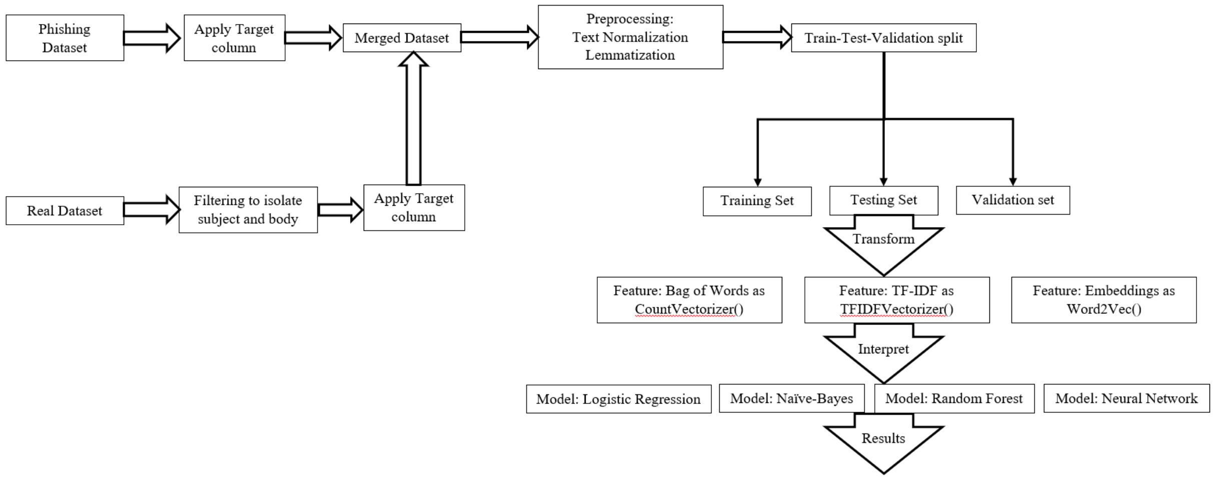

Phishing Dataset
Apply Target column
Merged Dataset
Preprocessing: Text Normalization Lemmatization
Train-Test-Validation split
Real Dataset
Filtering to isolate subject and body
Apply Target column
Training Set
Testing Set
Validation set
Transform
Feature: Bag of Words as CountVectorizer()
Feature: TF-IDF as TFIDFVectorizer()
Feature: Embeddings as Word2Vec()
Interpret
Model: Logistic Regression
Model: Naïve-Bayes
Model: Random Forest
Model: Neural Network
Results

Figure 1: Summary of Experimental process.

Preprocessing

Text Normalization

Among the very basic and commonly used preprocessing steps is text normalization. It puts all text into the standard case, either lower or upper, so that the same words are recognized, not distinguished by the instances in which they appear in the original emails. It also removes extraneous spaces like line breaks and special characters that may disturb text analysis. This is done to reduce the complexity of the text data and introduce uniformity into the data, which helps the following feature extraction steps.

Lemmatization

It is a process by which words with inflectional endings likely to alter the meaning of the words are removed and returned to their dictionary form, hence called the lemma. Unlike stemming, lemmatization is based on correctly identifying the intended part of speech, the meaning of a word in a sentence, and the greater context around that sentence, neighbouring sentences, or even the whole document. Then, to ensure that words with the same root were treated as the same word, such as "spamming" versus "spam," we implemented lemmatization from Spacy. Finally, this also helps reduce the feature set's dimensionality, making the extraction methods more efficient. This will make things like Bag of Words or TF-IDF efficient.

Merging of the datasets

This preprocessing was done to pair up two datasets: one with phishing emails and the other with legitimate emails. The reason for doing this was to balance the datasets so that we could train the machine-learning models used positively and negatively. We were careful to clean the data of personal/sensitive information at the time of merging to ensure the privacy and security of data.

Merging of subject and body columns

We combined each email's 'Subject' and 'Body' fields into a single text field. This step recognizes that both email components contain valuable information while detecting phishing. It treats the subject line as part of the email context, allowing the capture of more phishing indicators, which was why we did it.

Setting Target Column

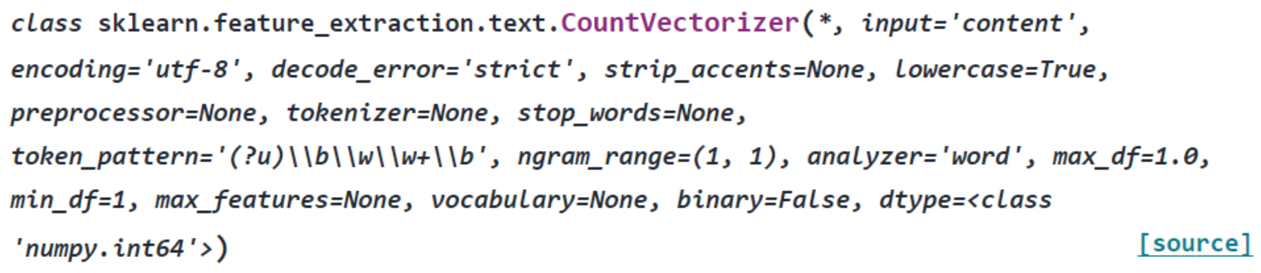
We added a column that identified the email as either a phishing attempt or an authentic one. The models learn based on this tag for characteristics of whether the email is a phishing or non-phishing mail. We also found that a target column is necessary for the training/evaluation of machine learning models to learn from examples and then tell new, unseen e-mails' classification, whether they are legit or spam.

These preprocessing steps become critical because they directly affect the effectiveness of the machine learning models. They use an approach to phishing detection by cleaning and structuring the data properly before it is passed into models for training and testing. Each step ensures that the data is in the best form to reveal patterns in distinguishing between phishing attempts and legitimate emails.

Feature Engineering

Bag-of-Words (BoW)

Bag of Words is a simple yet fundamentally effective way to represent text in natural language processing. In this technique, the email text data are converted into a fixed-length set of arrays representing the frequency of each word. This is implemented in Python using the CountVectorizer class from the sci-kit-learn Python library that creates a vocabulary of all the unique words from the training dataset and transforms each document into a vector where each element represents the frequency of the words.[16]



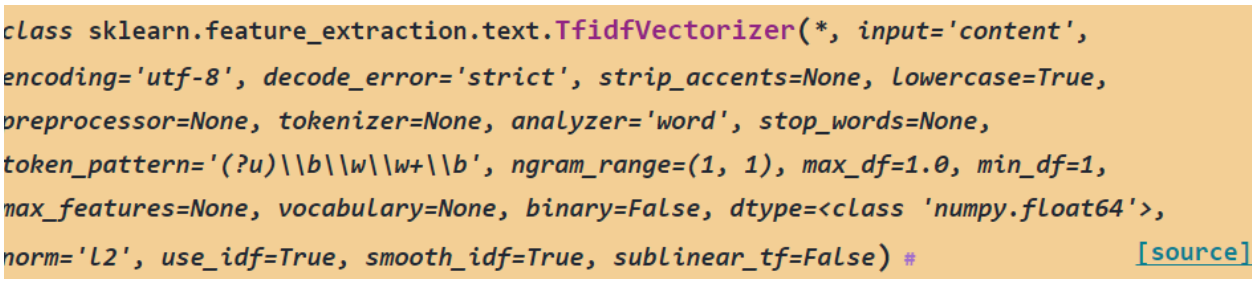
```
class sklearn.feature_extraction.text.CountVectorizer(*, input='content',
encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True,
preprocessor=None, tokenizer=None, stop_words=None,
token_pattern='(?u)\\b\\w\\w+\\b', ngram_range=(1, 1), analyzer='word', max_df=1.0,
min_df=1, max_features=None, vocabulary=None, binary=False, dtype=<class
'numpy.int64'>)                                                          [source]
```

Figure 2: CountVectorizer model parameters

We carefully selected the top 95 words based on their frequency and excluded punctuations and stop words that do not contribute meaningful information. Since almost all the email texts were within this limit, we set the maximum feature length to 500.

Term Frequency-Inverse Document Frequency (TF-IDF)

Term Frequency-Inverse Document Frequency (TF-IDF) is another important statistical technique for representing texts as numbers. This technique is often used in text mining and information retrieval. It computes the frequency of a word in a document and offsets it with the frequency of the word in the corpus of documents. This adjusts the fact that some common English words appear more frequently in general, and this offset helps prevent these words from skewing the output feature vectors.



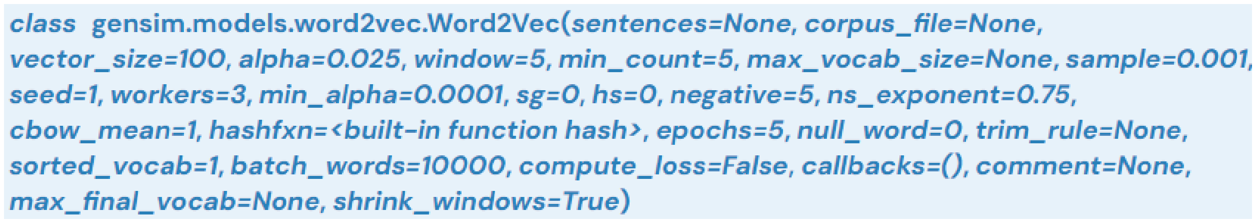
```
class sklearn.feature_extraction.text.TfidfVectorizer(*, input='content',
encoding='utf-8', decode_error='strict', strip_accents=None, lowercase=True,
preprocessor=None, tokenizer=None, analyzer='word', stop_words=None,
token_pattern='(?u)\\b\\w\\w+\\b', ngram_range=(1, 1), max_df=1.0, min_df=1,
max_features=None, vocabulary=None, binary=False, dtype=<class 'numpy.float64'>,
norm='l2', use_idf=True, smooth_idf=True, sublinear_tf=False) #          [source]
```

Figure 3: TfidfVectorizer model parameters.

We utilized sci-kit-learn's TfidfVectorizer to compute the TF-IDF scores for each word in the document and then converted text into their corresponding numerical representation. This vectorizer was configured to have a maximum of 5000 features and remove common English stop words. This approach is generally better than the Bag-of-Words technique as it emphasizes the words that are unique to particular documents, which can be very helpful in identifying unique phishing tactics that might not be common across all the emails in the dataset.[16]

Word Embeddings (Word2Vec)

Word Embeddings is the current state-of-the-art technique for representing text in a dense vector form. This technique outputs vectors such that two semantically similar words are closer in the vector space. We trained the Skip Gram (SG) version of the Word2Vec model on our training dataset using the Gensim library.[17] This model tries to predict the preceding and succeeding words given an input word. This version is beneficial for understanding the context in which each word appears in the phishing emails. We chose a vector size of 100, which means each word is converted into a 100-dimensional vector, which is a good balance between accuracy and efficiency. This technique is extremely powerful for detecting phishing emails because it can capture subtle semantic similarities between the words across different emails.



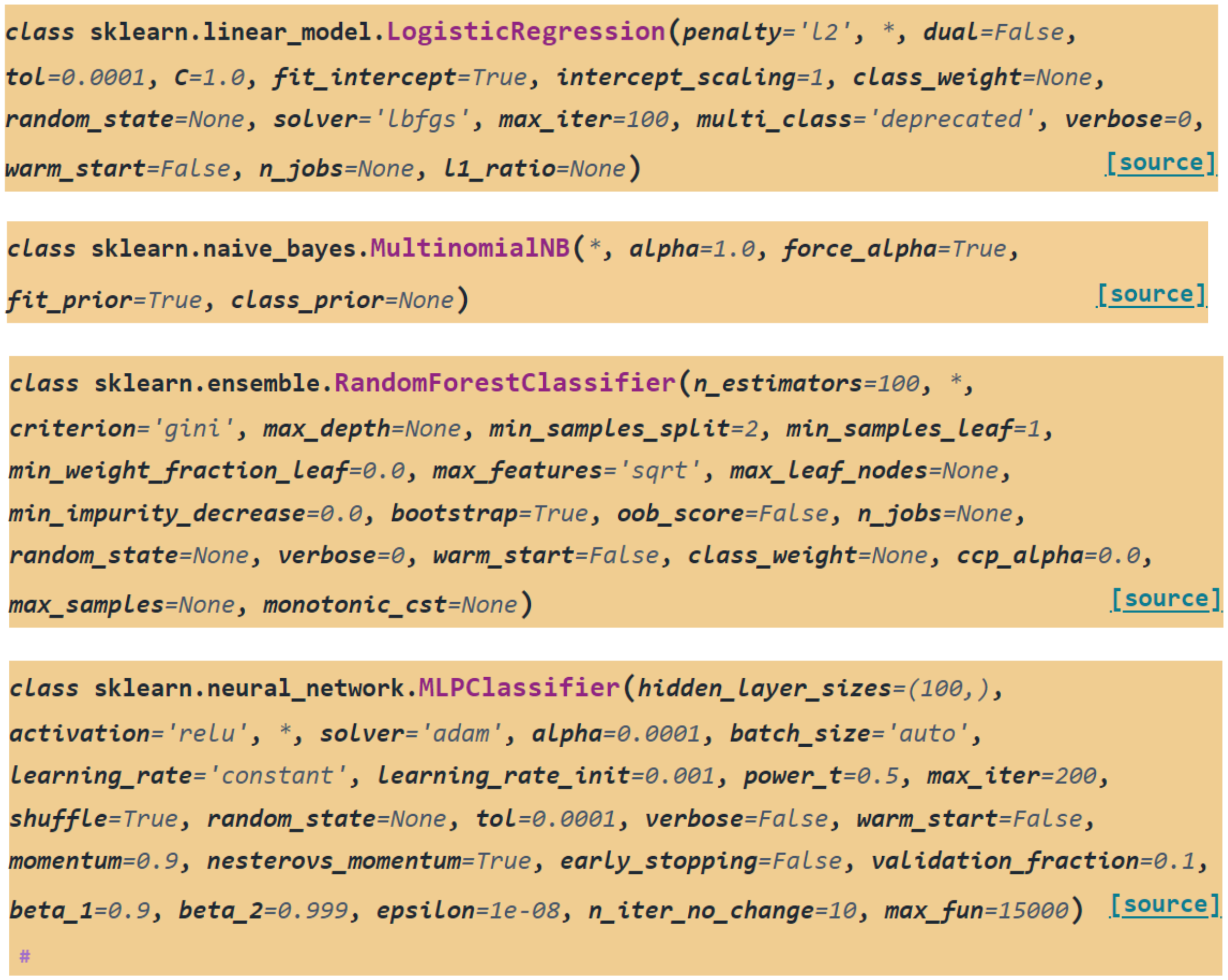
```
class  gensim.models.word2vec.Word2Vec(sentences=None, corpus_file=None,
vector_size=100, alpha=0.025, window=5, min_count=5, max_vocab_size=None, sample=0.001,
seed=1, workers=3, min_alpha=0.0001, sg=0, hs=0, negative=5, ns_exponent=0.75,
cbow_mean=1, hashfxn=<built-in function hash>, epochs=5, null_word=0, trim_rule=None,
sorted_vocab=1, batch_words=10000, compute_loss=False, callbacks=(), comment=None,
max_final_vocab=None, shrink_windows=True)
```

Figure 4: Word2Vec model parameters.

Model Construction and Tuning

When considering the construction of our models, we must admit that aside from our research, we contemplated another factor when deciding on our final four. As we are relatively novices in machine learning algorithms and the development of models, we looked to utilize libraries that streamlined and simplified the process for us. Luckily, there is an extensively developed scikit-learn library in Python. This library contains all commonly utilized Machine Learning classification algorithms ranging from simple linear and logistic regression models to ensemble-based models and even multi-layer perceptron neural networks, which perfectly fit our requirements for this assessment. Thus, we developed four base scripts using sci-kit-learn with LogisticRegression(), MultinomialNB(), RandomForestClassifier() and MLPClassifier() functions. We then expanded upon them to independently evaluate each of the three selected features, resulting in twelve scripts in total.[16]

A series of initial parameters and hyperparameters are set using the classifier functions above in their base form, as shown in Figure 2 from [16].



```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False,
tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None,
random_state=None, solver='lbfgs', max_iter=100, multi_class='deprecated', verbose=0,
warm_start=False, n_jobs=None, l1_ratio=None)                                  [source]
```

```
class sklearn.naive_bayes.MultinomialNB(*, alpha=1.0, force_alpha=True,
fit_prior=True, class_prior=None)                                              [source]
```

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *,
criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None,
min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None,
random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0,
max_samples=None, monotonic_cst=None)                                          [source]
```

```
class sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(100,),
activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto',
learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200,
shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False,
momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1,
beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)   [source]
#
```

Figure 5: Initial state of models and parameters.

As shown above, there are numerous parameters and hyperparameters we can adjust and fine-tune to improve the models' performance. However, tuning and evaluating every parameter in tandem or independently is difficult. Therefore, we selected a subset of parameters and hyperparameters to tune to see if they can improve performance, which will be listed in the results tables.

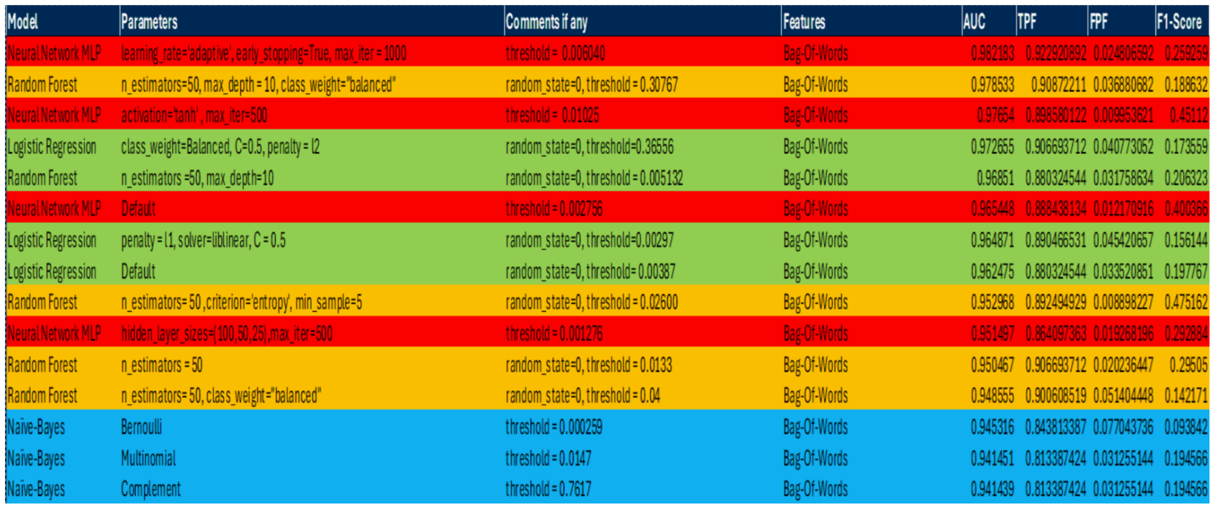
Finally, when constructing and tuning the models, we need to consider the threshold probabilities for classification. As we utilize the entire dataset, we inherently deal with imbalances as the ratio is heavily skewed regarding real and phishing emails. As such, in the initial testing phases, our results showed that the models all yielded high accuracy as they would accurately detect a much higher proportion of real emails. Yet, the phishing email detection would range between 50-70%, showing that on the ROC curve for the models, the False Positive Rate is minimized as opposed to maximizing the True Positive Rate. Therefore, we decided to employ a method that adjusts the curve's point to maximize the True Positive Rate. This brings us to Youden's J statistic, also known as Youden's index. This statistic is utilized to find the peak predicted probability cut-off that maximizes the distance between true positive and false positive rates. As the ROC curve function from sci-kit-learn generates an array of true positives, false positives and their corresponding thresholds, we can find the index of the maximum distance using the NumPy function maxarg(TPR-FPR) and utilize the resulting index to find the associated threshold probability. We can maximize the True Positive Rates by manually labelling the predicted probabilities generated by the models based on the threshold.[16],[18]-[20]

Results

Table of Results

Bag of words

The first model we used, Bag of Words (Bow), is the most common yet useful model for handling text data in applications of machine learning models. The first step was appending the email bodies and subjects to maximize the current data, as more data would help the model learn the underlying patterns better. Then, it is followed by removing stop words as they might skew the model's results. Then, the punctuation is removed to remove the unwanted characters and word differences. Then, we selected the top 95 words based on frequency to run the model efficiently. The parameter and the hyperparameters are then tweaked to maximize the performance of the models. From the results presented in the table, we can observe that the neural network with Multi-layer perceptron with a threshold of 0.006040 and adaptive learning rate with a maximum iteration of 1000 has produced the highest accuracy of 98.2%. Other models like the Random Forest, Neural Network with different parameters and hyper-parameters, and logistic regression have also consistently performed well with 97% accuracy.

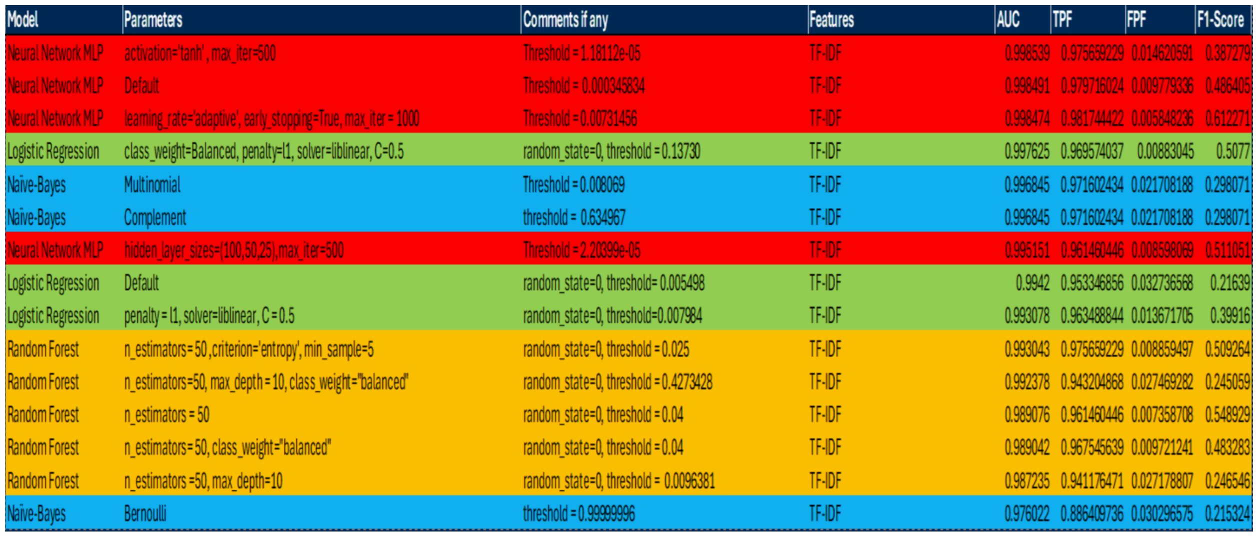


Model	Parameters	Comments if any	Features	AUC	TPF	FPF	F1-Score
Neural Network MLP	learning_rate='adaptive', early_stopping=True, max_iter = 1000	threshold = 0.006040	Bag-Of-Words	0.982183	0.922920892	0.024806592	0.259259
Random Forest	n_estimators=50, max_depth = 10, class_weight="balanced"	random_state=0, threshold = 0.30767	Bag-Of-Words	0.978533	0.90872211	0.036880682	0.188632
Neural Network MLP	activation='tanh' , max_iter=500	threshold = 0.01025	Bag-Of-Words	0.97654	0.898580122	0.009953621	0.45112
Logistic Regression	class_weight=Balanced, C=0.5, penalty = l2	random_state=0, threshold=0.36556	Bag-Of-Words	0.972655	0.906693712	0.040773052	0.173559
Random Forest	n_estimators =50, max_depth=10	random_state=0, threshold = 0.005132	Bag-Of-Words	0.96851	0.880324544	0.031758634	0.206323
Neural Network MLP	Default	threshold = 0.002756	Bag-Of-Words	0.965448	0.888438134	0.012170916	0.400366
Logistic Regression	penalty = l1, solver=liblinear, C = 0.5	random_state=0, threshold=0.00297	Bag-Of-Words	0.964871	0.890466531	0.045420657	0.156144
Logistic Regression	Default	random_state=0, threshold= 0.00387	Bag-Of-Words	0.962475	0.880324544	0.033520851	0.197767
Random Forest	n_estimators= 50 ,criterion='entropy', min_sample=5	random_state=0, threshold = 0.02600	Bag-Of-Words	0.952968	0.892494929	0.008898227	0.475162
Neural Network MLP	hidden_layer_sizes=(100,50,25),max_iter=500	threshold = 0.001276	Bag-Of-Words	0.951497	0.864097363	0.019268196	0.292884
Random Forest	n_estimators = 50	random_state=0, threshold = 0.0133	Bag-Of-Words	0.950467	0.906693712	0.020236447	0.29505
Random Forest	n_estimators= 50, class_weight="balanced"	random_state=0, threshold = 0.04	Bag-Of-Words	0.948555	0.900608519	0.051404448	0.142171
Naïve-Bayes	Bernoulli	threshold = 0.000259	Bag-Of-Words	0.945316	0.843813387	0.077043736	0.093842
Naïve-Bayes	Multinomial	threshold = 0.0147	Bag-Of-Words	0.941451	0.813387424	0.031255144	0.194566
Naïve-Bayes	Complement	threshold = 0.7617	Bag-Of-Words	0.941439	0.813387424	0.031255144	0.194566

Figure 6: Bag Of Words results table.

TF-IDF

The next model we opted for is the Term Frequency - Inverse Document Frequency Vectoriser. This model works by representing words as vectors. The model first finds out the frequency of each word and then the frequency of those words in a set of documents. The first step in implementing the model is removing stoop words, as these common English words do not have any relevant information. Then, we initialized the model with a maximum of five thousand features. From the observations from the results table, we could say that the Neural Networks outperformed every other model, even with default parameters and modified hyperparameters. The Neural Network model with TanH activation with max iterations set to 500 has achieved the highest accuracy of 99.8%.

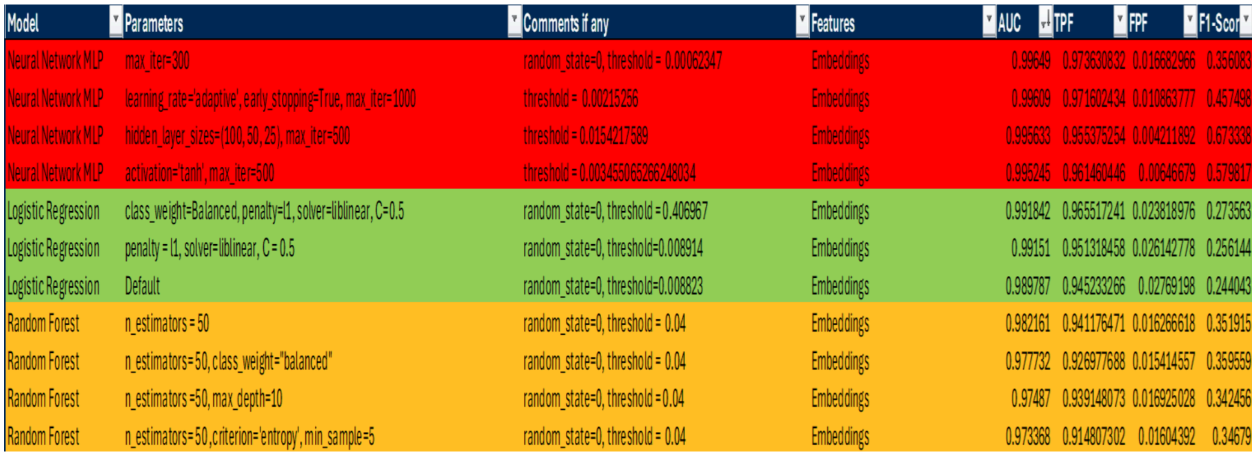


Model	Parameters	Comments if any	Features	AUC	TPF	FPF	F1-Score
Neural Network MLP	activation='tanh' , max_iter=500	Threshold = 1.18112e-05	TF-IDF	0.998539	0.975659229	0.014620591	0.387279
Neural Network MLP	Default	Threshold = 0.000345834	TF-IDF	0.998491	0.979716024	0.009779336	0.486405
Neural Network MLP	learning_rate='adaptive', early_stopping=True, max_iter = 1000	Threshold = 0.00731456	TF-IDF	0.998474	0.981744422	0.005848236	0.612271
Logistic Regression	class_weight=Balanced, penalty=l1, solver=liblinear, C=0.5	random_state=0, threshold = 0.13730	TF-IDF	0.997625	0.969574037	0.00883045	0.5077
Naïve-Bayes	Multinomial	Threshold = 0.008069	TF-IDF	0.996845	0.971602434	0.021708188	0.298071
Naïve-Bayes	Complement	threshold = 0.634967	TF-IDF	0.996845	0.971602434	0.021708188	0.298071
Neural Network MLP	hidden_layer_sizes=(100,50,25),max_iter=500	Threshold = 2.20399e-05	TF-IDF	0.995151	0.961460446	0.008598069	0.511051
Logistic Regression	Default	random_state=0, threshold= 0.005498	TF-IDF	0.9942	0.953346856	0.032736568	0.21639
Logistic Regression	penalty = l1, solver=liblinear, C = 0.5	random_state=0, threshold=0.007984	TF-IDF	0.993078	0.963488844	0.013671705	0.39916
Random Forest	n_estimators= 50 ,criterion='entropy', min_sample=5	random_state=0, threshold = 0.025	TF-IDF	0.993043	0.975659229	0.008859497	0.509264
Random Forest	n_estimators=50, max_depth = 10, class_weight="balanced"	random_state=0, threshold = 0.4273428	TF-IDF	0.992378	0.943204868	0.027469282	0.245059
Random Forest	n_estimators = 50	random_state=0, threshold = 0.04	TF-IDF	0.989076	0.961460446	0.007358708	0.548929
Random Forest	n_estimators= 50, class_weight="balanced"	random_state=0, threshold = 0.04	TF-IDF	0.989042	0.967545639	0.009721241	0.483283
Random Forest	n_estimators =50, max_depth=10	random_state=0, threshold = 0.0096381	TF-IDF	0.987235	0.941176471	0.027178807	0.246546
Naïve-Bayes	Bernoulli	threshold = 0.99999996	TF-IDF	0.976022	0.886409736	0.030296575	0.215324

Figure 7: TF-IDF results table.

Embeddings

The next model that we used was Word Embedding. For this, we used the word2vec model from the Python library Gensim. In the model, all the words are first converted into vectors, and then these vectors are represented in a high-dimensional feature space. For this model, we used a vector size of a hundred. Here, in word2vec, we substituted the bag of words with skiagrams to improve the model's performance. From the table, the Neural Network has outperformed other models, such as logistic regression and random forest. The Neural Network model with Random state set to 0 and threshold set to 0.00062347 with max iteration of 300 has produced a maximum accuracy of 99.6%.



Model	Parameters	Comments if any	Features	AUC	TPF	FPF	F1-Scor
Neural Network MLP	max_iter=300	random_state=0, threshold = 0.00062347	Embeddings	0.99649	0.973630832	0.016682966	0.356083
Neural Network MLP	learning_rate='adaptive', early_stopping=True, max_iter=1000	threshold = 0.00215256	Embeddings	0.99609	0.971602434	0.010863777	0.457498
Neural Network MLP	hidden_layer_sizes=(100, 50, 25), max_iter=500	threshold = 0.0154217589	Embeddings	0.995633	0.955375254	0.004211892	0.673338
Neural Network MLP	activation='tanh', max_iter=500	threshold = 0.0034550652662448034	Embeddings	0.995245	0.961460446	0.00646679	0.579817
Logistic Regression	class_weight=Balanced, penalty=l1, solver=liblinear, C=0.5	random_state=0, threshold = 0.406967	Embeddings	0.991842	0.965517241	0.023818976	0.273563
Logistic Regression	penalty = l1, solver=liblinear, C = 0.5	random_state=0, threshold=0.008914	Embeddings	0.99151	0.951318458	0.026142778	0.256144
Logistic Regression	Default	random_state=0, threshold=0.008823	Embeddings	0.989787	0.945233266	0.02769198	0.244043
Random Forest	n_estimators = 50	random_state=0, threshold = 0.04	Embeddings	0.982161	0.941176471	0.016266618	0.351915
Random Forest	n_estimators=50, class_weight="balanced"	random_state=0, threshold = 0.04	Embeddings	0.977732	0.926977688	0.015414557	0.359559
Random Forest	n_estimators =50, max_depth=10	random_state=0, threshold = 0.04	Embeddings	0.97487	0.939148073	0.016925028	0.342456
Random Forest	n_estimators=50 ,criterion='entropy', min_sample=5	random_state=0, threshold = 0.04	Embeddings	0.973368	0.914807302	0.01604392	0.34679

Figure 8: Embeddings results table.

F1 vs AUC score

As stated in a prior paragraph, we have utilized Youden's index to find an optimal predicted probability threshold to maximize the machine learning models' true positive rate. Let us reflect on this as we look at the AUC vs. F1 score graphs for the features in Figure 9. Innately, the predictive threshold for decisions in machine learning models is 0.5 in this binary classification case. By changing these thresholds to account for the imbalances in the dataset to those shown within the results table, we generally allow more entries to be classified as phishing emails. In doing so, we drastically increase the False Positive Rates, which explains why our F1 scores fall within drastically lower ranges than the AUC scores. This is because F1 depends on the models' recall and precision, skewed with a greater tendency for false positives.[21]

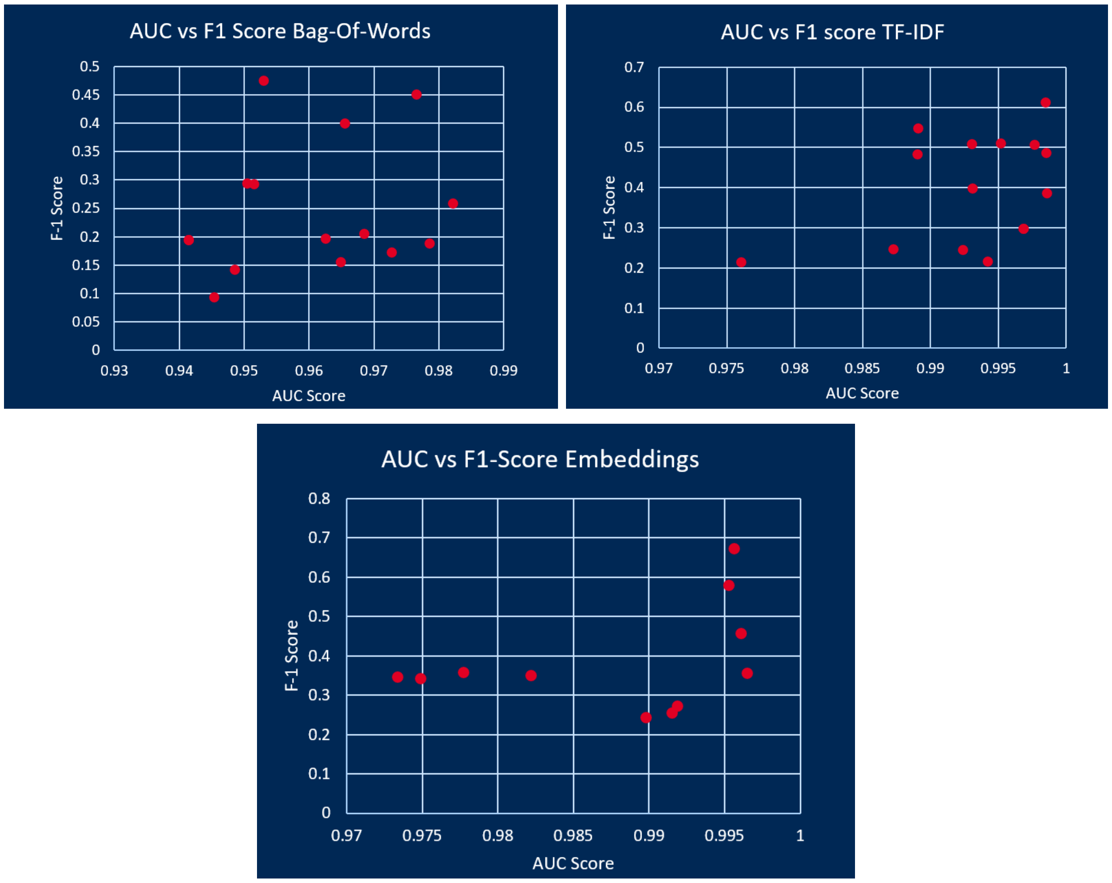


Figure 9: F1 vs AUC score Graphs.

Discussion

In this project, we experimented with several machine-learning models to detect phishing emails. We aim to examine Logistic Regression, naive Bayes, Neural Networks, and Random Forest Classifiers to classify phishing messages from genuine heuristics. After much experimentation and tweaking hyperparameters, we understood each type's power and limitations.

The logistic regression model is simple and powerful for solving linear relationships. Its results could be more adequate for larger datasets, but performance-wise, it is great when the data give a nice shape. This application showed that even simple models can provide state-of-the-art results when used correctly. However, this model performed poorly on more complex data distribution patterns and displayed a performance plateau, suggesting the necessity of advanced methods in those settings.

At best, naive Bayes was a niche technique in text classification, with methods as cited above applying the Bayes theorem and assuming the independence of features. This proved to be a strong model for email content analysis. A probabilistic nature enabled it to quickly and effectively make predictions, thus fitting for large-scale deployments. However, one of the major drawbacks is the assumption that features are independent, which rarely is true in real-world data, likely affecting performance in more complex datasets.

Neural networks, as they can learn much more intricate patterns by combining multiple neurons between layers, easily outperform SVMs. However, it had good accuracy as neural networks are very flexible and can be used to model complex data patterns. They were computationally intensive and suffered from overfitting, especially in the presence of small amounts of training data. The components built in regularization techniques for combating overfitting, but that only contributed to the complexity of model training.

By aggregating the predictions of multiple decision trees, Random Forest Classifiers provided high accuracy and robustness. This, together with their ensemble nature, makes them less prone to overfitting, leading to broader models able to catch a wider range of patterns in the data. Another challenge was the interpretability of Random Forests because, with increasing trees, the contribution of the single features to the final decision was no longer obvious.

Our results indicated that while all models had their merits, the Random Forest Classifier and Neural Networks stood out due to their ability to capture intricate patterns in the data. Unfortunately, they also require a lot of computation, leaving many out for use in real-time situations without adequate hardware. On the other hand, logistic regression and Naive Bayes also performed quite well, but in between, they were an ideal choice for scenarios with limited resources.

Complications/Future Works

Despite the promising results, our project encountered several challenges. The primary complication faced here was that the datasets needed to be more balanced, with a large discrepancy between the number of legitimate versus phishing emails. Further, since there were so many more legitimate e-mails than phishing e-mails, it resulted in very high accuracy scores that were misleading as the models were fairly accurate at classifying legitimate e-mails and poor at classifying phishing e-mails. These threshold probabilities could be calibrated to a certain extent using techniques such as Youden's J statistic; further improvements are due with this approach.

Another challenge was the need for substantial computational resources, particularly for training Neural Networks and Random Forest Classifiers. This limitation affected our ability to fully explore these models' potential and prevented us from investigating what they could do. In the future, we can use powerful computational infrastructure for larger experimentation.

Moreover, the preprocessing steps, including merging the subject and body of emails and tokenizing text, played a crucial role in the final model performance. Ensuring that the most informative features were retained and noise was not simultaneously increased was challenging. Advanced preprocessing with TF-IDF or word embeddings could have richer features and be more accurate for a model.

For future work, we propose the following directions:

1. **Improved Data Preprocessing**: Improving preprocessing processes for dealing with skewed datasets, as well as applying methods like SMOTE (Synthetic Minority Over-Sampling Technique) to generate new instances of the non-major class, which aids in dataset balancing and the effectiveness of the model.
2. **Feature Engineering**: Additional features like email metadata and sender information might have improved the functionality. Setting timestamps and more complex text representations (TF-IDF, word embeddings), which provide significantly more context and improve model accuracy, could have further aided the functionality. These might be coupled with feature selection approaches to find the best discriminative characteristics for classification.
3. **Hybrid Models**: Using ensemble learning techniques to combine the capabilities of many models might resolve many issues and strengthen the model. Using either the combination of various classifiers or a boosting approach such as AdaBoost or Gradient Boosting can increase the accuracy of predictions.
4. **Real-time Application**: Developing a high-performance and computationally efficient system for real-time phishing detection using approaches like incremental learning. This involves building models that can be updated and more adaptive to new data so businesses can continue building a responsive system in the long term.
5. **Model Interpretability**: Robust implementations of complex models, such as neural networks and random forests, are required to guarantee practical deployment. SHAP (Shapley Additive Explanations) and LIME (Local Interpretable Model-agnostic Explanations) are important strategies for understanding and explaining model predictions, making models more interpretable and trustworthy.

Understanding and developing these obstacles and examining potential future paths can help enhance machine-learning models that are more robust and reliable in real-world scenarios for phishing detection. This will boost security and make the digital world considerably safer for consumers.

Conclusion

To conclude, as businesses face an ever-increasing threat of social engineering attacks through email services, developing and employing machine learning models can serve as effective tools to combat this. Based on the articles reviewed, we noted that Logistic Regression, Naive Bayes, Random Forests and Neural Networks can serve as effective models for text-based datasets. As phishing emails target people, it can take time to discern the difference between them and authentic communications. Therefore, by extracting information through features such as Bag of Words, TF-IDF and word embeddings, we can train machine learning models to extract subtle contextual information to make the distinction. Through experimentation, we sanitized, preprocessed and merged real and phishing datasets provided by the UVic Systems office and a public database. We then split the datasets into a training, testing, and cross-validation subset and performed tests measuring each of the features above and models independently. Then, we compared their performance to assess the task's best possible feature and model combination. Although we dealt with imbalances in our dataset, which required us to manipulate thresholds of the models and yield high false positive rates, our results proved promising. We saw that TF-IDF, in conjunction with a Neural Network model, performs the best in terms of AUC score and word embeddings, closely following in performance, and another Neural Network Model using word embeddings yielded the best F1 score out of the bunch. Finally, it is important to note that our experimental process utilized much trial and error to find the best-performing model and feature by fine-tuning hyperparameters. Future extensions of this project could explore the best-performing feature and model in more depth to better understand its parameters and hyperparameters and improve general performance to a greater extent. Alternatively, we could have explored more complex methodologies, such as more hybridized ML models.

References

[1] "Phishing - glossary: CSRC," CSRC, https://csrc.nist.gov/glossary/term/phishing (accessed Aug. 4, 2024).

[2] C. S. E. Canada, "National Cyber Threat Assessment 2023-2024," Canadian Centre for Cyber Security, https://www.cyber.gc.ca/en/guidance/national-cyber-threat-assessment-2023-2024 (accessed Aug. 4, 2024).

[3] "Worldwide 2023 email phishing statistics and examples," Trend Micro, https://www.trendmicro.com/en_ca/ciso/23/e/worldwide-email-phishing-stats-examples-2023.html (accessed Aug. 4, 2024).

[4] "State of Cybercrime: Microsoft security," State of cybercrime | Microsoft Security, https://www.microsoft.com/en-ca/security/business/microsoft-digital-defense-report-2022-state-of-cybercrime (accessed Aug. 4, 2024).

[5] "Protecting against cyber threats during covid-19 and beyond | Google Cloud blog," Google, https://cloud.google.com/blog/products/identity-security/protecting-against-cyber-threats-during-covid-19-and-beyond (accessed Aug. 4, 2024).

[6] "Phishing attacks: Defending your organisation," NCSC, https://www.ncsc.gov.uk/guidance/phishing (accessed Aug. 4, 2024).

[7] "Microsoft Defender for Office 365," Microsoft Security, https://www.microsoft.com/en-ca/security/business/siem-and-xdr/microsoft-defender-office-365 (accessed Aug. 4, 2024).

[8] Kim, D., Lee, S., & Park, J. (2023). Logistic Regression Model for Phishing Detection: A Practical Approach. *International Journal of Information Security*, 22, 113-126.

[9] Sharma, A., & Pandey, R. (2022). Phishing Detection Using Logistic Regression: An Empirical Study. *Journal of Cybersecurity and Privacy*, 1(2), 45-58.

[10] Sahoo, D., Liu, C., & Hoi, S. C. (2023). Efficient Phishing Detection Using Naive Bayes Classifier. *ACM Transactions on Privacy and Security*, 16(1), 10.

[11] Gupta, R., & Bhushan, B. (2022). A Comprehensive Analysis of Phishing Detection Using Naive Bayes. *International Journal of Cyber-Security and Digital Forensics*, 11(4), 567-580.

[12] Huang, J., Chen, X., & Li, Y. (2023). Deep Learning Approaches for Phishing Detection: A Review. *Computers & Security*, 122, 102584.

[13] Singh, P., Kumar, A., & Jain, R. (2022). Neural Network Techniques for Phishing Detection: Challenges and Opportunities. *Cybersecurity and Privacy*, 2(3), 198-214.

[14] Zhou, Y., Wang, J., & Zhang, L. (2023). Phishing Detection Using Machine Learning Techniques: A Comparative Study. *Journal of Cybersecurity Research*, 45(3), 234-250.

[15] Lee, H., Park, S., & Kim, J. (2022). Enhancing Phishing Detection with Random Forest Classifiers. *IEEE Transactions on Information Forensics and Security*, 18, 1123-1136.

[16] "Scikit-Learn," scikit, https://scikit-learn.org/stable/ (accessed Aug. 4, 2024).

[17] “Gensim: Topic modelling for humans,” models.word2vec – Word2vec embeddings - gensim, https://radimrehurek.com/gensim/models/word2vec.html (accessed Aug. 4, 2024).
[18] Area under the curve, https://www.ibm.com/docs/en/spss-statistics/29.0.0?topic=schemes-area-under-curve (accessed Aug. 4, 2024).
[19] “Use Youden index to determine cut-off for classification,” Gist, https://gist.github.com/twolodzko/4fae2980a1f15f8682d243808e5859bb (accessed Aug. 4, 2024).
[20] NumPy, https://numpy.org/ (accessed Aug. 4, 2024).
[21] “What is classification threshold,” Deepchecks, https://deepchecks.com/glossary/classification-threshold/ (accessed Aug. 4, 2024).