



MASTER OF TECHNOLOGY

PATTERN RECOGNITION SYSTEMS (PRS)

Semester One 2021/2022

ElderGuard

Team Members

Student Name	Student ID
Lim Jun Ming	A0231523U
Mediana	A0231458E
Yeong Wee Ping	A0231533R

Contents

1	Executive Summary.....	3
2	Problem Statement.....	4
3	System Overview.....	5
4	Spam Text Module	6
4.1	Background and Related Works.....	6
4.2	Module System Pipeline	8
4.3	Dataset	9
4.4	Feature Engineering.....	13
4.5	Model Training.....	15
4.6	Model Evaluation	17
4.7	Model Selection & Deployment.....	20
5	Phishing URL Module	21
5.1	Background	21
5.2	Module System Pipeline	21
5.3	Phishing Dataset	21
5.4	Feature Extraction.....	23
5.5	Model Training.....	25
5.6	Model Evaluation	26
6	Malicious URL Module	31
6.1	Background	31
6.2	Module Workflow	31
6.3	Datasets	32
6.4	Feature Extraction.....	32
6.5	Model Training and Selection	34
6.6	Model evaluation	37
7	System Deployment.....	38
7.1	FrontEnd – Android.....	38
7.2	BackEnd - FlaskApp and Heroku	39
8	Conclusion and Future Works.....	40
9	References	41
10	Appendix A – Model Hyperparameter Tuning Evaluation	42
11	Appendix B – Proposal	46

1 Executive Summary

With Singapore moving towards digitalization, elderlies are starting to embrace the use of smart phones in their daily lives. This inadvertently exposes them to malicious content that send through their smart phones. When our elderlies, who may not be tech-savvy, receive messages containing malicious link or spam, they might unknowingly click on malicious contents without second thoughts. As a results, it leads to financial loss or identity theft if the link is unsafe or malicious.

Our team perceive the need for an application can help to detect text messages that may be fraudulent in nature, for example, spam, fraud messages or messages that contain malicious or phishing URLs. In the hope that elderlies can connect to a colourful digital life safely and prevent them from being scammed or cheated. Thus, we developed a machine learning driven application that provides the capability of predicting the safety of a text message content.

An Android application, **ElderGuard**, is built for users to forward messages and obtain the predicted results. The predictive engine which contains three machine learning modules, ie. Spam Detection, Phishing URL Detection, and Malicious URL Detection are deployed and hosted in Heroku. Prior deployment, the team has conducted model training for each of the modules using multiple supervised machine learning method including Naïve Bayes, Logistic Regression, Decision Tree, Random Forest, AdaBoost, Support Vector Machine (SVM), K-Nearest Neighbours as well as some deep learning neural network architecture like Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) Neural Network and the combination CNN-LSTM. Upon model training, the team performed model evaluation based on few performances' metric including F1-score, AUC-score and testing accuracy score for selecting the best model to be deployed in Heroku. When users forward messages for predicting, the predictive engine in Heroku will pass through the three machine learning modules, depends on if the messages containing text messages only or with URL, to come out with results. And finally, through **ElderGuard** Android Application, users will receive a green tick to show that the text message is safe or a red cross-sign to show that the text message is unsafe.

2 Problem Statement

Nowadays, the number of fraud cases from mobile text messages are on the rise. Scammers has been targeting elderlies who tends to be less tech-savvy and has been sending them text messages that may contains fraudulent content, malicious or phishing links.

The elderlies are usually the prime targets for scams related to credit cards, e-commerce transactions, health products, investment, loans, insurance, banking and wire transfers. The tactics used by the scammers has been evolving and becoming more sophisticated nowadays. Victims typically receive a mobile text message (SMS) or WhatsApp message requesting the victims to perform some actions such as replying the text message with their personal identity information, bank accounts credentials or clicking into malicious or phishing links. Despite government's effort in raising awareness for mobile text messages scams, elderlies still often find themselves falling into one of these scamming schemes and suffered financial losses.

Elderlies are often perceived as easy targets by the scammers because elderlies may be more willing to listen, and more trusting than the younger generation. Besides, elderlies who have fallen into these scams often will try to hide it from their families to avoid any form embarrassment or to avoid further increasing any burden on their families or children. The root cause here is simply because it is hard for them to differentiate and filter these fraudulent text messages. Often when they receive a text message, they do not have the knowledge nor the awareness to identify and verify if the text message is fraudulent or not. With the rise of text message mobile scams among the elderlies, there is a need to address the problem so that the elderlies continue to have confidence and to enjoy the convenience of mobile text messages.

Minimum Viable Product (MVP) - ElderGuard

As an MVP, we seek to address the difficulty of identifying fraudulent messages for the elderlies. The system is designed with ease of usage and readability in mind. This is to ensure that there is minimal learning curve for the elderlies to start using the application.

We have selected three types of fraudulent contents to tackle on in our application, namely spam text messages, malicious and phishing URL. We have explored various pattern recognition methods and approaches in identifying each of the fraudulent contents in a given text message. The best performing methods and models are then deployed in Flask and Heroku. The predictions of fraudulent contents can then be easily performed through an Android app - **ElderGuard** that will communicate with the Flask and Heroku backend for inference. In the remaining sections of this report, we will be discussing the overall design, methodology and processes for the application in detail.

3 System Overview

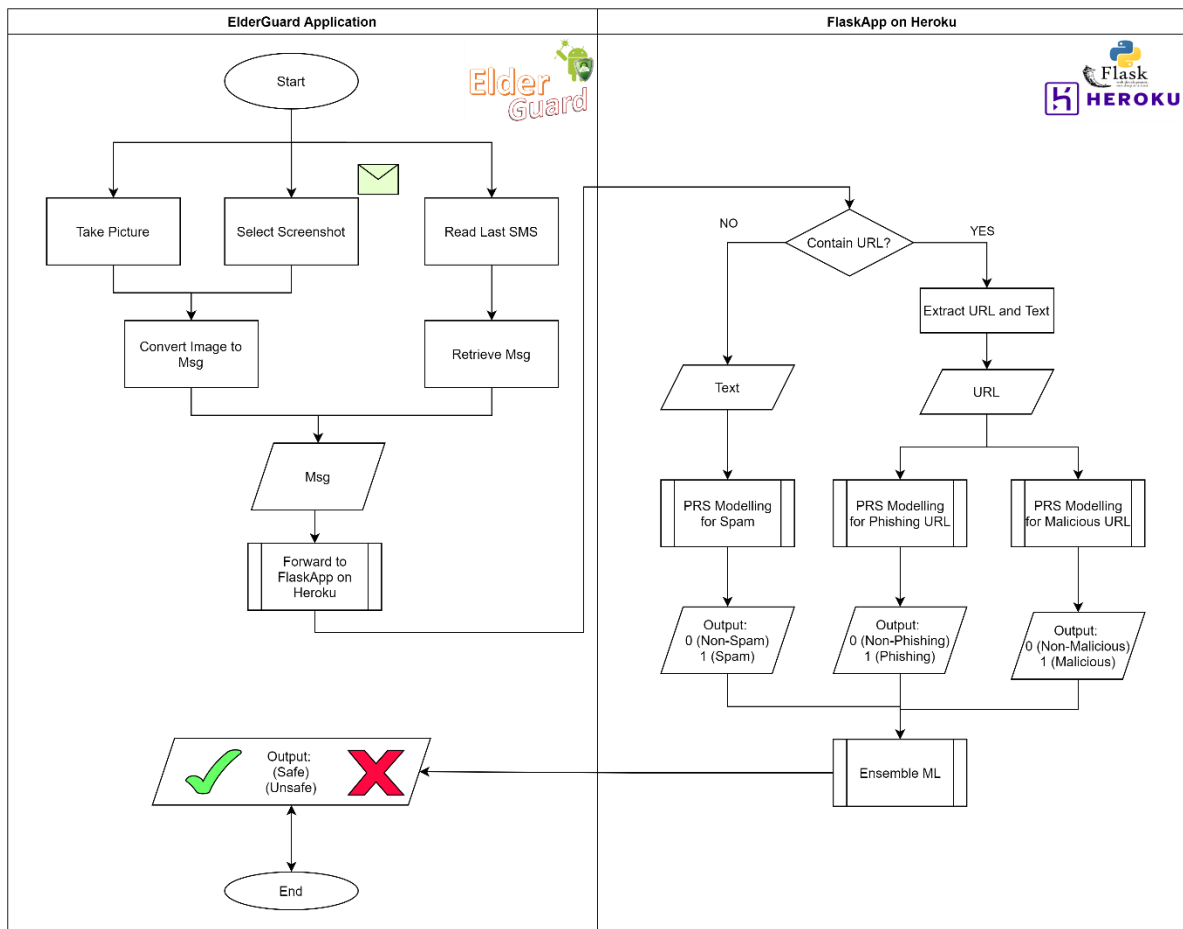


Figure 1. ElderGuard Application - System Overview

Figure 1 shows the System Overview of ElderGuard Application. It contains following TWO (2) main components:

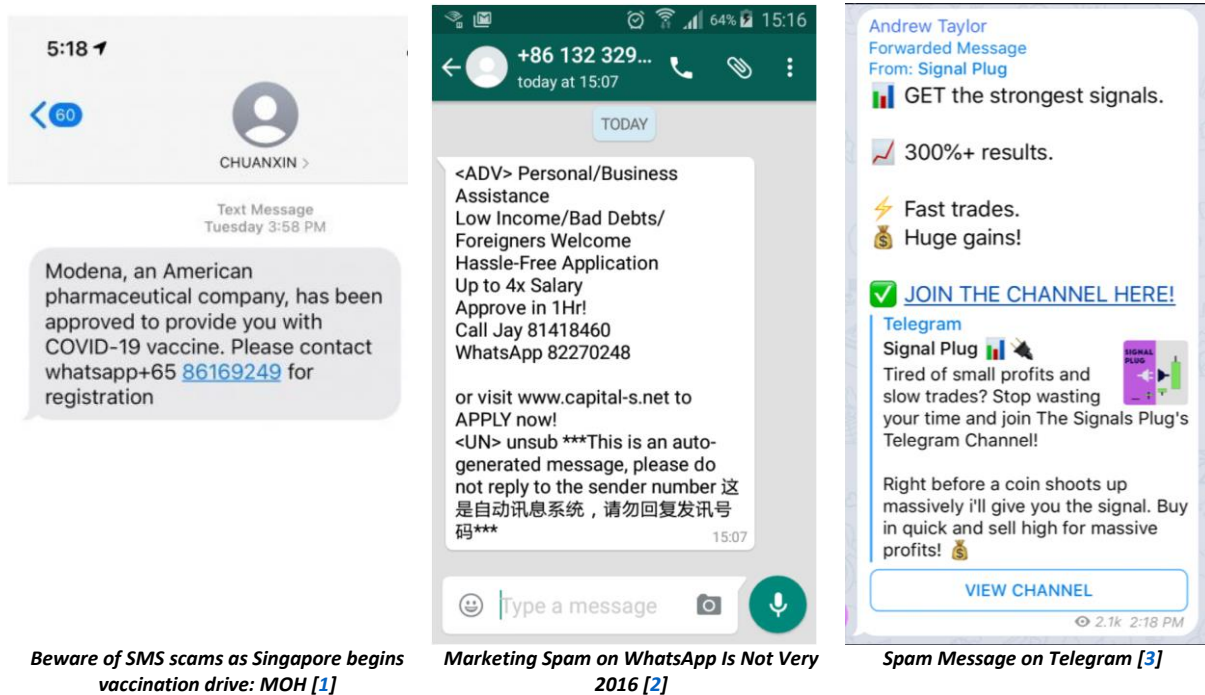
- ElderGuard – Android Application, interact with users and obtain messages to be classified as SAFE or UNSAFE.
- FlaskApp hosted on Heroku Platform, contains THREE (3) Pattern Recognition Modules for detecting Spam Text, Phishing URL and Malicious URL.

Given the text message feed into the FlaskApp from the Android Application, the text body of the message is extracted and will be sent to the Spam Text Module for inference. If there is any URL in the text message, they will be extracted and sent to the Phishing URL and Malicious URL module for inference. Finally, the outputs from the three modules are combined to form the output whether the text message is safe or unsafe. As long as one of the modules predicted spam or phishing or malicious URLs, the output will be unsafe. The final output is then sent back to the Android Application to present the results on screen to users.

4 Spam Text Module

4.1 Background and Related Works

The topic of spam text detection has been a classic machine learning problem in recent years. Spam text detector is often deployed in various forms of text-based interaction applications such as email and mobile SMS. Although spam email classification is a heavily researched topic, studies for spam text detection in mobile applications has been rising rapidly following the accelerated adoption of smart mobile phone globally. In general, spam text can come in multiple forms and channels including mobile SMS, messenger applications such as Facebook Messenger, WhatsApp, WeChat, Telegram etc. Despite most of the techniques used in spam email classification may be relevant in mobile spam text, different techniques are crafted and developed along the years to adapt to the nature of mobile text which is shorter in length and where spoken language are more loosely used.



Nowadays, we can see that there are many articles, literatures and published research papers that cover different traditional machine learning techniques being utilized in spam text classification problem. Some of the popular models used are like Naïve Bayes, Support Vector Machine (SVM), Decision Tree, Random Forest and K-Nearest Neighbours (KNN). These traditional machine learning techniques are simple and effective for the spam text classification problem and are able to achieve accuracy of more than 95% [4-6]. Hence, they are widely adopted in many applications of spam text classification.

On the other hand, adoption of Deep Neural Network (DNN) based methods in spam text classification problem only started to popularize in the past few years with the grow of research in the area of Natural Language Processing (NLP). Some of the common DNN model architectures used here are like Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM) and some other combination of DNN models like CNN-LSTM and Bi-LSTM. [7-8]

The development of the DNN based models in this context is very much related to the research in the techniques for feature extraction under NLP such as Bag-of-Words (BoW) representation and word embeddings. We can see that by utilizing these NLP based feature extraction with DNN models, the accuracy of the spam text detection is generally slightly higher than that of the traditional machine learning models. With the increasing complexity in the mobile text applications ecosystem, we observed that many literatures in spam text classification encounter some common challenges in their work. We have summarized 3 common challenges below which we attempt to address in our application:

1. Labelled Dataset Availability

Although text message corpus datasets are widely available, most of them are unlabelled dataset that are usually utilized for NLP related works. Besides, it is very costly to label each and every text message in these corpuses for the purpose of spam text classification.

2. Imbalance Dataset

The ratio of ham over spam messages is high and datasets used for spam classification are usually imbalance. This is simply because not everyone will receive spam messages in their daily usage of mobile text message applications.

3. Language and Slangs are loosely Used

Spam messages can come in different forms and languages depending on the geographical location as well as target victims. In addition, the language used on mobile text message are usually loosely spoken language. This is more apparent in Singapore where most of the mobile text messages are transmitted in “Singlish” where elements of multiple languages are incorporated.

4.2 Module System Pipeline

The spam text module is designed with the challenges mentioned above in mind and is incorporated as one of the important modules in our **ElderGuard** application. Figure 2 below is a visualization of the system pipeline within the Spam Text Module:

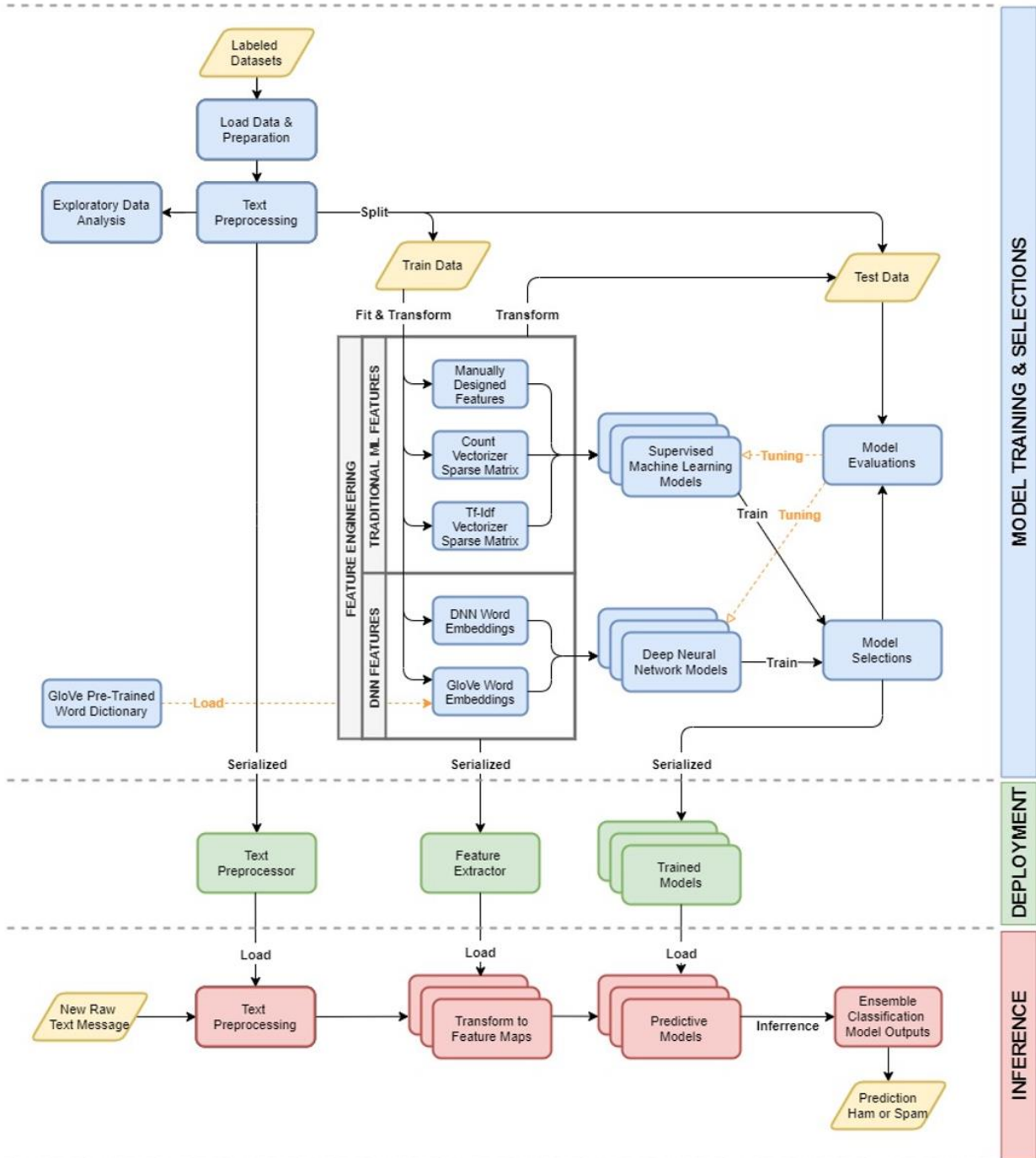


Figure 2. Process Pipeline for Spam Text Module

The training process starts with loading up the labelled datasets. The datasets will go through a text pre-processing process. This text pre-processing is mainly to remove some unnecessary information in the text messages. With pre-processed text messages, we split the dataset into training, validation and testing set.

One of the important steps in spam text classification task is feature engineering. As models usually only take numerical inputs, we will need to convert or transform the text messages into numerical representations that are able to capture the essential features for model training. We have designed and fitted various methods of feature extraction on the training datasets. With the trained feature extractor, we applied them on all 3 training, validation and testing dataset to transform them into feature maps which are in the form of numerical representations. The details on feature engineering will be discussed in Section 3.4.

The feature maps are then fitted into various machine learning models (traditional machine learning models and DNN based models) for training. We then go through the model selections and evaluation process to obtain a list of candidate models that return great performance in spam text classification. The process of model selections and evaluation will be discussed in Section 3.6.

From the pipeline diagram, we can see that the training process will produce 3 groups of objects namely the text pre-processor, feature extractors and shortlisted trained models which are then saved and serialized for deployment. Whenever a raw text message comes into the spam text module, the serialized objects will be called in sequence and inference will be performed. As there are multiple models selected for deployment, the final prediction of ham or spam are defined by the majority vote of the model inferences. The final output of the spam text module will be a binary score of 0 or 1 to indicate if the text message is ham or spam.

4.3 Dataset

To train and develop the spam text classification module, we have selected and utilized the following datasets:

1. SMS Spam Collection Data Set [9]

by the University of California Irvine (UCI) Machine Learning Repository.

The dataset comes in CSV format and is simple where each entry of data point consists of the full raw text message and its label, “ham” or “spam”. According to the source, the dataset is constructed from a few research corpora on the internet such as SMS spam messages from Grumbletext website, SMS ham messages from NUS SMS Corpus (NSC), SMS ham messages collected from Caroline Tag’s PhD Thesis and SMS Spam Corpus v0.1 Big. It has a total of 5,572 of text message with 4,825 labelled as “ham” and 747 labelled as “spam”. This dataset forms the majority of our labelled dataset for developing the spam text module.

2. SMS dataset [10]

From Kaggle open dataset

This dataset is much smaller in size as compared to the previous. It comes in the same format where each entry is a full raw text message and labelled as either “ham” or “spam”. The dataset has a total of 184 entries with 77 labelled as “ham” and 107 labelled as “spam”. The dataset is manually collected by the owner and is published on Kaggle.

3. Manually Collected Spam SMS

Collected from personal mobile phone SMS records

Total of 81 spam SMS that are manually downloaded and labelled from personal mobile phone SMS records. These spam SMS are collected on the basis that they must contain very pronounced element of spam message such as requesting receiver to sign up promotions or offering free prizes for replying or calling specified mobile number. The dataset is collected and saved in the same format as the previous 2 datasets used.

In the final combined dataset constructed based on the 3 sources listed above, we have a total of 5,837 text messages with 4,902 ham messages and 935 spam messages. The second and third dataset used here is our attempt to get more dataset for building the spam text module. We can see that the first dataset from UCI Machine Learning Repository is very useful for our application here as it contains ham messages from NSC which will help in addressing the issue of loosely used spoken "Singlish" in local context. We have also added in manually collected spam SMS based on personal mobile phone SMS records which ensure that our dataset includes spam messages which relate to the languages and types of local spam messages in Singapore.

Below are some examples of ham and spam messages extracted from the dataset:

Label	Raw Text
Ham	Beautiful Truth against Gravity.. Read carefully: "Our heart feels light when someone is in it.. But it feels very heavy when someone leaves it.." GOODMORNING
Ham	No..jst change tat only..
Ham	Yar lor wait 4 my mum 2 finish sch then have lunch lor... I whole morning stay at home clean my room now my room quite clean... Hee...
Spam	Thanks 4 your continued support Your question this week will enter u in2 our draw 4 ??100 cash. Name the NEW US President? txt ans to xxxxx
Spam	Dear Subscriber ur draw 4 ??100 gift voucher will b entered on receipt of a correct ans. When was Elvis Presleys Birthday? TXT answer to xxxxx
Spam	Phase 2. Amazon Alliance is Hiring now. Get your phone and start to earn 300-800 SGD daily. For more info, please click on the link : wa.me/xx-xxxxxxx

[^] SMS or phone number are masked off

Below is a summary of the construction of the dataset used:

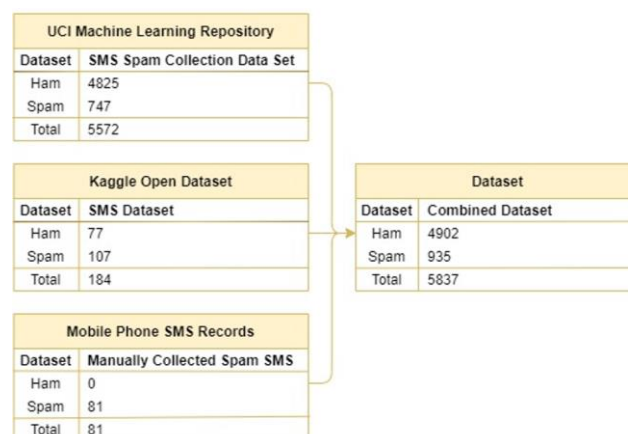


Figure 3. Spam Text Module Dataset Construction

Text Pre-processing

Before proceeding to feature engineering using the dataset, we applied some common text pre-processing to the data. A summary of text pre-processing performed is as follows:

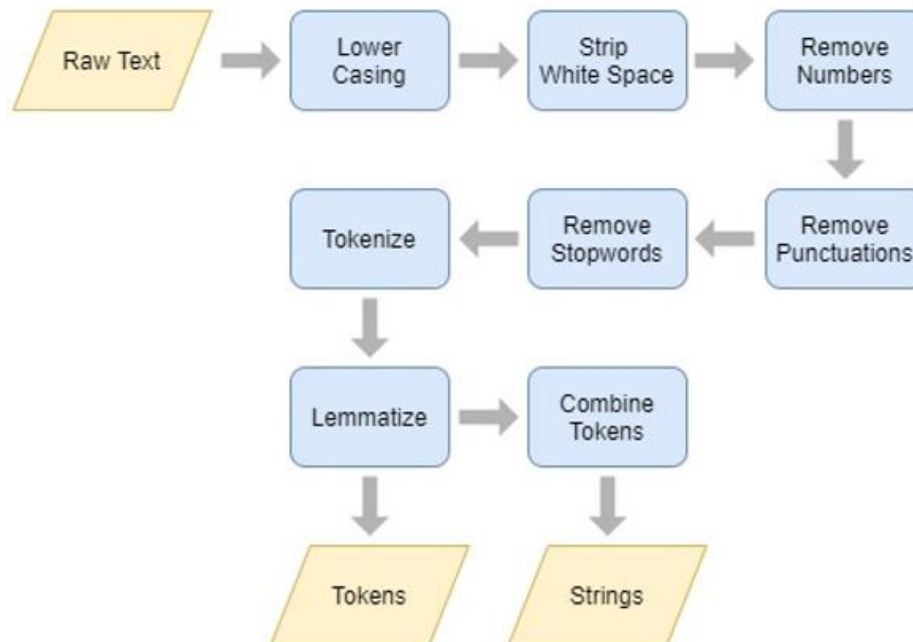


Figure 4. Text Pre-processing in Spam Text Module

Notice that we have 2 types of outputs from the text pre-processing. This is because the token-based and string-based format data will be utilized in different feature engineering methods in the later steps. The dataset is then split into the training, validation, testing dataset using 70%/15%/15% ratio. The same split index is applied to the raw full text dataset, token-based and string-based format dataset from the text pre-processing.

Exploratory Data Analysis (EDA)

This section documents the exploratory findings and statistical analysis of the dataset. The objective of EDA is to visualize the data statistically in order gain a better understand of the dataset.

1. Ham vs Spam Message Count

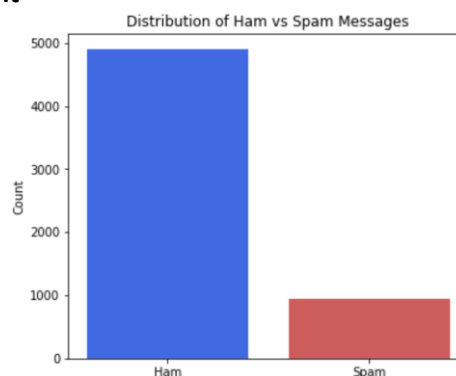


Figure 5. Ham vs Spam Message Count

We can see that the dataset is imbalance where the ratio of ham to spam message is almost close to 5:1. This will affect the choices of evaluation metrics for model selections.

2. Ham vs Spam Message Length

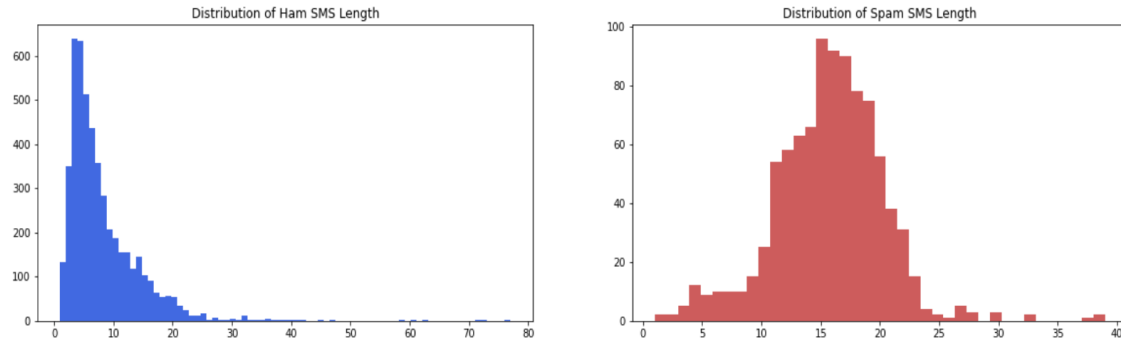


Figure 6. Ham vs Spam Message Length

Overall, ham text message is slightly shorter than spam text message. We can observe that majority of ham text message length range around 5 to 20 words while spam text messages range around 10 to 25 words.

3. Ham vs Spam Message Frequent Words

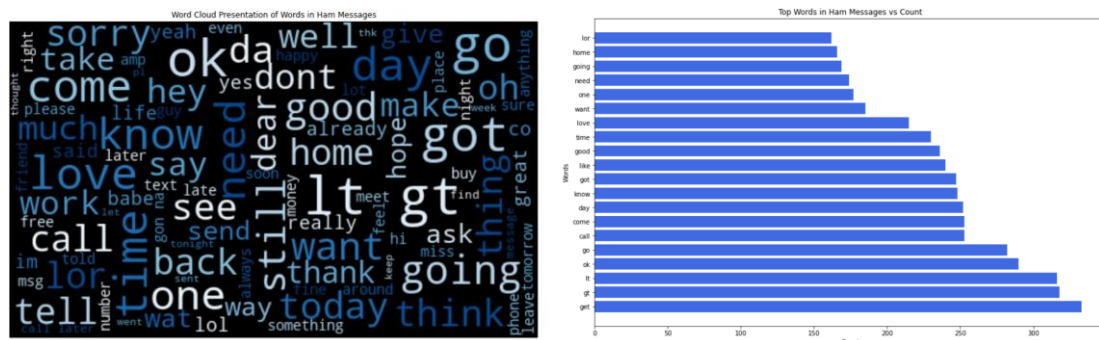


Figure 7. Word Cloud and Top 20 Frequent Words in Ham Messages

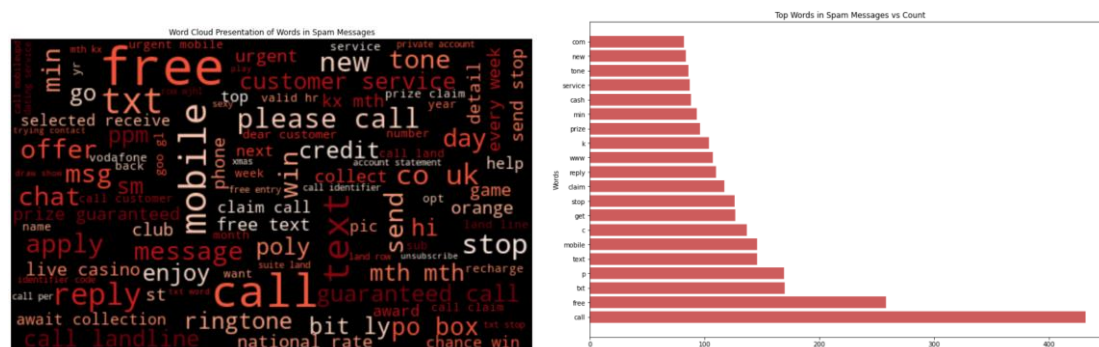


Figure 8. Word Cloud and Top 20 Frequent Words in Spam Messages

The group of frequent words in ham messages can be very different from that from spam messages. The frequent words in ham messages appears to be very common words in our daily communications such as “go”, “ok”, “know”, “come” etc. On the other hand, the frequent words in spam messages are more of a requesting or demanding tone such as “reply”, “free”, “send”, “call” etc.

4. Ham vs Spam Word Embedding 2D Visualization

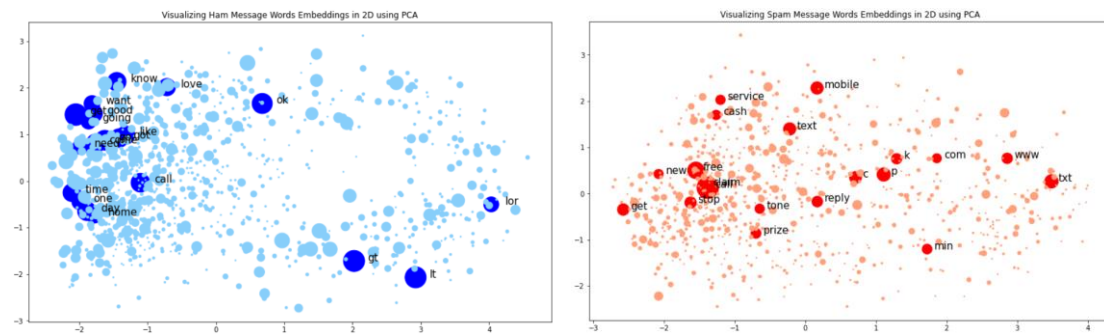


Figure 9. 2D Visualization of Word Embeddings Using PCA

Based on one of the feature engineering that we used in our pipeline, word embeddings, we can create 2-dimension visualization of the word embeddings. The above figure is the visualization of the word embeddings in 2D, achieved through reducing the 50-dimension pre-trained Global Vectors for Word Representation (GloVe)[11] word embeddings to 2-dimension using Principal Component Analysis (PCA). The size of the circle correlates to the frequency of the word in ham and spam messages and the top 20 frequent ham and spam words are highlighted in the diagram.

One observation we can see from the diagram is that most of the common ham words are repetitive and they are concentrated in one area. On the other hand, the common spam words are more spread out across in the 2-dimension space.

4.4 Feature Engineering

The methods of feature design and extraction is generally separated into 2 groups. One is for traditional ML models while the other is for DNN based models. Total of 5 feature representation methods are experimented, 3 of them are used in the traditional ML models and the remaining 2 in DNN based models.

The feature extractors are all fit based on the training dataset and are used to transform all 3 training, validation and testing dataset into feature maps. The feature maps are then used for training the classification models in the next step. Below are the 5 feature representation methods discussed in detail:

1. Manually Designed Features (Traditional ML Models)

The objective of manually designed features is to extract useful information or elements from text messages that are highly indicative of spam messages. From a user perspective, we are able to identify spam messages by just scanning through the text message. This is because there are traits that we can pick up from the text messages that are related to spam. Hence, the manually designed features will utilize these traits as features.

The handcrafted features include:

- Count of Mathematical Symbols
- Count of Special Symbols
- Count of Fully Uppercased Words
- Count of Occurrence of Top 20 Spam Words
- Indication of Occurrence of SMS or Phone Number in Text Message
- Length of Tokens after Text Pre-processing

For the feature (a), (b), (c) and (e), they are extracted using the original full raw text since this information are removed after text pre-processing. While feature (d) and (f) are extracted based on the data after text pre-processing. We can see that in many spam messages, mathematical or special symbols often occur in the text messages. Besides, most of the spam messages would also request or demand the receiver to call or reply to the given SMS or phone number. In the final feature map, we have 6 columns of data that corresponds to the 6 features described above. A min-max scaler is then applied to the feature map to normalize the data.

2. Bag-of-Words (BoW) – Count Vectorizer (Traditional ML Models)

We utilize Tensorflow and Keras python package, tokenizer function to convert the data into sparse matrices where each entry represents the occurrence count of different words in the text messages. The tokenizer is trained and fitted using the training dataset and vocabulary size is 6,389 of unique words. The sparse matrix will form the feature maps which will be used in various ML Models. As observed from the EDA process, the top frequent words in ham and spam messages can be very different. Hence, the word count information in the sparse matrix can be useful in identifying the spam text messages. The number of features in the sparse matrix is 6,390 which is equal the vocabulary size of the tokenizer plus an index for out of the vocabulary word.

3. Bag-of-Words (BoW) – Tf-Idf Vectorizer (Traditional ML Models)

Similar to Count Vectorizer methods, we utilize the same tokenizer function to convert the data into sparse matrices. However, the entries in the sparse matrix now represent the term frequency-inverse document frequency (Tf-Idf) score of different words in the text messages. The Tf-Idf vectorizer attempts to utilize the normalized word count information in identifying the spam text messages. This is slightly different compared to Count Vectorizer where only the count information of more unique words is captured here and will have a higher frequency score. The number of features in the sparse matrix is same as Count Vectorizer, 6,390 features as they share the same word vocabulary.

4. Word Embeddings trained from Data (DNN based Models)

We experimented with word embeddings in DNN based models as many literatures have shown that DNN based models perform slightly better than traditional ML models when they are trained with word embeddings. The word embeddings are trained from our own training dataset in the word embedding layer under the Tensorflow or Keras DNN model architecture. In practice, the word embeddings are actually trained together in the same procedure or epochs as the model training for classification task. To prepare the dataset for word embedding training in a DNN model architecture, we transformed the dataset into a word index based sparse matrix. Each data point which is a list of tokens is converted into a vector where each entry represents the corresponding token's or the word index in the tokenizer word vocabulary. As each text message in the dataset has different length (number of tokens) after text pre-processing, we pad the vectors with 0 up to a length of 30 tokens or words long. Data points with length (number of tokens) of more than 30 will have the word indices of the first 30 words in vectors. Data points with length (number of tokens) of less than 30 will have padded with value 0 to make up the vectors of length 30. The first 30 words are selected for constructing the sparse matrix as evident in EDA, most of the spam messages have length up to 30 words. Hence, the final word index feature map we used for word embeddings training has a size of 30 that represents the word index of the first 30 words.

5. Pre-trained Word Embeddings GloVe (DNN based Models)

We have also experimented the DNN based models using pre-trained word embeddings from Global Vectors for Word Representation (GloVe) [11]. A dictionary of word embedding is downloaded directly from the source website and is used to convert each token in the dataset into word embedding of 50-dimension vector. Similarly, we only applied word embeddings or pad with 0 up to the first 30 words in each data point. The word embedding dataset is then used to train various DNN based models in the next step. For DNN models that are taking pre-trained word embeddings as input, they will be no word embedding layer trainable in their model architecture. We can see that out of the 6,389 unique words found in the training dataset, the GloVe word embedding only has matching for 5,181 words. Hence, the remaining 1,208 words are converted into zero vectors of length 50. The feature map dimension of each data point under the word embedding dataset is 30x50 which corresponds to the first 30 words and vector of length 50 for each word.

Below is a summary of the 5 feature engineering methods in the Spam Text Module:

Feature Engineering Method	Model Type	Input Dataset Type	Feature Dimension (each data point)	Word Vocabulary Size
Manually Designed Features	Traditional ML	Raw full Text, Token-based	6	
BoW – Count Vectorizer	Traditional ML	String-based	6,390	6,389
BoW – Tf-Idf Vectorizer	Traditional ML	String-based	6,390	6,389
Word Embeddings trained from Data	DNN Based	String-based	30	6,389
Pre-trained Word Embeddings GloVe	DNN Based	Token-based	30x50	5,181

4.5 Model Training

Traditional Machine Learning Models

In traditional ML models, we utilized all 3 types of feature maps (manually design features, BoW – Count Vectorizer and BoW – Tf-Idf Vectorizer) in training 6 types of models namely Naïve Bayes, Logistic Regression, Random Forest, Support Vector Machine (SVM), AdaBoost and K-Nearest Neighbour (KNN). Both training and validation dataset are used for training and tuning the models hyperparameters. The testing dataset is used to evaluate the performance of the model on unseen dataset. This is done to prevent information leakage from the testing dataset during the hyperparameter tuning procedure. Besides, evaluation based on unseen data is very important here as our dataset has an imbalance ratio of ham vs spam.

Below table is a summary of the models and hyperparameters that are experimented in training:

Model	Feature Input Types	Hyperparameters Experimented
Naïve Bayes	Manually Designed Features BoW – Count Vectorizer BoW – Tf-Idf Vectorizer	-
Logistic Regression		Regularization Strength - 3, 2, 1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001
Random Forest		Number of Trees - 3, 5, 10, 25, 50 Number of Features - 1, 2, 4, 10, 20, 50
SVM		Gamma - 0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3
AdaBoost		Number of Trees - 25, 50, 100 Tree Depth - 1, 2, 4, 10
KNN		Number of Neighbour - 3, 5, 7, 9, 11

Deep Neural Network Models

Similarly, for DNN based model training, we utilized both the pre-trained word embeddings as well as word embeddings trained from dataset in various DNN model architecture. We have experimented with 4 types of model architecture namely Dense Classifier, CNN, LSTM and CNN-LSTM. Intuitively, the Dense Classifier would not be very useful here as there is no feature extraction layers in the model architecture. The CNN, LSTM and CNN-LSTM are among the popular DNN models that are used in classification task. All DNN models are constructed and experimented using the Tensorflow and Keras deep learning packages in Python.

We can see that the number of parameters in the DNN models with pre-trained word embeddings range around 100k and is much lower as compared to DNN models with word embedding training layer which can go up to 420k. This is simply because the bulk of the number of parameters to train in the later mainly comes from the word embedding layer. We are using vectors of length 50 to train each word in the vocabulary. As we are using the full vocabulary size of 6,389 words, there are a total of close to 320k parameters to train in the word embedding layer alone. The remaining 100k parameters are similar to that of the DNN models with pre-trained word embeddings. The activation function used are all ReLu except the last Dense layer which uses sigmoid as this is a binary classification problem.

Below is a summary of the specifications of the DNN models experimented:

Model	Word-Embeddings	Number of Parameters	Loss Function	Optimizer (Learning Rate, LR)
Dense	GloVe	844,545	binary_crossentropy	Adam (LR = 0.00015)
CNN	GloVe	70,641	binary_crossentropy	Adam (LR = 0.0002)
LSTM	GloVe	104,297	binary_crossentropy	Adam (LR = 0.0001)
CNN-LSTM	GloVe	118,225	binary_crossentropy	Adam (LR = 0.0001)
Dense	Trained From Data	725,319	binary_crossentropy	Adam (LR = 0.0002)
CNN	Trained From Data	359,435	binary_crossentropy	Adam (LR = 0.00015)
LSTM	Trained From Data	423,355	binary_crossentropy	Adam (LR = 0.00004)
CNN-LSTM	Trained From Data	407,851	binary_crossentropy	Adam (LR = 0.00004)

Many iterations and tuning are conducted to each of the DNN models experimented in order to optimize the performance. Below are some layers or modification methods that we have used in tuning the DNN models:

Tunings	Description
Dropout	Dropout layers can be applied after Convolutional layers or Dense layers to reduce overfitting
L2 Regularization	Regularizations are applied to Convolution layers or Dense layers to slow down the learning rate
Batch Training	Batch size of 10 to 50 are used for trainings in each epoch

4.6 Model Evaluation

As observed from the EDA process, the dataset has an imbalance ratio of ham over spam messages. Suitability of evaluation metrics becomes an important factor for model evaluation and selections. Besides the usual accuracy metric, metrics like precision, recall, F1 score and Area Under the Curve (AUC) score are taken into consideration for model evaluations.

Traditional Machine Learning Models

Based on these evaluation metrics, hyperparameters of the traditional ML models are tuned such that they produce optimized scores using the training and validation dataset. With the optimized hyperparameters models, the final evaluation is performed based on the testing dataset.

Below are the model evaluation findings for the traditional ML models experimented:

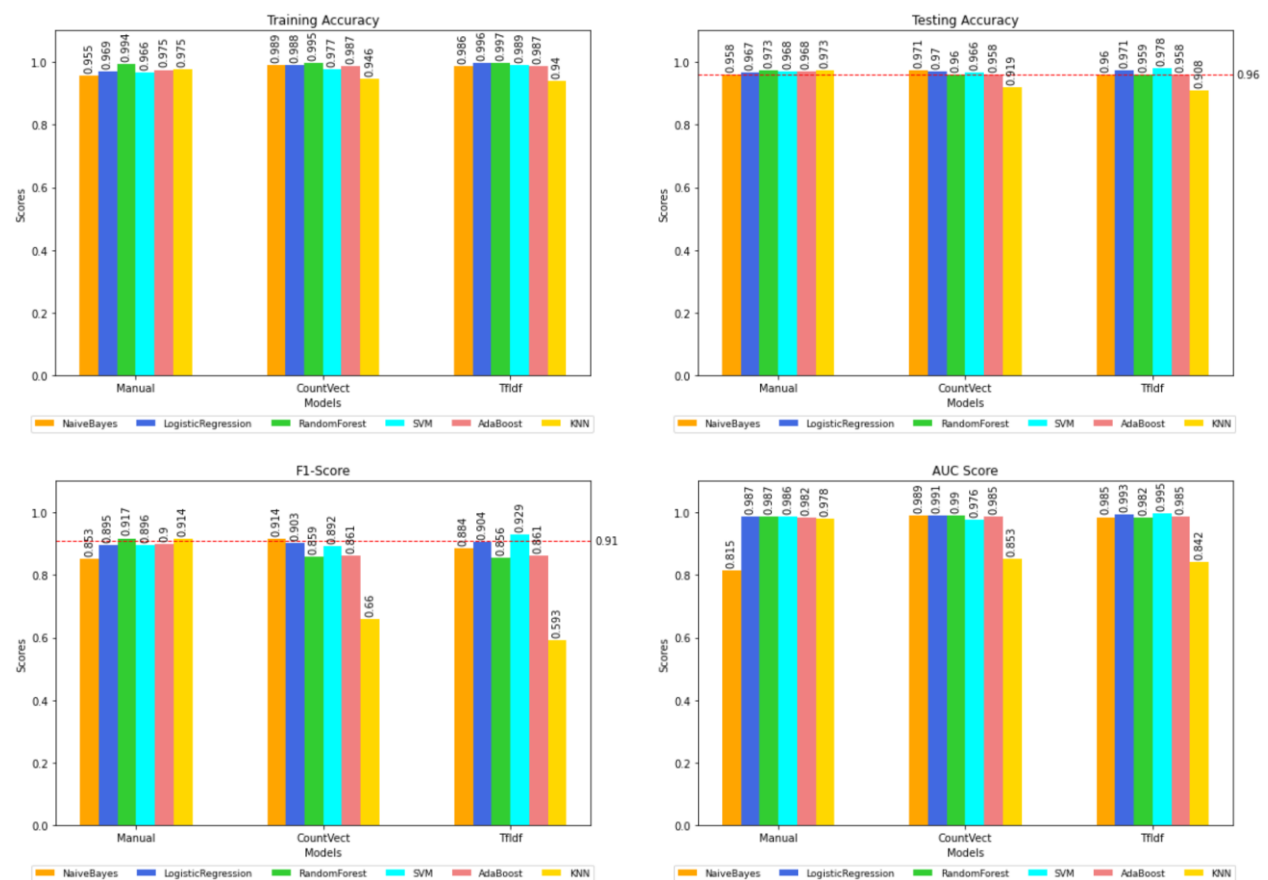


Figure 9. Model Evaluation for Traditional ML Models

Most of the models performed well for dataset and testing accuracies are at least 90%. Based on the testing accuracies and F1 scores, Random Forest and KNN performed the best using the manually designed features with the highest F1 scores at 91.7% and 91.4% respectively. Naïve Bayes performed best when using the BoW-Count Vectorizer features with testing accuracy at 97.1% and F1 score of 91.4%. SVM performed best among all comparison when using BoW-Tfidf Vectorizer features with both highest testing accuracy at 97.8% and F1 score at 92.9%. The performance of Logistic Regression is more average across the 3 types of feature maps used with accuracies range around 96-97% and F1 score around 89-90%. Finally, the AdaBoost performed the worst as compared to other models here with accuracies below 97% and F1 score below 90%.

The table below is a summary of the metric scores across the models:

Features	Hyperparameters	Training Accuracy	Testing Accuracy	F1 Score	AUC Score
Naives Bayes					
Manually Designed Features	-	95.5	95.8	85.3	81.5
BoW – Count Vectorizer	-	98.9	97.1	91.4	98.9
BoW – Tfidf Vectorizer	-	98.6	96.0	88.4	98.5
Logistic Regression					
Manually Designed Features	Regularization Strength = 3	96.9	96.7	89.5	98.7
BoW – Count Vectorizer	Regularization Strength = 0.3	98.8	97.0	90.3	99.1
BoW – Tfidf Vectorizer	Regularization Strength = 0.03	99.6	97.1	90.4	99.3
Random Forest					
Manually Designed Features	Number of Tress = 50, Features per Tree = 1	99.4	97.3	91.7	98.7
BoW – Count Vectorizer	Number of Tress = 10, Features per Tree = 20	99.5	96.0	85.9	99.0
BoW – Tfidf Vectorizer	Number of Tress = 10, Features per Tree = 50	99.7	95.9	85.6	98.2
SVM					
Manually Designed Features	Gamma = 1	96.6	96.8	89.6	98.6
BoW – Count Vectorizer	Gamma = 0.1	97.7	96.6	89.2	97.6
BoW – Tfidf Vectorizer	Gamma = 0.003	98.9	97.8	92.9	99.5
AdaBoost					
Manually Designed Features	Number of Trees = 50, Tree Depth = 1	97.5	96.8	90.0	98.2
BoW – Count Vectorizer	Number of Trees = 100, Tree Depth = 1	98.7	95.8	86.1	98.5
BoW – Tfidf Vectorizer	Number of Trees = 100, Tree Depth = 1	98.7	95.8	86.1	98.5
KNN					
Manually Designed Features	Number of Neighbor = 9	97.5	97.3	91.4	97.8
BoW – Count Vectorizer	Number of Neighbor = 3	94.6	91.9	66.0	85.3
BoW – Tfidf Vectorizer	Number of Neighbor = 3	94.0	90.8	59.3	84.2

Deep Neural Network Models

Below are the findings for the DNN based models experimented:

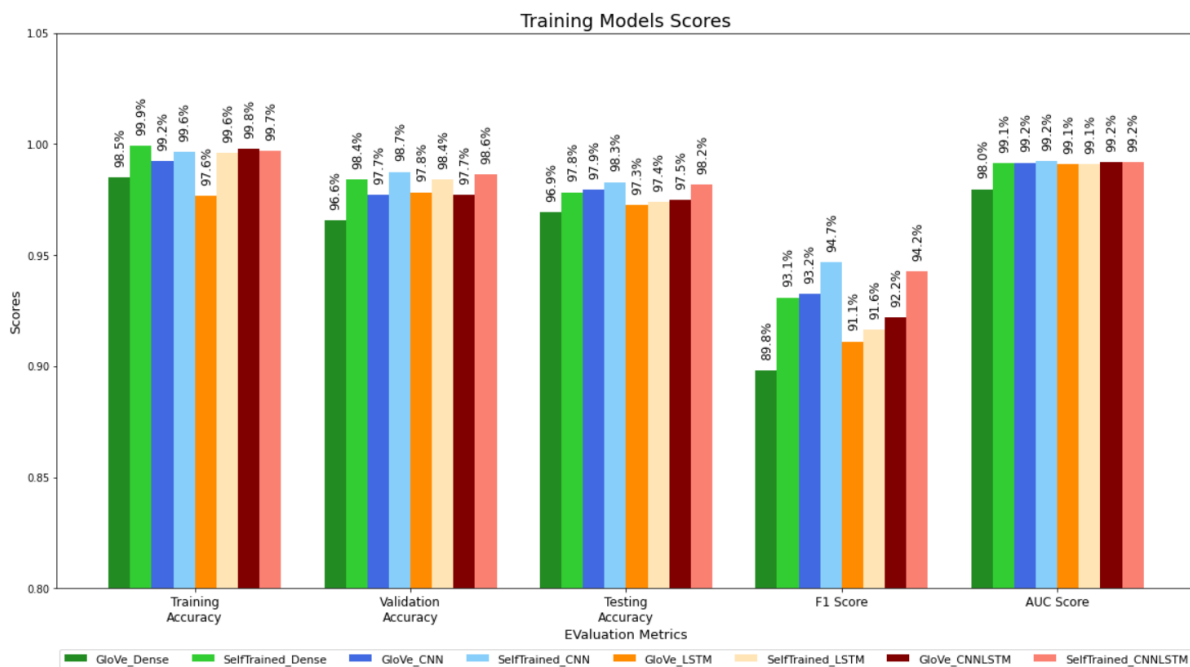


Figure 10. Model Evaluation for DNN Based Models

An observation based on the results is that the models with word embeddings trained from the dataset generally performed better the models with pre-trained GloVe word embeddings. This is expected because the GloVe embeddings were trained using unsupervised learning algorithm to

obtain a general vector representation of words and the corpus used in the training was Wikipedia 2014 and Gigaword 5. On the other hand, the word embeddings trained based on our dataset will be more useful for the spam text classification task as the backpropagation mechanism will steer the embeddings trained to be more catered to nature of ham and spam text messages. We can see that the CNN and CNN-LSTM model with word embeddings training have the highest testing accuracies of more than 98% and f1-score of more than 94% among the models experimented. This is aligned with the literature findings where DNN based models with word embeddings can perform slightly better than traditional ML models.

Model	Word-Embeddings	Training Accuracy	Testing Accuracy	F1 Score	AUC Score
Dense	GloVe	98.5	96.9	89.8	98.0
CNN	GloVe	99.2	97.9	93.2	99.2
LSTM	GloVe	97.6	97.3	91.1	99.1
CNN-LSTM	GloVe	99.8	97.5	92.2	99.2
Dense	Trained From Data	99.9	97.8	93.1	99.1
CNN	Trained From Data	99.6	98.3	94.7	99.2
LSTM	Trained From Data	99.6	97.4	91.6	99.1
CNN-LSTM	Trained From Data	99.7	98.2	94.2	99.2

To reduce the effect of randomness in model training and evaluations, we performed cross validation on the CNN and CNN-LSTM model with word embedding trained from dataset. The training, validation and testing dataset are combined to form the full dataset. A 5-fold stratified cross-validation are repeated 5 times using the full dataset. There are total of 25 cross validation training performed and the final results are obtained by averaging the evaluation metrics across the 25 sets of training. Below are the results of the cross validations performed on CNN and CNN-LSTM model with word embedding trained from dataset:

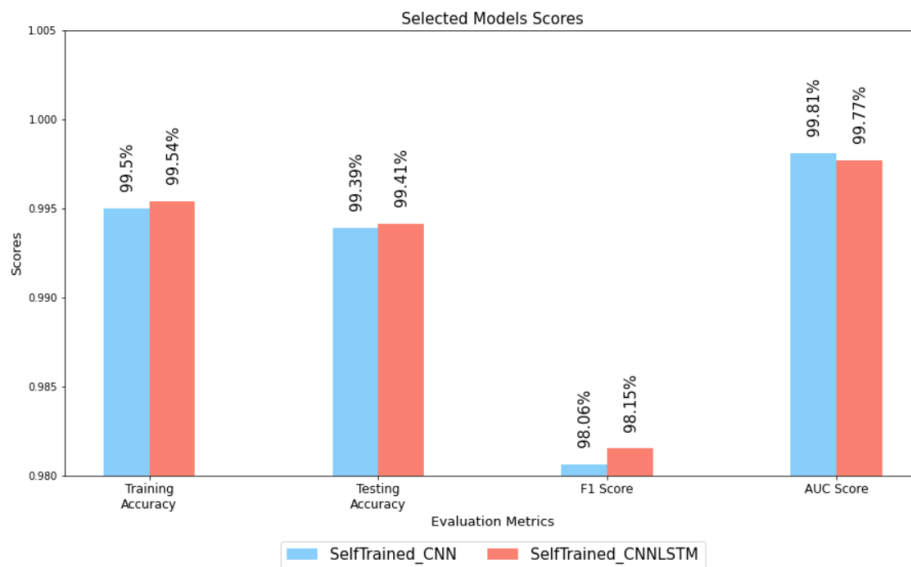


Figure 11. Cross Validation Results

Model	Word-Embeddings	Average Training Accuracy	Average Testing Accuracy	Average F1 Score	Average AUC Score
CNN	Trained From Data	99.50	99.39	98.06	99.81
CNN-LSTM	Trained From Data	99.54	99.41	98.15	99.77

Based on the cross-validation results, we can observe that both CNN and CNN-LSTM have almost the same performance with average testing accuracy at around 99.4% and average F1 score at around 98.1%.

4.7 Model Selection & Deployment

Although most of models trained here have great performance with accuracies above 90%, there is a different set of properties that we need to consider when selecting models for deployment. Below is a list of consideration that we take into account when selecting the final list of models for deployment:

Consideration	Description
Model Evaluation	To select models with best performance based on the training results.
Type of Features	To cover all types of features as different features may capture different important information that may be useful in spam text classifications
Light Weight	Models with less weights or parameters trained. This ensure that file size of the serialized models is small for deployment.
Fast Inference	The inference should be fast enough to ensure that the app can run smoothly

For traditional ML models, we have selected 3 models for deployment namely Naïve Bayes with BoW-Count Vectorizer, SVM with BoW-Tfidf Vectorizer and KNN with manually designed features. The selection criteria for model evaluation metrics are defined as >96% of testing accuracy and >91% of f1 score. We can see that aside of the 3 selected models mentioned here, Random Forest with manually designed features satisfied the selection criteria also. However, it is not selected for deployment mainly due to the large file size (>60mb) of the serialized model file. The 3 models selected here have file size of less than 40mb in total.

For DNN based models, we have selected CNN with word embedding trainings for deployment. Even though both CNN and CNN-LSTM with word embedding trainings have almost the same performance and are the best among the models trained, the CNN is selected due to the number of parameters is slightly lower at 359,435 as compared to CNN-LSM at 407,851. This will help to ensure that the total file size for deployment is lower and at the same time will allow a slightly faster inference time.

The selected CNN model has 1 word embedding layer, 3 Conv1D layers with MaxPooling and 5 Dense layers for the classification tasks. All Conv1D and Dense layers use ReLu activation function except the last Dense layer which uses the Sigmoid activation function for a binary class of non-spam or spam.

Below is a summary of the models selected for deployment:

Model	Type	Features
Naïve Bayes	Traditional ML	BoW – Count Vectorizer
SVM	Traditional ML	BoW – Tfidf Vectorizer
KNN	Traditional ML	Manually Designed Features
CNN	DNN Based	Word Embeddings Trained from Dataset

In the deployment process, the text pre-processor, feature extractors and selected models are serialized and saved. Whenever the application called the spam text module, these 3 groups of objects will be loaded into memory to perform the inference given the raw text message. The final output of the inference will be a majority vote of the 4 models deployed whether the given text message is non-spam or spam.

5 Phishing URL Module

5.1 Background

Phishing is one of the most popular attack conducted by phishers to steal personal information and financial details like usernames, credentials. It can cost internet and mobile users billions of dollars. According to Phishing and Email Fraud Statistics 2019 conducted by ReTruster[12], phishing attempts have grown 65% in the last year and 30% of phishing messages get opened by targeted users. This suggested the need of an application that can help to identify or warn the users whether visited sites are potential phishing sites.

There have been multiple methods used to detecting phishing sites, either based on the content of the pages, URL page rank and so forth. In this project, we presented Machine Learning approach to detect phishing sites, in particular a supervised machine learning with labelled datasets is conducted.

5.2 Module System Pipeline

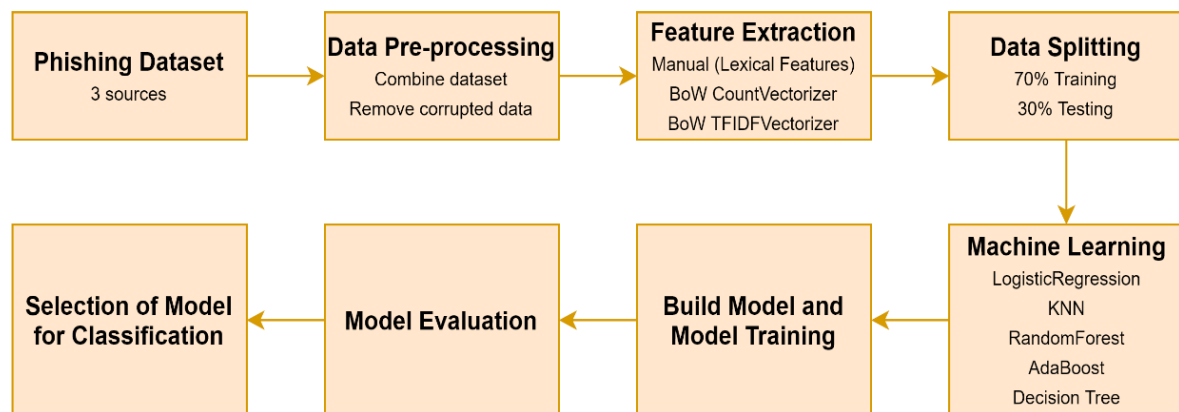


Figure 11. Pattern Recognition Process for Phishing Module

5.3 Phishing Dataset

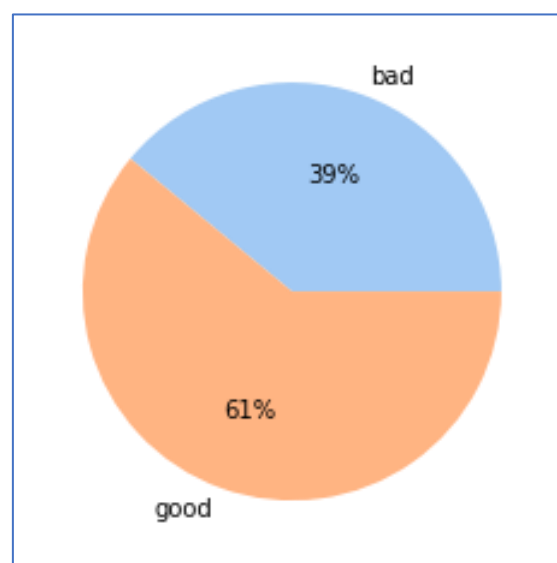


Figure 12. Distribution of Good and Bad Phishing URLs

Three URLs dataset are combined and used in this project for training Phishing Classification Model. Table 1 shows the sources of datasets and distribution of good and bad URLs for each source. All the datasets contain two columns, URL and Label. The label columns are the prediction column that has two categories, Good and Bad. Good refers to site is a non-phishing site, while Bad refers to site is a phishing site. There is no missing value in the dataset.

Three datasets are combined to ensure no imbalanced data when train the phishing classification model. Figure 13 shows the distribution of Good and Bad label upon combining the three datasets mentioned above. 61% of entries are categorized as good URLs and 39% are categorised as bad URLs.

Kaggle Phishing Dataset

The first dataset is sourced from Kaggle - Phishing Site URLs[13]. It contains 549,346 entries with 392,924 labelled as Good, and 156,422 labelled as Bad.

Kaggle Malicious URLs

The second dataset is sourced from Kaggle – Malicious URLs[14]. It contains defacement URLs, phishing URLs, and malware URLs. Only Phishing URLs are extracted and used in this project. Total of 94,111 entries of Bad URLs (phishing URLs) are added.

OpenPhish Dataset

The third phishing dataset is sourced from OpenPhish[15]. It contains 1014 entries URLs categorized as Phishing sites.

Source	Good	Bad
Kaggle Phishing Site ¹³	392,924	156,422
Kaggle Malicious URL (Phishing) ¹⁴		94,111
OpenPhish ¹⁵	-	1,014
Total	392,924	251,547

5.4 Feature Extraction

Few features extraction methods are used in this project to find a new representation of the important features of a phishing site, including lexical features (manual extraction), and Bag-of-Words (BoW) with CountVectorizer, and TF-IDF as feature representation.

1. Lexical Features - Manual Feature Extraction

URL → http://www.forums.news.cnn.com/%7Eguido/Python.html'
 Netloc → www.forums.news.cnn.com
 First Directory → /%7Eguido
 Path → /%7Eguido/Python.html
 Subdomain → forums.news
 Domain → cnn
 Suffix → com

Figure 13. Sample of URL Components

24 different lexical features are used to differentiate between phishing and non-phishing URLs. Table below shows the lexical features extracted from the URL for classification and its variable name used in coding.

URL Components	Lexical Features	Variable Name
URL	Total URL Length	tot_url_length
Netloc	NetLoc Length	netloc_length
Path	Path Length	path_length
First Directory	First Directory Length	fd_length
Suffix	Suffix Length	suffix_length
Subdomains	Number of Subdomains	num_subdomains
Domain	Count of "-" in Domain	domaindesk
URL	Count of "-" in URL	urldesk
Path	Count of "-" in Path	pathdesk
Domain	Count of "_" in Domain	domain_
URL	Count of "_"	countunderscore
Path	Count of "_" in Path	path_
URL	Count of directory	count_dir
Path	Count of "." in Path	path_fsdot
URL	Count of "@"	countat
URL	Count of "?"	countquest
URL	Count of "%"	countpercent
URL	Count of "."	countdot
URL	Count of "="	counteq
Path	Count of HTTP in Path	count_http_inpath
Path	Count of HTTPS in Path	count_https_inpath
Path	Count of WWW in Path	count_www_inpath
URL	Count of digits	countdigits
URL	Count of letters	countletters

Based on the figure below, it is observed that there are a number of URLs labelled with phishing sites have one or more count of HTTP/WWW/HTTPS in path as compared to URLs labelled with good sites.

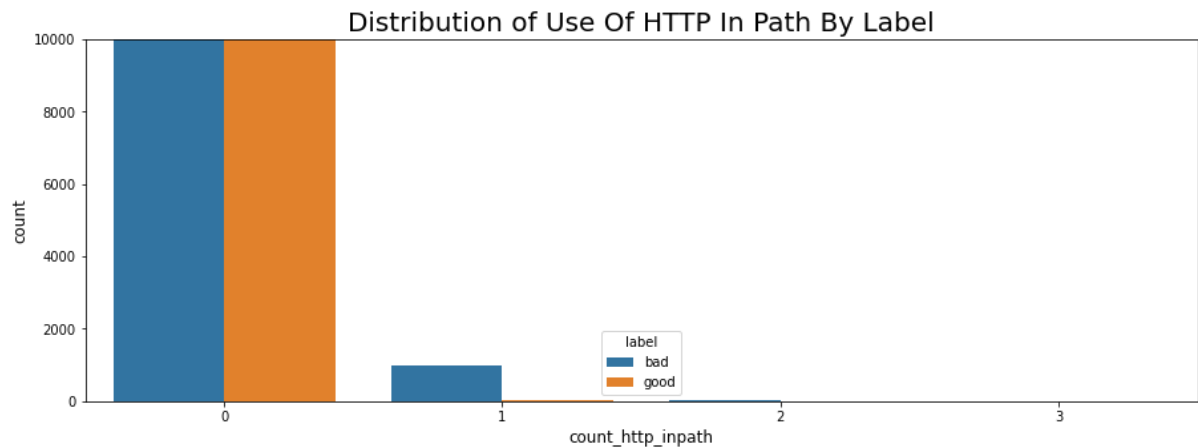


Figure 14. Distribution of HTTP in PATH

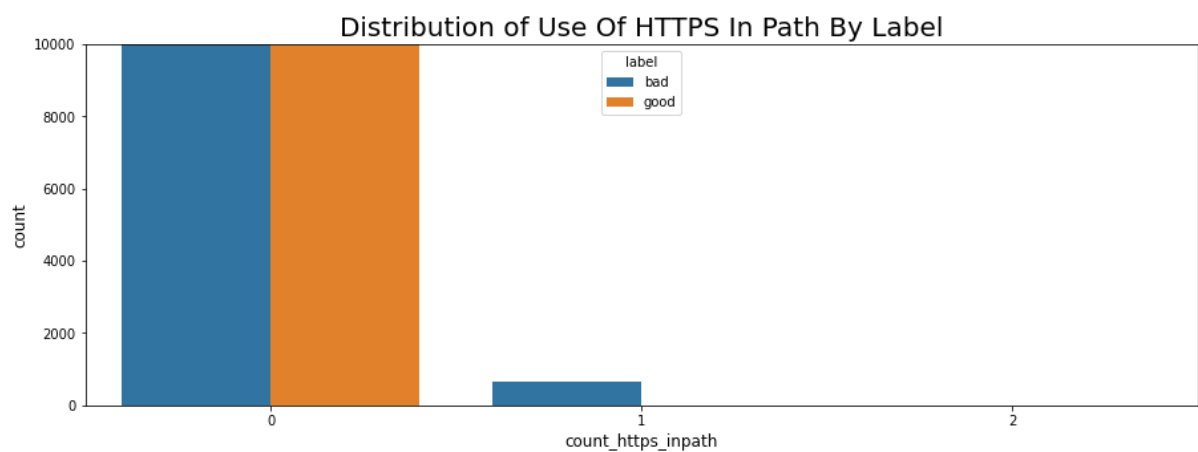


Figure 15. Distribution of Use of HTTPS in PATH

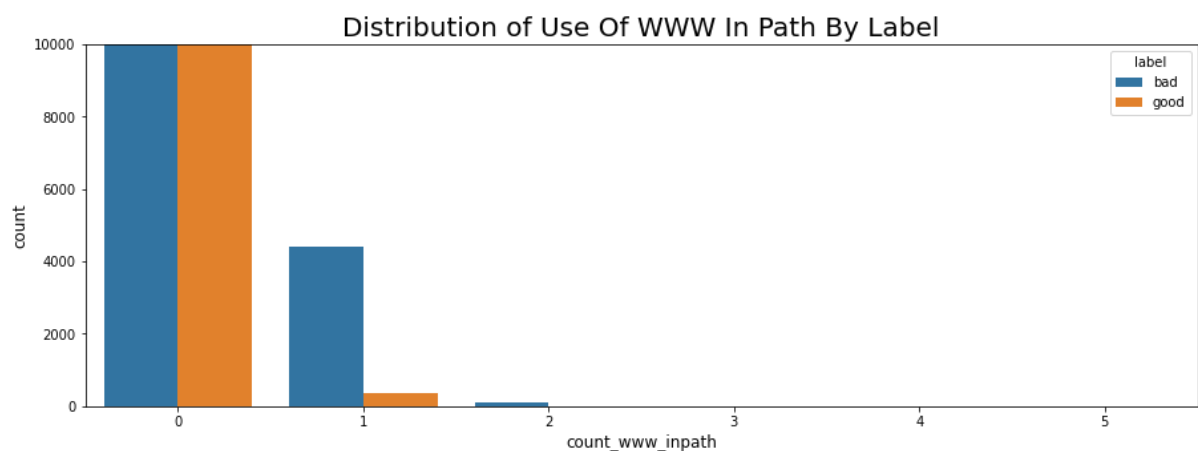


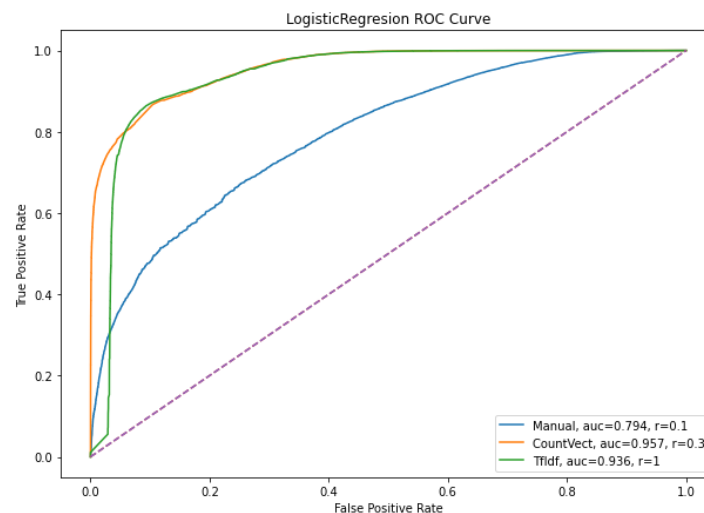
Figure 16. Distribution of Use of WWW in PATH

Machine Learning Techniques	Features	Hyperparameters
Logistic Regression	Manual Lexical, CountVectorizer, TFIDFVectorizer	Regularization Strength: {3, 2, 1, 0.3, 0.1, 0.03, 0.01, 0.003, 0.001}
K-Nearest Neighbour	Manual Lexical, CountVectorizer, TFIDFVectorizer	N_Neighbours: {3, 5, 7}
Decision Tree	Manual Lexical, CountVectorizer, TFIDFVectorizer	Via GridSearch to search for best score and parameters. Criterion: {gini, entropy} Max_depth: {2 to 10} Min_samples_split: {2 to 100}
Random Forest	Manual Lexical, CountVectorizer, TFIDFVectorizer	N_trees: {3, 5, 10, 25, 50} Tree_depth: {4, 10, 20, 50}
AdaBoost	Manual Lexical, CountVectorizer, TFIDFVectorizer	N_trees: {25, 50, 100} Tree_depth: {1,2,4,10}
Naïve Bayes	CountVectorizer TFIDFVectorizer	alpha=1.0

5.6 Model Evaluation

Based on the performance results of model training, we selected the top performance for each of the category (Model & Input Feature Type Pairs) for model evaluation. The selection of model and input pairs is based on the testing accuracy and Area Under the Curve (AUC) score for each of the category pairs. Following shows the result of model evaluation for each of the machine learning model.

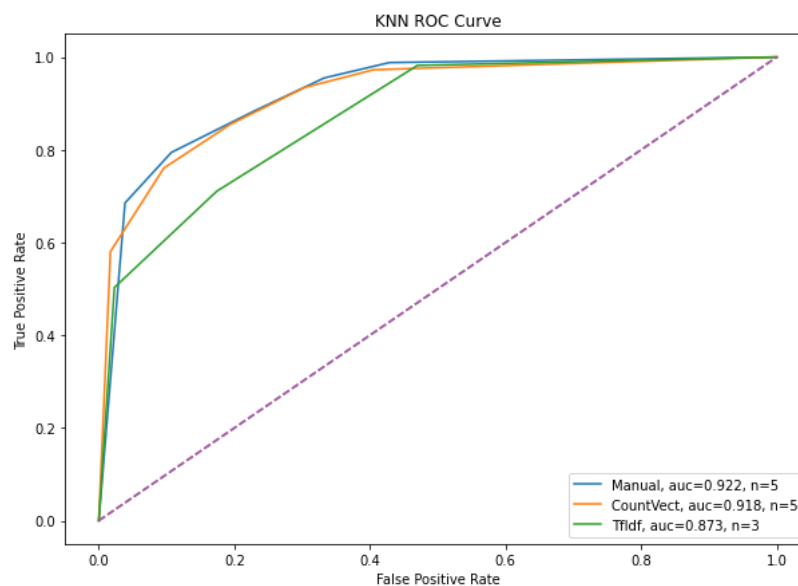
Logistic Regression



	Input Feature Types	Regularization Strength	Training Accuracy	Testing Accuracy	F1-Score	AUC Score
0	Manual	0.1	0.722	0.721	0.794	0.794
1	CountVect	0.3	0.893	0.872	0.898	0.958
2	Tfidf	1	0.906	0.873	0.899	0.936

Observation:

- Using Logistic Regression model, CountVectorizer feature extraction performs better than Manual and TFIDF feature extraction in terms of the AUC Score.
- When used Manual Lexical Features in Logistic Regression, Regularization Strength ($C = 0.1$) outperforms all others regularization strength mentioned in Hyperparameter above
- When used CountVectorizer Features in Logistic Regression, Regularization Strength ($C = 0.3$) performs better than all others regularization strength mentioned in Hyperparameter above
- When used TFIDF Features in Logistic Regression, Regularization Strength ($C = 1$) performs the best among all others regularization strength mentioned in Hyperparameter above

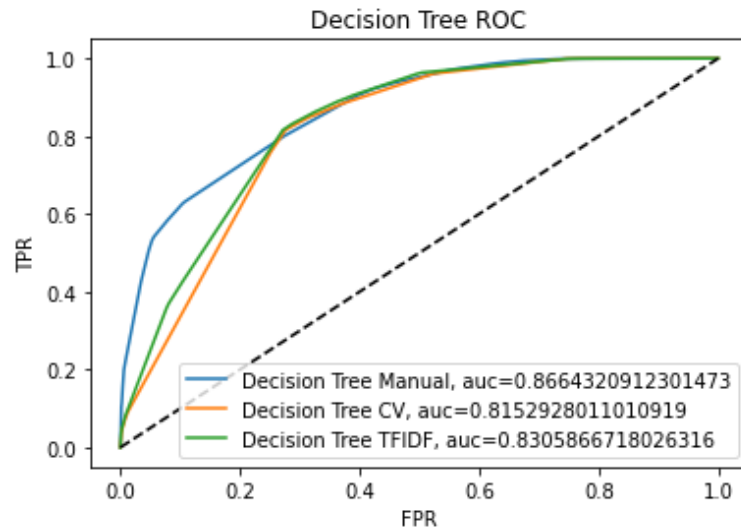
K-Nearest Neighbour (KNN)

	Input Feature Types	Nearest Neighbors	Training Accuracy	Testing Accuracy	F1-Score	AUC Score
0	Manual	5	0.881	0.839	0.870	0.923
1	CountVect	5	0.886	0.836	0.864	0.919
2	Tfidf	3	0.938	0.756	0.780	0.873

Observation:

- Using KNN model, manual feature extraction performs better than CountVectorizer and TfidfVectorizer feature extraction in terms of the AUC Score and testing accuracy score.
- When using Manual Lexical Features in KNN, N_Neighbor ($n = 5$) outperforms the other $n: \{3, 5, 7\}$
- When using CountVectorizer Features in KNN, N_Neighbor ($n = 5$) outperforms the other $n: \{3, 5, 7\}$
- When using TFIDF Features in KNN, N_Neighbor ($n = 3$) outperforms the other $n: \{3, 5, 7\}$

Decision Tree

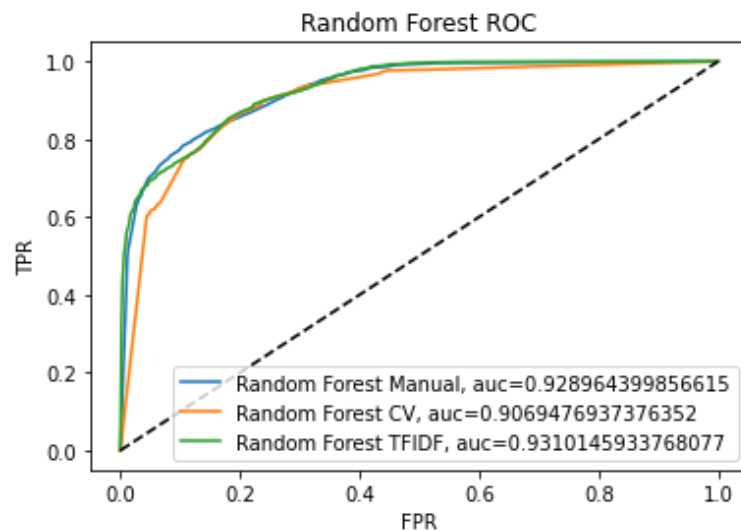


	Input Feature Types	Criterion	Max Depth	Min Samples Split	Training Accuracy	Testing Accuracy	F1-Score	AUC Score
0	Manual	Entropy	9	35	0.7888	0.787	0.84	0.866
1	CountVect	Entropy	9	35	0.7860	0.784	0.83	0.815
2	Tfidf	Entropy	10	50	0.7920	0.790	0.84	0.830

Observation:

- Using Decision Tree model, manual feature extraction performs better than CountVectorizer and TFIDF in terms of AUC score, however there is slight decrease in terms of testing accuracy when compare with TFIDF.

Random Forest

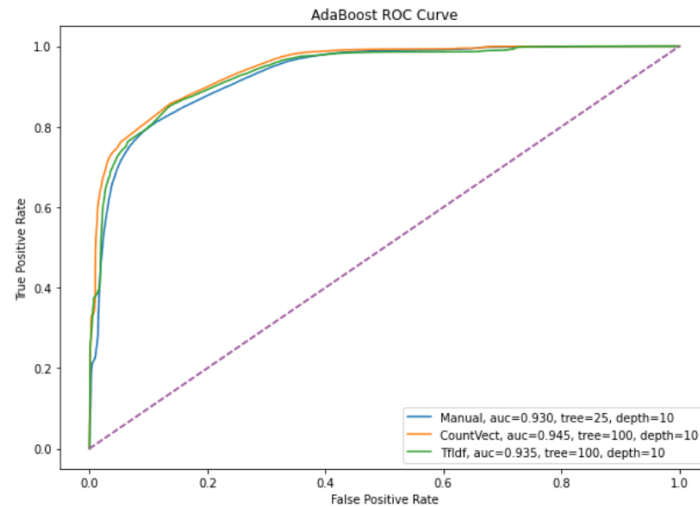


	Input Feature Types	n_tree	n_features	Training Score	Testing Accuracy	F1-Score	AUC Score
0	Manual	50	10	0.917	0.836	0.87	0.929
1	CountVect	5	10	0.941	0.840	0.87	0.907
2	Tfidf	50	4	0.945	0.846	0.88	0.931

Observation:

- Using Random Forest model, TFIDF feature extraction (with hyperparameter $n_trees=50$, $n_features=4$) outperforms manual and CountVectorizer feature representation in terms of AUC score.
- Random Forest with TFIDF has similar performance in terms of testing accuracy, F1-score and AUC score with Manual, with only minor difference.

AdaBoost

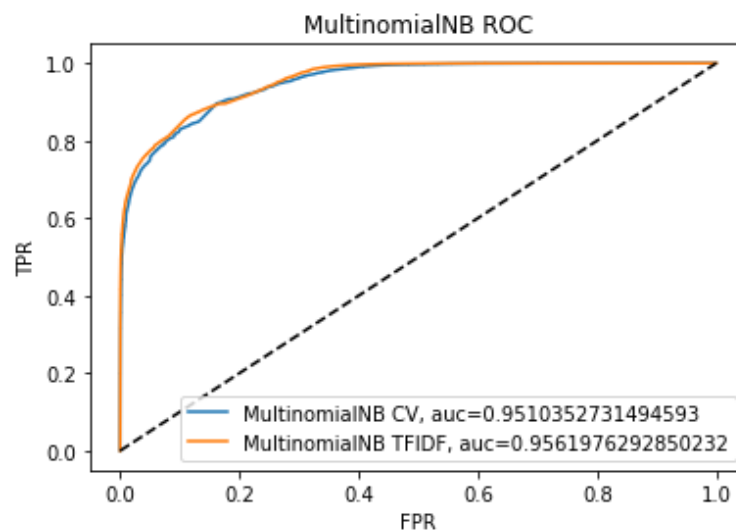


	Input Feature Types	Number of Trees	Max Features Per Tree	Training Accuracy	Testing Accuracy	F1-Score	AUC Score
0	Manual	25	10	0.875	0.848	0.878	0.930
1	CountVect	100	10	0.874	0.861	0.888	0.945
2	Tfidf	100	10	0.878	0.856	0.884	0.936

Observation:

- AdaBoost model using CountVect as the feature representation and Hyperparameter ($n_tree=100$, $tree_depth=10$) outperform the other input feature and hyperparameter setup, in terms of F1-score, AUC Score and testing accuracy.

Naïve Bayes



	Input Feature Types	Alpha	Training Acc	Testing Acc	F1-Score	AUC Score
0	CountVectorizer	1	0.911804	0.867761	0.87	0.951
1	Tfidf	1	0.919398	0.865951	0.89	0.956

Observation:

- The performance of using Multinomial Naïve Bayes model with CountVectorizer and TFIDF performs almost similar. With TFIDF performs only slightly better than CountVectorizer in terms of AUC score and F1 Score

Model Comparison and Selection for Deployment

With the performance criteria of at least 85% of testing accuracy, 85% of F1-Score, and 92% of AUC-score, the team has selected following model to be deployed. Ensembles combining approach with weighted voting based on accuracy is used to get the final prediction.

	ML-Features	Training Accuracy	Testing Accuracy	F1-Score	AUC Score
0	Logistic Regression-CountVectorizer	0.870000	0.896000	0.880	0.958
1	NB-Tfidf	0.919398	0.865951	0.890	0.956
2	NB-CountVectorizer	0.911804	0.867761	0.870	0.951
3	AdaBoost-CountVectorizer	0.874000	0.860000	0.887	0.944
4	Logistic Regression-Tfidf	0.855000	0.848000	0.878	0.936

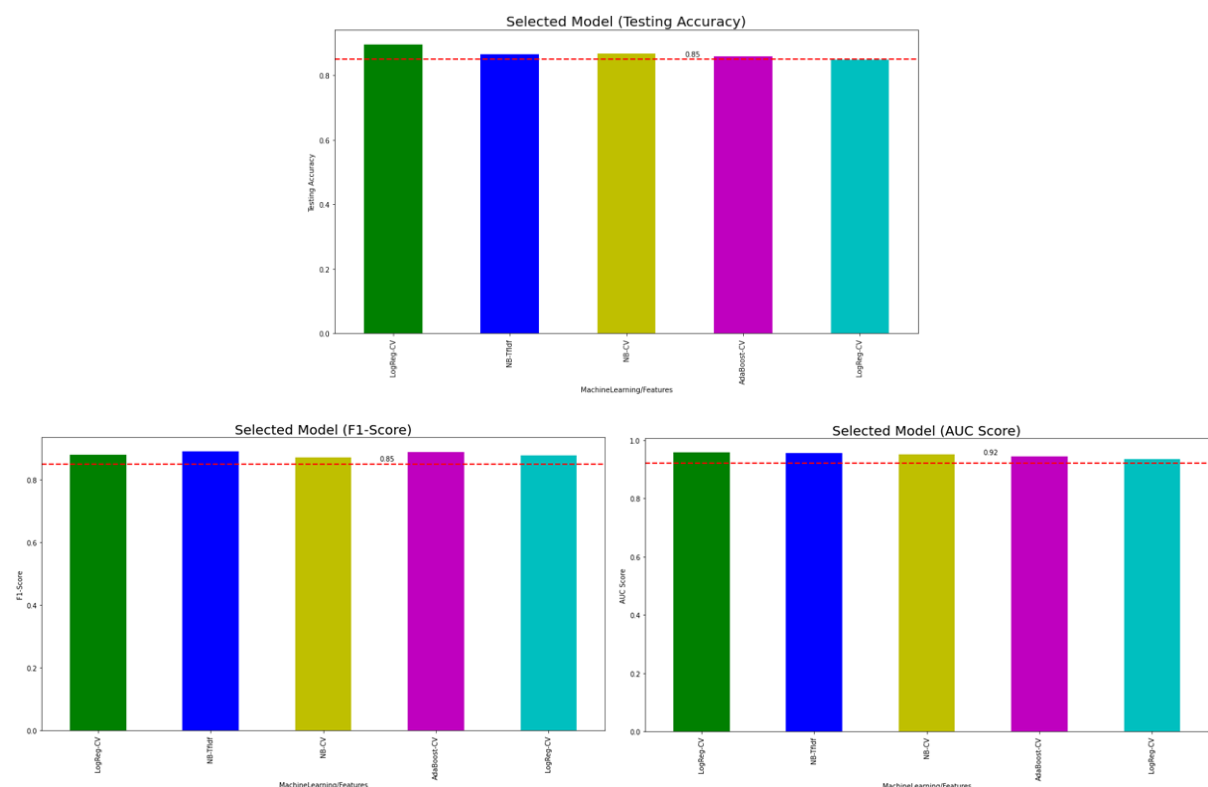


Figure 18. Testing, F-1 and AUC Score

6 Malicious URL Module

6.1 Background

With the ever-continuous growth of illicit activities on the Internet coupled with the increasing trend of elderly adopting smart devices, there is a growing need for intelligent systems to help our vulnerable elderly to identify malicious URL in their mobile devices. A lot of studies have performed URL classification using a combination of lexical features, network traffic, hosting information, and other strategies. However, such methods are time consuming which results in significant delay in real-time systems. In this project, we adopted a purely lexical approach done on the URL in order to implement a light-weight and fast classification system [16-18].

6.2 Module Workflow

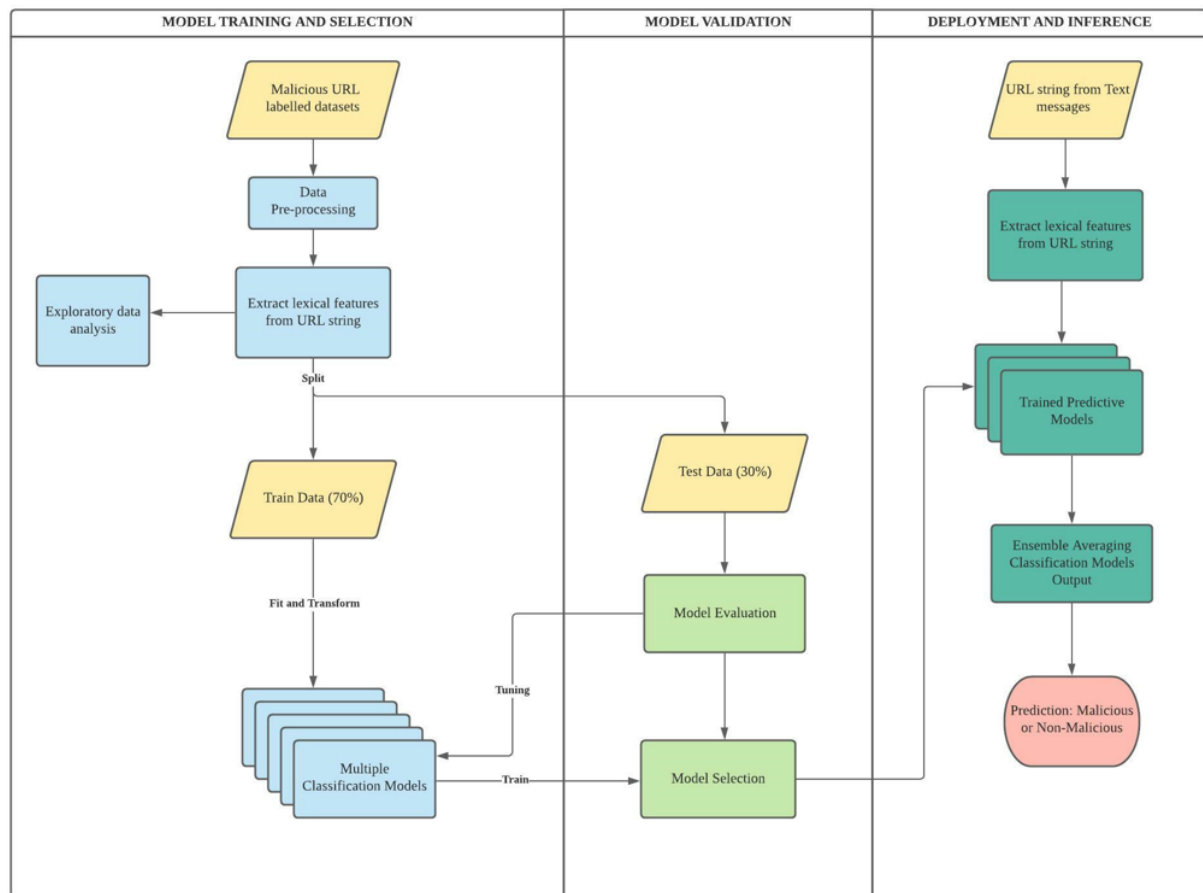


Figure 19. Technical workflow chart for Malicious URL detection

6.3 Datasets

Since there are already numerous works in this area of research, there are quite a few publicly available datasets to work with. The following are the datasets resource that was used in the model training and testing.

<https://www.kaggle.com/antonyj453/urldataset/>

<https://www.kaggle.com/sid321axn/malicious-urls-dataset/>

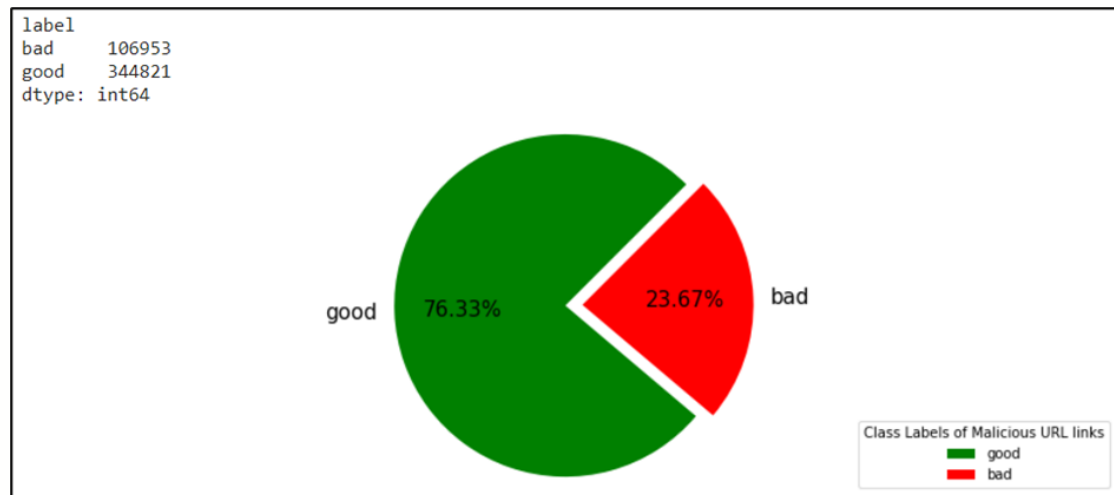


Figure 20. Distribution of Malicious URL Class labels used as training and testing datasets

6.4 Feature Extraction

As mentioned in section 5.1 in order to build a light-weight classification system that can produce fast prediction result, only lexical features from the URL string is extracted and analysed. A total of 23 different lexical features will be extracted and used as feature maps to train our malicious classification model.

During the model tuning and validation, it was discovered that the binary features extracted did not have any significant impact to any of the model performance. This is likely due to the dataset that was available to be used in this project did not actually have sufficient cases to make the binary feature useful. Hence in order to save time during the feature extraction and model training stage, it was decided that this binary feature will be discarded.



Figure 21. Example of URL components

Lexicon Length (5)	Lexicon Count (18)	Binary Features (not used)
<ul style="list-style-type: none"> Length of entire URL Length of Hostname Length of Path Length of First Directory (FD) Length of Top Level Domain (TLD) 	<ol style="list-style-type: none"> Count of subdomains Count of domain - Count of URL - Count of path - Count of URL _ Count of path _ Count of directory Count of path Count of @ Count of ? Count of % Count of . Count of = Count of 'http' Count of 'https' Count of Digits Count of Letters 	<ul style="list-style-type: none"> Use of IP address, or not Use of Shortening URL, or not

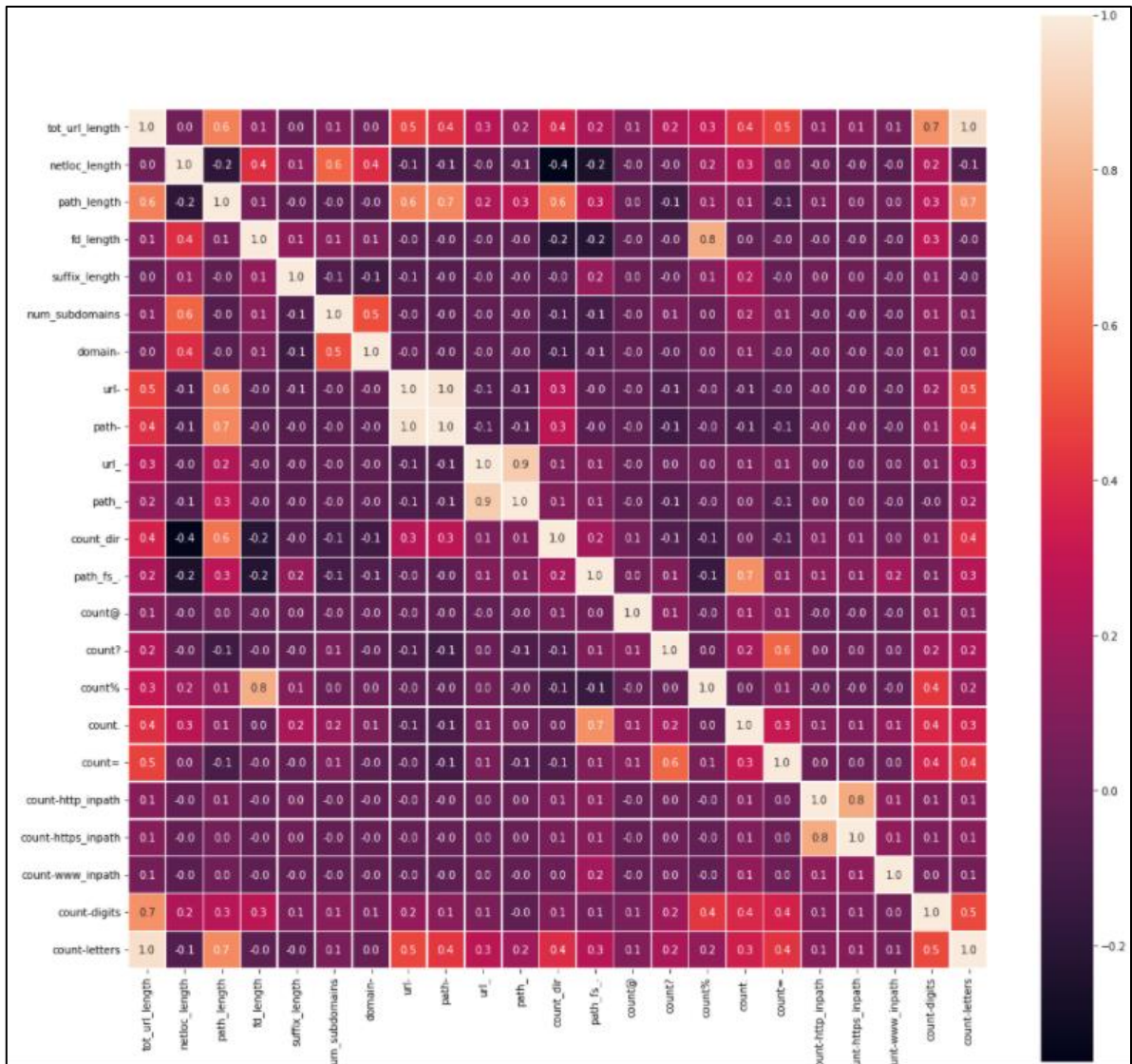
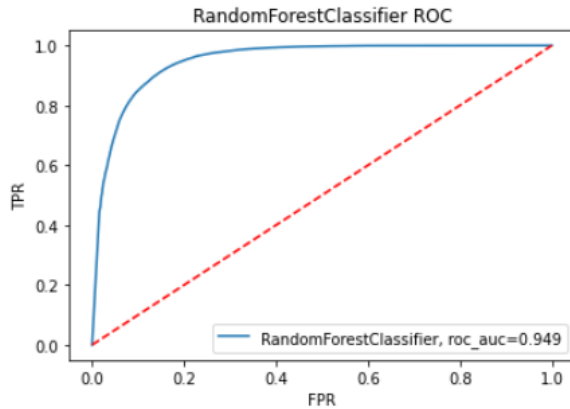


Figure 22. Heatmap data of correlation between the 23 URL lexical feature map

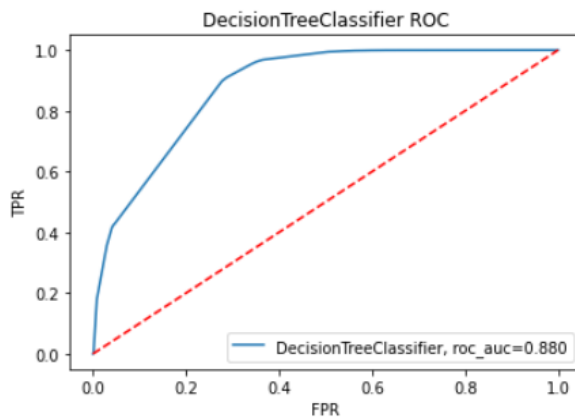
6.5 Model Training and Selection

Using the 23 lexical features as explained in the previous section, the entire dataset was split into 70% for training and remaining 30% for testing. 5 different traditional Machine Learning models were used in this training phase, in order of best train accuracy

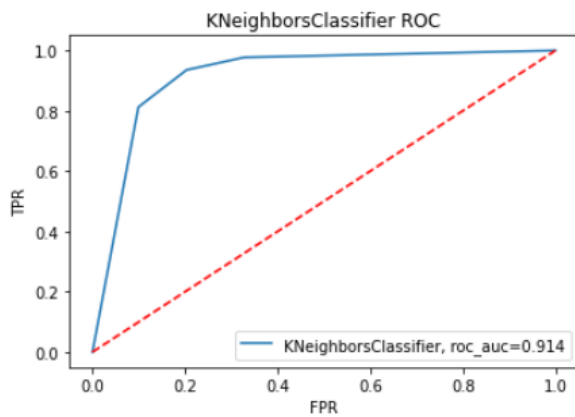
1. Random Forest



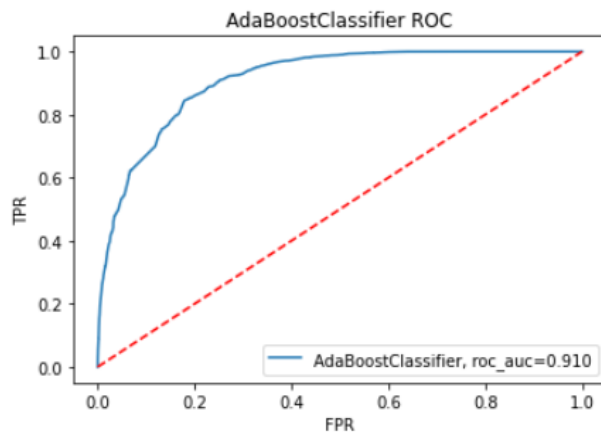
2. Decision Tree



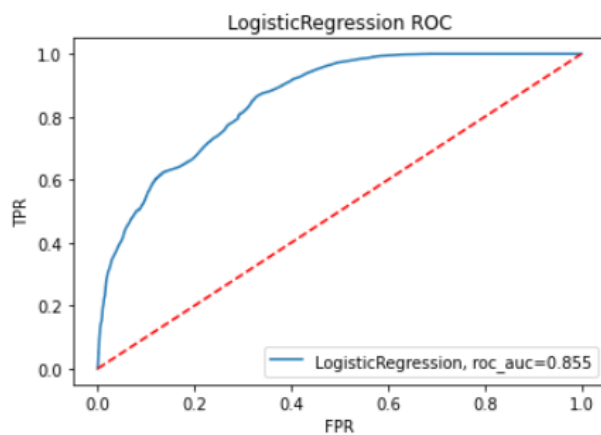
3. KNN



4. AdaBoost



5. Logistic Regression



Machine Learning Techniques	Features	Hyperparameters
Random Forest	Manual Lexical	N_trees: {10, 25, 50} Tree_depth: {5, 20, 50}
Decision Tree	Manual Lexical	GridSearch Criterion: {gini, entropy} Max_depth: {2 to 10} Min_samples_split: {2 to 100}
K-Nearest Neighbour (KNN)	Manual Lexical	N_Neighbours: {3, 5, 10}
AdaBoost	Manual Lexical	-
Logistic Regression	Manual Lexical	-

Besides the training and testing accuracy, and given the class imbalance dataset, the F1 score is also considered since the False Positives and False Negatives are also crucial since we are predicting whether a given URL is malicious (ignore and even report to authorities) or non-malicious (safe to click and enter the website). From the charts below, 3 of the models gave consistently higher performance compared to the rest. They are

1. Random Forest
2. Decision Tree
3. KNN

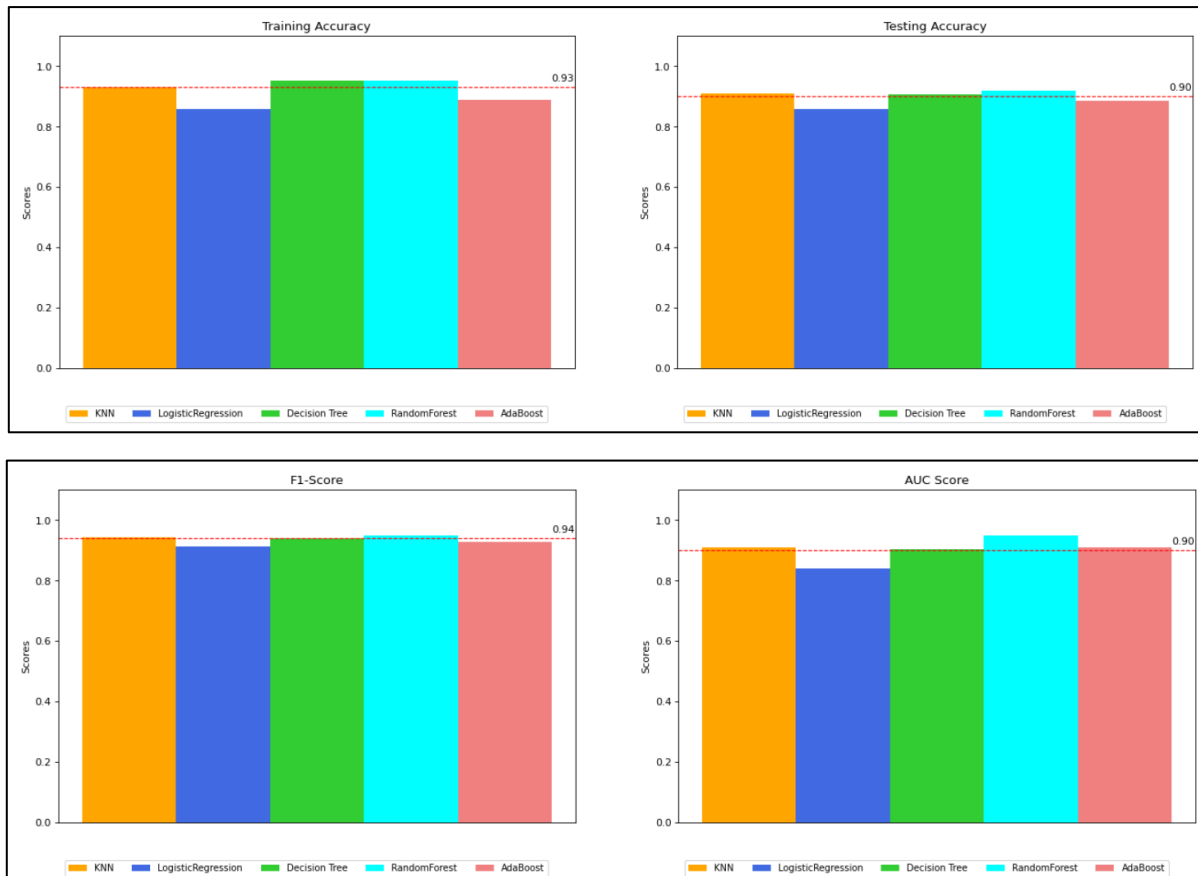


Figure 23. Charts showing the different performance metric of the models

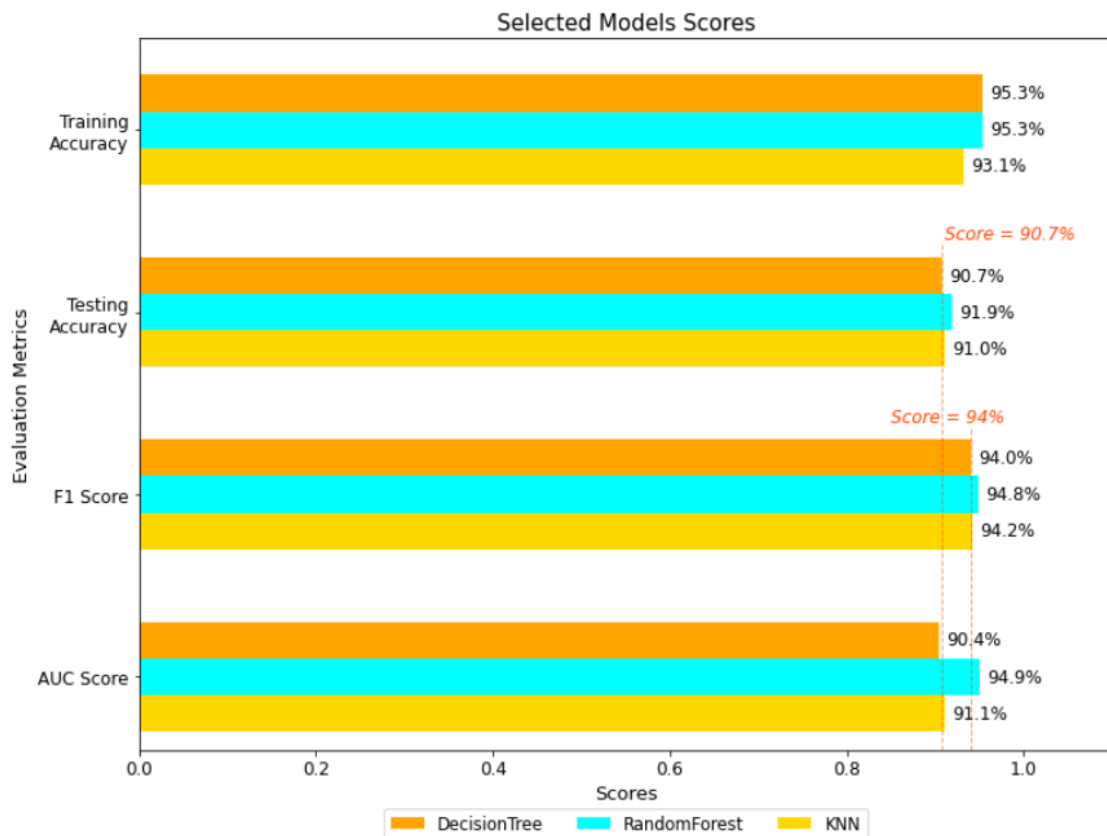


Figure 24 Performance metrics of top 3 performing models based on labelled datasets

In an attempt to further fine tune the hyper-parameters of the models, scikit-learn's GridSearchCV methodology was employed [19-20]. This a method that, instead of sampling randomly from a distribution, evaluates all combinations the user defines. The reason why GridSearchCV was chosen over RandomizedSearchCV was mainly because the intention was to exhaustively evaluate all combinations possible given the sufficient amount of time on this project.

6.6 Model evaluation

To simulate a real case, 10 known safe URL sites and another 10 known malicious URL sites are input into our Malicious URL detection system to test our system. The following are the actual observations from this simulation

- Decision Tree turned out to be the best performing classification model
- KNN model did not work well at all
- Instead AdaBoost model proved to be the next best detection model
- Random Forest did not always provide a consistent prediction score despite being the top performing model during both the training and testing phase using the datasets.

Finally, an ensemble averaging approach was adopted between 1) Decision Tree 2) AdaBoost and 3) Random Forest in order to output the prediction on whether the URL is malicious or otherwise. Below is the validation result of Malicious URL detection system using Ensemble Model Averaging.

Label	URL	Decision Tree	Random Forest	AdaBoost	Average Score (Threshold = 0.55)
Safe	www.channelnewsasia.com/	0.0%	96.1%	19.9%	38.7%
Safe	www.infineon.com/cms/singapore/en/	0.0%	95.3%	33.3%	42.9%
Safe	www.apple.com/sg/	0.0%	89.2%	19.8%	36.3%
Safe	https://www.google.com/	0.0%	88.6%	19.8%	36.1%
Safe	www.nus.edu.sg/	0.0%	91.0%	18.8%	36.6%
Safe	https://www.youtube.com/	0.0%	94.5%	19.8%	38.1%
Safe	https://www.youtube.com/watch?v=78rSqtKw3Gk	0.0%	96.0%	19.8%	38.6%
Safe	https://www.atome.sg/blog/how-to-not-burn-a-hole-in-your-	0.0%	72.0%	20.0%	30.7%
Safe	https://www.singaporepools.com.sg/en/product/Pages/toto_res	0.0%	91.0%	31.6%	40.9%
Safe	https://www.fairprice.com.sg/promotions	0.0%	97.0%	19.0%	38.7%
Malicious	//adserving.favorit-network.com/eas?camp=19320;cre=mu&grpid=1738&tag_id=618	96.5%	78.6%	52.2%	75.8%
Malicious	//officeon.ch.ma/office.js?google_ad_format=728x90_as	87.6%	66.4%	50.8%	68.3%
Malicious	www.pawthereum.com/claim-rewards-	0.0%	99.0%	19.1%	39.4%
Malicious	//diaryofagameaddict.com	0.0%	88.6%	33.6%	40.7%
Malicious	//crackspider.us/toolbar/install.php?pack=exe	37.6%	80.6%	50.1%	56.1%
Malicious	//callingcardsinstantly.com/webalizer/050709wareza/crack=17=k	73.0%	80.5%	51.7%	68.4%
Malicious	//wigglewoo.com/portfolio/contests/contest-006.png	86.8%	88.6%	50.7%	75.4%
Malicious	//kjbbs.net/bbs/data/gallery/1207290228/inbox.txt	86.8%	86.9%	51.1%	74.9%
Malicious	//webcashmaker.com/v2/members/trade_traffic.php	96.5%	77.8%	51.2%	75.1%
Malicious	//webcom-software.ws/links/?153646e8b0a88	37.6%	90.0%	50.3%	59.3%

As can be seen from the results in the Table above, although there are 2 cases of False negative from the 20 validation URLs, the Random Forest model did manage to output a high score for these 2, hence the decision was made to include this model as part of the ensemble approach. The Decision Tree model was able to consistently give a 0% probability score on URL that are safe for the 10 known safe URLs.

7 System Deployment

7.1 FrontEnd – Android

For the front-end, the team has developed an Android application for users. Figure 25 shows the ElderGuard User Interface. Users can interact with the application using following functions provided by the ElderGuard.

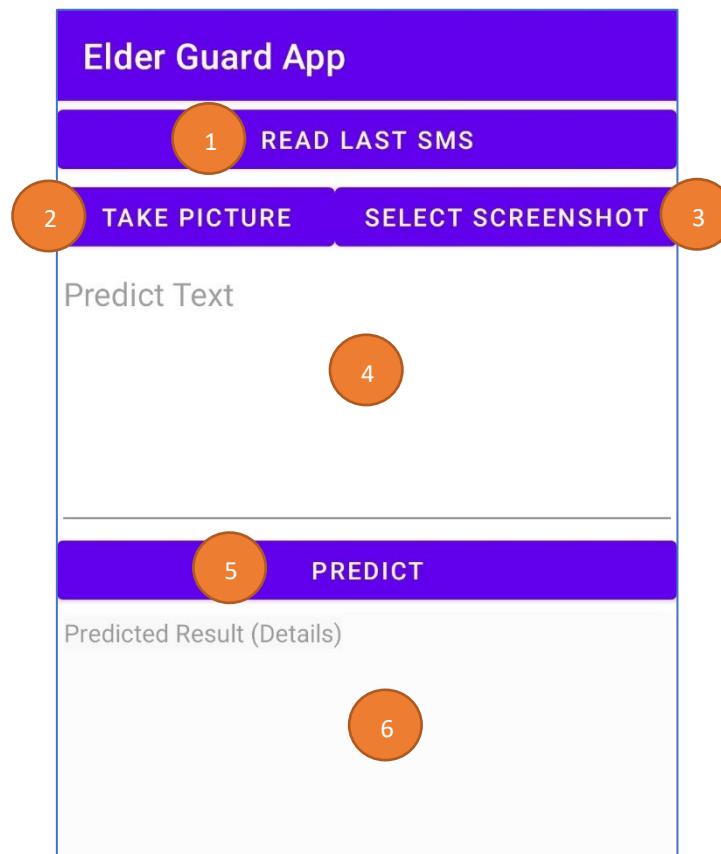


Figure 25 ElderGuard Android App User Interface

Read Last Message

When users tap on the (1) READ LAST SMS, ElderGuard App retrieves the latest Message received. It is used for users who would like to check the nature of the last SMS whether the message is safe or contains any malicious URLs, phishing URLs, or spam text.

Take Picture and Select Screenshot

Machine Learning Kit provided by Android Firebase is used in this project to recognize text in images. Users may either (2) take an image or (3) select an image from Gallery, then Android Firebase will perform Text Recognition, one of the features in the ML Kit, to convert the images into text. It can automate the tedious URL or text entry that users may experience when they observe any text or URLs that they would like to check.

Few packages that are used to accomplish the conversion of image to text includes FirebaseVisionImage, FirebaseVisionText, and FirebaseVisionTextRecognizer. These packages are used to send the image to Firebase, analyses the text, and upon successes, it processes the return text and sets it to a text view.

Predict

When users tap on (5) PREDICT, ElderGuard App will extract the text displayed in (4) PredictText EditText widget, create a post request and forward to backend server in Heroku for checking. The PredictText (4) is a EditText widget, it allows users to make modification of the text before tapping on the (5) PREDICT button to forward it to backend server.

The results obtained from the backend server regarding the text messages will be shown in (6) Predicted Result (Detail) box. (6) Predicted Result contains two widgets, TextView and ImageView. In the TextView widget, it will show the predicted results details containing the JSON format shown in table below, while in the ImageView it will show the GREEN TICK sign or RED CROSS sign to indicate whether the text message is good or bad.

Result	Details
Text	Text messages, including the text and url forwarded to backend for checking
Text_Predict	Prediction results of the text whether is spam or not spam
URLPhish_Predict	Prediction results of URL whether it is phishing sites or not. It is a dictionary list containing the label of each URLs {URL:Label}
URLMalicious_predict	Prediction results of URL whether it is malicious sites or not. It is a dictionary list containing the label of each URLs {URL:Label}
Final_Predict	Final results, when either of the Text_Predict, URLPhish,, and URLMalicious are bad, then it is considered as bad

7.2 BackEnd - FlaskApp and Heroku

For backend, a Flask application is deployed and hosted on Heroku. All of the three modules, ie. Spam Text Module, Phishing Module and Malicious URL Module are contained in this Flask application. The extracted URL and text obtained from the ElderGuard Android Application will be wrapped as a query and forwarded to Flask Server hosted on Heroku.

8 Conclusion and Future Works

Nowadays, with the Smart Nation Digitalization initiative across the country, more elderlies are starting to embrace the use of smart mobile device in their daily lives. In addition, the Covid-19 global pandemic has accelerated that even more. This has inevitably expose them to malicious content through their smart phones in the form of spam text messages or messages containing phishing or malicious URLs.

We have proposed a machine learning driven application that provides the capability of predicting the safety of a text message content through an Android application with the predictive engine deployed and hosted on Heroku. We have explored various feature engineering methods for extracting or transforming useful features that represents the text messages or URLs. Based on the features extracted, we have evaluated the performance of various machine learning algorithms in detecting the malicious message contents such as Naïve Bayes, Logistic Regression, Decision Tree, Random Forest, AdaBoost, Support Vector Machine (SVM), K-Nearest Neighbours as well as some deep learning neural network architecture like Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) Neural Network and the combination CNN-LSTM. Finally, we have selected performing algorithms for deployment while taking into account of various considerations such as the feature representations, size, inference speed and complexity of the models. The final inference from the 3 pattern recognition model are then output

In addition, we have identified some limitations and future improvements for our project as follows:

Limitations	Improvements
The predictions from the three pattern recognition modules are by no means perfect and is dependent on the languages and the style of URLs being used in text messages. The languages and style in text message can change overtime which may reduce the performance of our model predictions.	We may consider using feature representations that can better represents more generic but meaningful features for the predictions. We can also explore more advanced algorithms such as transformer-based NLP deep learning models to perform both feature extractions and classification tasks.
The imbalance dataset remains an issue across the three pattern recognition modules even though we utilized various ways to address and reduce its impact in the project. This is more apparent in the spam text classification module where the level of imbalance is higher.	We would need to collect more data especially for spam text class or to simulate more data for the minority class. These methods may be costly in terms of time and resources.
Our MVP ElderGuard application currently only able to make predictions for English language text messages. We know that some elderlies in Singapore reads and writes in Mandarin in their daily life. Hence, the ElderGuard may not useful for these group of people as our application doesn't work with Chinese characters.	To consider separate spam text modules that handles other languages such as Chinese and Malay. This is can be a different algorithms and methods alone by itself under the scope of NLP.
Currently, our app requires user to manually put in the text they would like to check for spam, phishing or malicious URLs in text messages. There is no functionality that automatically detects and perform predictions on all incoming text messages.	To build in a functionality within the Android app that automatically retrieve incoming text messages for predictions and to warn users if spam, phishing or malicious URLs are detected. This will require deeper integration and interaction between the ElderGuard App and Android OS Backend system.

9 References

- [1] Beware of SMS scams as Singapore begins vaccination drive: MOH
<https://www.businesstimes.com.sg/government-economy/beware-of-sms-scams-as-singapore-begins-vaccination-drive-moh>
- [2] Marketing Spam on WhatsApp Is Not Very 2016 <https://medium.com/@liva/spamming-on-whatsapp-to-market-stuff-is-not-very-2016-72c32783a55>
- [3] Spam Message on Telegram <https://github.com/OriginProtocol/telegram-moderator/issues/17>
- [4] Pavas Navaney, Gaurav Dubey, Ajay Rana (January 2018) Spam Filtering using Supervised *Machine Learning Algorithms* <https://ieeexplore.ieee.org/document/8442564>
- [5] Houshmand Shirani-Mehr *SMS Spam Detection using Machine Learning Approach*
<http://cs229.stanford.edu/proj2013/ShiraniMehr-SMSSpamDetectionUsingMachineLearningApproach.pdf>
- [6] Sandhya Mishra, Devpriya Soni (July 2020) *Smishing Detector: A security model to detect smishing through SMS content analysis and URL behavior analysis*
<https://www.sciencedirect.com/science/article/pii/S0167739X19318758>
- [7] Pradeep Kumar Roy, Jyoti Prakash Singh, Snehasish Banerjee (January 2020) *Deep learning to filter SMS Spam* <https://www.sciencedirect.com/science/article/pii/S0167739X19306879>
- [8] Wael Hassan Gomaa (January 2020) *The Impact of Deep Learning Techniques on SMS Spam Filtering*
https://www.researchgate.net/publication/339025894_The_Impact_of_Deep_Learning_Techniques_on_SMS_Spam_Filtering
- [9] University of California Irvine (UCI) Machine Learning Repository *SMS Spam Collection Data Set*
<https://archive.ics.uci.edu/ml/datasets/sms+spam+collection>
- [10] Uddeshya Singh *Kaggle SMS Dataset* <https://www.kaggle.com/uds5501/sms-dataset>
- [11] Jeffrey Pennington, Richard Socher, Christopher D. Manning *Global Vectors for Word Representation (GloVe)* <https://nlp.stanford.edu/projects/glove/>
- [12] 2019 Phishing Statistics and Email Fraud Statistics <https://retruster.com/blog/2019-phishing-and-email-fraud-statistics.html>
- [13] Phishing Site URLs – Malicious and Phishing attacks urls <https://www.kaggle.com/taruntiwarihp/phishing-site-urls>
- [14] Malicious URLs dataset - Huge dataset of 6,51,191 Malicious URLs
<https://www.kaggle.com/sid321axn/malicious-urls-dataset>
- [15] OpenPhish Dataset <https://openphish.com/feed.txt>
- [16] Michael Darling (September 2015) *A Lexical Approach for Classifying Malicious URLs*.
<https://ieeexplore.ieee.org/abstract/document/7237040>
- [17] Clayton Johnson, Bishal Khadka, Ram B. Basnet¹, Tenzin Doleck (December 2020) *Towards Detecting and Classifying Malicious URLs Using Deep Learning. (PDF) Towards Detecting and Classifying Network Intrusion Traffic Using Deep Learning Frameworks (researchgate.net)*
- [18] Apoorva Joshi, Levi Lloyd, Paul Westin, Srini Cenesthopathy (October 2021) *Using Lexical Features for Malicious URL Detection - A Machine Learning Approach*. <https://arxiv.org/abs/1910.06277>
- [19] Python Scikit-Learn Grid Search https://scikit-learn.org/stable/modules/grid_search.html
- [20] Random Forest Hyperparameter Tuning <https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74>

10 Appendix A – Model Hyperparameter Tuning Evaluation

	Input Feature Types	Regularization Strength	Training Accuracy	Validation Accuracy	F1-Score	AUC Score
0	Manual	3.000	0.723	0.719	0.793	0.792
1	Manual	2.000	0.723	0.719	0.793	0.792
2	Manual	1.000	0.723	0.719	0.793	0.791
3	Manual	0.300	0.723	0.719	0.793	0.791
4	Manual	0.100	0.722	0.719	0.793	0.791
5	Manual	0.030	0.720	0.717	0.791	0.789
6	Manual	0.010	0.717	0.715	0.790	0.788
7	Manual	0.003	0.713	0.710	0.787	0.785
8	Manual	0.001	0.710	0.708	0.786	0.782
9	CountVect	3.000	0.910	0.868	0.895	0.956
10	CountVect	2.000	0.907	0.868	0.895	0.957
11	CountVect	1.000	0.905	0.870	0.896	0.958
12	CountVect	0.300	0.893	0.871	0.897	0.957
13	CountVect	0.100	0.880	0.869	0.896	0.954
14	CountVect	0.030	0.866	0.862	0.891	0.948
15	CountVect	0.010	0.851	0.848	0.882	0.940
16	CountVect	0.003	0.835	0.833	0.872	0.928
17	CountVect	0.001	0.816	0.814	0.859	0.911
18	Tfidf	3.000	0.929	0.867	0.894	0.937
19	Tfidf	2.000	0.924	0.869	0.896	0.937
20	Tfidf	1.000	0.906	0.872	0.898	0.936
21	Tfidf	0.300	0.883	0.870	0.897	0.932
22	Tfidf	0.100	0.868	0.862	0.892	0.926
23	Tfidf	0.030	0.843	0.840	0.877	0.915
24	Tfidf	0.010	0.814	0.814	0.860	0.902
25	Tfidf	0.003	0.756	0.761	0.831	0.885
26	Tfidf	0.001	0.688	0.696	0.799	0.873

Figure 26. Logistic Regression Training (Phishing Module)

	Input Feature Types	n_tree	n_features	Validation Acc	AUC Score
0	CountVect	5	10	0.844744	0.909507
1	CountVect	5	20	0.844175	0.908343
2	CountVect	5	50	0.840575	0.907260
3	CountVect	5	4	0.841195	0.907189
4	CountVect	3	20	0.838920	0.892374
5	CountVect	3	50	0.841175	0.892244
6	CountVect	3	4	0.834668	0.891691
7	CountVect	3	10	0.835682	0.891585
8	Manual	50	10	0.837895	0.932350
9	Manual	50	20	0.836954	0.932134
10	Manual	50	4	0.838402	0.931900
11	Manual	25	10	0.837730	0.930399
12	Manual	25	4	0.837554	0.929979
13	Manual	25	20	0.836654	0.929886
14	Manual	10	10	0.835640	0.923913
15	Manual	10	20	0.834233	0.923837
16	Manual	10	4	0.835547	0.923834
17	Manual	5	10	0.834202	0.914288
18	Manual	5	4	0.835423	0.914161
19	Manual	5	20	0.834182	0.913462
20	Manual	3	10	0.832247	0.901872
21	Manual	3	20	0.832402	0.901590
22	Manual	3	4	0.830520	0.900348
23	Tfidf	25	20	0.846709	0.929848
24	Tfidf	25	4	0.844464	0.929208
25	Tfidf	25	10	0.844620	0.928823
26	Tfidf	25	50	0.841899	0.928358
27	Tfidf	10	20	0.842623	0.919444
28	Tfidf	10	4	0.840989	0.919026
29	Tfidf	10	10	0.844620	0.918485
30	Tfidf	10	50	0.840937	0.918193
31	Tfidf	5	10	0.835723	0.904676
32	Tfidf	5	50	0.836602	0.904648
33	Tfidf	5	20	0.835330	0.904063
34	Tfidf	5	4	0.840958	0.903906
35	Tfidf	3	50	0.836571	0.891052
36	Tfidf	3	10	0.834668	0.890138
37	Tfidf	3	4	0.835578	0.888209
38	Tfidf	3	20	0.831182	0.887426
39	Tfidf	50	4	0.849000	0.932000
40	Tfidf	50	20	0.848000	0.932000
41	Tfidf	50	50	0.843000	0.931000

Figure 27. Random Forest Training (Phishing Module)

	Input Feature Types	Nearest Neighbors	Validation Acc	AUC Score
0	Manual	3	0.835433	0.903900
1	Manual	5	0.838951	0.922829
2	Manual	7	0.842209	0.929550
3	CountVect	3	0.807000	0.900000
4	CountVect	5	0.836000	0.920000
5	CountVect	7	0.841000	0.927000
6	Tfidf	3	0.757000	0.874000
7	Tfidf	5	0.713000	0.833000
8	Tfidf	7	0.688000	0.834000

Figure 28. KNN Training (Phishing Module)

	Input Feature Types	Number of Trees	Max Features Per Tree	Training Accuracy	Validation Accuracy	F1-Score	AUC Score
0	Manual	25	1	0.783	0.780	0.829	0.864
1	Manual	25	2	0.804	0.803	0.847	0.887
2	Manual	25	4	0.824	0.823	0.860	0.911
3	Manual	25	10	0.874	0.850	0.880	0.932
4	Manual	50	1	0.790	0.789	0.838	0.875
5	Manual	50	2	0.814	0.814	0.854	0.901
6	Manual	50	4	0.835	0.833	0.867	0.921
7	Manual	50	10	0.891	0.847	0.876	0.925
8	Manual	100	1	0.798	0.797	0.843	0.883
9	Manual	100	2	0.822	0.821	0.859	0.909
10	Manual	100	4	0.844	0.841	0.873	0.928
11	Manual	100	10	0.905	0.840	0.870	0.917
12	CountVect	25	1	0.756	0.755	0.823	0.832
13	CountVect	25	2	0.806	0.804	0.847	0.875
14	CountVect	25	4	0.827	0.824	0.862	0.906
15	CountVect	25	10	0.851	0.844	0.874	0.932
16	CountVect	50	1	0.804	0.802	0.846	0.871
17	CountVect	50	2	0.825	0.824	0.859	0.903
18	CountVect	50	4	0.840	0.837	0.869	0.924
19	CountVect	50	10	0.863	0.853	0.882	0.940
20	CountVect	100	1	0.823	0.822	0.858	0.900
21	CountVect	100	2	0.837	0.836	0.869	0.922
22	CountVect	100	4	0.853	0.849	0.880	0.937
23	CountVect	100	10	0.874	0.860	0.887	0.944
24	Tfidf	25	1	0.779	0.778	0.827	0.832
25	Tfidf	25	2	0.805	0.780	0.821	0.859
26	Tfidf	25	4	0.828	0.825	0.862	0.907
27	Tfidf	25	10	0.852	0.840	0.873	0.925
28	Tfidf	50	1	0.807	0.782	0.825	0.857
29	Tfidf	50	2	0.826	0.802	0.837	0.886
30	Tfidf	50	4	0.840	0.836	0.869	0.923
31	Tfidf	50	10	0.866	0.848	0.878	0.932
32	Tfidf	100	1	0.826	0.800	0.838	0.885
33	Tfidf	100	2	0.839	0.835	0.868	0.921
34	Tfidf	100	4	0.855	0.848	0.878	0.936
35	Tfidf	100	10	0.880	0.856	0.884	0.936



Figure 29. AdaBoost

11 Appendix B – Proposal

Project Proposal

Date of proposal: 22 August 2021
Project Title: “ElderGuard”: Suspicious Text Messages Detector for the Elderly
Group ID (As Enrolled in LumiNUS Class Groups): 18 Group Members (Name, Student ID): Lim Jun Ming, A0231523U Mediana, A0231458E Yeong Wee Ping, A0231533R
Sponsor/Client: <i>(Company Name, Address and Contact Name, Email, if any)</i> N.A
Background/Aims/Objectives: <p>With Singapore moving towards Smart Nation digitalization, more elderlies are starting to embrace the use of smart mobile phones in their daily lives. However, this inadvertently exposes them to malicious content through their smart phones. Adversaries are targeting elderly who tend to be less tech-savvy by sending them spam text messages or messages with malicious URL that they might unknowingly click. This not only results in financial loss or identity theft but may also cause them to lose confidence and abandon the use of smart phones.</p> <p>Our main aim is to build an app that helps them to detect text messages that may be fraudulent in nature. Examples are spam messages, fraudulent messages or messages that contain malicious or phishing URLs. This app shall protect elderly from being scammed or cheated by alerting them whenever they receive a suspicious text message.</p>
Project Descriptions: <p>An app developed in the Android OS environment to identify any incoming text messages which are suspicious and to alert the phone user. Whenever a text message is received on the phone, the data will be fed into the app backend to check whether if it is a spam or fraudulent message or if it contains any malicious or phishing URL.</p> <p>The classification or identification task is done by running the data through a pattern recognition system. The classification result will then be sent back to the phone. The app will then show and tag the text message as safe or unsafe in the form of trivial symbols for representation.</p> <p>Various machine learning techniques are used in this project to determine whether the text messages are spam or non-spam and URL whether is malicious or benign:</p> <ol style="list-style-type: none"> 1. Supervised Learning, using labelled data to classify if an URL is benign or malicious 2. Deep Learning, to learn the text messages and classify if spam or non-spam 3. Ensemble techniques to integrate the machine learning model results

Scope:

- Target: Senior Citizens
- Target Inputs: SMS text messages
- Target Platform: Android Phones
- Language: English
- Classification Output: Spam/Not Spam, Malicious URL / Not Malicious URL, Phishing URL / Not Phishing URL.
- App Output to user:  /  (symbols for ease of understanding)

Requirements Overview:

- Research ability: Machine Learning Methods, Approaches for Suspicious Text Messages detection
- Programming ability: Python for ML model, Java/C++ for Android App development
- System integration ability: Integrate our Machine Learning model and deploy through Android App.

Resource Identified:

- Dataset for model training: Benign URL, Phishing URL, Malicious URL, Normal Text Messages and Spam Text
- Hardware proposed: Android mobile phone, server to host backend program and web application
- Software/Libraries proposed: Python Flask for web application, SpaCy for NLP, Scikit-Learn for Machine Learning in Python