Nanyang Technological University

Wee Kim Wee School of Communication and Information

SMS Spam Detection

IS6751 2023 Group Project

Group members:

Tang Kaiyuan	G2304420G
Tang Dengcheng	G2304480F
Chen Yunyi	G2203055A
Li Jiachen	G2304541B

Table of Contents

1. Introduction

In the digital age, the proliferation of mobile devices has led to an unprecedented increase in the volume of Short Message Service (SMS) communications. While SMS offers a convenient medium for personal and professional communication, it has also become a channel exploited by malicious actors for spamming purposes. SMS spam, much like email spam, encompasses unsolicited messages sent in bulk, often with deceptive content aimed at promoting products, services, or even scams. Such messages not only inundate users' inboxes but can also pose security threats and financial risks.

Detecting and filtering SMS spam is crucial for enhancing the overall user experience and ensuring the safety and privacy of mobile device users. Traditional filtering methods, which rely predominantly on blacklists or simple keyword-based detection, have shown limitations in their effectiveness. With the advancements in machine learning and natural language processing, there is an opportunity to create more sophisticated models capable of detecting spam messages with higher accuracy.

For our project, we have selected the "SMS Spam Collection Dataset" from Kaggle, which comprises a diverse set of Singapore related SMS messages labeled as 'spam' or 'ham' (legitimate). Using this dataset, our objective is to implement several predictive models that can efficiently differentiate between spam and legitimate messages. This project aims to explore various machine learning and deep learning algorithms, evaluate their performance, and build a best model that can be integrated into mobile devices or messaging platforms to aid in real-time spam detection.

2. Background and Related Work

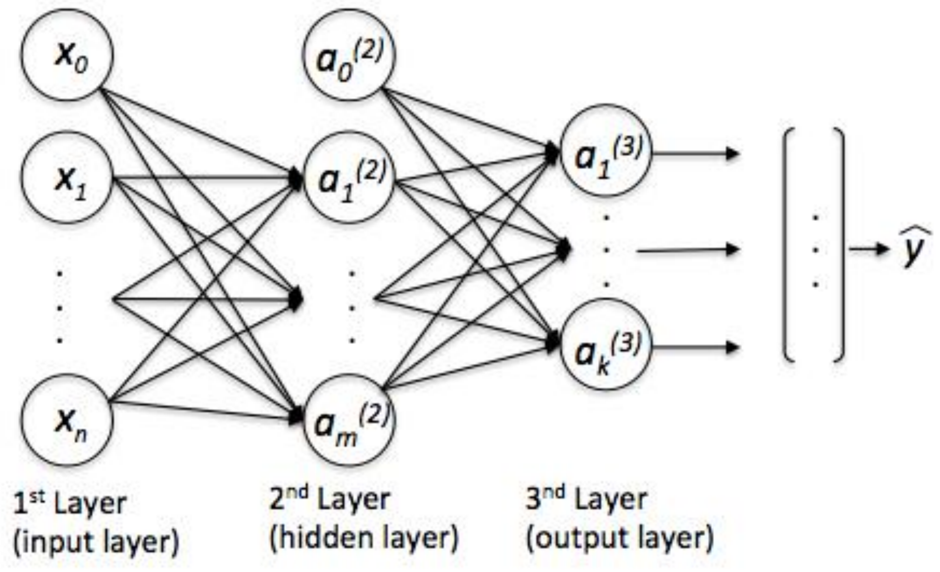
2.1. Literature Review

In the age of mobile communication, Short Message Service (SMS) has seamlessly integrated into our daily routines [1]. However, amid the legitimate exchanges, the uncontrolled proliferation of unsolicited and potentially harmful SMS spam has raised considerable apprehension [2]. Effective SMS spam detection is paramount in safeguarding users against unwanted messages, fraudulent activities, and potential security threats [3]. This section provides a comprehensive overview of the various methods and techniques employed in SMS spam detection, as well as the practical experimental processes and results. It compares the performance strengths and weaknesses of different models in handling this task [4].

2.2. Traditional Rule-Based Approaches

One of the earliest strategies for SMS spam detection lies in the rule-based methods, a foundation that hinges on predetermined criteria, keywords, and patterns to categorize messages [5]. Despite their ease of implementation, these rule-based systems often grapple with the dynamic and evolving nature of spam messages, rendering them less effective in identifying novel forms of SMS spam. Consequently, they necessitate frequent updates to maintain their relevance [6].

2.3. Machine Learning-Based Approaches

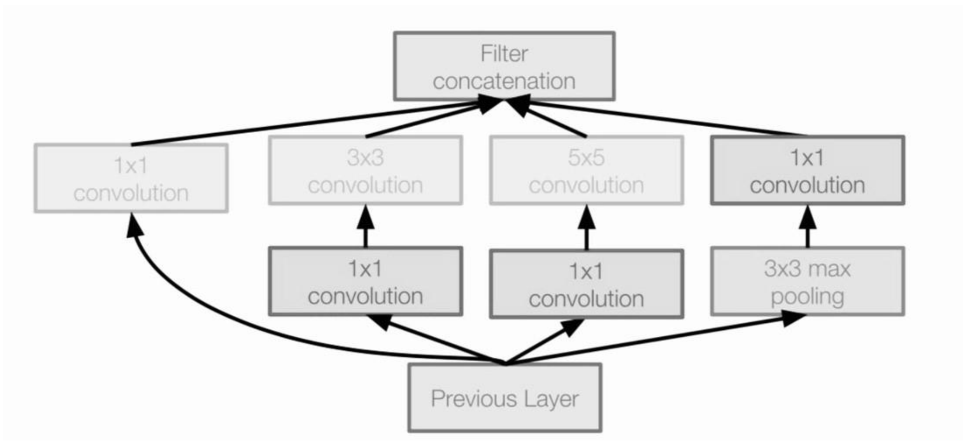


Machine learning techniques play a pivotal role in SMS spam detection, harnessing algorithms such as Naive Bayes, Support Vector Machines (SVM), and Multi-layers Neural Network to categorize SMS messages [7]. These models exploit diverse features encompassing text content, sender attributes, and timing to distinguish spam messages. Their adaptability and ability to accommodate shifting spam patterns render them indispensable tools in the fight against SMS spam [8].

2.4. Deep Learning Approaches

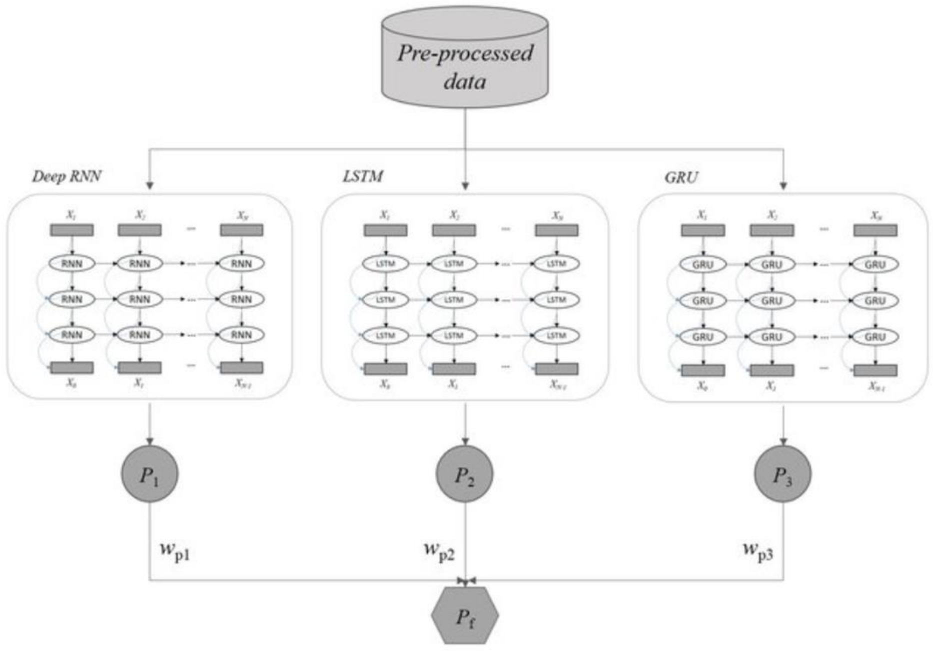
The advent of deep learning has ushered in the application of recurrent neural networks (RNNs) and convolutional neural networks (CNNs) for SMS spam detection. RNNs are tailor-made for the temporal nuances of message sequences, adept at capturing the contextual evolution over time [9]. On the other hand, CNNs, originally designed for image analysis, have found their footing in text feature extraction, rendering them highly efficient for spam classification [10]. These deep learning models enjoy automated feature extraction and have set the benchmark for performance in SMS spam detection tasks.

2.4.1. Convolutional Neural Networks (CNN)



Originally developed for image processing, Convolutional Neural Networks (CNNs) have seamlessly transitioned into the domain of natural language processing [11]. When applied in the context of spam SMS detection, CNNs have firmly demonstrated their efficacy. These networks leverage convolutional layers to analyze text sequences, meticulously capturing localized patterns and features inherent in the data. This feature extraction proficiency proves particularly valuable in identifying recurring keywords or phrases commonly found in spam messages. CNNs have consistently exhibited exceptional performance, especially when handling concise text inputs, and have emerged as a reliable tool in spam SMS filtering [12].

2.4.2. Recurrent Neural Networks (RNNs)

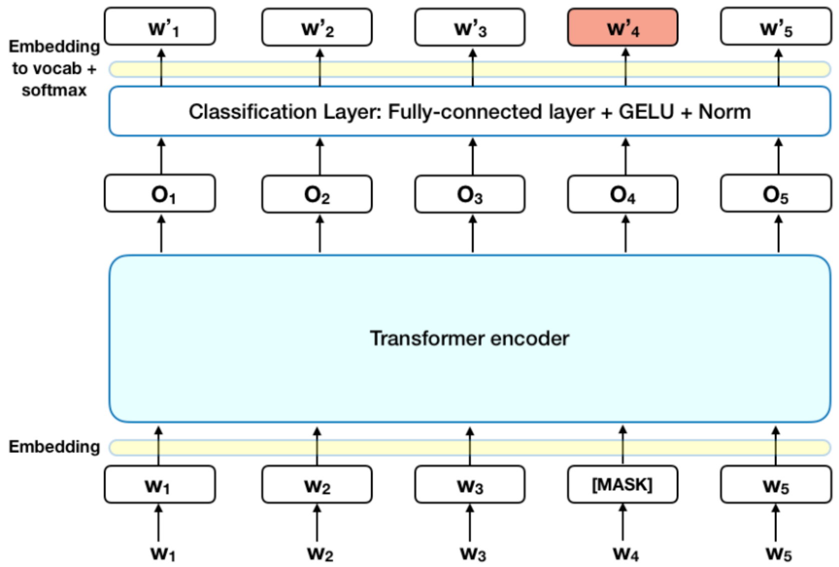


When it comes to recurrent neural networks (RNNs), one prominent member of the family is the Long Short-Term Memory network (LSTM). LSTM, as one of the distinguished representatives of the RNN family, demonstrates significant potential in handling sequential data. These networks are intricately designed to capture long-range contextual information while overcoming the vanishing gradient problem, enhancing their capability to analyze sequential data. Researchers have achieved remarkable results, particularly in tasks involving complex content, utilizing LSTM models.

On the other hand, recurrent neural networks (RNNs) represent a class of neural network architectures commonly employed in the processing of sequential data. RNNs serve as versatile tools for various applications, such as natural language processing and time series analysis, allowing them to capture temporal dependencies in data. However, they are susceptible to the vanishing gradient problem. Researchers have been working diligently to address these challenges and enhance RNNs' capabilities for improved sequential data processing.

Gated Recurrent Unit (GRU), a variant of recurrent neural networks (RNNs), also exhibits promise in the nuanced interpretation of sequential data, including SMS messages [13]. GRUs are ingeniously designed to fathom dependencies and grasp long-range contextual information nested within the text. Their unique gating mechanisms mitigate the vanishing gradient issue, thereby enhancing their capacity to analyze extended text sequences. Researchers have reported notable achievements with GRU models in spam SMS classification, particularly when confronted with messages containing diverse content [14].

2.4.3. Bidirectional Encoder Representations from Transformers (BERT)


w'1
w'2
w'3
w'4
w'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
w1
w2
w3
[MASK]
w5
w1
w2
w3
w4
w5

Bidirectional Encoder Representations from Transformers (BERT), a cutting-edge transformer-based model, has emerged as a focal point of attention due to its exceptional performance across various natural language understanding tasks. When enlisted for spam SMS detection, BERT leverages its contextual comprehension and attentive mechanisms to discern the intricate nuances within text messages [15]. This model excels in understanding the contextual dynamics, making it exquisitely suited for identifying subtle indicators of spam within messages. While it is computationally intensive and demands substantial computational resources, BERT has substantiated its mettle through remarkable accuracy in the classification of spam SMS [16].

2.5. Model Training and Evaluation

Having explored the various approaches and methodologies for SMS spam detection, the next crucial phase involves model constructing, training, and evaluation. The effectiveness of these models relies heavily on the quality and quantity of training data, fine-tuning, and rigorous evaluation metrics. In the following section, we delve into the specifics of model training techniques and the criteria used to evaluate the performance of SMS spam detection models.

3. Preparatory Work

3.1. Dataset

The dataset used in this project can be found in Kaggle (SMS Spam Collection Dataset (kaggle.com)). It contains 5574 SMS messages in total, tagged according to being ham or spam. Many of the messages are Singapore-related, hence models fit on the datasets could be very useful in helping Singapore SMS customers to detect spam messages.

3.2. Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is an important step in data mining task. We conducted exploratory data analysis from the following aspects.

We first reviewed the data distribution. As we can see from Figure 1, the dataset is labeled into two categories, with 4819 ham messages and 747 spam messages in total.

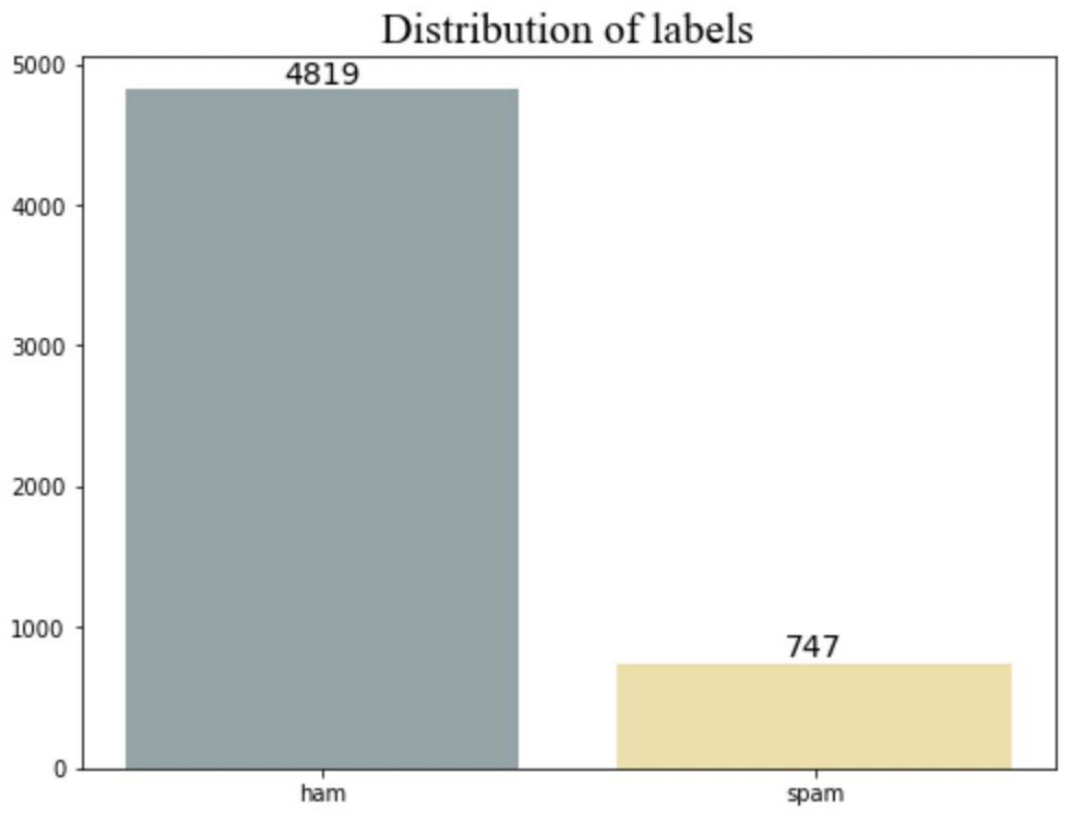


Figure 1. Distribution of Labels

Secondly, figure 2 below illustrates the distribution of message lengths in the two categories. Both ham and spam messages predominantly fall within the range of approximately 1 to 20 words. Ham messages can extend beyond 70 words, indicating longer content compared to spam messages.

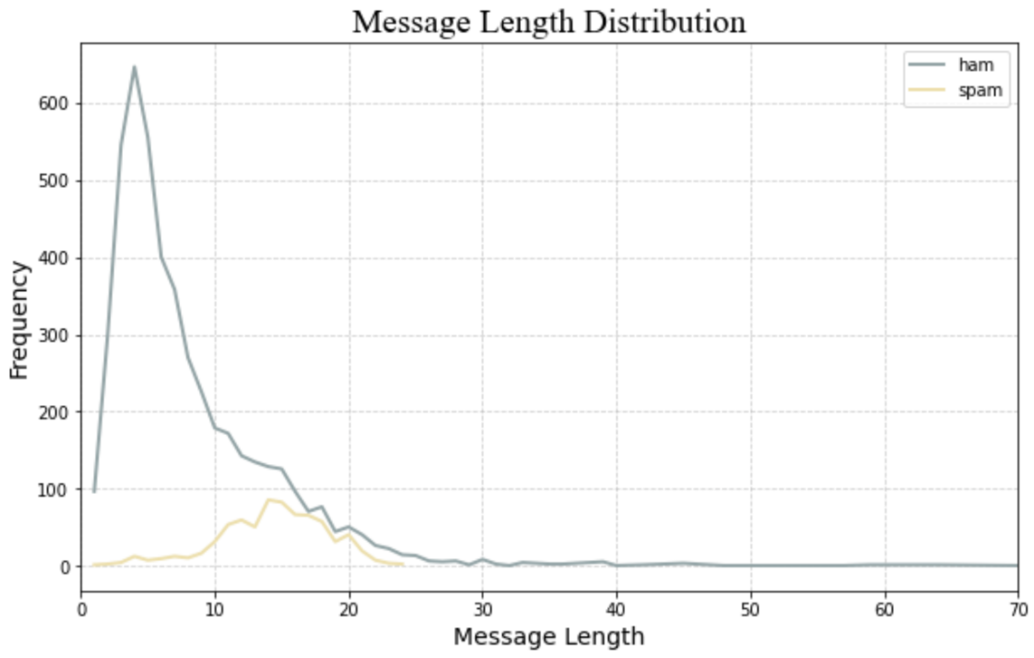
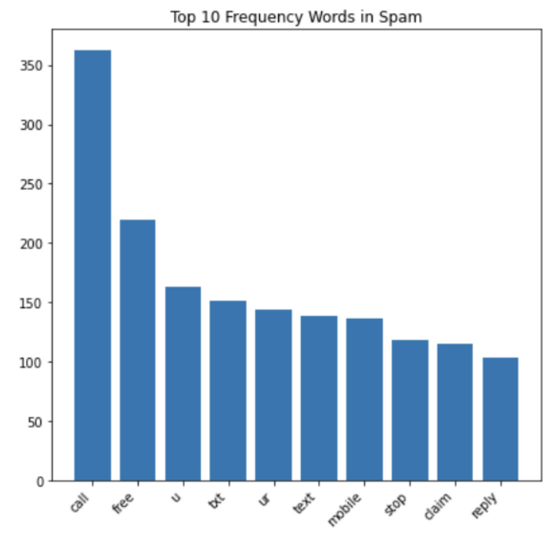


Figure 2. Message Length Distribution

Thirdly, we continue to explore the top-frequency words of different categories. We calculated the top 10 words of each label. Word frequency statistics regarding different categories of messages are also conducted and presented with a word cloud. From Figures 3 and 4, we can see "call", "free", "mobile", "reply" is predominant in spam messages. And words like "come", "call", "dont" can be commonly found in ham messages. Figure 3, 4, and 5 visualized the info.



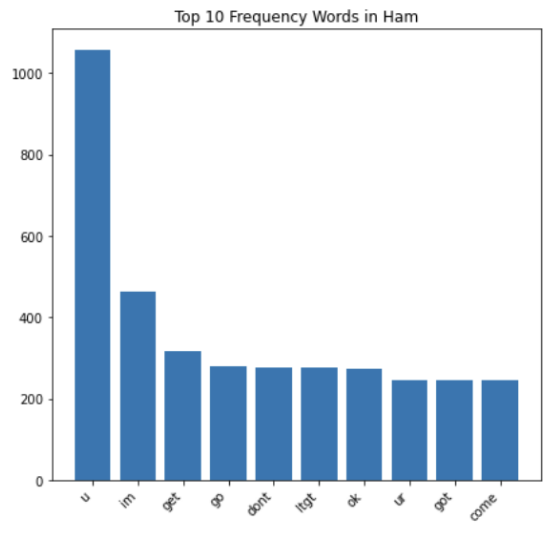


Figure 3. Top 10 Frequency Words

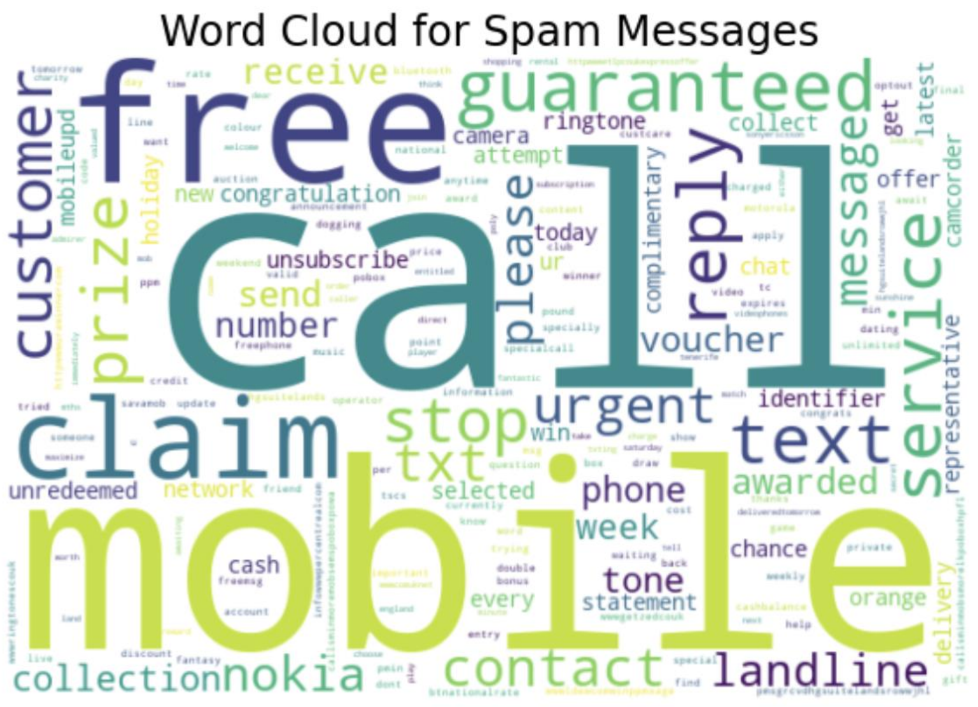


Figure 4. Word Cloud for Spam Messages

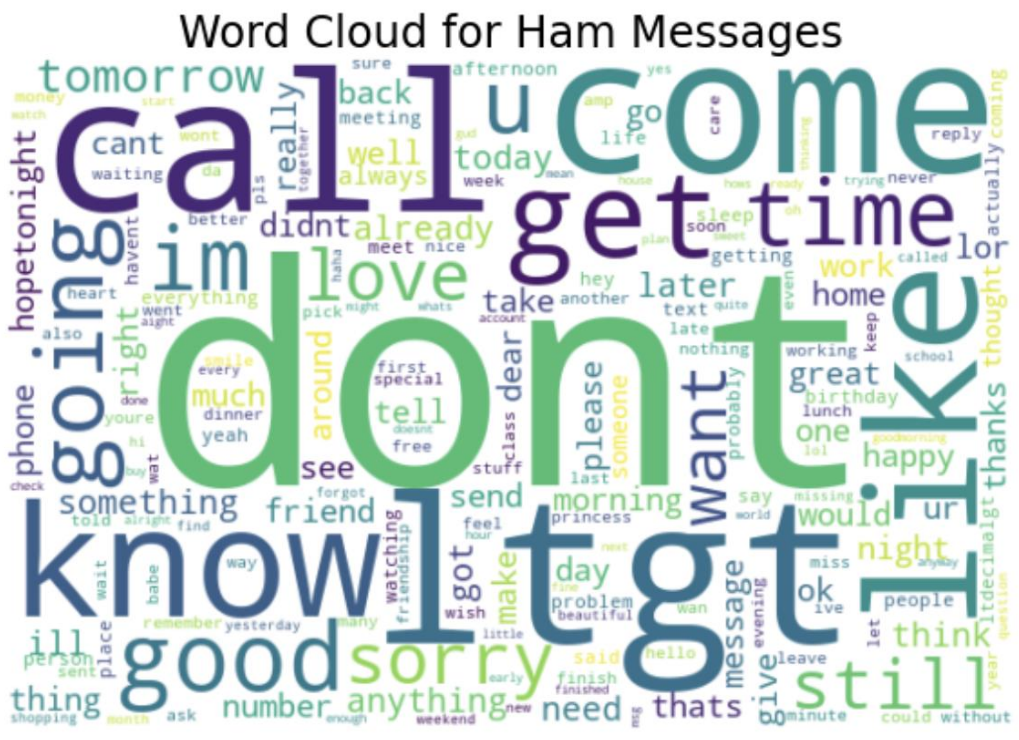


Figure 5. Word Cloud for Ham Messages

3.3. Data Preprocessing

Data preprocessing is a vital step in a text classification task. It ensures all data are thoroughly cleaned, enhances the contextual richness of information provided to the model, and helps to make a more accurate classification. In this project, we mainly applied the following data preprocessing strategies.

1. **Remove special characters**. Special characters like punctuation and symbols may not carry meaningful contextual information in a text analysis task and may affect the accuracy of the model.

2. **Remove stop words**. In this project, stop words like "and", "the", etc. are removed since they frequently appear in a message but are often meaningless in a text analysis task.

3. **Lemmatization**. In this project, we used the 'WordNetLemmatizer' from NLTK for lemmatization. It helps to conduct linguistic analysis and reduce words to their root or base form. This method helps improve model performance by providing more accurate and semantically meaningful word representations.

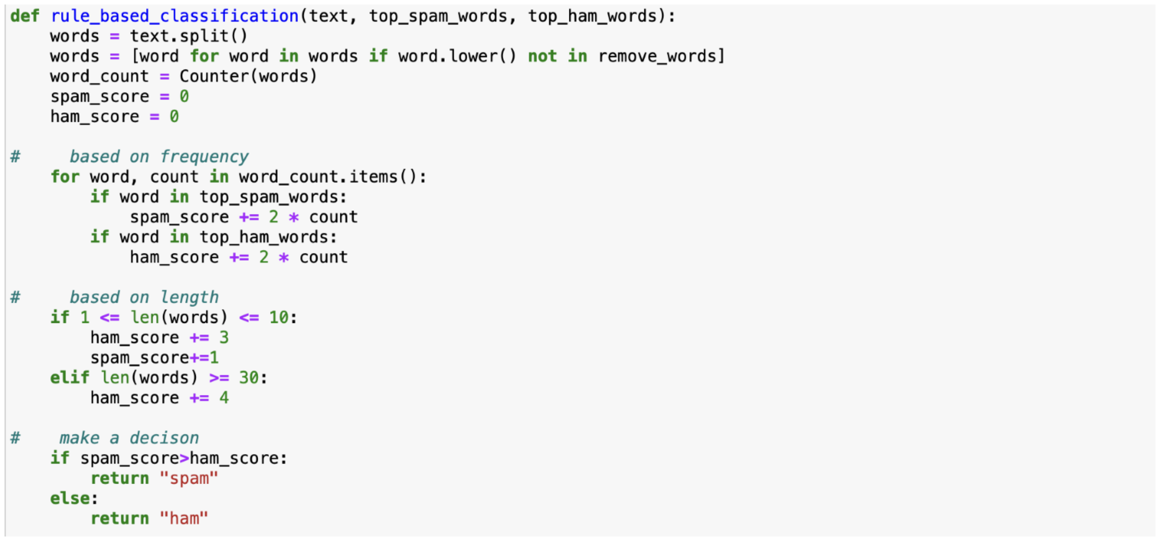
4. **Remove empty messages**. Null values and invalid records are recognized and deleted in the data preprocessing step.

After preprocessing, the data is now ready to be used as training and test sets.

4. Implementation and Comparison

4.1. Traditional Rule-based Method

In order to get more insight into the effectiveness of traditional approaches in a spam message detection task, we tried this rule-based model which simply detects spam messages with the self-defined rules.

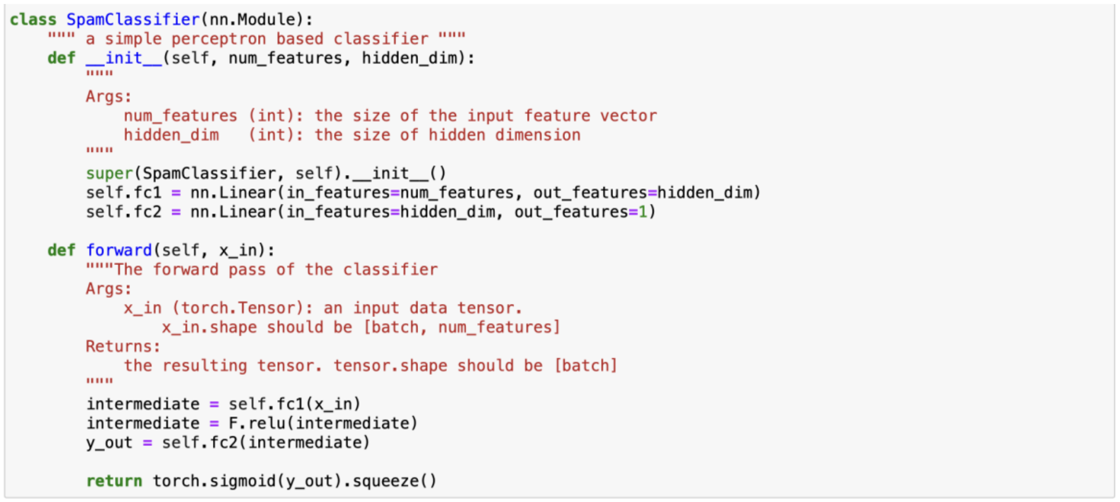


```
def rule_based_classification(text, top_spam_words, top_ham_words):
    words = text.split()
    words = [word for word in words if word.lower() not in remove_words]
    word_count = Counter(words)
    spam_score = 0
    ham_score = 0

#     based on frequency
    for word, count in word_count.items():
        if word in top_spam_words:
            spam_score += 2 * count
        if word in top_ham_words:
            ham_score += 2 * count

#     based on length
    if 1 <= len(words) <= 10:
        ham_score += 3
        spam_score+=1
    elif len(words) >= 30:
        ham_score += 4

#    make a decison
    if spam_score>ham_score:
        return "spam"
    else:
        return "ham"
```

The rule-based model implemented in the paper is mainly focused on word frequency and text length. Each message category has unique high-frequency words. Besides, based on the EDA, we found there is an obvious text length disparity within the two categories. Given these reasons, we calculated the score of each message and classified the category of the message based on the score. As a result, the rule-based approach gets an 85.95% score in accuracy.

4.2. Multi-layer Neural Network

Next, we attempt to use a simple multilayer neural network to identify spam messages. The following code shows the construction of the model.



```python
class SpamClassifier(nn.Module):
    """ a simple perceptron based classifier """
    def __init__(self, num_features, hidden_dim):
        """
        Args:
            num_features (int): the size of the input feature vector
            hidden_dim   (int): the size of hidden dimension
        """
        super(SpamClassifier, self).__init__()
        self.fc1 = nn.Linear(in_features=num_features, out_features=hidden_dim)
        self.fc2 = nn.Linear(in_features=hidden_dim, out_features=1)

    def forward(self, x_in):
        """The forward pass of the classifier
        Args:
            x_in (torch.Tensor): an input data tensor.
                x_in.shape should be [batch, num_features]
        Returns:
            the resulting tensor. tensor.shape should be [batch]
        """
        intermediate = self.fc1(x_in)
        intermediate = F.relu(intermediate)
        y_out = self.fc2(intermediate)

        return torch.sigmoid(y_out).squeeze()
```

Simple tuning process:

	Accuracy	Loss	Apply 5-Pretrainned gram layer	Pretrained	Frequency_cutoff	Hidden dim	Batch_size	Learning_rt
1	98.24	0.058	No	None	25	20	128	0.001
2	98.05	0.077	No	None	25	20	128	0.0005
3	98.24	0.073	No	None	25	40	128	0.0005
4	98.63	0.070	No	None	25	40	128	0.001

Training curve:

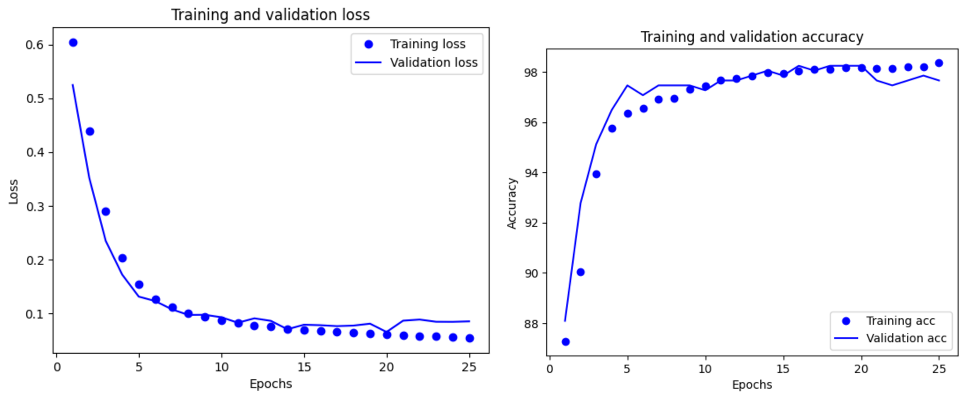


Figure 6. Multilayer Neural Network Training Curve

We can see that the introduction of the neural network has significantly improved the accuracy of SMS classification. The figures below display the classification results. The test accuracy can reach 98.63.

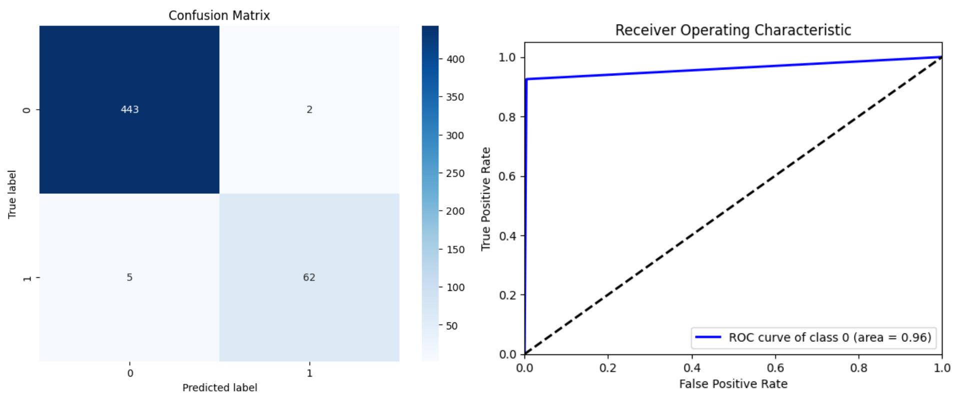
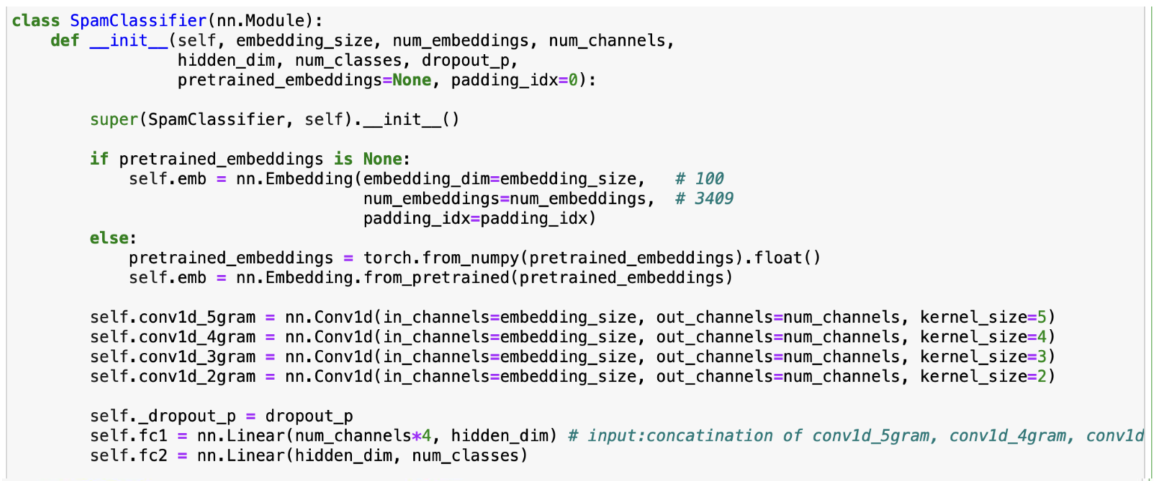


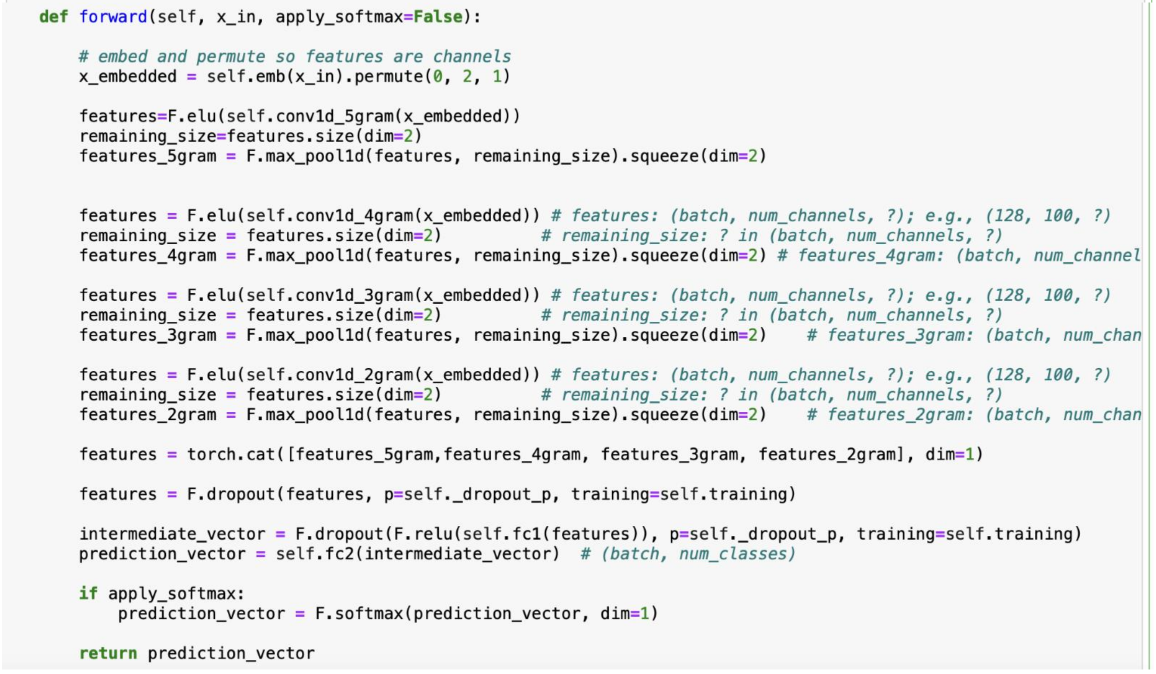
Figure 6. Multilayer Neural Network Classification Result

4.3. CNN

We also tried the CNN model since it’s able to provide contextual information which is helpful in this spam detection task. The implementation of the CNN model is as follows.



```
class SpamClassifier(nn.Module):
    def __init__(self, embedding_size, num_embeddings, num_channels,
                 hidden_dim, num_classes, dropout_p,
                 pretrained_embeddings=None, padding_idx=0):

        super(SpamClassifier, self).__init__()

        if pretrained_embeddings is None:
            self.emb = nn.Embedding(embedding_dim=embedding_size,   # 100
                                    num_embeddings=num_embeddings,  # 3409
                                    padding_idx=padding_idx)
        else:
            pretrained_embeddings = torch.from_numpy(pretrained_embeddings).float()
            self.emb = nn.Embedding.from_pretrained(pretrained_embeddings)

        self.conv1d_5gram = nn.Conv1d(in_channels=embedding_size, out_channels=num_channels, kernel_size=5)
        self.conv1d_4gram = nn.Conv1d(in_channels=embedding_size, out_channels=num_channels, kernel_size=4)
        self.conv1d_3gram = nn.Conv1d(in_channels=embedding_size, out_channels=num_channels, kernel_size=3)
        self.conv1d_2gram = nn.Conv1d(in_channels=embedding_size, out_channels=num_channels, kernel_size=2)

        self._dropout_p = dropout_p
        self.fc1 = nn.Linear(num_channels*4, hidden_dim) # input:concatination of conv1d_5gram, conv1d_4gram, conv1d
        self.fc2 = nn.Linear(hidden_dim, num_classes)
```



```
   def forward(self, x_in, apply_softmax=False):

       # embed and permute so features are channels
       x_embedded = self.emb(x_in).permute(0, 2, 1)

       features=F.elu(self.conv1d_5gram(x_embedded))
       remaining_size=features.size(dim=2)
       features_5gram = F.max_pool1d(features, remaining_size).squeeze(dim=2)


       features = F.elu(self.conv1d_4gram(x_embedded)) # features: (batch, num_channels, ?); e.g., (128, 100, ?)
       remaining_size = features.size(dim=2)          # remaining_size: ? in (batch, num_channels, ?)
       features_4gram = F.max_pool1d(features, remaining_size).squeeze(dim=2) # features_4gram: (batch, num_channel

       features = F.elu(self.conv1d_3gram(x_embedded)) # features: (batch, num_channels, ?); e.g., (128, 100, ?)
       remaining_size = features.size(dim=2)          # remaining_size: ? in (batch, num_channels, ?)
       features_3gram = F.max_pool1d(features, remaining_size).squeeze(dim=2)    # features_3gram: (batch, num_chan

       features = F.elu(self.conv1d_2gram(x_embedded)) # features: (batch, num_channels, ?); e.g., (128, 100, ?)
       remaining_size = features.size(dim=2)          # remaining_size: ? in (batch, num_channels, ?)
       features_2gram = F.max_pool1d(features, remaining_size).squeeze(dim=2)    # features_2gram: (batch, num_chan

       features = torch.cat([features_5gram,features_4gram, features_3gram, features_2gram], dim=1)

       features = F.dropout(features, p=self._dropout_p, training=self.training)

       intermediate_vector = F.dropout(F.relu(self.fc1(features)), p=self._dropout_p, training=self.training)
       prediction_vector = self.fc2(intermediate_vector)  # (batch, num_classes)

       if apply_softmax:
           prediction_vector = F.softmax(prediction_vector, dim=1)

       return prediction_vector
```

The model computes vectors for word sequences of lengths 5, 4, 3 and 2 to make good use of contextual information. In addition, we compared the model with a simpler one without the 5-gram layer, to verify the word sequence information extracted in this model is helpful to this specific task. Besides, we tried to use Glove as the pre-trained embedding layer and compared its effectiveness. What’s more, the hyperparameters including embedding_size, hidden_dim, num_channels, drop_out rate, learning_rate, batch_size are fine-tuned to get the best parameters combination and obtain the most robust model.

Tuning process:

	Accuracy	**Loss**	**Apply 5-gram layer**	**Pretrainned**	**Embedding_size**	**Num_channels**	**Hidden_dim**	**dropout**	**Batch_size**	**Learning_rt**
1	98.242	0.10	Yes	None	100	100	100	0.1	128	0.001
2	96.094	0.13	Yes	Glove	100	100	100	0.1	128	0.001
3	98.047	0.12	Yes	None	100	200	200	0.1	256	0.0005
4	98.438	0.11	Yes	None	100	100	200	0.1	128	0.001
5	97.070	0.15	Yes	Glove	100	100	200	0.1	128	0.001
6	97.852	0.08	No	None	100	100	200	0.1	128	0.001
7	**98.828**	**0.08**	**Yes**	**None**	**100**	**100**	**200**	**0.4**	**128**	**0.001**
8	97.461	0.16	Yes	Glove	100	100	200	0.4	128	0.001
9	97.070	0.12	No	None	100	100	200	0.4	128	0.001
10	98.242	0.09	Yes	None	100	100	200	0.4	256	0.001
11	98.438	0.08	Yes	None	100	100	200	0.4	128	0.0005

The best parameter combination and the final result are highlighted in bold font. From the tuning process, the effectiveness of multiple word sequences vectors can provide rich contextual information and lead to better model performance with more useful features. The simpler model without the 5-gram layer always obtains poor performance compared with the one that concatenates the 5-gram, 4-gram, 3-gram, and 2-gram features. Besides, it's noticeable that Glove pre-trained embeddings don't help in this specific task as expected. The result shows the model trained with Glove embeddings obtains a less satisfying accuracy and loss score when other parameters remain the same. Finally, the CNN model achieves an accuracy of 98.828 and a loss of 0.08 when the embedding size is set to 100, the number of channels is 100, the hidden dimension is 200, the dropout rate is 0.4, and the learning rate is 0.001.

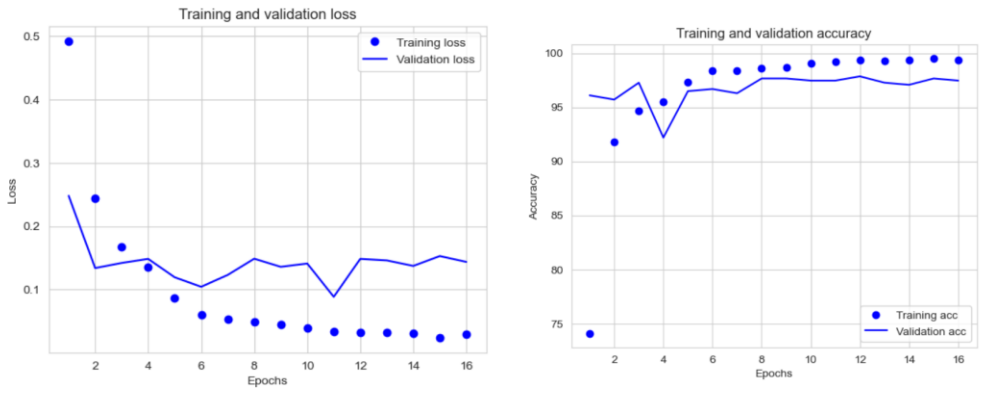


Figure 7. CNN Training Curve

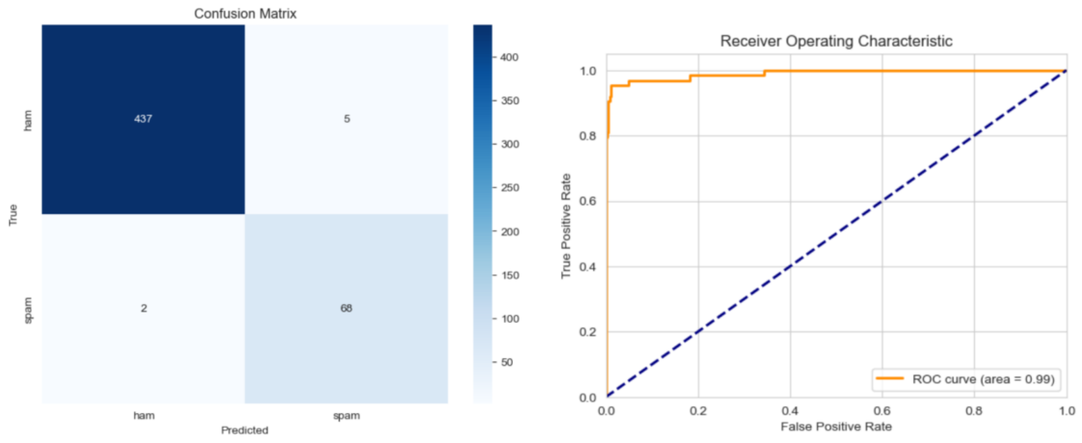
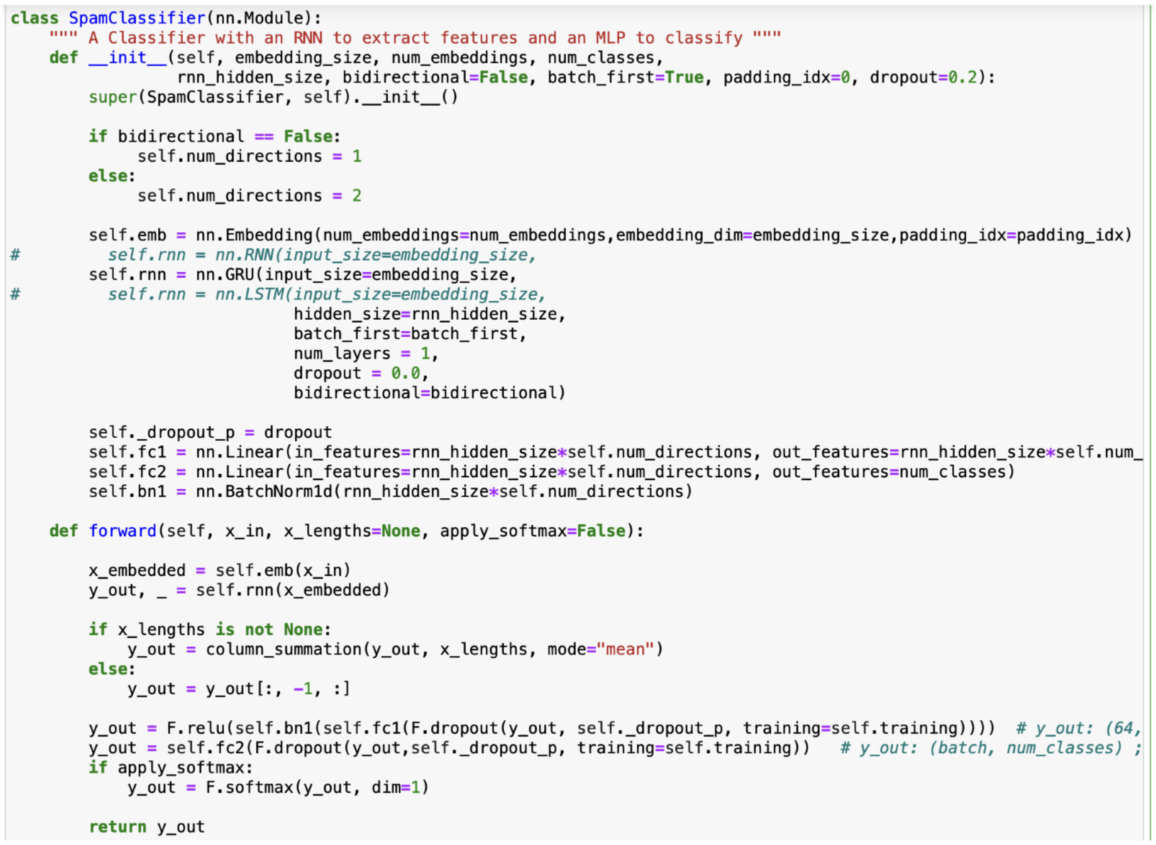


Figure 8. CNN Classification Result

4.4. RNN, GRU, and LSTM

The RNN model is also compared in this task. The detailed model definition can be seen below.



```python
class SpamClassifier(nn.Module):
    """ A Classifier with an RNN to extract features and an MLP to classify """
    def __init__(self, embedding_size, num_embeddings, num_classes,
                 rnn_hidden_size, bidirectional=False, batch_first=True, padding_idx=0, dropout=0.2):
        super(SpamClassifier, self).__init__()

        if bidirectional == False:
             self.num_directions = 1
        else:
             self.num_directions = 2

        self.emb = nn.Embedding(num_embeddings=num_embeddings,embedding_dim=embedding_size,padding_idx=padding_idx)
#         self.rnn = nn.RNN(input_size=embedding_size,
        self.rnn = nn.GRU(input_size=embedding_size,
#         self.rnn = nn.LSTM(input_size=embedding_size,
                             hidden_size=rnn_hidden_size,
                             batch_first=batch_first,
                             num_layers = 1,
                             dropout = 0.0,
                             bidirectional=bidirectional)

        self._dropout_p = dropout
        self.fc1 = nn.Linear(in_features=rnn_hidden_size*self.num_directions, out_features=rnn_hidden_size*self.num_
        self.fc2 = nn.Linear(in_features=rnn_hidden_size*self.num_directions, out_features=num_classes)
        self.bn1 = nn.BatchNorm1d(rnn_hidden_size*self.num_directions)

    def forward(self, x_in, x_lengths=None, apply_softmax=False):

        x_embedded = self.emb(x_in)
        y_out, _ = self.rnn(x_embedded)

        if x_lengths is not None:
            y_out = column_summation(y_out, x_lengths, mode="mean")
        else:
            y_out = y_out[:, -1, :]

        y_out = F.relu(self.bn1(self.fc1(F.dropout(y_out, self._dropout_p, training=self.training))))  # y_out: (64,
        y_out = self.fc2(F.dropout(y_out,self._dropout_p, training=self.training))   # y_out: (batch, num_classes) ;
        if apply_softmax:
            y_out = F.softmax(y_out, dim=1)

        return y_out
```

We explored the efficiency of RNN, GRU, and LSTM. What's more, the bidirectional choice is also considered in the tuning process to compare whether this strategy can feed more rich feature information to the model and lead to more accurate results. Two linear layers and one batch norm layer are added afterward to generate a robust model.

Tuning process:

	Accuracy	**Loss**	**Model**	**Embedding_size**	**Hidden_size**	**Bidirectional**	**Max/Mean Vector**	**dropout**	**Batch_size**	**Learning_rt**
1	96.484	0.16	RNN	100	64	False	Max	0.2	64	0.001
2	96.289	0.13	RNN	100	64	True	Max	0.2	64	0.001
3	96.682	0.13	RNN	100	64	False	Mean	0.2	64	0.001
4	98.047	0.18	RNN	100	64	True	Mean	0.2	64	0.001
5	98.047	0.13	GRU	100	64	True	Mean	0.2	64	0.001
6	93.945	0.14	LSTM	100	64	True	Mean	0.2	64	0.001
7	96.094	0.19	RNN	100	128	True	Mean	0.2	64	0.001
8	98.047	0.11	GRU	100	128	True	Mean	0.2	64	0.001
9	**98.242**	**0.11**	**GRU**	**100**	**32**	**True**	**Mean**	**0.4**	**128**	**0.001**
10	95.898	0.13	GRU	100	32	False	Mean	0.4	128	0.001
11	96.680	0.15	GRU	100	128	True	Mean	0.4	128	0.001
12	97.656	0.10	GRU	300	128	True	Mean	0.2	256	0.001
13	98.047	0.11	GRU	300	200	True	Mean	0.2	256	0.001

From the first four records, it's obvious that taking the mean vector always leads to a higher accuracy score and a lower loss score. The superiority of the mean vector can be proved. Additionally, we can see that applying a bidirectional strategy is also helpful to achieve better performance. In the following trials, we continued to compare GRU and LSTM models and found GRU has more satisfying results in both accuracy and loss. Then we tried to fine-tune the model by adjusting other hyperparameters. After multiple attempts, the GRU model obtains its best performance when the embedding_size=100, hidden_size=32, dropout=0.4, batch_size=128, with an accuracy of 98.242 and a loss of 0.11.

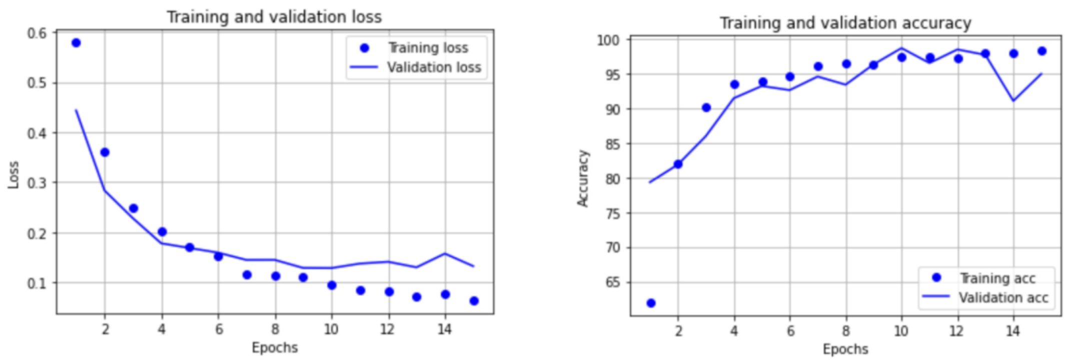


Figure 9. RNN Training Curve

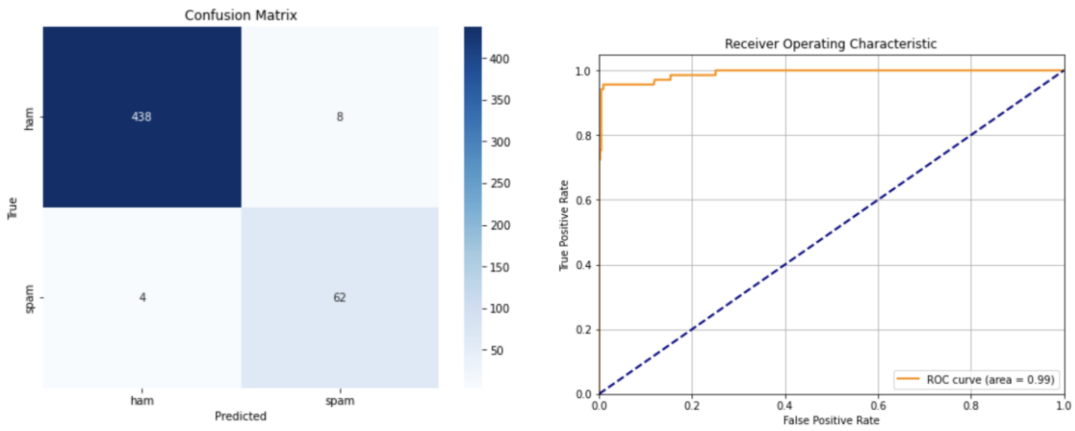
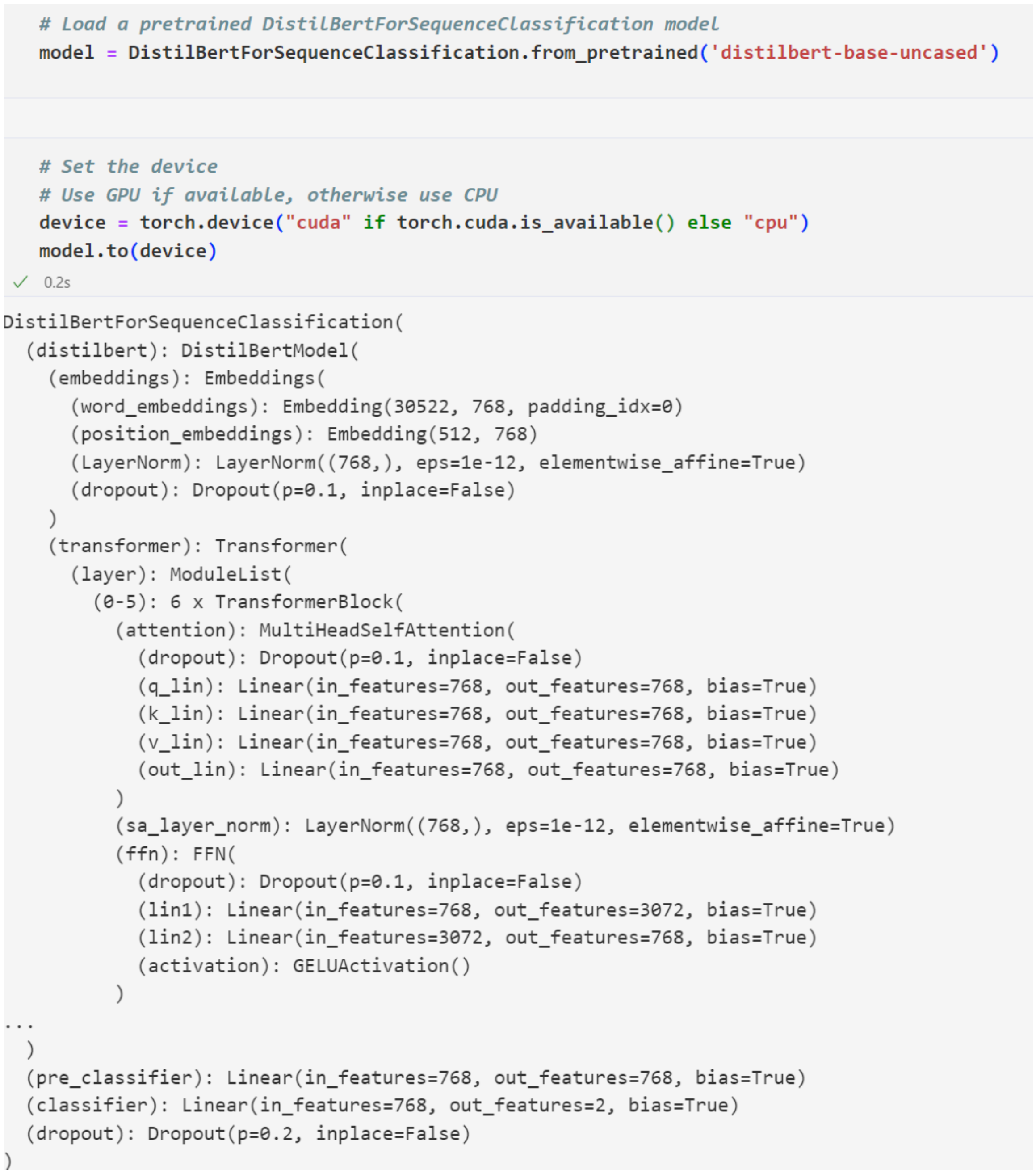


Figure 10. RNN Classification Result

4.5. BERT

In the last stage of the project, we explored BERT (Bidirectional Encoder Representations from Transformers). BERT is a pre-trained language representation model developed by Google AI researchers.

We utilized DistilBERT in the classification task. DistilBERT is a lightweight version of BERT, developed by the Hugging Face team. It is distilled from the BERT model using a technique called knowledge distillation. In this process, a smaller model is trained to replicate the behavior of a larger, more complex model. The result is that DistilBERT retains much of BERT's performance while reducing the size and complexity of the model, making it easier to deploy and less computationally expensive.



```
# Load a pretrained DistilBertForSequenceClassification model
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased')
```

```
# Set the device
# Use GPU if available, otherwise use CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

✓ 0.2s

```
DistilBertForSequenceClassification(
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
...
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

During training, we can observe that the BERT model incurs very high training costs, requiring a significant amount of time and extensive resources (especially the need for GPU acceleration to shorten the duration of training). However, at the same time, DistilBERT achieves the best training results out of all the methods we tried, with a test accuracy of 99.64 and a test loss of 0.01.

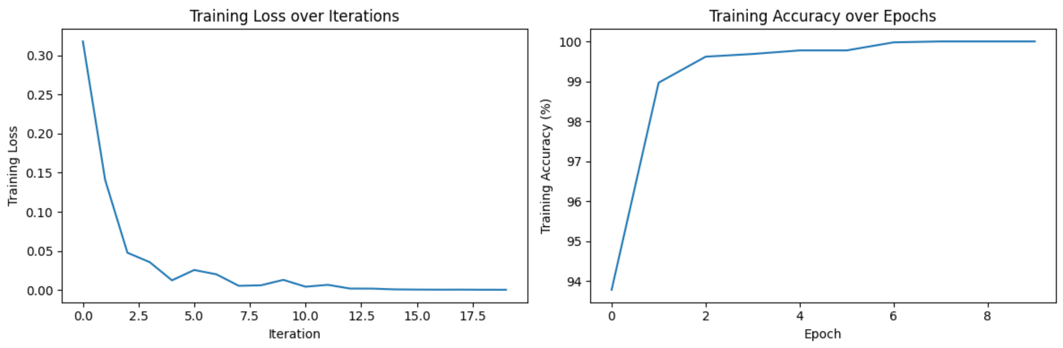


Figure 11. BERT Training Curve

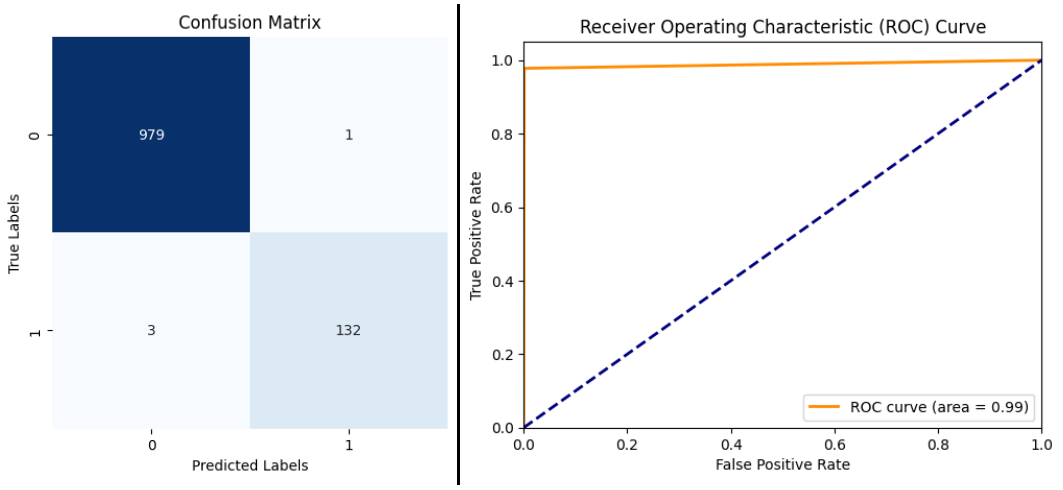


Figure 12. BERT Classification Result

5. Conclusion

In this project, we have attempted to classify spam and legitimate messages in SMS using a variety of methods. Overall, the dataset we used has achieved good classification results across multiple algorithms. The table below is a simple summary of the performance of several different algorithms and models:

Model	Test loss	Test accuracy
Rule-based	/	85.95
Multilayer Neural Network	0.07	98.63
RNN (GRU)	0.08	98.24
CNN	0.07	98.83
BERT (DistilBERT)	**0.01**	**99.64**

From the results, we confirmed that the BERT model exhibits outstanding performance in text sequence classification tasks. However, from the perspective of training costs, a simple multilayer neural network also achieved good classification results on our dataset with a very low training cost. In the summary after the completion of the project, we believe that the classification of this dataset is relatively simple, and the data sample size is not large, which is an important reason for the good results under simple models. If faced with larger datasets, or more complex classification tasks, the advantages of deep learning models such as CNN and BERT would become more pronounced.

6. References

[*] IS6751 Lecture notes; Some illustrating images are from the web

[1] Gadde, S., Lakshmanarao, A., & Satyanarayana, S. (2021, March). SMS spam detection using machine learning and deep learning techniques. In 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS) (Vol. 1, pp. 358-362). IEEE.

[2] Lota, L. N., & Hossain, B. M. (2017). A systematic literature review on sms spam detection techniques. International Journal of Information Technology and Computer Science (IJITCS), 9(7), 42-50.

[3] Shafi'I, M. A., Abd Latiff, M. S., Chiroma, H., Osho, O., Abdul-Salaam, G., Abubakar, A. I., & Herawan, T. (2017). A review on mobile SMS spam filtering techniques. IEEE Access, 5, 15650-15666.

[4] Karami, A., & Zhou, L. (2014). Improving static SMS spam detection by using new content-based features.

[5] Chan, P. P., Yang, C., Yeung, D. S., & Ng, W. W. (2015). Spam filtering for short messages in adversarial environment. Neurocomputing, 155, 167-176.

[6] Nagwani, N. K. (2017). A Bi-Level Text Classification Approach for SMS Spam Filtering and Identifying Priority Messages. International Arab Journal of Information Technology (IAJIT), 14(4).

[7] Alzahrani, A., & Rawat, D. B. (2019, April). Comparative study of machine learning algorithms for SMS spam detection. In 2019 SoutheastCon (pp. 1-6). IEEE.

[8] Aliza, H. Y., Nagary, K. A., Ahmed, E., Puspita, K. M., Rimi, K. A., Khater, A., & Faisal, F. (2022, March). A comparative analysis of SMS spam detection employing machine learning methods. In 2022 6th International Conference on Computing Methodologies and Communication (ICCMC) (pp. 916-922). IEEE.

[9] Annareddy, S., & Tammina, S. (2019, December). A comparative study of deep learning methods for spam detection. In 2019 third international conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC) (pp. 66-72). IEEE.

[10] Popovac, M., Karanovic, M., Sladojevic, S., Arsenovic, M., & Anderla, A. (2018, November). Convolutional neural network based SMS spam detection. In 2018 26th Telecommunications forum (TELFOR) (pp. 1-4). IEEE.

[11] Roy, P. K., Singh, J. P., & Banerjee, S. (2020). Deep learning to filter SMS Spam. Future Generation Computer Systems, 102, 524-533.

[12] Gomaa, W. H. (2020). The impact of deep learning techniques on SMS spam filtering. International Journal of Advanced Computer Science and Applications, 11(1).

[13] Poomka, P., Pongsena, W., Kerdprasop, N., & Kerdprasop, K. (2019). SMS spam detection based on long short-term memory and gated recurrent unit. International Journal of Future Computer and Communication, 8(1), 11-15.

[14] Odera, D., & Odiaga, G. (2023). A comparative analysis of recurrent neural network and support vector machine for binary classification of spam short message service. World Journal of Advanced Engineering Technology and Sciences, 9(1), 127-152.

[15] Sahmoud, T., & Mikki, D. M. (2022). Spam detection using BERT. arXiv preprint arXiv:2206.02443.

[16] Raga, S. S., & Chaitra, B. L. (2022, December). A bert model for sms and twitter spam ham classification and comparative study of machine learning and deep learning technique. In 2022 IEEE 7th International Conference on Recent Advances and Innovations in Engineering (ICRAIE) (Vol. 7, pp. 355-359). IEEE.