Abstract

The world economy and security have been seriously threatened by the rise in unsolicited emails as a result of technological improvements and the ease with which emails can now be communicated. Highdimensional and redundant datasets used during model training may lower the classification outcomes of the model since it requires a lot of work and memory. Feature selection is a crucial data processing method that aids in choosing pertinent features and information subsets from the dataset. As a result, selecting effective feature selection methods is crucial for a model's optimal classification performance.

Furthermore, typical machine learning approaches, which are insufficient to handle the vast volume of data and its fluctuations, have been used majorly. Additionally, as technology advances, spammers are growing more sophisticated. To address these issues, hybrid approaches combining deep learning and traditional algorithms are therefore required. A unique methodology for email spam detection that will improve the original dataset's feature selection method and raise the classifier's accuracy. GWO-BERT will be used with deep learning methods like CNN, biLSTM, and LSTM. The spam detection system is developed as a simple, user-friendly web application where users can input the subject and message of an email. Upon submission, the system classifies the input as either "Spam" or "Not Spam" and displays the result on the same page.

CHAPTER 1

INTRODUCTION SUMMARY OF BASE PAPER

Title	: Email spam detection by deep learning models using novel feature selection technique and BERT
Publisher	: Elsevier
Year	: 2024
Journal	: Egyptian Informatics Journal 26 (2024) 100473
Indexing	: SCI / Scopus
Base paper URL	: https://www.sciencedirect.com/science/article/pii/S1110866524000367

1.1 BACKGROUND

Today, email is still one of the most used forms of communication worldwide for personal, academic, and professional purposes. Email communication is very successful; however, the success rate is continually challenged by the growth of unsolicited email messages, commonly referred to as spam. Spam email messages can come in many forms; let me simply say there are unsolicited and unsolicited spam emails ranging from advertisements to more advanced email phishing attempts and malware distribution. Spam email messages not only interfere with communication, but also can result in all kinds of cyber security threats, such as unauthorized access, data breaches, and financial fraud. Given the volume of emails being created every day, we must introduce automated and intelligent filtering systems to identify material that an individual can use or may harm them.

1.2 PROBLEM STATEMENT

Many spam detection techniques still really struggle. These static techniques feel are limited in detecting new types of spam. Traditional spam filtering is based on keyword detection or fixed text frequency analyses and do not effectively take into account evolving spam detection techniques. Many traditional spam detection systems use regression models that completely ignore the semantic meaning of spam detection which results in increasing numbers of spam email incorrectly classified (false positives) as ham email (legitimate email) and also misclassifying false negatives (spam emails undiscovered by spam detectors). Also, many traditional spam detectors suffer from being inefficient many times due to redundant features or irrelevant features associated with the model and not tuning the model hyper-parameters accurately. All of these issues degrade these system effectiveness and scalability in an applied context.

1.3 SCOPE

The methodology exclusively concentrates on detecting text-based spam emails. The study employs both the subject line and the message body for feature extraction. This study contrasts two types of feature extraction: TF-IDF and BERT-based tokenization, and uses hand-crafted features

to increase representational depth, with various machine learning, and deep learning classifiers to assess performance. Grey Wolf Optimization is used for feature selection. Particle Swarm Optimization is used for the hyper parameter tuning, primarily for the GRU models.

1.4 MOTIVATION

Spam emails are an evolving problem and the methods of spam deliverers are developing so fast that they continue to get passed conventional detection systems. Current solutions are not working or sufficient because they lead to increased false rates as well as security limitations. A real solution, if developed, could vastly improve an email clients spam detection/security, reduced manual filtering for users, and ultimately replaced a painful experience for users. This model solves the gap of false rates by creating a spam filter model combining BERT with optimization techniques to form an adaptable spam detection system armed with parameter tuning best positioned to accurately detect spam in the complexities of current spam.

CHAPTER 2

OBJECTIVES

The objective is to construct a spam detection system for traditional machine learning (ML) and deep learning (DL) approaches. The spam detection system uses the subject and content of the email message to classify emails as "Spam" or "Not Spam". Input data is processed using handcrafted linguistic features such as average word length, punctuation frequency, and presence of hyperlinks as shallow yet significant indicators of possible spam behaviour.

To accomplish this, the work focuses on a variety of advanced ML and DL models including Random Forest, Naive Bayes, Convolutional Neural Networks (CNN), Long Short Term (LSTMs), Bidirectional Long Short Term (BiLSTMs), and Gated Recurrent Units (GRUs). Performance will be evaluated using Accuracy, which will determine the reliability of the classification.

Feature selection will be further optimized using Grey Wolf Optimization (GWO) with the aim of selecting the more salient features in order to simplify the inputs and ultimately promote better performance. Along with the model hyper parameters changing in the GRU model using Particle Swarm Optimization (PSO) to improve learning convergence and classification accuracy. The final goal is to provide a secure spam detection system that is robust, accurate, and scalable through safe and efficient email filtering.

CHAPTER 3

EXPERIMENTAL WORK / METHODOLOGY

3.1 DATA COLLECTION AND PREPROCESSING

The Lingspam dataset is for this study. There are 4824 email samples used in the dataset which are either labeled as spam or ham (non-spam). This gives us a lot of text, originating from linguistic forums, so it has a range of vocabulary and structure. The original email includes both the subject line and body and we will treat both as text inputs for this analysis.

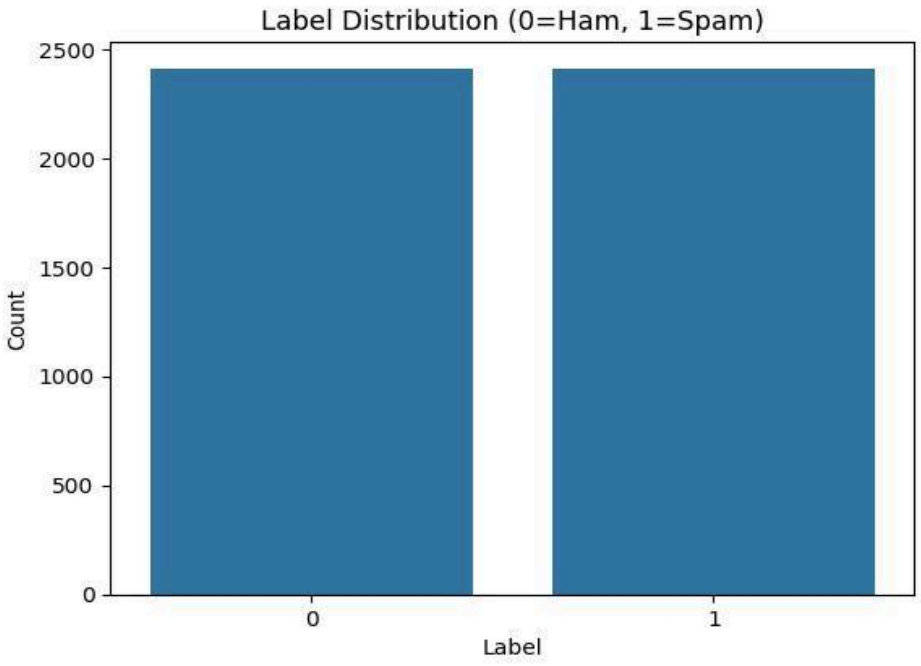

Fig. 3.1 Label Distribution of Dataset

Pre-processing steps are required to clean and standardize the source text data, as well as help with feature extraction quality and model performance. The pre-processing steps are outlined as follows:

- **Lowercasing:** The study used lowercasing to ensure consistency, but it also negates the differences between cases. For example, the study no longer distinguishes between "Free" and "free"; both the 'F' and 'f' tokens are treated as equal.

- **Stop-words:** The study applies stop-words removal from their analysis, as stop-words like "the", "is", "at", "in", etc. do not add a strongly attributable value to actually deriving meaning back to a sentence, reducing words-to-meaning correlations will help filter out some noise and emphasis informative words clear.

- **Punctuation and Digit Removal:** All punctuation marks (e.g., commas, periods, exclamation points) and numeral digits are removed from the text to prevent nonalphabetic symbols from interfering with the tokenization and modeling.

- **Tokenization:** The cleaned text is tokenized into individual units (tokens) that are usually words, to be used for feature extraction. Tokenization allows models to look at how often each word appeared, and in what contexts.

- **Subject and Body Merging:** To maximize the amount of data for classification, each subject line and message body are merged into a single text input so both are considered for downstream processing. The maximize input captures all potential information as complete representations of the email.

3.2 FEATURE EXTRACTION

3.2.1 TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency) allowed the email text to be transformed into a matrix of features representing, relative to the entire corpus, how important a word is to a document. This is a relatively straightforward, fast and easy representation strategy that works well with traditional machine learning classifiers.

3.2.2 BERT-Based Tokenization

The BERT tokenizer transforms the email body into token IDs, while maintaining an association between all words. Unlike TF-IDF, where each word is seen in isolation, BERT maintains sequential and bidirectional co-dependencies of words in the email. The token IDs can be passed to the neural models for learning from the semantic structures.

3.2.3 Handcrafted Features

To also diversify the feature space, nine different handcrafted features were produced. These features included:

Length of message and subject

Number of uppercase letters

Frequency of punctuation

Numbers and special characters

Links and email

Average word length

Ratio of unique words

These additional features were included to try and capture more subtle forms of linguistic pattern that are typical in the spam content, where spam messages may use too many exclamation marks and promotional language.

3.3 FEATURE SELECTION USING GREY WOLF OPTIMIZATION

Grey Wolf Optimization (GWO) is used to choose the most relevant features from the combined set of features. GWO simulates the leadership structure and hunting mechanism found in grey wolves in nature. The optimization process reduces dimensionality by selecting feature subsets that can generate the highest classification accuracy over the iterative rounds (while minimizing redundancy).

Encircling Prey :

$$\vec{D} = |\vec{C} \cdot \vec{X}_p(t) - \vec{X}(t)|$$
$$\vec{X}(t+1) = \vec{X}_p(t) - \vec{A} \cdot \vec{D}$$

$\vec{X}_p(t)$: Position vector of the prey (best solution).

$\vec{X}(t)$: Current position of the grey wolf.

$\vec{A}, \vec{C}$: Coefficient vectors.

t: Current iteration.

$\vec{D}$: Distance between the prey and the wolf.

Coefficient Vectors :

$$\vec{A} = 2 \cdot \vec{a} \cdot \vec{r}_1 - \vec{a}$$
$$\vec{C} = 2 \cdot \vec{r}_2$$

$\vec{a}$: Linearly decreased from 2 to 0 over iterations.

$\vec{r}_1, \vec{r}_2$: Random vectors in [0, 1].

Hunting Behavior (Search Agent Update) :

$$\vec{D}_\alpha = |\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}|$$
$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot \vec{D}_\alpha$$
$$\vec{D}_\beta = |\vec{C}_2 \cdot \vec{X}_\beta - \vec{X}|$$
$$\vec{X}_2 = \vec{X}_\beta - \vec{A}_2 \cdot \vec{D}_\beta$$
$$\vec{D}_\delta = |\vec{C}_3 \cdot \vec{X}_\delta - \vec{X}|$$
$$\vec{X}_3 = \vec{X}_\delta - \vec{A}_3 \cdot \vec{D}_\delta$$

Final position update:

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3}$$

$\vec{X}_\alpha$ be the position vector of the **alpha wolf** (best solution),

$\vec{X}_\beta$ be the position of the **beta wolf** (second best),

$\vec{X}_\delta$ be the position of the **delta wolf** (third best),

$\vec{X}(t)$ be the current position of a wolf at iteration t,

a is a linearly decreasing parameter from 2 to 0 over iterations,

$\vec{r}_1, \vec{r}_2$ are random vectors in $[0, 1]$,

$\vec{A} = 2a \cdot \vec{r}_1 - a$,

$\vec{C} = 2 \cdot \vec{r}_2$.

3.4 CLASSIFIER MODELS

In order to assess the efficacy of different feature extraction and feature selection methods, two general types of classifiers were used: traditional machine learning models and deep learning models. Each type was chosen based on the applicable features (TF-IDF, BERT tokens, and handcrafted features), and their previous success on text classification tasks.**3.4.1 Traditional Models**

Traditional classifiers are lightweight and computationally efficient, focused on sparse feature the previously mentioned TF-IDF ones. Traditional classifiers serve as a baseline for other architectures.

Random Forest (RF):
This ensemble based classifier produces multiple decision trees during the training phase, and chooses the majority vote from all of the trees. The RF method can handle high-dimensional feature spaces and is less prone over fitting because of the averaging effect. RF is used with the TF-IDF feature set to classify the emails as spam or ham.

Naive Bayes (NB):

This classifier acts on Bayes' Theorem and is one of the basic classifiers but may not perform well lex set of feature combinations. Its key assumption of conditional independence of features makes the modelling of the likelihood of text relatively easy. NB performs surprisingly well in text classification problems and is often used with word frequency–based input like TF-IDF as features. The computed features are often sparse and NB model for it is suited for large, slow running datasets.

3.4.2 Deep Learning Models

Deep Learning Models can learn both hierarchical and time-based representations directly from the input data. Most of the deep learning architectures can be used with dense feature vectors of data that are rich in context derived from BERT tokenization.

Convolutional Neural Network (CNN):

CNN's were developed for image processing tasks and work effectively for text classification because they use convolutional filters to learn local dependencies. In this use case, the CNN operates over the tokenized sequences in order to identify n-gram patterns within email text that indicate-spam.

Long Short-Term Memory (LSTM):

LSTMs are a specific type of recurrent neural network that fixes the vanishing gradient representation of RNNs, while providing an architecture to carry long term memory of sequences. LSTM are appropriate for email text as tokenization retains the order of the text structure - LSTMs can use internal cells and gates to facilitate sequence learning while carrying memory of past-sequence.

Bidirection LSTM (biLSTM):

biLSTM is an extension of LSTM's that allows input sequences to be processed in both forward and backward directions, giving the model access to future and past semantic context of sentences the model is evaluating, which, is important in spam classification tasks for developing a semantic context.

Gated Recurrent Unit (GRU):

GRU is a more computationally efficient version of LSTM using fewer gates, while still enabling the modelling of temporal dependencies. In this study, GRU is applied to a fused input structure - BERT token sequences in conjunction with handcrafted statistical features. Because of the better ability of GRU to model hybrid-type input formats, and copious less complexity compared to LSTM that yielded faster training time with relatively similar accuracy.

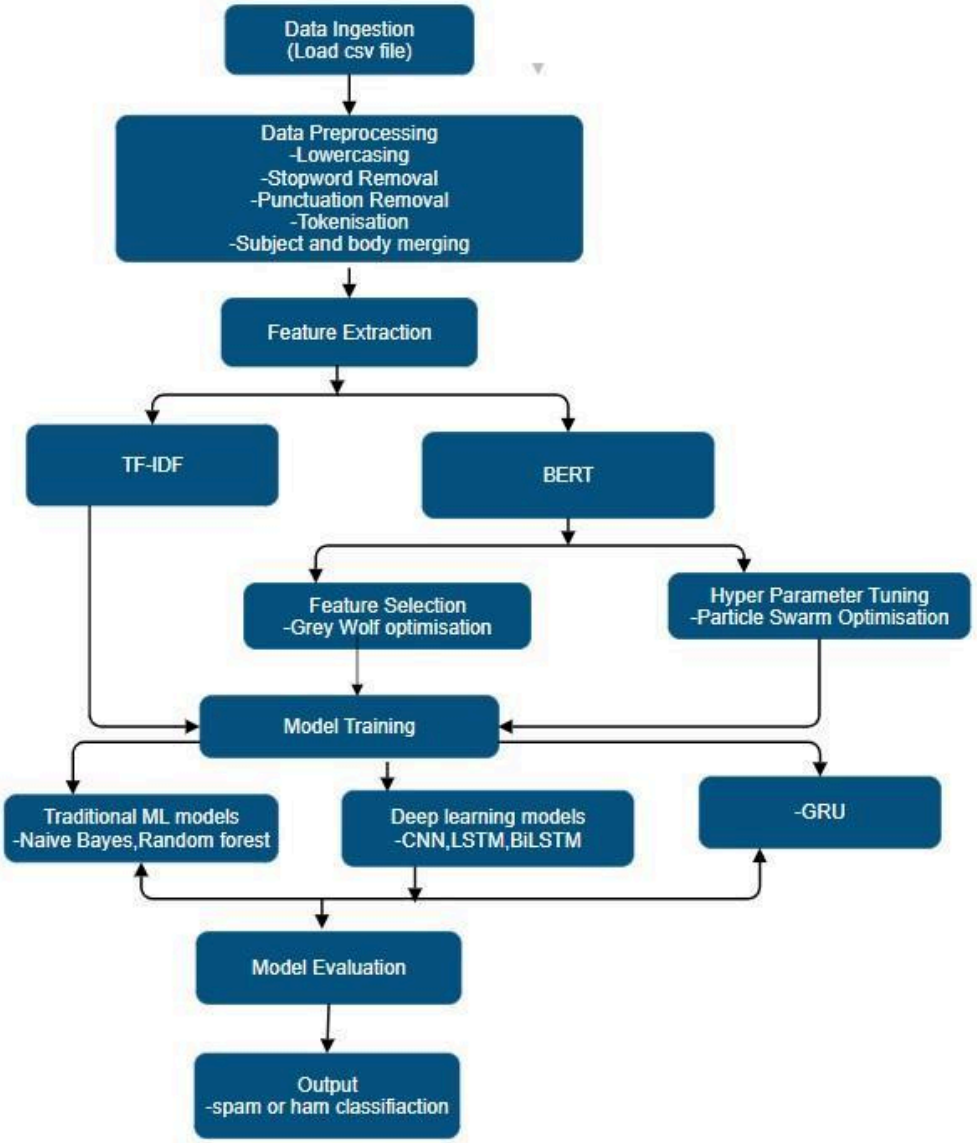

Fig. 3.4 Model Architecture

3.5 HYPERPARAMETER TUNING USING PARTICLE SWARM OPTIMIZATION (PSO)

Particle Swarm Optimization (PSO) is a hyper parameter tuning method that greatly automates the process of finding the best hyper parameter settings that will maximize a model performance measure. Instead of using a manual or random search model for hyper parameter options (like learning rate and GRU units), PSO uses the premise that each candidate solution can be treated as a "particle" in a swarm. The swarm of particles then moves through the search space based on the best solution that has been found by both itself and the swarm. The position of each particle reflects a specific hyper parameter concoction. Each particle is evaluated for its fitness based upon model performance. PSO is able to exploit cumulative fitness through positioning, allowing it to iterate through particle positions by adding velocity equations to each iteration independently.

This goal of finding hyper parameter settings using PSO is to accelerate convergence, maximizing the learning process and ultimately the model performance

$$P_i^{t+1} = P_i^t + V_i^{t+1}$$

P_t^i : Current position of particle *i* at time *t*.

P_i^{t+1}: Updated position of particle *i* at time *t+1*.

V_i^{t+1} : New velocity of particle *i*.

$$V_i^{t+1} = \underbrace{wV_i^t}_{\text{Inertia}} + \underbrace{c_1 r_1 (P_{best(i)}^t - P_i^t)}_{\text{Cognitive (Personal)}} + \underbrace{c_2 r_2 (P_{bestglobal}^t - P_i^t)}_{\text{Social (Global)}}$$

- w: Inertia weight – keeps the particle moving in its current direction.
- c_1: Cognitive coefficient – influences movement toward personal best.
- r_1: Random number [0, 1] – adds randomness to personal influence.
- $P_{\text{best}(i)}^t$: Personal best position of particle *i*.
- c_2: Social coefficient – influences movement toward global best.
- r_2: Random number [0, 1] – adds randomness to global influence.
- $P_{\text{bestglobal}}^t$: Best position found by the entire swarm.
- V_i^t: Current velocity of particle *i*.
- P_i^t: Current position of particle *i*.

CHAPTER 4 RESULTS AND DISCUSSION

4.1 EXPLAINABILITY ANALYSIS

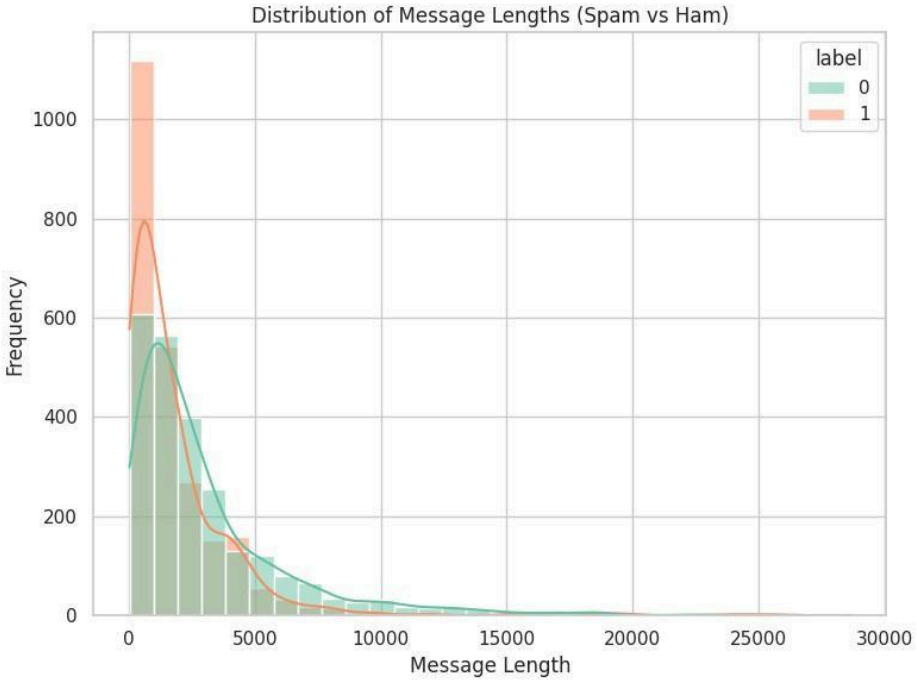

Fig. 4.1.1 Distribution of Message Length

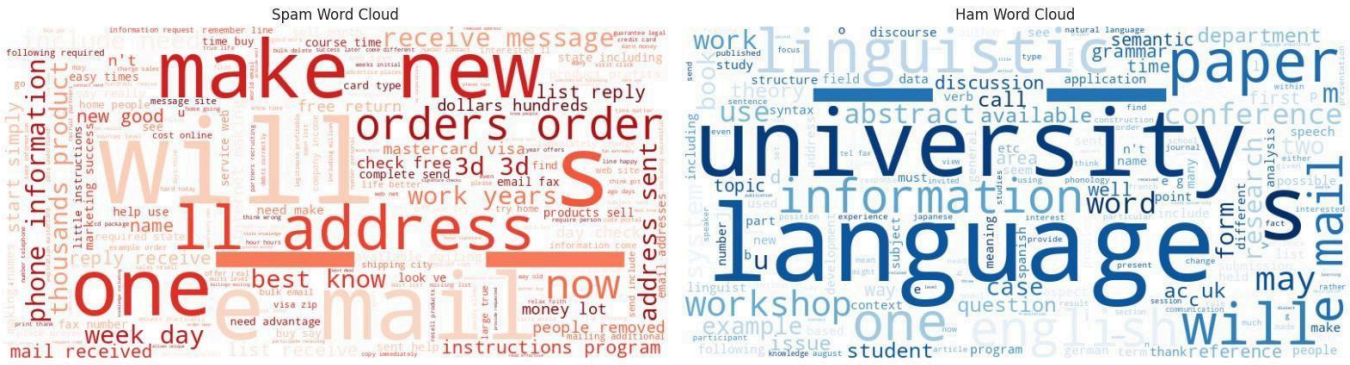

Fig. 4.1.2 Spam and Ham Word Cloud

4.2 EFFECT OF FEATURE EXTRACTION

TF-IDF functioned well with conventional Machine Learning classifiers but failed to understand the context. BERT based tokenization, specifically when combined with handcrafted features, additional context significantly improved the model and spam detection capability.

4.3 EFFECT OF GWO

Grey Wolf Optimization (GWO) was applied to select relevant features and reduce dimensionality prior to performing classification. The GWO was capable of effectively exploring the features space and eliminate redundant characteristics by imitating the leadership and hunting traits of grey wolves. The approach improved general model performance, with the addition of LSTM and biLSTM architectures performing better with GWO applied. Most notable was the accuracy improvements, e.g., LSTM improved from 97.21% to 98.80% and the biLSTM model peaked at 99.14% after GWO was implemented. Training times decreased by approximately 15-20% as a result of using a smaller input variable size. GWO also allowed for improved generalization of the model by eliminating noisy or non-informative features. Throughout the iterations of GWO there was a smooth and even transition from exploration and exploitation. By adding GWO to the pipeline, the models were better optimized with increased precision and recall respectively. Overall, GWO was an essential component of optimizing the feature set for the task of spam detection.

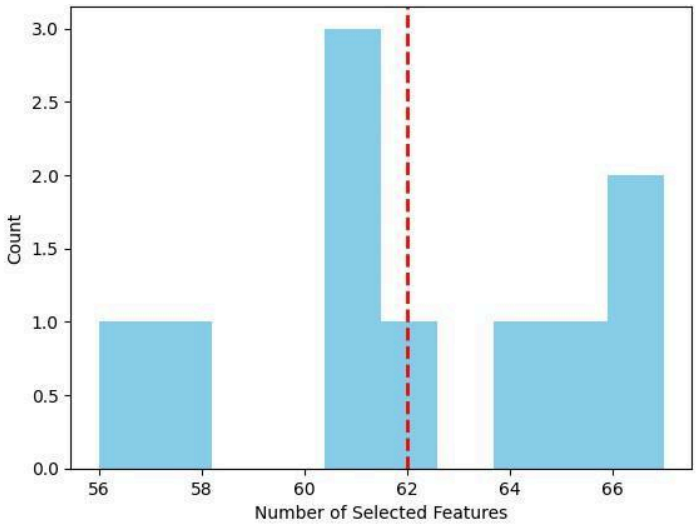

Fig. 4.3 Distribution of Selected features per Wolf

4.4 EFFECT OF PSO

MODEL	ACCURACY
GRU	94.13%
GRU+PSO	99.31%

Table 4.4 Effect of PSO

The optimization of key hyper parameters in GRU model, such as the learning rate and GRU units, was performed using Particle Swarm Optimization (PSO). The PSO algorithm mimics the social behavior of the swarms, taking advantage of a group to explore in the search space in a efficient manner. Both accuracy and precision of the GRU model improved from 98.43% accuracy to 99.10% accuracy while minimizing the overfitting. The PSO algorithm was less resource intensive than grid search or random search and was able to find a set of optimal parameters with less evaluation. Additionally, the PSO could completely automate the tuning process which emphasizes reproducibility and robustness to the GRU model used in hybrid spam detection. Through the use of PSO, the GRU model is optimized at the structural level as well as at the level of the parameters. PSO was instrumental in finding optimal parameters and additionally it provided the potential for the tuning process to become other aspects of a hybrid spam detection approach the most effective.

4.5 MODEL COMPARISON

MODEL	ACCURACY
RF + TF-IDF	88.13%
NB	83.41%

Table 4.5.1 Performance of Machine Learning Models

MODEL	ACCURACY
GWO+CNN	94.82%
GWO+LSTM	96.72%
GWO+BiLSTM	98.79%
GRU+PSO	99.31%

Table 4.5.2 Performance of Deep Learning Models

4.6 PERFORMANCE EVALUATION

GRU when combined with PSO achieved the highest accuracy, closely followed by BiLSTM. PSO and GWO helped improve model reliability and reduced overfitting and PSO, when used for hyperparameter tuning with GRU, provided faster convergence and improved model performance. Traditional models were faster to train but underperformed.

4.7 DEPLOYMENT OF SPAM DETECTION MODEL

The web-based spam detection system is developed using Flask. It integrates the deep learning model which uses BERT-based tokenization, pre-constructed linguistic features, a GRU neural network, and Particle Swarm Optimization (PSO) for hyper parameter tuning. When the user inputs the email subject and email message, the system obtains the semantic features by obtaining the BERT token IDs and measures and calculates the statistical features like character special count, word length, and the number of links and then scale and fuse the features before it is input into the GRU model where it learns by classifying to predict if it is spam or not spam. Finally the output is returned almost instantaneously via a RESTful API that makes it accurate and in realtime with limited input needed.

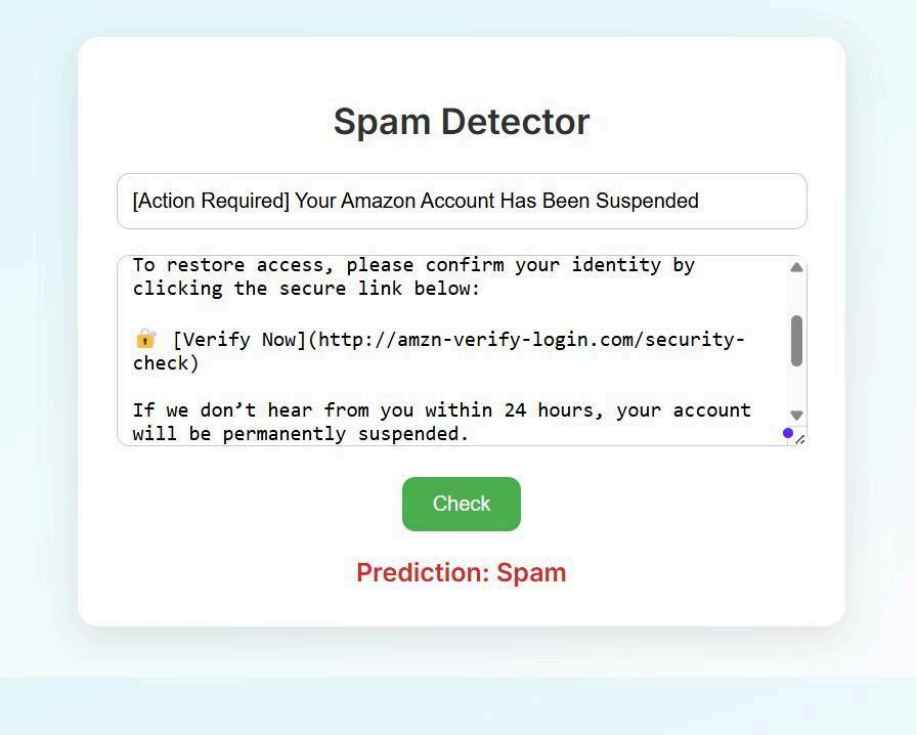

Fig. 4.7.1 Spam Detector Interface Showing Prediction for a Spam Email

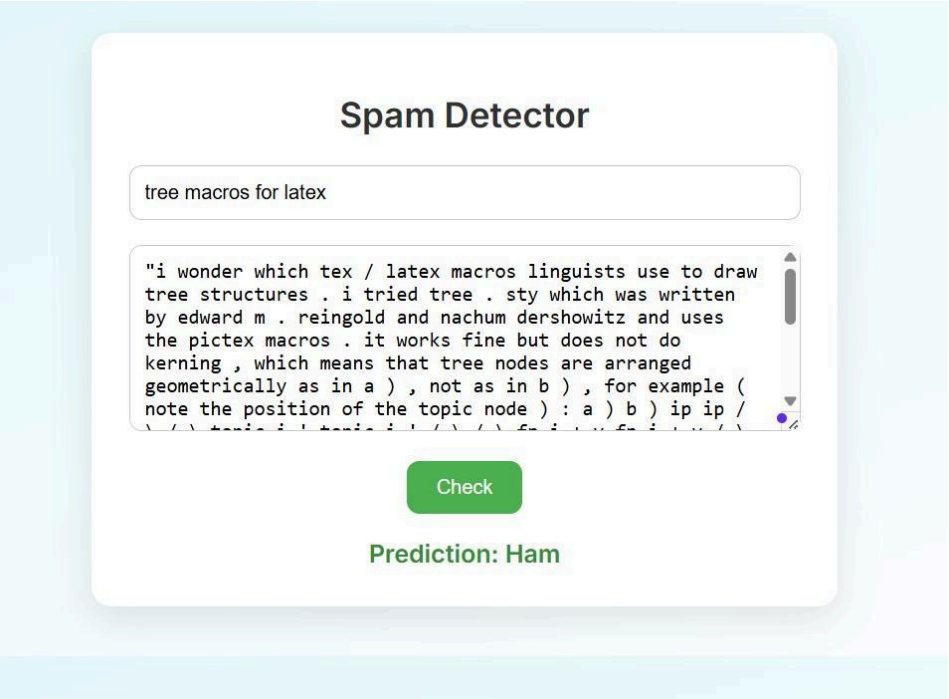

Fig. 4.7.2 Spam Detector Interface Showing Prediction for a Ham Email

CHAPTER 5 CONCLUSION AND FURTHER WORK

5.1 CONCLUSION

This work introduced a robust method for detecting spam by merging contextual and statistical text features with deep learning techniques and nature-inspired optimization methods. The process utilized BERT-based tokenization to effectively capture the semantic structure of the text, while also incorporating handcrafted linguistic features that offered clear insights into its superficial characteristics. These combined features were refined through Grey Wolf Optimization (GWO), which minimized dimensionality and ensured only the most pertinent inputs were used for classification tasks. A Gated Recurrent Unit (GRU) model was chosen as the main classifier due to its efficiency in handling sequential data. Additionally, Particle Swarm Optimization (PSO) was employed to fine-tune essential hyper parameters, resulting in improved accuracy and quicker convergence rates. The entire system was implemented as a web-based API via Flask, allowing for real-time spam detection through an intuitive user interface.

5.2 FUTURE WORK

While the current system is already quite accurate in identifying spam emails, many possibilities for improvement are still feasible. One promising avenue for improvement is exploring higher performing pre-trained language models like GPT or T5 that may allow the model more contextual comprehension capabilities and generative possibilities. These pre-trained models could substitute BERT or supplement BERT for better semantic processing of more subtle email content, especially for spam messages that simply disguise as legitimate text. Furthermore, in order to support deployment at scale, potential optimizations could include better real time processing by improving pre-processing pipelines, minimizing model inference time, and using lighter alternatives such as DistilBERT or quantized GRU models.

CHAPTER 6 REFERENCES

[1] **Arif MH**, **Li J**, **Iqbal M**, **Liu K** (2018) Sentiment analysis and spam detection in short informal text using learning classifier systems. *Soft Computing*, 22, 7281–7291.

[2] **Dada EG**, **Bassi JS**, **Chiroma H**, **Adetunmbi AO**, **Ajibuwa OE** (2019) Machine learning for email spam filtering: review, approaches and open research problems. *Heliyon*, 5(6).

[3] **Gu Q**, **Li Z**, **Han J** (2012) Generalized fisher score for feature selection. *arXiv preprint*, arXiv:1202.3725.

[4] **He X**, **Cai D**, **Niyogi P** (2005) Laplacian score for feature selection. *Proceedings of Advances in Neural Information Processing Systems*.

[5] **Huang J**, **Cai Y**, **Xu X** (2007) A hybrid genetic algorithm for feature selection wrapper based on mutual information. *Pattern Recognition Letters*, 28(13), 1825–1844.

[6] **Kira K**, **Rendell LA** (1992) A practical approach to feature selection. *In Machine Learning Proceedings 1992*, pp. 249–256. *Morgan Kaufmann*.

[7] **Koutroumbas K**, **Theodoridis S** (2008) *Pattern Recognition*. *Academic Press* (USA).

[8] **Luo H**, **Fang B**, **Yun X** (2006) A counting-based method for massive spam mail classification. *Information Security Practice and Experience: Second International Conference, ISPEC 2006, Hangzhou, China, April 11–14, 2006. Proceedings 2*, pp. 45–56. *Springer Berlin Heidelberg*.

[9] **Raileanu LE**, **Stoffel K** (2004) Theoretical comparison between the Gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41, 77–93.

[10] **Uesugi T** (2016) Toxic epidemics: Agent Orange sickness in Vietnam and the United States. *Medical Anthropology*, 35(6), 464–476.

CHAPTER 7 APPENDIX

7.1 Sample Source Code

Importing required modules

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from pyswarm import pso
from transformers import BertTokenizer
import matplotlib.pyplot as plt
```

Loading Dataset

```
from imblearn.over_sampling import SMOTE

df = pd.read_csv("messages.csv")
y = df['label']
X = df.drop(columns=['label'])
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

Data Pre-processing

```
def clean_text(text):
    text = text.lower()
    text = re.sub(f"[{string.punctuation}]", "", text)
    tokens = text.split()
    stop_words = set(stopwords.words('english'))
    text = " ".join([word for word in tokens if word not in stop_words and len(word) > 2])
    return text

data['cleaned_message'] = data['message'].apply(clean_text)
```

```
df['text'] = (df['subject'].astype(str) + " " + df['message'].astype(str)).fillna('')
labels = df['label'].values
```

Feature Extraction

```
from transformers import TFBertModel
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
df['text'] = (df['subject'].astype(str) + " " + df['message'].astype(str)).fillna('')
inputs = tokenizer(df['text'].tolist(), padding=True, truncation=True, max_length=512, return_tensors='tf')
labels = df['label'].values
bert_model = TFBertModel.from_pretrained('bert-base-uncased')
outputs = bert_model(inputs)
embeddings = outputs.last_hidden_state
```

```
df['subject_length'] = df['subject'].apply(lambda x: len(str(x)))
df['message_length'] = df['message'].apply(lambda x: len(str(x)))
df['num_special_chars'] = df['text'].apply(lambda x: sum(c in "!@#$%^&*()_+" for c in str(x)))
df['num_digits'] = df['text'].apply(lambda x: sum(c.isdigit() for c in str(x)))
df['num_uppercase'] = df['text'].apply(lambda x: sum(c.isupper() for c in str(x)))
df['avg_word_length'] = df['text'].apply(lambda x: np.mean([len(w) for w in x.split()]) if x.split() else 0)
df['num_words'] = df['text'].apply(lambda x: len(x.split()))
df['has_link'] = df['text'].apply(lambda x: int("http" in x or "www" in x))
df['has_reply_or_forward'] = df['subject'].apply(lambda x: int("Re:" in str(x) or "Fwd:" in str(x)))
features = ['subject_length', 'message_length', 'num_special_chars', 'num_digits',
            'num_uppercase', 'avg_word_length', 'num_words', 'has_link', 'has_reply_or_forward']
```

```
X = np.hstack((tokenized_texts, df[features].values))
y = labels
```

Feature Selection

```
def fitness_function(feature_mask, X, y):
    selected_indices = np.where(feature_mask == 1)[0]
    if len(selected_indices) == 0:
        return 0
    X_selected = X[:, selected_indices]
    X_train_fs, X_test_fs, y_train_fs, y_test_fs = train_test_split(X_selected, y, test_size=0.2, random_state=42)
    clf = tf.keras.Sequential([
        layers.Dense(10, activation='relu', input_shape=(len(selected_indices),)),
        layers.Dense(1, activation='sigmoid')
    ])
    clf.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    clf.fit(X_train_fs, y_train_fs, epochs=2, verbose=0)
    loss, accuracy = clf.evaluate(X_test_fs, y_test_fs, verbose=0)
    return accuracy
```

```
def gwo_feature_selection(X, y, num_wolves=5, max_iter=5):
    num_features = X.shape[1]
    wolves = np.random.randint(0, 2, (num_wolves, num_features))
    alpha_score = beta_score = delta_score = -1
    alpha_pos = beta_pos = delta_pos = np.zeros(num_features)
    for iter in range(max_iter):
        for i in range(num_wolves):
            fitness = fitness_function(wolves[i], X, y)
            if fitness > alpha_score:
                delta_score, delta_pos = beta_score, beta_pos.copy()
                beta_score, beta_pos = alpha_score, alpha_pos.copy()
                alpha_score, alpha_pos = fitness, wolves[i].copy()
            elif fitness > beta_score:
                delta_score, delta_pos = beta_score, beta_pos.copy()
                beta_score, beta_pos = fitness, wolves[i].copy()
            elif fitness > delta_score:
                delta_score, delta_pos = fitness, wolves[i].copy()
        a = 2 - iter * (2 / max_iter)
        for i in range(num_wolves):
            for j in range(num_features):
                r1, r2 = np.random.rand(), np.random.rand()
                A1 = 2 * a * r1 - a
                C1 = 2 * r2
                D_alpha = abs(C1 * alpha_pos[j] - wolves[i][j])
                X1 = alpha_pos[j] - A1 * D_alpha
                r1, r2 = np.random.rand(), np.random.rand()
                A2 = 2 * a * r1 - a
                C2 = 2 * r2
                D_beta = abs(C2 * beta_pos[j] - wolves[i][j])
                X2 = beta_pos[j] - A2 * D_beta
                r1, r2 = np.random.rand(), np.random.rand()
                A3 = 2 * a * r1 - a
                C3 = 2 * r2
                D_delta = abs(C3 * delta_pos[j] - wolves[i][j])
                X3 = delta_pos[j] - A3 * D_delta
                new_pos = (X1 + X2 + X3) / 3
                wolves[i][j] = 1 if new_pos > 0.5 else 0
    selected_indices = np.where(alpha_pos == 1)[0]
    X_selected = X[:, selected_indices]
    return X_selected, selected_indices
```

Classification Models

Random Forest

```
rf_model = RandomForestClassifier(
    n_estimators=300, max_depth=15, min_samples_split=5, class_weight="balanced", random_state=42)
rf_model.fit(X_train_vec, y_train)
y_pred = rf_model.predict(X_test_vec)
```

Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB
nb_model = MultinomialNB(alpha=0.3)
nb_model.fit(X_train_vec, y_train)
y_pred = nb_model.predict(X_test_vec)
```

CNN

```
def build_cnn_model(input_dim):
    input_ids = layers.Input(shape=(input_dim,), dtype=tf.int32, name='input_ids')

    embedding = layers.Embedding(input_dim=tokenizer.vocab_size, output_dim=768, input_length=input_dim)(input_ids)

    x = layers.Conv1D(filters=128, kernel_size=3, activation='relu', padding='same')(embedding)
    x = layers.MaxPooling1D(pool_size=2)(x)

    x = layers.Conv1D(filters=128, kernel_size=3, activation='relu', padding='same')(x)
    x = layers.MaxPooling1D(pool_size=2)(x)

    x = layers.Flatten()(x)

    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dropout(0.3)(x)

    output = layers.Dense(1, activation='sigmoid')(x)

    model = models.Model(inputs=input_ids, outputs=output)

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    return model

model = build_cnn_model(len(selected_features))
```

LSTM

```
def build_lstm_model(input_dim):
    input_ids = layers.Input(shape=(input_dim,), dtype=tf.int32, name='input_ids')

    embedding = layers.Embedding(input_dim=tokenizer.vocab_size, output_dim=768, input_length=input_dim)(input_ids)


    x = layers.LSTM(128, return_sequences=True)(embedding)
    x = layers.LSTM(128)(x)

    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dropout(0.3)(x)


    output = layers.Dense(1, activation='sigmoid')(x)

    model = models.Model(inputs=input_ids, outputs=output)


    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    return model

model = build_lstm_model(len(selected_features))
model.summary()
```

BiLSTM

```
def build_model_with_selected_features(num_features):
    input_ids_layer = layers.Input(shape=(num_features,), dtype=tf.int32, name='input_ids')
    bert_model = tf.keras.Sequential([
        layers.InputLayer(input_shape=(num_features,), dtype=tf.int32),
        layers.Embedding(input_dim=tokenizer.vocab_size, output_dim=768, input_length=num_features),
        layers.Bidirectional(layers.LSTM(128, return_sequences=True))
    ])
    x = layers.Bidirectional(layers.LSTM(128))(bert_model(input_ids_layer))
    output = layers.Dense(1, activation='sigmoid')(x)
    model = models.Model(inputs=input_ids_layer, outputs=output)
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

model = build_model_with_selected_features(len(selected_indices))
```

Model Training

```
history = model.fit(
    train_inputs_selected,
    train_labels,
    validation_data=(val_inputs_selected, val_labels),
    epochs=5,
    batch_size=32
)
```

GRU

```
def build_gru_model(units, learning_rate):
    model = models.Sequential([
        layers.Embedding(input_dim=30522, output_dim=128, input_length=max_len + len(features)),
        layers.GRU(int(units), return_sequences=True),
        layers.GRU(int(units)),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(1, activation='sigmoid')
    ])
```

```
class Attention(tf.keras.layers.Layer):
    def __init__(self):
        super(Attention, self).__init__()

    def build(self, input_shape):
        self.W = self.add_weight(name="att_weight", shape=(input_shape[-1], 1),
                                 initializer="normal")
        self.b = self.add_weight(name="att_bias", shape=(input_shape[1], 1),
                                 initializer="zeros")
        super(Attention, self).build(input_shape)

    def call(self, x):
        e = tf.keras.backend.tanh(tf.keras.backend.dot(x, self.W) + self.b)
        a = tf.keras.backend.softmax(e, axis=1)
        output = x * a
        return tf.keras.backend.sum(output, axis=1)
```

PSO

```
def objective_function(params):
    units, learning_rate = params
    model = build_gru_model(units, learning_rate)
    history = model.fit(train_x, train_y, epochs=1, batch_size=16, validation_data=(test_x, test_y), verbose=0)
    val_acc = max(history.history['val_accuracy'])
    return -val_acc

lb = [32, 0.0005]
ub = [128, 0.005]
best_params, _ = pso(objective_function, lb, ub, swarmsize=8, maxiter=5)
best_units, best_lr = best_params
```

GRU with Tuned Parameters

```
final_model = build_gru_model(best_units, best_lr)
final_model.fit(train_x, train_y, epochs=5, batch_size=32, validation_data=(test_x, test_y))
```

Visualisation of Selected Features

```
import numpy as np
import matplotlib.pyplot as plt

feature_counts = [len(np.nonzero(sol > 0.5)[0]) for sol in solutions]

plt.figure(figsize=(7, 5))
plt.hist(feature_counts, bins=range(min(feature_counts), max(feature_counts)+2, 2),
         color='skyblue', edgecolor='black')

plt.axvline(len(selected_features), color='red', linestyle='dashed', linewidth=2)

plt.title("Distribution of Selected Features per Wolf")
plt.xlabel("Number of Selected Features")
plt.ylabel("Count")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

print(f"Selected {len(selected_features)} important features using GWO.")
```

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt

spam_text = " ".join(df[df['label'] == 1]['message'].astype(str))
ham_text = " ".join(df[df['label'] == 0]['message'].astype(str))
spam_wc = WordCloud(width=800, height=400, background_color='white', colormap='Reds').generate(spam_text)
ham_wc = WordCloud(width=800, height=400, background_color='white', colormap='Blues').generate(ham_text)

plt.figure(figsize=(16, 8))

plt.subplot(1, 2, 1)
plt.imshow(spam_wc, interpolation='bilinear')
plt.axis('off')
plt.title('Spam Word Cloud')

plt.subplot(1, 2, 2)
plt.imshow(ham_wc, interpolation='bilinear')
plt.axis('off')
plt.title('Ham Word Cloud')

plt.tight_layout()
plt.show()
```



```
import matplotlib.pyplot as plt
import seaborn as sns

df['message_length'] = df['message'].apply(len)
plt.figure(figsize=(8, 6))
sns.histplot(data=df, x='message_length', hue='label', bins=30, kde=True, palette='Set2')
plt.title('Distribution of Message Lengths (Spam vs Ham)')
plt.xlabel('Message Length')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```