3rd Semester Mini Project Report on

Sentimental Analysis using wrapper and filter-based Bio-inspired Algorithms

Submitted in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

Submitted by:

Ishita Tomar **2023391**

Under the Guidance of

Mr. Rehan

Assistant Professor

Department of Computer Science and Engineering
Graphic Era (Deemed to be University)
Dehradun, Uttarakhand

2024-25

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the project report entitled **"Sentimental Analysis using wrapper and filter-based Bio-inspired Algorithms"** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering in the Department of Computer Science and Engineering of the Graphic Era (Deemed to be University), Dehradun shall be carried out by the undersigned under the supervision of **Mr. Rehan, Assistant Professor**, Department of Computer Science and Engineering, Graphic Era (Deemed to be University), Dehradun.

Ishita Tomar 2023391 signature

The above mentioned student shall be working under the supervision of the undersigned on the **"Sentimental Analysis using wrapper and filter-based Bio-inspired Algorithms"**

Supervisor **Head of the Department**

Examination

Name of the Examiners: **Signature with Date**

1.

2.

Table of Contents

Chapter 1

Introduction and Problem Statement

1.1 Introduction

Sentimental analysis is the process of extracting conclusions from the opinions of the people on a particular topic. People around the world are connected to each other through an enormous network of social networking sites such as Facebook, Instagram, twitter and some more [1].

Twitter with a wide range of users (more than 300 million users) and huge data is used by most of the researchers to get informative data to process the public opinion and collect the required feedback. Users communicate through their tweets having abbreviations, slangs, sarcasm, emoticons and much more.

Depression is a mental common illness faced worldwide by people of all ages. It is a type of illness which causes lack of interest in various activities, be it physical, emotional, social or anything leading to reducing one's work efficiency [2].

Machine Learning (ML) methods such as Logistic Regression and Naive Bayes algorithms are used for classification.

Here, we have performed sentimental analysis on a dataset acquired from Kaggle which contains tweets expressing the depression level of the people (0 for low and 1 for high) with the help of some wrapper and filter-based bio-inspired algorithms viz. Particle Swarm Optimization (PSO) [3], Grey Wolf Optimization (GWO) [1] and Naive Bayes Classification [5] and Logistic Regression [6].

These models were used to analyze the public opinions by comparing their accuracy to each other along with scatter plot.

1.2 Problem Statement

Social Media is the hub of public opinion where millions of opinions are shared every few minutes. As more and more people are expressing their thoughts on social media which can be both neutral and polarizing. The problem is to plot public sentiments on the relative topic and monitor it.

Objective of the project is to extract tweets from twitter and extract sentiments out of them and develop a sentimental analysis system using Bio inspired Algorithms using PSO and GWO and various other algorithms to optimize the parameters of various machine learning classifier. The system will analyse text and data and classify it a positive, negative or neutral based on its sentiment.

Developing a sentimental analysis model using Bio inspired Algorithms, utilizing some machine learning algorithms as the base model and optimizing its parameters using PSO and GWO and other bio inspired algorithms with the help of python as the programming language.

Through this we will be able to analyse sentiments of various twitter handles to get what emotions they are inciting and what sentiments they are pointing to, also increasing the accuracy.

Chapter 2

Methodology

The methodologies used are as follows:

2.1 Natural Language Processing (NLP)

Term Frequency-Inverse Document Frequency (TF-IDF) was used for converting raw data to numerical form so that the machine learning model can understand it.

2.2 Supervised Learning

Machine learning model-Logistic regression was used to predict whether the message indicated depression or no depression.

Naive Bayes is an algorithm for text classification based on Bayes' theorem, here it is used to classify the sentiment of tweets as depression (1) and or no depression (0).

2.3 Swarm Intelligence

Particle Swarm optimisation (PSO) is an optimisation technique inspired by the flocking of birds, which searches for optimal feature weights for logistic regression and hence, resulting in the best accuracy.

Grey Wolf Optimization (GWO) is an optimisation technique inspired by the pack of grey wolves and is used for feature selection for machine learning model (here, logistic regression).

2.3 Flowcharts for depicting methodologies

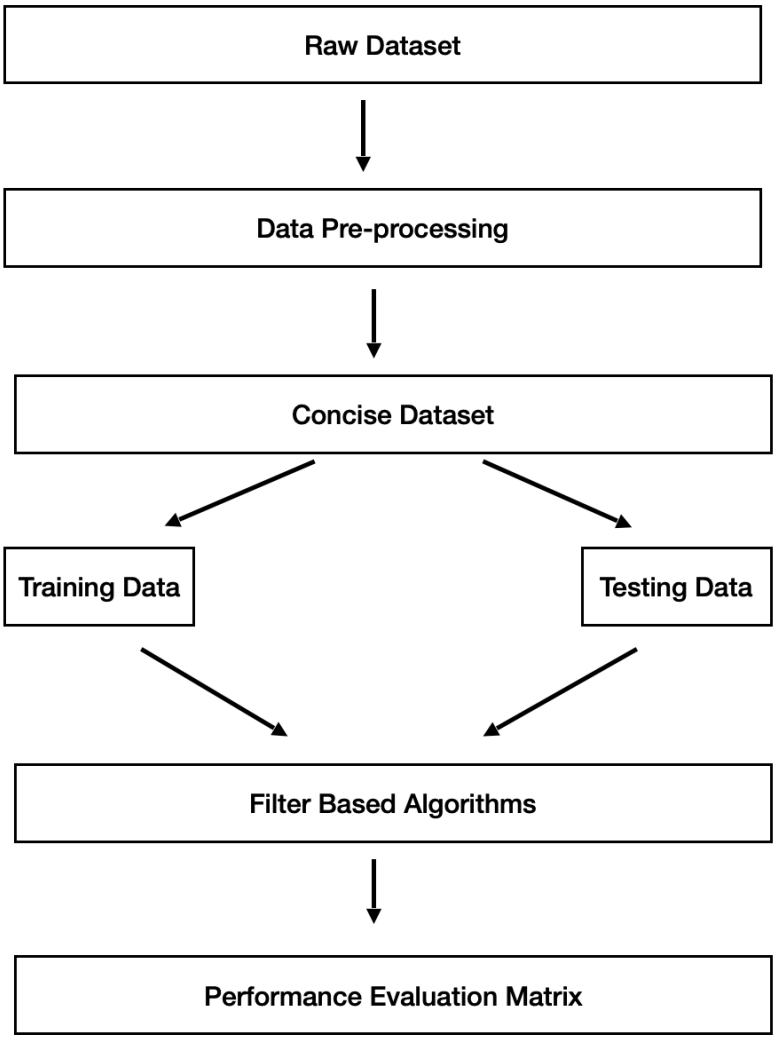

Fig 2.1: Methodology

The figure 2.1 given above depicts the methodology for the project, here we have processed a raw dataset to make it concise dataset which then is split into training and testing data. With the help of filter based PSO and GWO algorithm, we have calculated the accuracies for analyzing sentiments.

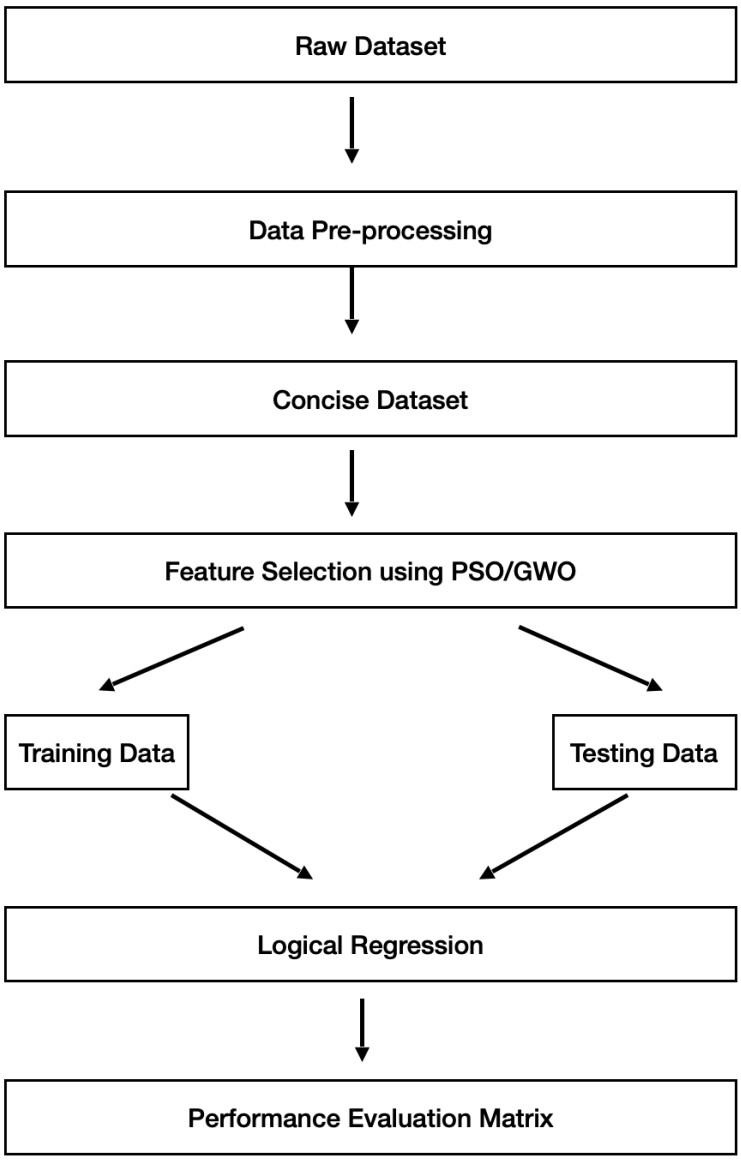

Fig 2.2: Methodology using logistic regression

The figure 2.2 given above depicts another methodology for the project; here we have processed a raw dataset to concise dataset. With the help of PSO and GWO we have selected the feature (wrapper based implementation of PSO and GWO), which then is split into training and testing data. Applying logistic regression, as machine learning algorithm, we have calculated the accuracy for analyzing the sentiments.

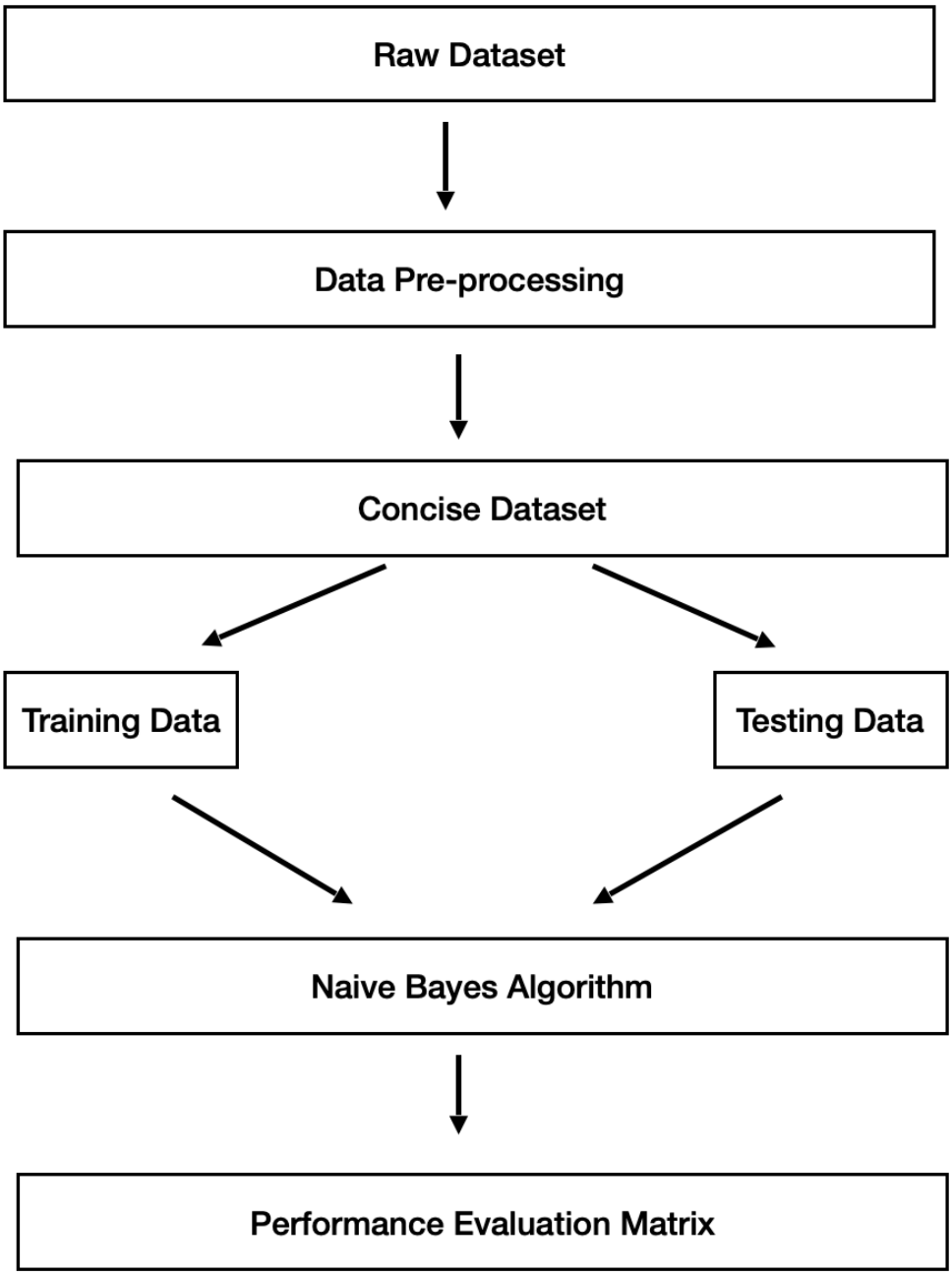

Fig 2.3: Methodology using Naive Bayes Algorithm

The figure 2.3 given above depicts the another methodology for the project; here we have processed a raw dataset to concise dataset, which then is split into training and testing data. Applying Naïve Bayes, as machine learning algorithm, we have calculated the accuracy for analyzing the sentiments. This implementation is without feature selection.

Chapter 3

Project Work Carried Out

3.1 Language Used

We used Python for the analysis of the dataset.

3.2 Dataset

We acquired a dataset (sentiment_tweets3) from Kaggle which was based on twitter tweets and were labelled as per on depression (1 for high and 0 for low). The dataset had more than 10,000 tweets labelled as “message to examine” which were classified into two categories (depression or no depression)[4].

3.3 Feature Extraction Technique

We used TF-IDF as an vectorizer technique. We imported ‘Tfidfvectorizer’ from ‘sklearn.model_selection’ that converts raw data into numerical features.

3.4 Model

We used logistic regression model of machine learning. We imported ‘LogisticRegression’ model from the ‘sklearn.linear_model’. We performed classification on the pre-processed and vectorized data. It is within the objective function of PSO and GWO.

3.5 Algorithms

3.5.1 PSO

It is inspired by flocking nature of birds. The birds always fly together following some kind of pattern and abiding by some rules to look for food. The continuously adjust their speed and positions in order to achieve their goal of finding food. The bird closest to food is followed by the other birds. Similarly, the particles here act as the birds and the particle’s fitness value is calculated by a fitness function and velocity. It has two variables: pBest for personal best and gBest for global variable that indicates the particle closest to food. The particle’s velocity is calculated as [3]:

$v_{(t+1)}=v_t+w_1*rand\ *\ (pBest-p)+w_2*rand*\ (gBest-p)$ (1)

here,

- v_{t+1} is the particle's velocity at time t+1
- v_t is particle's velocity at time t
- w_1 is weight of local information
- w_2 is weight of global information
- p is particle's position
- rand is some random value
- pBest is personal best
- gBest is global best

The particle's position is calculated as:

$x_{t+1}=x_t+v_{t+1}$ (2)

here,

- x_{t+1} is particle's position at t+1
- x_t is particle's position at time t

Table 3.1: Pseudo code of PSO

Initialize a swarm of particles:

- Initialize each particle's position (x) and velocity (v) randomly within the search space.
- Evaluate the fitness of each particle using the objective function.
- Set the personal best position (pBest) of each particle to its initial position.
- Set the global best position (gBest) to the best pBest among all particles.

While termination criteria not met:

- Update the velocity of each particle: $v_{(t+1)}=v_t+w_1*rand\ *\ (pBest-p)+w_2*rand*\ (gBest-p)$
- Update the position of each particle: $x_{t+1}=x_t+v_{t+1}$
- Evaluate the fitness of each particle.
- Update the personal best position (pBest) of each particle if necessary. -
- Update the global best position (gBest) if necessary.

3.5.1.1 Implementation:

Fig 3.1: PSO Algorithm

```
[10] !pip install pyswarms
     import numpy as np
     import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.feature_extraction.text import TfidfVectorizer
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score
     import pyswarms as ps

     # Load the dataset
     file_path = 'sentiment_tweets3.csv'
     data = pd.read_csv(file_path)

     # Preprocess the data
     X_raw = data['message to examine']  # Text messages
     y = data['label (depression result)']  # Labels

     # Vectorize the text data using TF-IDF
     vectorizer = TfidfVectorizer(max_features=1000)  # Limit to 1000 features for simplicity
     X = vectorizer.fit_transform(X_raw).toarray()

     # Split the dataset into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

     # Initialize lists to store history of weights and accuracies
     weights_history = []
     accuracies_history = []
```

We imported ‘ps’ from ‘pyswarms’ for carrying out the PSO algorithm. Then we loaded the dataset and preprocessed the data and converted it into numerals using tf-idf. The data was split into training and testing data.

```
# Define the PSO objective function with history tracking
def objective_function_with_history(weights):
    # Reshape weights to (n_particles, 1, n_features)
    weights = weights.reshape(weights.shape[0], 1, weights.shape[1])

    # Initialize an array to store accuracy for each particle
    accuracies = []

    # Track each particle's weights and corresponding accuracy
    for particle_weights in weights:
        # Apply weights to features
        X_train_weighted = X_train * particle_weights
        X_test_weighted = X_test * particle_weights

        # Train a Logistic Regression model
        model = LogisticRegression(max_iter=500)
        model.fit(X_train_weighted, y_train)

        # Predict and calculate accuracy
        y_pred = model.predict(X_test_weighted)
        accuracy = accuracy_score(y_test, y_pred)
        accuracies.append(accuracy)

    # Store the history of weights and accuracies
    weights_history.append(weights.flatten())  # Flatten to 1D array for simplicity
    accuracies_history.append(np.mean(accuracies))  # Store the average accuracy for each iteration

    # Return negative accuracies for minimization
    return -np.array(accuracies)
```

Fig 3.2: Objective function of PSO Algorithm with logistic regression

Here, the objective function of PSO is defined where an array accuracy is created to store accuracy of each particle. Logistic Regression Model is trained for prediction and calculation of the accuracy.

```
# PSO Optimization with history tracking
def optimize_with_pso_with_history():
    # Define bounds for weights (0 to 1 for each feature)
    dimensions = X_train.shape[1]  # Number of features
    bounds = (np.zeros(dimensions), np.ones(dimensions))

    # Define modified PSO options
    options = {
        'c1': 0.5,  # Cognitive component
        'c2': 0.3,  # Social component
        'w': 0.9,   # Inertia weight
    }

    # Define the optimizer
    optimizer = ps.single.GlobalBestPSO(
        n_particles=30,  # Number of particles
        dimensions=dimensions,  # Number of features
        options=options,  # PSO options
        bounds=bounds  # Bounds for the weights
    )

    # Run optimization with history tracking
    best_cost, best_weights = optimizer.optimize(objective_function_with_history, iters=50)

    return best_weights, -best_cost  # Return weights and accuracy

# Run PSO with history tracking
best_weights, best_accuracy = optimize_with_pso_with_history()

# Output the results
print("Best Accuracy Score:", best_accuracy)
```

Fig 3.3: Function of PSO Algorithm

Here, we are tracking the history of weights and accuracies, analyzing the optimization process for the results.The change in options would result in change of best positions. The results are printed in terms of best position and the accuracy.

3.5.2 GWO

This algorithm is based on the leadership hierarchy of grey wolves and their hunting habits. Four distinct grey wolves' alpha, beta, delta and omega (in hierarchical order) are employed to implement their hierarchy. There are three main steps in this algorithm, of which one is hierarchy management, followed by encircling prey and finally hunting. Grey wolves surround the prey and mathematically it is represented as[1]:

$$A = 2*\alpha*r_1 - a \quad (3)$$

$$C = 2*r_2 \quad (4)$$

$$X_{t+1} = X_t - A*D \quad (5)$$

here,

- A and C are coefficient vectors.
- a linearly decreases from 2 to 0 over the iterations.
- r1 and r2 are random numbers between 0 and 1.
- D is the distance between the prey and the wolf.
- X_t is the current position of the wolf.
- X_{t+1} is the updated position of the wolf.

The hunting process involves the alpha, beta, and delta wolves guiding the omega wolves towards the prey. The omega wolves change their position as per the position of alpha, beta and delta.

$$D\alpha = |C*X\alpha - X(t)| \quad (6)$$

$$D\beta = |C*X\beta - X(t)| \quad (7)$$

$$D\delta = |C*X\delta - X(t)| \quad (8)$$

$$X1 = X\alpha - A*D\alpha \quad (9)$$

$$X2 = X\beta - A*D\beta \quad (10)$$

$$X3 = X\delta - A*D\delta \quad (11)$$

$$X(t+1) = (X1 + X2 + X3) / 3 \quad (12)$$

Xα, Xβ and Xδ are the positions of the alpha, beta, and delta wolves, respectively. Dα, Dβ and Dδ are the distances between prey and alpha, beta and delta respectively[1].

Table 3.2: Pseudo code of GWO

Initialize alpha, beta, and delta wolves randomly.
Initialize omega wolves randomly.
While termination condition not met:
  Update alpha, beta, and delta wolves:
  - Calculate fitness of each wolf.
  - Sort wolves based on fitness.
  - Assign the top three wolves as alpha, beta, and delta.

  Update omega wolves:
    For each omega wolf:
  - Calculate the distance between the omega wolf and alpha, beta, and delta wolves.
  - Update the position of the omega wolf : $X_{t+1} = X_t - A*D$

3.5.2.1. Implementation:

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import re

# Step 1: Data Collection
data = pd.read_csv('sentiment_tweets3.csv')
tweets = data['message to examine']
y = data['label (depression result)']

# Step 2: Preprocessing
def preprocess_tweet(tweet):
    tweet = tweet.lower()  # Convert to lowercase
    tweet = re.sub(r'http\S+|www\S+|https\S+', '', tweet, flags=re.MULTILINE)  # Remove URLs
    tweet = re.sub(r'@\w+', '', tweet)  # Remove mentions
    tweet = re.sub(r'#', '', tweet)  # Remove hashtags
    tweet = re.sub(r'[^a-zA-Z\s]', '', tweet)  # Remove special characters
    return tweet

cleaned_tweets = tweets.apply(preprocess_tweet)

# Step 3: Feature Extraction
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(cleaned_tweets)
```

Fig3.4: GWO Algorithm

We loaded the dataset and preprocessed the data removing any special characters, URLs, mentions and hashtags, also converting it into lower case; converted it into numerals using tf-idf.

```
# Step 4: GWO Algorithm Implementation
class GWO:
    def __init__(self, objective_function, num_wolves, max_iter):
        self.objective_function = objective_function
        self.num_wolves = num_wolves
        self.max_iter = max_iter
        self.alpha_pos = None
        self.alpha_score = float("inf")
        self.wolves_pos = np.random.rand(num_wolves, X.shape[1])  # Random positions
        self.error_history = []  # To store error history for plotting

    def optimize(self):
        for iter in range(self.max_iter):
            for i in range(self.num_wolves):
                score = self.objective_function(self.wolves_pos[i])
                if score < self.alpha_score:
                    self.alpha_score = score
                    self.alpha_pos = self.wolves_pos[i]

            # Update positions of wolves
            a = 2 - iter * (2 / self.max_iter)  # Decrease a from 2 to 0
            for i in range(self.num_wolves):
                r1, r2 = np.random.rand(2)
                A = 2 * a * r1 - a
                C = 2 * r2
                D = np.abs(C * self.alpha_pos - self.wolves_pos[i])
                self.wolves_pos[i] = self.alpha_pos - A * D

            # Track the error (1 - accuracy) for the current iteration
            self.error_history.append(self.alpha_score)
        return self.alpha_pos, self.alpha_score
```

Fig 3.5: GWO Algorithm Implementation with logistic regression

Here, we wrote the code for GWO algorithm by creating a class and defining two functions in it.

```
def objective_function(wolf_position):
    # Here you can define how to evaluate the position
    # For example, using a simple logistic regression model
    model = LogisticRegression()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    model.fit(X_train, y_train)
    accuracy = model.score(X_test, y_test)  # Calculate accuracy
    return 1 - accuracy  # Minimize error (1 - accuracy)

# Step 5: Evaluation
gwo = GWO(objective_function, num_wolves=10, max_iter=100)
best_position, best_score = gwo.optimize()

# Print the best position and the corresponding accuracy
best_accuracy = 1 - best_score  # Convert error to accuracy
print(f'Best Position: {best_position}')
print(f'Best Score (Error): {best_score}')
print(f'Best Accuracy: {best_accuracy}')
```

Fig 3.6: Objective Function of GWO

We defined an objective function and later called the functions defined in class for evaluation. Then the accuracy was printed along with the best position and best score.

A scatter plot was plotted for comparing the results of PSO and GWO for accuracies.

3.4.3 Naive Bayes Algorithms

It is a classification algorithm based on Bayes' Theorem

Bayes' Theorem:

$$Pr(B|A)=(Pr(B)*Pr(A|B)) / Pr(A) \quad (13)$$

here,

- Pr(A|B): Probability of event A occurring, given that event B has occurred.
- Pr(B|A): Probability of event B occurring, given that event A has occurred.
- Pr(A): Prior probability of event A.
- Pr(B): Prior probability of event B.

Table 3.3: Pseudo code of Naive Bayes Algorithm

Training: • Calculate the prior probability of each class • For each feature and class, calculate the conditional probability Prediction: • For a new data point X, calculate the posterior probability for each class. • Select the class with the highest posterior probability as the predicted

3.5.3.1. Implementation:

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.decomposition import PCA  # Import PCA for dimensionality reduction
import re
from sklearn.metrics import classification_report, accuracy_score, precision_score, recall_score, f1_score
import matplotlib.pyplot as plt

# Step 1: Data Collection
data = pd.read_csv('sentiment_tweets3.csv')
tweets = data['message to examine']
y = data['label (depression result)']  # Assuming 'label' column contains sentiment labels

# Step 2: Preprocessing
def preprocess_tweet(tweet):
    tweet = tweet.lower()
    tweet = re.sub(r'http\S+|www\S+|https\S+', '', tweet, flags=re.MULTILINE)  # Remove URLs
    tweet = re.sub(r'@\w+', '', tweet)  # Remove mentions
    tweet = re.sub(r'#', '', tweet)  # Remove hashtags
    tweet = re.sub(r'[^a-zA-Z\s]', '', tweet)  # Remove special characters
    return tweet

cleaned_tweets = tweets.apply(preprocess_tweet)

# Step 3: Feature Extraction
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(cleaned_tweets)
```

Fig 3.7: Naive Bayes' Algorithm

We imported 'MultinomialNB' from 'sklearn.naive_bayes'.We loaded the dataset, processed it to remove URLs, mentions, hashtags and special characters also converted it to lower case. Extracted the features using tf-idf.

```
# Step 5: Evaluation
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Get the classification report
report = classification_report(y_test, y_pred, output_dict=True)

# Print evaluation metrics including accuracy, precision, recall, and F1-score
print(f'Accuracy: {accuracy_score(y_test, y_pred):.4f}')


print('\nClassification Report:\n')
# Print class-wise precision, recall, F1-score, and support
for class_label, metrics in report.items():
    if class_label not in ['accuracy', 'macro avg', 'weighted avg']:
        print(f'Class: {class_label}')
        print(f'Precision: {metrics["precision"]:.4f}')
        print(f'Recall: {metrics["recall"]:.4f}')
        print(f'F1-Score: {metrics["f1-score"]:.4f}')
        print(f'Support: {metrics["support"]}\n')
```

Fig 3.8: Implementation of Naive Bayes Model

Here, we used the Naive Bayes Model for the purpose of classification and also printed the classification report along with the accuracy of this model.

Chapter 4

Results and Discussion

4.1 Output of the PSO

We found the accuracy and the best positions by using three different values for the hyper-parameters and the classification is been performed using LR.

CASE I:

```
# Define modified PSO options
options = {
    'c1': 0.5,  # Cognitive component
    'c2': 0.3,  # Social component
    'w': 0.9,   # Inertia weight
}
```

Fig 4.1: Case I of PSO hyper-parameters

```
| best cost: -0.9912748424624334, best pos: [7.26211013e-02 9.47820273e-01 2.97791830e-01 4.32223375e-01
 6.92486612e-01 6.01371870e-01 2.32227037e-01 4.89459746e-01
 2.32615023e-01 4.13173314e-01 4.54636423e-01 5.87818927e-01
 9.26605155e-01 6.11719264e-01 3.11069371e-01 4.26555288e-01
 2.96218497e-01 7.49257479e-01 4.35412061e-01 1.05306243e-01
 7.14100601e-01 3.25938269e-01 3.69875887e-01 4.69298742e-01
 5.53833035e-01 7.58301385e-01 5.30162168e-01 8.24844589e-01
 7.41890125e-01 5.38138658e-01 1.42572494e-01 4.44869499e-01
 1.80872038e-01 3.57412077e-01 9.73322307e-02 6.75754462e-01
 2.11396348e-01 4.62597332e-02 4.85147744e-01 7.25802749e-01
 8.43919783e-01 4.13607468e-01 4.96418925e-01 7.83143173e-01
 7.28531273e-01 5.41014662e-01 6.37734691e-01 2.59370065e-01
 8.49045701e-01 3.80937259e-01 4.35276895e-01 4.95477423e-01
 8.69874950e-01 2.17714605e-01 9.44255782e-01 4.85311974e-01
 1.41544128e-01 5.01189114e-01 4.64930287e-01 4.93719677e-01
 2.29384119e-01 8.31976710e-01 3.30399936e-01 9.82777618e-01
 7.50176604e-01 3.81732547e-01 2.42844766e-01 6.01483039e-01
 6.83221811e-01 7.10312747e-01 9.85323882e-02 7.64700204e-01
 8.06020123e-01 6.40605976e-01 6.00108668e-01 4.00296024e-01
 9.34501837e-01 6.33245814e-01 3.28704587e-01 3.83107683e-01
 8.75077405e-01 7.66204968e-01 4.65875529e-01 6.15119231e-01
 5.60381374e-01 8.25351249e-01 3.56040102e-01 8.85987254e-01
 7.54494717e-01 7.62622936e-01 4.90704980e-01 7.26775660e-01
 1.38842689e-01 4.29711650e-01 5.72982823e-01 9.34248356e-01
 7.66346900e-01 2.08223409e-01 7.36391633e-01 2.66973545e-01
 7.25498750e-02 4.78638937e-01 9.26131585e-01 3.06298148e-01
 2.00991796e-01 6.51519205e-01 3.58056441e-01 4.40958407e-01
 1.22090136e-01 4.83739564e-01 3.30798253e-01 3.33165707e-01
 3.64359784e-01 7.87413228e-01 4.90271559e-01 1.35029389e-01
 4.34484096e-01 8.15360509e-01 8.09618170e-01 2.11364218e-01]
Best Accuracy Score: 0.9903053805138148
```

Fig 4.2: Output for case I

Here, the feature selection is done through PSO and the accuracy is calculated using logical regression. These were the output of the best positions along with the accuracy of about 0.9903 or 99.03%.

CASE II:

```
# Define modified PSO options
options = {
    'c1': 0.7,  # Cognitive component
    'c2': 0.8,  # Social component
    'w': 0.9,   # Inertia weight
}
```

Fig 4.3: Case II of PSO hyper-parameters

```
best cost: -0.9903053805138148, best pos: [0.12771419 0.19901822 0.910068
 0.90937276 0.02211143 0.6707271  0.70260848 0.82171889 0.35942537
 0.37894995 0.56708477 0.89542375 0.51878257 0.98691065 0.85159061
 0.78417144 0.93684849 0.2743123  0.91997974 0.25968622 0.09198891
 0.26143865 0.2062842  0.64695029 0.65004552 0.0976109  0.00429641
 0.3696562  0.30850729 0.25815767 0.25222721 0.01185788 0.16876915
 0.17440345 0.78058214 0.77281629 0.85332546 0.76971482 0.81423343
 0.47988468 0.26010305 0.46020625 0.19225617 0.58642884 0.83818493
 0.74143327 0.43779104 0.41069838 0.04438112 0.55833866 0.83908487
 0.02281913 0.71486786 0.11370402 0.79037051 0.86965031 0.65659727
 0.88322397 0.63657211 0.9300842  0.29889973 0.04595306 0.17978197
 0.49323964 0.86499558 0.35160366 0.27862656 0.98526671 0.40526422
 0.89526354 0.9118169  0.18370176 0.38426151 0.51547044 0.16349136
 0.01189706 0.15535608 0.8899669  0.73813589 0.01628964 0.60131654
 0.29136449 0.22077565 0.31857277 0.46539003 0.23511022 0.44704314
 0.39473767 0.27502644 0.83729732 0.12922797 0.48166563 0.42892202
 0.02936819 0.58290017 0.99204578 0.62960771 0.92885389 0.86496874
 0.91438521 0.04257974 0.00948265 0.91627631 0.04155748 0.19807425
 0.47964406 0.9488106  0.70824905 0.50884998 0.2581594  0.27476337
 0.87908553 0.47752257 0.13331438 0.70429023 0.71218724 0.32013229
 0.50851782 0.95473302 0.85573226 0.42068006 0.53495121 0.54280131
 0.0337175  0.90889257 0.7420241  0.50393299 0.99105399 0.21643233
 0.29015337 0.96015856 0.24437021 0.89928264 0.46108497 0.43628536
 0.70668776 0.02870784 0.1015262  0.26573563 0.6773675  0.67870187
 0.80808948 0.30001471 0.60101074 0.29550101]
Best Accuracy Score: 0.9903053805138148
```

Fig 4.4: Output for case II

Here, the feature selection is done through PSO and the accuracy is calculated using logical regression. After implementing PSO (after changing the hyper-parameters) the accuracy acquired was about 0.9903 or 99.03%

CASE III:

```
# Define modified PSO options
options = {
    'c1': 0.1,  # Cognitive component
    'c2': 0.1,  # Social component
    'w': 0.5,   # Inertia weight
}
```

Fig 4.5: Case III of PSO hyper-parameters

```
best cost: -0.9907901114881241, best pos: [0.42593564 0.48137198 0.294979
 0.87470569 0.45759271 0.27468821 0.42635321 0.35863163 0.67676066
 0.4379546  0.79500555 0.83479636 0.6495039  0.46872308 0.83046998
 0.73987508 0.51806934 0.81423398 0.76285422 0.51793502 0.49069046
 0.72387949 0.65853856 0.65399611 0.68559896 0.31513041 0.9128946
 0.83008299 0.7186165  0.23466327 0.57937626 0.73395323 0.56816591
 0.87099912 0.54846132 0.75260625 0.6899452  0.33401726 0.8518662
 0.67067976 0.52538972 0.19878583 0.38057727 0.47344675 0.68047317
 0.52113887 0.54534057 0.35384804 0.54524362 0.59815443 0.64961311
 0.66188104 0.63223962 0.76984815 0.60125212 0.17187053 0.67976253
 0.66607804 0.28979975 0.52849589 0.55441503 0.70212189 0.79289778
 0.61687771 0.65353802 0.63346814 0.58472513 0.44878213 0.73400501
 0.69678629 0.53039929 0.71924805 0.32165091 0.57516793 0.47705526
 0.285077   0.46275568 0.50412091 0.27113846 0.51779243 0.55115325
 0.41646419 0.52957961 0.25869775 0.48350049 0.60608767 0.76277867
 0.4801811  0.47005787 0.78527958 0.67359306 0.56315245 0.69238746
 0.59217159 0.84226119 0.23698726 0.60825324 0.35743863 0.48541809
 0.83616659 0.6672969  0.65243581 0.71957763]
Best Accuracy Score: 0.9907901114881241
```

Fig 4.6: Output for case III

Now the feature selection is done through PSO and the accuracy is calculated using logical regression. These were the output of the best positions when the hyper-parameters were changed again, along with the accuracy of about 99.07%.

Now only PSO is used without any machine learning algorithm which gave the accuracy of about 81.44%.

```
Best Weights: [0.60515347 0.79106784 0.75133675 0.60162502 0.76880859 0.96126473
 0.8266647  0.9676838  0.93677117 0.93382929]
Best Fitness Score (Sum of Weights): 8.144205341154018
Simulated Accuracy (%): 81.44205341154019
```

Fig 4.7: Output for PSO without LR

4.2 Output of the GWO

```
Best Position: [-9.61588346e-07 -1.27757967e-06 -3.04865855e-07 ... -4.53546841e-07
 -8.47789351e-07 -3.81071818e-07]
Best Score (Error): 0.014541929229277772
Best Accuracy: 0.9854580707707222
```

Fig 4.8: Output for GWO

Here, the feature selection is done through GWO and the accuracy is calculated using logical regression. After implementing GWO the accuracy acquired was about 0.9854 or 98.54% which is less as compared to PSO but higher than the Naive Bayes model.

```
 4.22838556e-02 7.60933174e-01 7.45517934e-01 8.89336372e-01
 2.28690309e-01 1.41296594e-03 8.16042837e-01 7.44446335e-01
 3.59087639e-01 2.06640567e-01 9.68867391e-01 9.76105241e-01
 6.66748787e-01 6.04699630e-01 8.70802551e-01 1.32158132e-01
 2.97296649e-01 7.19304411e-01 2.71486447e-01 8.38407899e-01
 4.53971334e-01 9.72396546e-01 2.25924036e-01 2.79709283e-01]
GWO - Simulated Accuracy (%): 54.07073103702829
```

Fig 4.9: Output for GWO without LR

Here, only GWO is used without any machine learning algorithm which gave the accuracy of about 54.07%.

4.3 Scatter plot of GWO and PSO accuracies

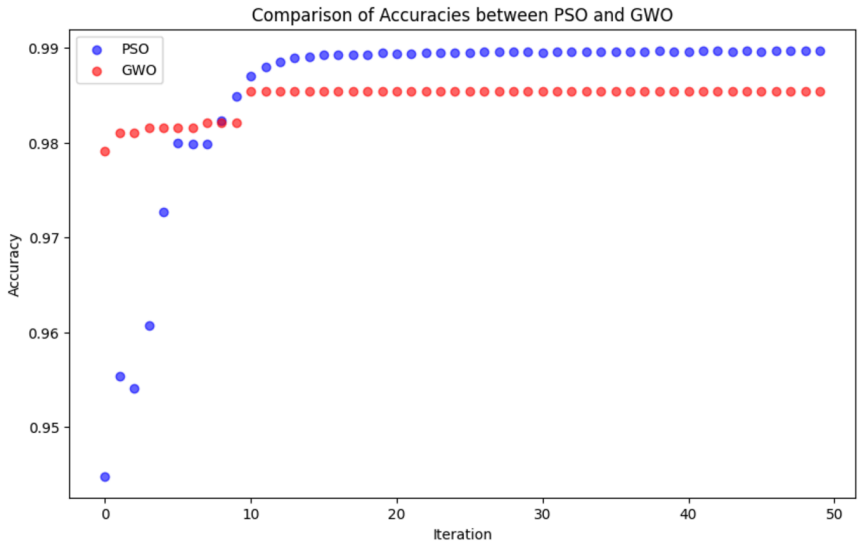

Fig 4.10: Scatter plot of GWO and PSO accuracies

In the scatter plot the accuracies of the PSO and GWO were measured over 50 iterations, initially the accuracies of GWO was higher then gradually the accuracy of PSO increased and appeared to be more than GWO at last.

4.5 Output of the Naive Bayes

```
Accuracy: 0.8822

Classification Report:

Class: 0
Precision: 0.8690
Recall: 0.9994
F1-Score: 0.9297
Support: 1607.0

Class: 1
Precision: 0.9953
Recall: 0.4693
F1-Score: 0.6379
Support: 456.0
```

Fig 4.11: Output for Naive Bayes

After implementing Naive Bayes Model the accuracy acquired was about 0.8822 or 88.22% which is less as compared to PSO and GWO.

4.4 Comparison of accuracies

This table shows the accuracies of different models. This bar chart shows the comparison between the accuracies.

Table 4.1: Accuracy Table

Models	**Accuracy**
PSO	0.8144
GWO	0.5407
PSO using LR	0.9907
GWO using LR	0.9854
Naive-Bayes	0.8822

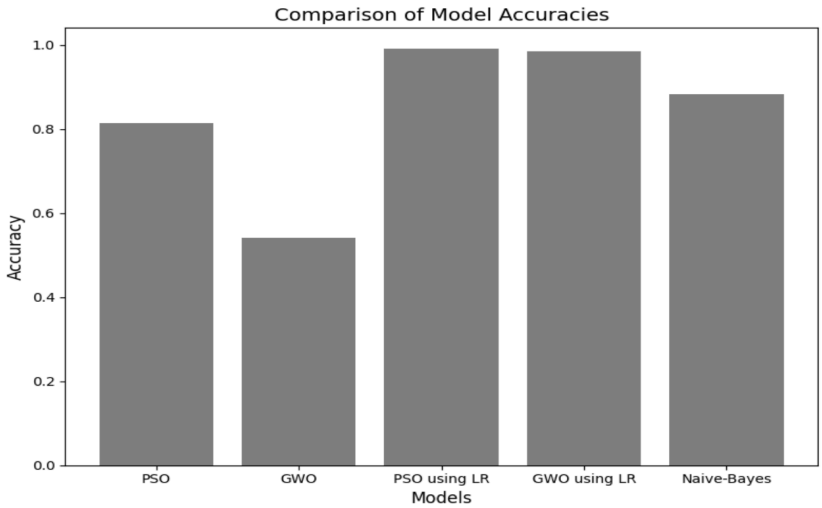

Fig 4.12: Accuracy comparison

Chapter 5

Conclusion and Future Work

Analysis of Twitter sentiment falls under the category of mining of text and opinion. We processed the dataset using three different algorithms (PSO, GWO and Naive Bayes) and then acquired the accuracy. On comparing the accuracies with each other we found out that PSO had the highest accuracy of 99.07% followed by GWO with 98.54% and Naive Bayes with the least accuracy of 88.22%. Results presented in the report indicate the performance of PSO, GWO (with the help of Logistic Regression which is a model) and Naive Bayes model of machine learning.

For the future work, we will be performing sentimental analysis using various other bio-inspired algorithms such as Ant Colony Optimization (ACO) and others. We will also do the sentimental analysis using random forest and XGBoost. We will also try to include images and videos for more comprehensive understanding of sentiments.

Guide interaction form

Guide Interaction Form

Name of the Student : ISHITA TOMAR
University Id of the Student: 23022128O
Section : A
Name of the Guide : REHAN.

S. No.	Date	Task Assigned	Task Status	Guide's Sign.
1.	05/9/24	Read research paper	Done.	Ref.
2.	07/9/24	Start understanding the maths of method	Done	Ref.
3.	12/9/24.	Implementation of PSO, GWO for SA.	Done	Ref.
4.	28/11/24	Improvements	Done	Ref
5.	30/11/24	Further improvements	Done	Ref.
6.	02/12/24	PSO filter + wrapper	done	Ref
7.	12/12/24	GWO filter, NB	done	Ref
8.	16/12/24	final Implement.	done	Ref
9.	18/12/24	Report check	done	Ref
10.	21/12/24	Report finalizing	done	Ref.

References

[1] Salam, Mustafa Abdul, and Mahmoud Ali. "Optimizing extreme learning machine using GWO algorithm for sentiment analysis." Int J Comput Appl 176.38 (2020): 22-28.

[2] Jawad, Khurram, et al. "Novel cuckoo search-based metaheuristic approach for deep learning prediction of depression." Applied Sciences 13.9 (2023): 5322.

[3] Yadav, A., Vishwakarma, D.K. A comparative study on bio-inspired algorithms for sentiment analysis. *Cluster Comput* **23**, 2969–2989 (2020).

[4] Sentiment_tweets3 [Online]. Accessed on 1st November 2024: https://www.kaggle.com/code/gargmanas/sentimental-analysis?scriptVersionId=76110084&cellId=1

[5] Ige, Tosin, et al. "An investigation into the performances of the Current state-of-the-art Naive Bayes, Non-Bayesian and Deep Learning Based Classifier for Phishing Detection: A Survey." *arXiv preprint arXiv:2411.16751* (2024).

[6] Li, Qing, et al. "Logistic Regression Matching Pursuit algorithm for text classification." *Knowledge-Based Systems* 277 (2023): 110761.

[7] Wikipedia [Online]. Accessed on 1st December 2024: https://en.wikipedia.org/wiki/Particle_swarm_optimization

[8] Percy Voughn, “Essentials of Machine Learning”, Larsen & Keller Education, 2023