In [1]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from deap import base, creator, tools, algorithms
import random
```

In [2]:

```
df = pd.read_csv('email_spam.csv')
```

In [3]:

```
df.head()
```

Out[3]:

	title	text	type
0	?? the secrets to SUCCESS	Hi James,\n\nHave you claim your complimentary...	spam
1	?? You Earned 500 GCLoot Points	\nalt_text\nCongratulations, you just earned\n...	not spam
2	?? Your GitHub launch code	Here's your GitHub launch code, @Mortyj420!\n ...	not spam
3	[The Virtual Reward Center] Re: ** Clarifications	Hello,\n \nThank you for contacting the Virtua...	not spam
4	10-1 MLB Expert Inside, Plus Everything You Ne...	Hey Prachanda Rawal,\n\nToday's newsletter is ...	spam

In [4]:

```
df.shape
```

Out[4]:

```
(84, 3)
```

In [5]:

```
df.describe()
```

Out[5]:

	title	text	type
count	84	84	84
unique	78	82	2
top	English	Model Casting Call\nThank you for taking the t...	not spam
freq	3	2	58

In [6]:

```
df.isna().sum()
```

Out[6]:

```
title    0
text     0
type     0
dtype: int64
```

In [7]:
```
tfidf = TfidfVectorizer(stop_words='english', max_features=5000)
X = tfidf.fit_transform(df['text']).toarray()
```

In [8]:
```
y = df['type'].apply(lambda x: 1 if x == 'spam' else 0)  # Converts 'spam' to
```

In [9]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
```

In [10]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [11]:
```
# Genetic Algorithm Parameters
POP_SIZE = 10  # Population size
GENS = 5  # Number of generations
```

In [19]:
```
# Fitness function to evaluate neural network performance
def eval_nn(individual):
    hidden_layer_size = int(individual[0])  # Hidden layer size
    learning_rate = individual[1]  # Learning rate

    learning_rate = max(0.0001, min(0.1, learning_rate))

    # Create and train a neural network model
    model = MLPClassifier(hidden_layer_sizes=(hidden_layer_size,),
                          learning_rate_init=learning_rate,
                          max_iter=300, random_state=42)

    model.fit(X_train, y_train)  # Train the model
    preds = model.predict(X_test)  # Predict on test set

    # Calculate accuracy
    acc = accuracy_score(y_test, preds)
    return (acc,)
```

In [20]:
```
# DEAP setup for Genetic Algorithm
creator.create("FitnessMax", base.Fitness, weights=(1.0,))  # Maximizing fitne
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
```

```
C:\Python310\lib\site-packages\deap\creator.py:185: RuntimeWarning: A class n
amed 'FitnessMax' has already been created and it will be overwritten. Consid
er deleting previous creation of that class or rename it.
  warnings.warn("A class named '{0}' has already been created and it "
C:\Python310\lib\site-packages\deap\creator.py:185: RuntimeWarning: A class n
amed 'Individual' has already been created and it will be overwritten. Consid
er deleting previous creation of that class or rename it.
  warnings.warn("A class named '{0}' has already been created and it "
```

In [21]:

```
toolbox.register("attr_int", random.randint, 10, 100)       # Random integer f
toolbox.register("attr_float", random.uniform, 0.001, 0.1)  # Random float for
toolbox.register("individual", tools.initCycle, creator.Individual,
                 (toolbox.attr_int, toolbox.attr_float), n=1)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("evaluate", eval_nn)  # Evaluate function
toolbox.register("mate", tools.cxBlend, alpha=0.5)  # Crossover method
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=5, indpb=0.2)  # Mut
toolbox.register("select", tools.selTournament, tournsize=3)  # Selection
```

In [22]:

```
# Run the Genetic Algorithm
pop = toolbox.population(n=POP_SIZE)
algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=GENS, verbose=True
```

```
3       6
4       7
5       5
```

Out[22]:

```
([[37.0, 0.0019069096986697445],
  [37.0, 0.0019069096986697445],
  [37.0, 0.0019069096986697445],
  [37.0, 0.0019069096986697445],
  [37.0, 0.0019069096986697445],
  [37.0, 0.0019069096986697445],
  [37.0, 0.0019069096986697445],
  [37.0, 0.0019069096986697445],
  [37.0, 0.0019069096986697445],
  [37.0, 0.0019069096986697445]],
 [{'gen': 0, 'nevals': 10},
  {'gen': 1, 'nevals': 6},
  {'gen': 2, 'nevals': 9},
  {'gen': 3, 'nevals': 6},
  {'gen': 4, 'nevals': 7},
  {'gen': 5, 'nevals': 5}])
```

In [23]:

```
# Best individual after GA optimization
best_ind = tools.selBest(pop, k=1)[0]
print("\nBest Parameters (Hidden Layer, Learning Rate):", best_ind)
```

```

Best Parameters (Hidden Layer, Learning Rate): [37.0, 0.0019069096986697445]
```

In [24]:

```
# Train final model with the best parameters
final_model = MLPClassifier(hidden_layer_sizes=(int(best_ind[0]),),
                            learning_rate_init=best_ind[1], max_iter=300)
final_model.fit(X_train, y_train)
```

Out[24]:

```
▾                          MLPClassifier

MLPClassifier(hidden_layer_sizes=(37,),
              learning_rate_init=0.0019069096986697445, max_iter=300)
```

In [25]:

```
# Predict and calculate accuracy on the test set
final_preds = final_model.predict(X_test)
final_acc = accuracy_score(y_test, final_preds)
print("Final Accuracy: {:.2f}%".format(final_acc * 100))
```

```
Final Accuracy: 64.71%
```

In [ ]: