EMAIL SPAM CLASSIFICATION MODEL USING ARTIFICIAL NEURAL NETWORKS

Live Website - https://spamclassifier.azurewebsites.net/

Dataset Source - https://archive.ics.uci.edu/dataset/94/spambase

```
# Importing all the necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import confusion_matrix
```

```
# Load the dataset
file_path = 'spambase.data'

# Since the dataset does not include header information, we need to create column names
# The dataset description indicates there are 57 attributes followed by a class label
attribute_names = 'word_freq_make, word_freq_address, word_freq_all, word_freq_3d, word_freq_our, word_freq_over, word_freq_remove, word_

# Read the dataset
spambase_df = pd.read_csv(file_path, names=attribute_names)

# Display the head of the dataframe
print(spambase_df.head())

# Display the shape of the dataframe
print(spambase_df.shape)
```

```
   word_freq_make  word_freq_address  word_freq_all  word_freq_3d  \
0            0.00               0.64           0.64           0.0
1            0.21               0.28           0.50           0.0
2            0.06               0.00           0.71           0.0
3            0.00               0.00           0.00           0.0
4            0.00               0.00           0.00           0.0

   word_freq_our  word_freq_over  word_freq_remove  word_freq_internet  \
0           0.32            0.00              0.00                0.00
1           0.14            0.28              0.21                0.07
2           1.23            0.19              0.19                0.12
3           0.63            0.00              0.31                0.63
4           0.63            0.00              0.31                0.63

   word_freq_order  word_freq_mail  ...  char_freq_;  char_freq_(  \
0             0.00            0.00  ...         0.00        0.000
1             0.00            0.94  ...         0.00        0.132
2             0.64            0.25  ...         0.01        0.143
3             0.31            0.63  ...         0.00        0.137
4             0.31            0.63  ...         0.00        0.135

   char_freq_[  char_freq_!  char_freq_$  char_freq_#  \
0          0.0        0.778        0.000        0.000
1          0.0        0.372        0.180        0.048
2          0.0        0.276        0.184        0.010
3          0.0        0.137        0.000        0.000
4          0.0        0.135        0.000        0.000

   capital_run_length_average  capital_run_length_longest  \
0                       3.756                          61
1                       5.114                         101
2                       9.821                         485
3                       3.537                          40
4                       3.537                          40

   capital_run_length_total  class_label
0                       278            1
1                      1028            1
2                      2259            1
3                       191            1
4                       191            1
```

```
[5 rows x 58 columns]
(4601, 58)
```

EXPLORATORY DATA ANALYSIS

```
# Display a summary of the dataframe
print(spambase_df.describe())
```

```
std         0.305358          1.290575       0.504143      1.395151
min         0.000000          0.000000       0.000000      0.000000
25%         0.000000          0.000000       0.000000      0.000000
50%         0.000000          0.000000       0.000000      0.000000
75%         0.000000          0.000000       0.420000      0.000000
max         4.540000         14.280000       5.100000     42.810000

       word_freq_our  word_freq_over  word_freq_remove  word_freq_internet  \
count    4601.000000     4601.000000       4601.000000         4601.000000
mean        0.312223        0.095901          0.114208            0.105295
std         0.672513        0.273824          0.391441            0.401071
min         0.000000        0.000000          0.000000            0.000000
25%         0.000000        0.000000          0.000000            0.000000
50%         0.000000        0.000000          0.000000            0.000000
75%         0.380000        0.000000          0.000000            0.000000
max        10.000000        5.880000          7.270000           11.110000

       word_freq_order  word_freq_mail  ...  char_freq_;  char_freq_(  \
count      4601.000000     4601.000000  ...  4601.000000  4601.000000
mean          0.090067        0.239413  ...     0.038575     0.139030
std           0.278616        0.644755  ...     0.243471     0.270355
min           0.000000        0.000000  ...     0.000000     0.000000
25%           0.000000        0.000000  ...     0.000000     0.000000
50%           0.000000        0.000000  ...     0.000000     0.065000
75%           0.000000        0.160000  ...     0.000000     0.188000
max           5.260000       18.180000  ...     4.385000     9.752000

       char_freq_[  char_freq_!  char_freq_$  char_freq_#  \
count  4601.000000  4601.000000  4601.000000  4601.000000
mean      0.016976     0.269071     0.075811     0.044238
std       0.109394     0.815672     0.245882     0.429342
min       0.000000     0.000000     0.000000     0.000000
25%       0.000000     0.000000     0.000000     0.000000
50%       0.000000     0.000000     0.000000     0.000000
75%       0.000000     0.315000     0.052000     0.000000
max       4.081000    32.478000     6.003000    19.829000

       capital_run_length_average  capital_run_length_longest  \
count                 4601.000000                 4601.000000
mean                     5.191515                   52.172789
std                     31.729449                  194.891310
min                      1.000000                    1.000000
25%                      1.588000                    6.000000
50%                      2.276000                   15.000000
75%                      3.706000                   43.000000
max                   1102.500000                 9989.000000

       capital_run_length_total  class_label
count               4601.000000  4601.000000
mean                 283.289285     0.394045
std                  606.347851     0.488698
min                    1.000000     0.000000
25%                   35.000000     0.000000
50%                   95.000000     0.000000
75%                  266.000000     1.000000
max                15841.000000     1.000000

[8 rows x 58 columns]
```

```
# Check for any missing values
missing_values = spambase_df.isnull().sum()
print('Missing values in each column:\n', missing_values)
```

```
word_freq_address             0
word_freq_all                 0
word_freq_3d                  0
word_freq_our                 0
word_freq_over                0
word_freq_remove              0
```

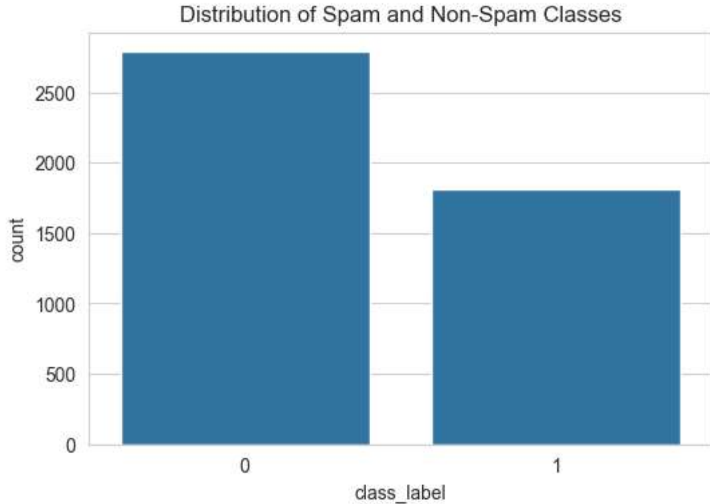
```
word_freq_mail                0
word_freq_receive             0
word_freq_will                0
word_freq_people              0
word_freq_report              0
word_freq_addresses           0
word_freq_free                0
word_freq_business            0
word_freq_email               0
word_freq_you                 0
word_freq_credit              0
word_freq_your                0
word_freq_font                0
word_freq_000                 0
word_freq_money               0
word_freq_hp                  0
word_freq_hpl                 0
word_freq_george              0
word_freq_650                 0
word_freq_lab                 0
word_freq_labs                0
word_freq_telnet              0
word_freq_857                 0
word_freq_data                0
word_freq_415                 0
word_freq_85                  0
word_freq_technology          0
word_freq_1999                0
word_freq_parts               0
word_freq_pm                  0
word_freq_direct              0
word_freq_cs                  0
word_freq_meeting             0
word_freq_original            0
word_freq_project             0
word_freq_re                  0
word_freq_edu                 0
word_freq_table               0
word_freq_conference          0
char_freq_;                   0
char_freq_(                   0
char_freq_[                   0
char_freq_!                   0
char_freq_$                   0
char_freq_#                   0
capital_run_length_average    0
capital_run_length_longest    0
capital_run_length_total      0
class_label                   0
dtype: int64
```

```
# Check the balance of the classes
spam_class_distribution = spambase_df['class_label'].value_counts()
print('Spam class distribution:\n', spam_class_distribution)
```

```
Spam class distribution:
 class_label
0    2788
1    1813
Name: count, dtype: int64
```

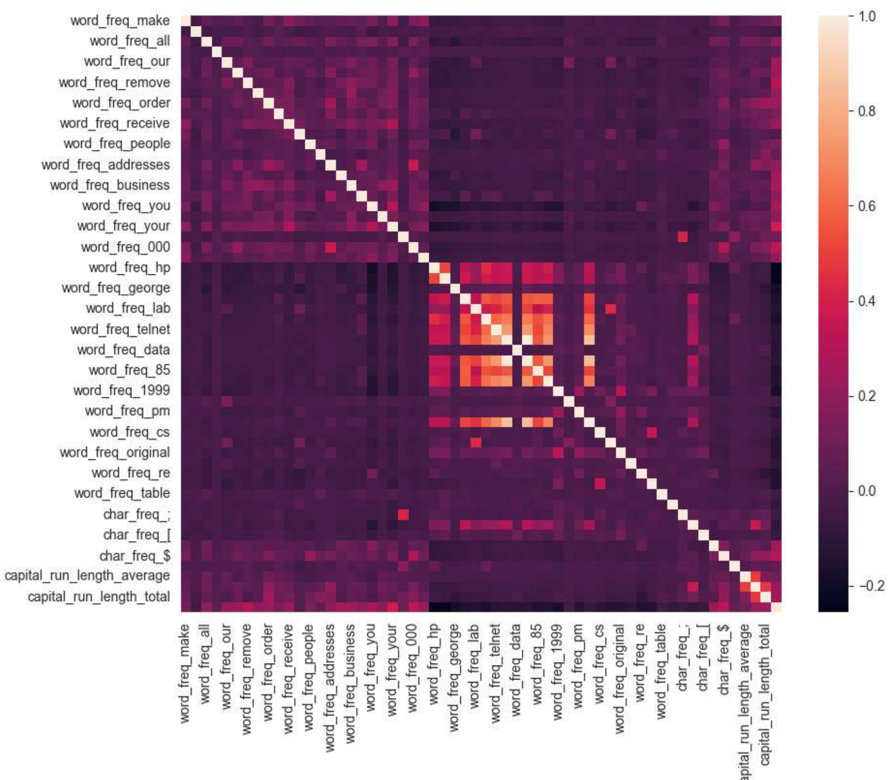
```
# Set the aesthetic style of the plots
sns.set_style('whitegrid')

# Plotting the distribution of the spam and non-spam classes
plt.figure(figsize=(6, 4))
sns.countplot(x='class_label', data=spambase_df)
plt.title('Distribution of Spam and Non-Spam Classes')
plt.show()
```




```
# Plotting the correlation matrix
plt.figure(figsize=(10, 8))
corr_matrix = spambase_df.corr()
sns.heatmap(corr_matrix)
```

```
<Axes: >
```



TRAIN - TEST SPLIT

```
# Split the data into features and target variable
X = spambase_df.drop('class_label', axis=1)
y = spambase_df['class_label']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Output the shape of the train and test sets
print('Training set shape:', X_train.shape)
print('Testing set shape:', X_test.shape)
```

```
Training set shape: (3220, 57)
Testing set shape: (1381, 57)
```

NORMALISATION

```
# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform both the training and testing data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

BUILDING NEURAL NETWORK

```
# Check if GPU is available and set the device accordingly
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
# Convert the scaled data to torch tensors
X_train_tensor = torch.tensor(X_train_scaled.astype(np.float32)).to(device)
y_train_tensor = torch.tensor(y_train.values.astype(np.float32)).to(device)
X_test_tensor = torch.tensor(X_test_scaled.astype(np.float32)).to(device)
y_test_tensor = torch.tensor(y_test.values.astype(np.float32)).to(device)
```

```
# Create TensorDatasets for the training and testing data
train_data = TensorDataset(X_train_tensor, y_train_tensor)
test_data = TensorDataset(X_test_tensor, y_test_tensor)

# Create DataLoaders for the training and testing data
train_loader = DataLoader(train_data, batch_size=64, shuffle=True)
test_loader = DataLoader(test_data, batch_size=64, shuffle=False)
```

```
# Define the neural network architecture with 3 hidden layers
class SpamClassifier1(nn.Module):
    def __init__(self, activation_fn):
        super(SpamClassifier1, self).__init__()
        self.fc1 = nn.Linear(57, 32)
        self.fc2 = nn.Linear(32, 16)
        self.fc3 = nn.Linear(16, 1)
        self.activation_fn = activation_fn()

    def forward(self, x):
        x = self.activation_fn(self.fc1(x))
        x = self.activation_fn(self.fc2(x))
        x = torch.sigmoid(self.fc3(x))
        return x
```

```
# Define the neural network architecture with 4 hidden layers
class SpamClassifier2(nn.Module):
    def __init__(self, activation_fn):
        super(SpamClassifier2, self).__init__()
        self.fc1 = nn.Linear(57, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 16)
        self.fc4 = nn.Linear(16, 1)
        self.activation_fn = activation_fn()

    def forward(self, x):
        x = self.activation_fn(self.fc1(x))
        x = self.activation_fn(self.fc2(x))
```

```
        x = self.activation_fn(self.fc2(x))
        x = self.activation_fn(self.fc3(x))
        x = torch.sigmoid(self.fc4(x))
        return x

```

```
# Initialize the neural network

# Model selection
classifier_options = {
    "1": SpamClassifier1(nn.ReLU),
    "2": SpamClassifier1(nn.Tanh),
    "3": SpamClassifier1(nn.Sigmoid),
    "4": SpamClassifier1(lambda: nn.Identity()),
    "5": SpamClassifier2(nn.ReLU),
    "6": SpamClassifier2(nn.Tanh),
    "7": SpamClassifier2(nn.Sigmoid),
    "8": SpamClassifier2(lambda: nn.Identity())
}

# Select the model
Option = input("Select the model:\n 1 - SpamClassifier_ReLU_3Layer,\n 2 - SpamClassifier_Tanh_3Layer,\n 3 - SpamClassifier_Sigmoid_3Layer

model = classifier_options[Option].to(device)

# Define the loss function and optimizer
loss_function = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
# Training the neural network
num_epochs = 10
model.train()
for epoch in range(num_epochs):
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = loss_function(output, target.view(-1, 1))
        loss.backward()
        optimizer.step()

    # Print progress
    print('Epoch ', epoch+1, '/', num_epochs, ': Loss -', loss.item())
```

```
Epoch  1 / 10 : Loss - 0.17934352159500122
Epoch  2 / 10 : Loss - 0.07509439438581467
Epoch  3 / 10 : Loss - 0.10006360709667206
Epoch  4 / 10 : Loss - 0.03437989205121994
Epoch  5 / 10 : Loss - 0.11822696030139923
Epoch  6 / 10 : Loss - 0.0790012925863266
Epoch  7 / 10 : Loss - 0.012757109478116035
Epoch  8 / 10 : Loss - 0.004362315870821476
Epoch  9 / 10 : Loss - 0.002448256593197584
Epoch  10 / 10 : Loss - 0.0585327222943306
```

EVALUATION

```
# Evaluate the model
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        outputs = model(data)
        predicted = outputs.ge(.5).view(-1)
        total += target.size(0)
        correct += (predicted == target).sum().item()

accuracy = correct / total
print('Test Accuracy: ', accuracy)
```

```
Test Accuracy:  0.939898624185373
```

```
# Get the predicted labels for the test data
model.eval()
with torch.no_grad():
    y_pred = []
    for data, target in test_loader:
        data = data.to(device)
        outputs = model(data)
        predicted = outputs.ge(.5).view(-1)
        y_pred.extend(predicted.cpu().numpy())

# Create the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

```