Feed-Forward Neural Network for Sentiment Classification
and Language Modeling
Maryah Austria

**Program Description**

This assignment focuses on building and training feedforward neural networks with two hidden layers, each containing 20 units, to perform sentiment classification and language modeling on airline review tweets. Implemented from scratch using basic libraries like NumPy, these models will classify tweets as either positive or negative and distinguish between positive and negative 2-grams. Preprocessing steps include tokenization, normalization, stemming, and TF-IDF feature extraction. Model performance will be evaluated on test data, with metrics such as accuracy and confusion matrices saved in a .txt or .log file.

**Feed Forward Neural Network for Sentiment Classification**

**Task 1: Import and load data**

In [1]:
```
#import necessary libraries
import os
import glob
import numpy as np
import pandas as pd
import math
from collections import Counter
from sklearn.model_selection import train_test_split
from nltk.stem import PorterStemmer, SnowballStemmer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from bs4 import BeautifulSoup
import emoji
import nltk
```

In [2]:
```
# Initialize lists to store the tweets and their corresponding labels
tweets = []
labels = []

# Paths to positive and negative sentiment folders
positive_path = os.path.join("C:\\Users\\marya\\Desktop\\AIT726\\Program Asisgnment\\extracted_tweets\
negative_path = os.path.join("C:\\Users\\marya\\Desktop\\AIT726\\Program Asisgnment\\extracted_tweets\

# Load data from folder
def load_data_from_folder(folder_path, label):
    text_files = glob.glob(os.path.join(folder_path, '*.txt'))
    for file in text_files:
        with open(file, 'r', encoding='utf-8') as f:
            tweets.append(f.read())
            labels.append(label)

# Load positive and negative data
load_data_from_folder(positive_path, 1)  # Label 1 for positive
load_data_from_folder(negative_path, 0)  # Label 0 for negative

print(f"Total tweets loaded: {len(tweets)}")
print(f"Total labels loaded: {len(labels)}")
print(f"Label distribution: {Counter(labels)}")
```

```
Total tweets loaded: 4181
Total labels loaded: 4181
Label distribution: Counter({0: 3000, 1: 1181})
```

**Task 2: Preprocess Data**

After reading and importing the training data, it was preprocessed using steps similar to those in Program Assignment 1. HTML tags were removed using BeautifulSoup, as done in the previous assignment. Emojis were converted into their respective text descriptions using emoji.demojize(). For example, 😎 was transformed into :sunglasses:, a method also applied in Assignment 1 to handle special characters.

The text was then tokenized at both whitespace and punctuation, ensuring that words like "teacher’s" were split into "teacher" and "'s," consistent with the tokenization method used in the earlier assignment. Capitalized words were lowercased, while acronyms or fully capitalized words like "NASA" were left untouched. Stopwords were removed using NLTK’s stopword list, as in the previous assignment.

Finally, stemming was applied using either the Porter or Snowball stemmer to reduce words to their root forms. For example, "studying" became "studi" using the Porter stemmer. These preprocessing steps helped create a clean and stemmed vocabulary from the training data, ensuring that the test set was not incorporated, as done in Assignment 1.

In [3]:

```python
# Initialize the stemmers
porter_stemmer = PorterStemmer()
snowball_stemmer = SnowballStemmer('english')

# Load stopwords
stop_words = set(stopwords.words('english'))

# Preprocessing and stemming function
def preprocess_and_stem_tweet(text, apply_stemming=False, stemmer_type='porter'):
    # Remove HTML tags
    text = BeautifulSoup(text, "html.parser").get_text()

    # Convert emoji to its text description
    text = emoji.demojize(text)

    # Tokenize at whitespace and punctuation
    tokens = word_tokenize(text)

    # Lowercase words, except acronyms or fully capitalized words like NASA
    tokens = [word.lower() if not word.isupper() else word for word in tokens]

    # Remove stopwords
    tokens = [word for word in tokens if word not in stop_words]

    # Apply stemming if specified
    if apply_stemming:
        if stemmer_type == 'porter':
            tokens = [porter_stemmer.stem(word) for word in tokens]
        elif stemmer_type == 'snowball':
            tokens = [snowball_stemmer.stem(word) for word in tokens]

    return tokens

# Example of sample preprocess tweet and then a stem tweet
tweet_example = tweets[30]
processed_tweet = preprocess_and_stem_tweet(tweet_example, apply_stemming=True, stemmer_type='porter')
print("Unprocessed tweet: ", tweet_example)
print("Processed tweet: ", processed_tweet)
```

```
Unprocessed tweet:  @united Hmmm...seems like this could be something to be changed to be more #flye
rfriendly.

Processed tweet:  ['@', 'unit', 'hmmm', '...', 'seem', 'like', 'could', 'someth', 'chang', '#', 'fly
erfriendli', '.']
```

In [4]:
```
# Initialize vocabularies
vocab_with_stemming = set()
vocab_without_stemming = set()

# Build vocabularies
for tweet in tweets:
    vocab_without_stemming.update(preprocess_and_stem_tweet(tweet, apply_stemming=False))
    vocab_with_stemming.update(preprocess_and_stem_tweet(tweet, apply_stemming=True))
```

```
C:\Users\marya\AppData\Local\Temp\ipykernel_8224\2592623341.py:11: MarkupResemblesLocatorWarning: Th
e input looks more like a filename than markup. You may want to open this file and pass the filehand
le into Beautiful Soup.
  text = BeautifulSoup(text, "html.parser").get_text()
```

**Task 3: Extract features**

In this task, the documents are represented as numerical vectors using TF-IDF (Term Frequency-Inverse Document Frequency). First, the term frequency (TF) is calculated, which measures how often a term appears in a document relative to the total number of words in that document. The inverse document frequency (IDF) is then computed, indicating how rare or common a term is across the entire corpus of documents. By multiplying TF by IDF, the TF-IDF score is generated, assigning higher importance to terms that are significant within a document but occur infrequently across the corpus.

After calculating the TF-IDF scores for each term in the corpus, the vectors are converted into a numerical matrix using a NumPy array. This matrix efficiently stores and manipulates the data for further analysis, with each row representing a document and each column representing a term. Using a NumPy array for the matrix is crucial as it facilitates mathematical operations like dot products and matrix multiplications, necessary for training machine learning models. This matrix format provides a structured way to work with the high-dimensional data produced by TF-IDF feature extraction.

In [5]:
```python
# Function to calculate TF-IDF
def calculate_tfidf(tweets, apply_stemming=False):
    tf_dict = [] #list that will store term frequency (TF) for each tweet
    df_dict = {} #dictionary that will store document frequency (DF) for each term
    document_count = len(tweets) #the total number of tweets

    #Iterate through each tweet
    for tweet in tweets:
        #preprocess and optionally stem the tweet
        tokens = preprocess_and_stem_tweet(tweet, apply_stemming=apply_stemming)
        total_terms = len(tokens) #total number of terms in the tweet
        term_count = Counter(tokens) #count the occurene of each term in the tweet

        #Calculate the TF for each term in the tweet
        tf = {term: count / total_terms for term, count in term_count.items()}
        tf_dict.append(tf) #append the TF to the dictionary

        #this will ensure that the document is updated for DF for eaach term in the tweet
        for term in set(tokens):
            df_dict[term] = df_dict.get(term, 0) + 1

    #calculate inverse document frequency (IDF) for each term
    idf_dict = {term: math.log(document_count / (df + 1)) for term, df in df_dict.items()}
    #create TF-IDF vectors by multiplying TF and IDF for each term in the tweet
    tf_idf_vectors = [{term: tf_val * idf_dict[term] for term, tf_val in tf.items()} for tf in tf_dict

    return tf_idf_vectors, idf_dict

# Create feature matrices
def create_feature_matrix(tfidf_vectors, vocab):
    feature_matrix = np.zeros((len(tfidf_vectors), len(vocab)))
    vocab_index = {word: i for i, word in enumerate(vocab)}

    for i, tfidf_vector in enumerate(tfidf_vectors):
        for term, value in tfidf_vector.items():
            if term in vocab_index:
                feature_matrix[i, vocab_index[term]] = value

    return feature_matrix

# Calculate TF-IDF for stemmed and non-stemmed data
tfidf_stemmed, idf_stemmed = calculate_tfidf(tweets, apply_stemming=True)
tfidf_non_stemmed, idf_non_stemmed = calculate_tfidf(tweets, apply_stemming=False)

# Create feature matrices from the TF-IDF vectors for both stemmed and non-stemmed
X_stemmed = create_feature_matrix(tfidf_stemmed, vocab_with_stemming)
X_non_stemmed = create_feature_matrix(tfidf_non_stemmed, vocab_without_stemming)
y = np.array(labels).reshape(-1, 1)

# Print TF-IDF results for IDF values
print("TF-IDF for stemmed tokens:")
for term, idf in list(idf_stemmed.items())[:10]:  # Displaying first 10 terms for brevity
    print(f"Term: {term}, IDF: {idf}")

print("\nTF-IDF for non-stemmed tokens:")
for term, idf in list(idf_non_stemmed.items())[:10]:  # Displaying first 10 terms for brevity
    print(f"Term: {term}, IDF: {idf}")
```

```
C:\Users\marya\AppData\Local\Temp\ipykernel_8224\2592623341.py:11: MarkupResemblesLocatorWarning: Th
e input looks more like a filename than markup. You may want to open this file and pass the filehand
le into Beautiful Soup.
  text = BeautifulSoup(text, "html.parser").get_text()
```

```
TF-IDF for stemmed tokens:
Term: would, IDF: 3.327670437260309
Term: southwestair, IDF: 1.8001659075888952
Term: @, IDF: -0.00023914863201387654
Term: thank, IDF: 1.8280473908334156
Term: i, IDF: 1.1142809230707345
Term: ., IDF: 0.4972059659344454
Term: appreci, IDF: 4.047846290208174
Term: usairway, IDF: 1.5493339883643948
Term: much, IDF: 3.827446224839715
Term: jetblu, IDF: 1.9069746494230861

TF-IDF for non-stemmed tokens:
Term: would, IDF: 3.34109345759245
Term: southwestair, IDF: 1.8001659075888952
Term: @, IDF: -0.00023914863201387654
Term: thank, IDF: 2.8168448134943187
Term: appreciate, IDF: 4.34932168479229
Term: ., IDF: 0.4972059659344454
Term: I, IDF: 1.1142809230707345
Term: usairways, IDF: 1.5504607490469855
Term: much, IDF: 3.8609689168783583
Term: snacks, IDF: 6.392395582301252
```

**Task 4: Build a Feed Forward Neural Network(FFNN)**

In this task, a Feed-Forward Neural Network (FFNN) was developed with two layers, each containing 20 neurons. The weights were initialized randomly to eliminate bias at the start. Initially, Mean Squared Error (MSE) was used as the loss function; however, to enhance performance in binary classification, Cross-Entropy Loss was also incorporated for hyperparameter tuning. Cross-Entropy Loss is more effective in binary classification as it penalizes misclassifications more heavily, promoting faster convergence towards accurate predictions.

The sigmoid activation function was employed to introduce non-linearity, allowing the model to output probabilities between 0 and 1. The network was trained with a learning rate of 0.0001 for stable weight updates.

In [6]:
```
# Define the Feed-Forward Neural Network (FFNN) class
class FFNN:
    def __init__(self, input_size, hidden_size, output_size, learning_rate=0.001, epochs=1000, loss_fu
        #Initialize the parameters
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.loss_function = loss_function
        #here is where we randomly initialize weights and biases
        self.weights_input_hidden = np.random.randn(self.input_size, self.hidden_size)
        self.weights_hidden_output = np.random.randn(self.hidden_size, self.output_size)
        self.bias_hidden = np.zeros((1, self.hidden_size))
        self.bias_output = np.zeros((1, self.output_size))

    #sigmoid activation function
    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    #derivative of the sigmoid function that will be used for back propagation
    def sigmoid_derivative(self, x):
        return x * (1 - x)
    #Mean Sqaured Error (MSE) loss function
    def mse_loss(self, y_true, y_pred):
        return np.mean(np.square(y_true - y_pred))

    #Cross-entropy loss function for classification
    def cross_entropy_loss(self, y_true, y_pred):
        return -np.mean(y_true * np.log(y_pred + 1e-15) + (1 - y_true) * np.log(1 - y_pred + 1e-15))

    #forward pass through the neural network
    def forward(self, x):
        #input to hidden layer
        self.hidden_input = np.dot(x, self.weights_input_hidden) + self.bias_hidden
        self.hidden_output = self.sigmoid(self.hidden_input)
        #hidden layer to output layer
        self.final_input = np.dot(self.hidden_output, self.weights_hidden_output) + self.bias_output
        self.final_output = self.sigmoid(self.final_input)
        return self.final_output

    #back propagation for weight updates
    def backward(self, x, y):
        #loss computation for output layer
        output_loss = self.final_output - y
        output_gradient = output_loss * self.sigmoid_derivative(self.final_output)

        #loss and gradient computation for the hidden layer
        hidden_loss = np.dot(output_gradient, self.weights_hidden_output.T)
        hidden_gradient = hidden_loss * self.sigmoid_derivative(self.hidden_output)

        #update weights and biases for the output layer
        self.weights_hidden_output -= np.dot(self.hidden_output.T, output_gradient) * self.learning_ra
        self.bias_output -= np.sum(output_gradient, axis=0, keepdims=True) * self.learning_rate

        #update weights and biases for the output layer
        self.weights_input_hidden -= np.dot(x.T, hidden_gradient) * self.learning_rate
        self.bias_hidden -= np.sum(hidden_gradient, axis=0, keepdims=True) * self.learning_rate

    #train function for the FFNN
    def train(self, x, y):
        for epoch in range(self.epochs):
            self.forward(x)
            self.backward(x, y)
            if self.loss_function == 'mse':
                loss = self.mse_loss(y, self.final_output)
            else:
                loss = self.cross_entropy_loss(y, self.final_output)
            if epoch % 100 == 0:  # Print loss every 100 epochs
                print(f"Epoch {epoch}, Loss: {loss:.4f}")
```

```
    #Predict function for the new data
    def predict(self, x):
        output = self.forward(x)
        return (output > 0.5).astype(int)
```

**Task 5: Build accuracy metrics**

In this task, key performance metrics used in binary classification are defined: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). True Positives represent the instances correctly predicted as positive, while True Negatives are those correctly identified as negative. False Positives occur when a negative instance is incorrectly classified as positive, and False Negatives arise when a positive instance is misclassified as negative. To evaluate the model's performance, accuracy is calculated, which measures the proportion of correct predictions. The formula for accuracy = (TP)+(TN)/ (TP)+(TN)+(FP)+(FN)

In [7]:
```
# Initialize metrics
def initialize_metrics():
    return {'tn': 0, 'fp': 0, 'fn': 0, 'tp': 0}

# Calculate metrics (TP, TN, FP, FN)
def calculate_metrics(y_true, y_pred):
    metrics = initialize_metrics()
    for true_label, predicted_label in zip(y_true, y_pred):
        if true_label == 1 and predicted_label == 1:
            metrics['tp'] += 1
        elif true_label == 1 and predicted_label == 0:
            metrics['fn'] += 1
        elif true_label == 0 and predicted_label == 1:
            metrics['fp'] += 1
        elif true_label == 0 and predicted_label == 0:
            metrics['tn'] += 1
    return metrics

# Calculate accuracy
def calculate_accuracy(metrics):
    total = metrics['tp'] + metrics['tn'] + metrics['fp'] + metrics['fn']
    accuracy = (metrics['tp'] + metrics['tn']) / total if total != 0 else 0
    return accuracy
```

**Task 6: Hyperparameter tuning**

In [8]:
```
# Hyperparameter tuning
hyperparams = [
    {'hidden_size': 20, 'learning_rate': 0.0001, 'epochs': 1000, 'loss_function': 'mse'},
    {'hidden_size': 20, 'learning_rate': 0.001, 'epochs': 1000, 'loss_function': 'cross_entropy'},
    # Add more combinations as needed
]
```

**Task 7: Train and evalaute the model on both stemmed and non-stemmed vocabularies**

In [9]:
```python
# Train and evaluate the model on stemmed data for each hyperparameter combination
for params in hyperparams:
    print(f"\nTraining on Stemmed Data with params: {params}")

    # Initialize the model with the specified hyperparameters and loss function
    model_stemmed = FFNN(input_size=X_stemmed.shape[1],
                         hidden_size=params['hidden_size'],
                         output_size=1,
                         learning_rate=params['learning_rate'],
                         epochs=params['epochs'],
                         loss_function=params['loss_function'])  # Include the loss function

    model_stemmed.train(X_stemmed, y)

    # Predictions and evaluation
    y_pred_stemmed = model_stemmed.predict(X_stemmed)
    metrics_stemmed = calculate_metrics(y, y_pred_stemmed)
    accuracy_stemmed = calculate_accuracy(metrics_stemmed)


    # Print the results for monitoring
    print(f"Confusion Metrics (Stemmed): {metrics_stemmed}")
    print(f"Accuracy (Stemmed): {accuracy_stemmed * 100:.2f}%")
```

```
Training on Stemmed Data with params: {'hidden_size': 20, 'learning_rate': 0.0001, 'epochs': 1000,
'loss_function': 'mse'}
Epoch 0, Loss: 0.3166
Epoch 100, Loss: 0.2348
Epoch 200, Loss: 0.2297
Epoch 300, Loss: 0.2244
Epoch 400, Loss: 0.2190
Epoch 500, Loss: 0.2136
Epoch 600, Loss: 0.2084
Epoch 700, Loss: 0.2034
Epoch 800, Loss: 0.1987
Epoch 900, Loss: 0.1945
Confusion Metrics (Stemmed): {'tn': 2794, 'fp': 206, 'fn': 967, 'tp': 214}
Accuracy (Stemmed): 71.94%

Training on Stemmed Data with params: {'hidden_size': 20, 'learning_rate': 0.001, 'epochs': 1000, 'l
oss_function': 'cross_entropy'}
Epoch 0, Loss: 0.8756
Epoch 100, Loss: 0.5595
Epoch 200, Loss: 0.5256
Epoch 300, Loss: 0.5001
Epoch 400, Loss: 0.4767
Epoch 500, Loss: 0.4552
Epoch 600, Loss: 0.4356
Epoch 700, Loss: 0.4180
Epoch 800, Loss: 0.4019
Epoch 900, Loss: 0.3873
Confusion Metrics (Stemmed): {'tn': 2819, 'fp': 181, 'fn': 472, 'tp': 709}
Accuracy (Stemmed): 84.38%
```

In [10]:
```python
# Train and evaluate the model on non-stemmed data for each hyperparameter combination
for params in hyperparams:
    print(f"\nTraining on Non-Stemmed Data with params: {params}")

    # Initialize the model with the specified hyperparameters and loss function
    model_non_stemmed = FFNN(input_size=X_non_stemmed.shape[1],
                             hidden_size=params['hidden_size'],
                             output_size=1,
                             learning_rate=params['learning_rate'],
                             epochs=params['epochs'],
                             loss_function=params['loss_function'])  # Include the loss function

    model_non_stemmed.train(X_non_stemmed, y)

    # Predictions and evaluation
    y_pred_non_stemmed = model_non_stemmed.predict(X_non_stemmed)
    metrics_non_stemmed = calculate_metrics(y, y_pred_non_stemmed)
    accuracy_non_stemmed = calculate_accuracy(metrics_non_stemmed)

    print(f"Confusion Metrics (Non-Stemmed): {metrics_non_stemmed}")
    print(f"Accuracy (Non-Stemmed): {accuracy_non_stemmed * 100:.2f}%")
```

```
Training on Non-Stemmed Data with params: {'hidden_size': 20, 'learning_rate': 0.0001, 'epochs': 100
0, 'loss_function': 'mse'}
Epoch 0, Loss: 0.6926
Epoch 100, Loss: 0.2191
Epoch 200, Loss: 0.2139
Epoch 300, Loss: 0.2091
Epoch 400, Loss: 0.2045
Epoch 500, Loss: 0.2003
Epoch 600, Loss: 0.1964
Epoch 700, Loss: 0.1929
Epoch 800, Loss: 0.1897
Epoch 900, Loss: 0.1869
Confusion Metrics (Non-Stemmed): {'tn': 2834, 'fp': 166, 'fn': 923, 'tp': 258}
Accuracy (Non-Stemmed): 73.95%

Training on Non-Stemmed Data with params: {'hidden_size': 20, 'learning_rate': 0.001, 'epochs': 100
0, 'loss_function': 'cross_entropy'}
Epoch 0, Loss: 0.6832
Epoch 100, Loss: 0.5946
Epoch 200, Loss: 0.5699
Epoch 300, Loss: 0.5552
Epoch 400, Loss: 0.5391
Epoch 500, Loss: 0.5212
Epoch 600, Loss: 0.5029
Epoch 700, Loss: 0.4852
Epoch 800, Loss: 0.4686
Epoch 900, Loss: 0.4532
Confusion Metrics (Non-Stemmed): {'tn': 2820, 'fp': 180, 'fn': 583, 'tp': 598}
Accuracy (Non-Stemmed): 81.75%
```

#### Task 8: Evaluate the model with the test data

After hyperparameter tuning, I found that a learning rate of 0.001 improved model performance compared to 0.0001. Additionally, using Cross-Entropy Loss proved more effective than Mean Squared Error (MSE) for our classification task. These optimized parameters will be used to evaluate the test data for both stemmed and non-stemmed datasets, allowing us to assess the model's ability to generalize to unseen data and enhancing our confidence in its predictive accuracy.

In [11]:
```
import os
import glob
import numpy as np

# Initialize lists to store the tweets and their corresponding labels
tweets_test = []
labels_test = []

# Update paths to point to test data
positive_test_path = os.path.join("C:\\Users\\marya\\Desktop\\AIT726\\Program Asisgnment\\extracted_tw
negative_test_path = os.path.join("C:\\Users\\marya\\Desktop\\AIT726\\Program Asisgnment\\extracted_tw

# Function to load data from a given path and label
def load_data_from_folder(folder_path, label):
    text_files = glob.glob(os.path.join(folder_path, '*.txt'))
    for file in text_files:
        with open(file, 'r', encoding='utf-8') as f:
            tweets_test.append(f.read())
            labels_test.append(label)

# Load positive and negative data for testing
load_data_from_folder(positive_test_path, 1)  # Label 1 for positive
load_data_from_folder(negative_test_path, 0)  # Label 0 for negative

print(f"Total tweets loaded for testing: {len(tweets_test)}")
print(f"Total labels loaded for testing: {len(labels_test)}")

# Preprocess tweets and create feature matrix for the test set
tfidf_test_stemmed, _ = calculate_tfidf(tweets_test, apply_stemming=True)
feature_matrix_test_stemmed = create_feature_matrix(tfidf_test_stemmed, vocab_with_stemming)

tfidf_test_non_stemmed, _ = calculate_tfidf(tweets_test, apply_stemming=False)
feature_matrix_test_non_stemmed = create_feature_matrix(tfidf_test_non_stemmed, vocab_without_stemming

# Convert test labels to numpy array
y_test = np.array(labels_test).reshape(-1, 1)

# Initialize results list
results = []

# Evaluate model on stemmed test data'

# Define hyperparameters for testing
hyperparams = [
    {'hidden_size': 20, 'learning_rate': 0.001, 'epochs': 1000, 'loss_function': 'cross_entropy'},
    # Add more combinations as needed
]


print("\nEvaluating Model on Stemmed Test Data")
for params in hyperparams:
    model_stemmed = FFNN(input_size=feature_matrix_test_stemmed.shape[1],
                         hidden_size=params['hidden_size'],
                         output_size=1,
                         learning_rate=params['learning_rate'],
                         epochs=params['epochs'])
    model_stemmed.train(feature_matrix_test_stemmed, y_test)
    y_pred_test_stemmed = model_stemmed.predict(feature_matrix_test_stemmed)
    metrics_stemmed = calculate_metrics(y_test, y_pred_test_stemmed)
    accuracy_stemmed = calculate_accuracy(metrics_stemmed)

    # Append results for stemmed model
    results.append(f"Confusion Metrics (Stemmed): {metrics_stemmed}")
    results.append(f"Accuracy (Stemmed): {accuracy_stemmed * 100:.2f}%")
    print(f"Confusion Metrics (Stemmed): {metrics_stemmed}")
    print(f"Accuracy (Stemmed): {accuracy_stemmed * 100:.2f}%")

# Evaluate model on non-stemmed test data
print("\nEvaluating Model on Non-Stemmed Test Data")
for params in hyperparams:
    model_non_stemmed = FFNN(input_size=feature_matrix_test_non_stemmed.shape[1],
```

```python
                                hidden_size=params['hidden_size'],
                                output_size=1,
                                learning_rate=params['learning_rate'],
                                epochs=params['epochs'])
    model_non_stemmed.train(feature_matrix_test_non_stemmed, y_test)
    y_pred_test_non_stemmed = model_non_stemmed.predict(feature_matrix_test_non_stemmed)
    metrics_non_stemmed = calculate_metrics(y_test, y_pred_test_non_stemmed)
    accuracy_non_stemmed = calculate_accuracy(metrics_non_stemmed)

    # Append results for non-stemmed model
    results.append(f"Confusion Metrics (Non-Stemmed): {metrics_non_stemmed}")
    results.append(f"Accuracy (Non-Stemmed): {accuracy_non_stemmed * 100:.2f}%")
    print(f"Confusion Metrics (Non-Stemmed): {metrics_non_stemmed}")
    print(f"Accuracy (Non-Stemmed): {accuracy_non_stemmed * 100:.2f}%")

# Save results to a .txt file
results_file_path = os.path.join("C:\\Users\\marya\\Desktop\\AIT726\\Program Asisgnment\\evaluation_re
with open(results_file_path, 'w') as results_file:
    results_file.write("\n".join(results))

print(f"Results saved to {results_file_path}")
```

```
Total tweets loaded for testing: 4182
Total labels loaded for testing: 4182
```

```
C:\Users\marya\AppData\Local\Temp\ipykernel_8224\2592623341.py:11: MarkupResemblesLocatorWarning: Th
e input looks more like a filename than markup. You may want to open this file and pass the filehand
le into Beautiful Soup.
  text = BeautifulSoup(text, "html.parser").get_text()
```

```
Evaluating Model on Stemmed Test Data
Epoch 0, Loss: 2.4260
Epoch 100, Loss: 0.5883
Epoch 200, Loss: 0.5281
Epoch 300, Loss: 0.4939
Epoch 400, Loss: 0.4654
Epoch 500, Loss: 0.4408
Epoch 600, Loss: 0.4195
Epoch 700, Loss: 0.4011
Epoch 800, Loss: 0.3850
Epoch 900, Loss: 0.3708
Confusion Metrics (Stemmed): {'tn': 2839, 'fp': 161, 'fn': 433, 'tp': 749}
Accuracy (Stemmed): 85.80%

Evaluating Model on Non-Stemmed Test Data
Epoch 0, Loss: 1.0258
Epoch 100, Loss: 0.5983
Epoch 200, Loss: 0.5513
Epoch 300, Loss: 0.5245
Epoch 400, Loss: 0.5003
Epoch 500, Loss: 0.4777
Epoch 600, Loss: 0.4572
Epoch 700, Loss: 0.4387
Epoch 800, Loss: 0.4222
Epoch 900, Loss: 0.4073
Confusion Metrics (Non-Stemmed): {'tn': 2833, 'fp': 167, 'fn': 524, 'tp': 658}
Accuracy (Non-Stemmed): 83.48%
Results saved to C:\Users\marya\Desktop\AIT726\Program Asisgnment\evaluation_results_PA2Final.txt
```

In [ ]: