2311MC04_assignment_01

February 15, 2024

[27]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
import numpy as np
import random
import matplotlib.pyplot as plt



# Function to perform 5-fold cross-validation
def k_fold_cross_validation(pairs, k=5):
    random.shuffle(pairs)
    fold_size = len(pairs) // k
    f1_macro_scores = []
    f1_micro_scores = []
    accuracy_scores = []
    precision_scores = []
    recall_scores = []
    f_score_scores = []
    confusion_matrices = []  # List to store confusion matrices

    for i in range(k):
        test_data = pairs[i * fold_size: (i + 1) * fold_size]
        train_data = pairs[:i * fold_size] + pairs[(i + 1) * fold_size:]

        # Train the model
        pi, A, B = train_model(train_data)

        # Test the model
        y_true_fold, y_pred_fold = test_model(test_data, pi, A, B)

        # Calculate F1 scores for this fold
        f1_macro_fold = f1_score(y_true_fold, y_pred_fold, average='macro')
        f1_micro_fold = f1_score(y_true_fold, y_pred_fold, average='micro')

        f1_macro_scores.append(f1_macro_fold)
```

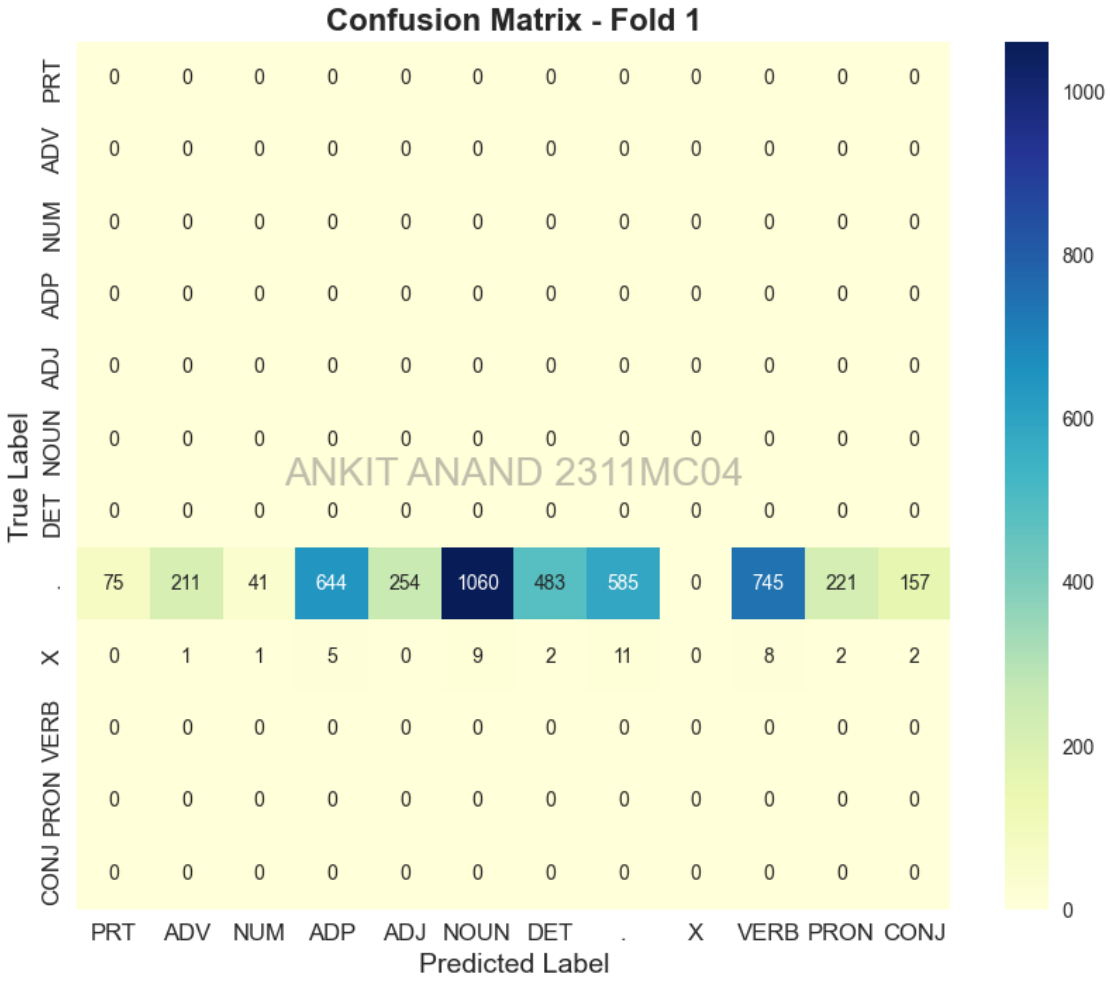
```
        f1_micro_scores.append(f1_micro_fold)

        # Calculate accuracy, precision, recall, and F-score
        accuracy = accuracy_score(y_true_fold, y_pred_fold)
        precision = precision_score(y_true_fold, y_pred_fold, average='macro',␣
↪zero_division=1)
        recall = recall_score(y_true_fold, y_pred_fold, average='macro',␣
↪zero_division=1)
        f_score = f1_score(y_true_fold, y_pred_fold, average='macro')

        # Calculate confusion matrix for this fold
        cm = confusion_matrix(y_true_fold, y_pred_fold,␣
↪labels=list(unique_tags))
        confusion_matrices.append(cm)

        accuracy_scores.append(accuracy)
        precision_scores.append(precision)
        recall_scores.append(recall)
        f_score_scores.append(f_score)

    avg_f1_macro = np.mean(f1_macro_scores)
    avg_f1_micro = np.mean(f1_micro_scores)
    avg_accuracy = np.mean(accuracy_scores)
    avg_precision = np.mean(precision_scores)
    avg_recall = np.mean(recall_scores)
    avg_f_score = np.mean(f_score_scores)

    return avg_f1_macro, avg_f1_micro, avg_accuracy, avg_precision, avg_recall,␣
↪avg_f_score, confusion_matrices



# Function to train the model
def train_model(train_data):
    # Initialize parameters
    pi = np.ones(num_tags, dtype=float)
    A = np.ones((num_tags, num_tags), dtype=float)
    B = np.ones((num_tags, num_words), dtype=float)

    prev_tag = None

    # Update parameters
    for word, tag in train_data:
        pi[tag_to_number[tag]] += 1
        if len(word) > 0:
            B[tag_to_number[tag]][word_to_number.get(word,␣
↪word_to_number['<UNK>'])] += 1
```

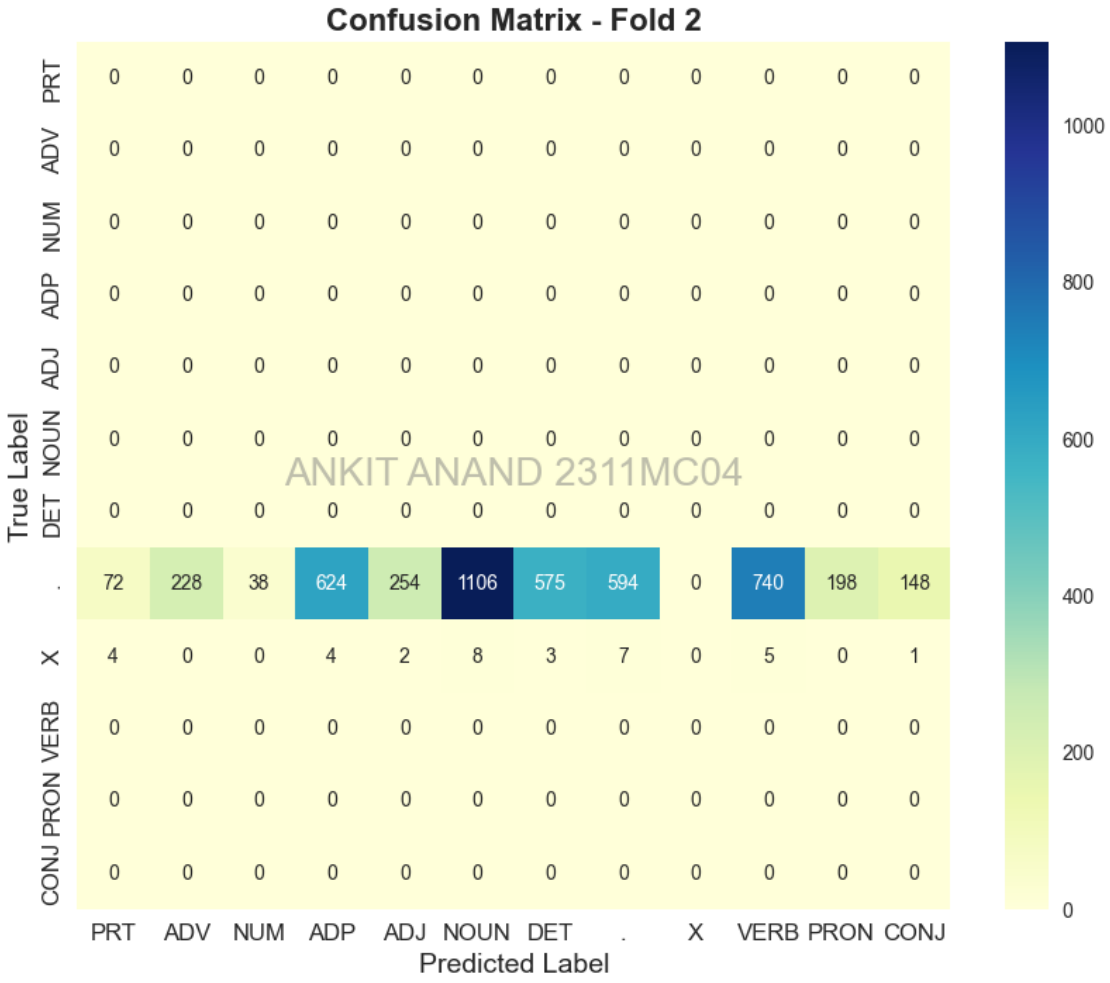
```
            if prev_tag is not None:
                A[tag_to_number[prev_tag]][tag_to_number[tag]] += 1
            prev_tag = tag

    # Normalize probabilities
    pi /= pi.sum()
    A /= A.sum(axis=1, keepdims=True)
    B /= B.sum(axis=1, keepdims=True)

    return pi, A, B

# Function to test the model
def test_model(test_data, pi, A, B):
    y_true = []
    y_pred = []

    for sentence, true_tag in test_data:
        predicted_tags = viterbi(sentence.split(), pi, A, B)
        y_true.extend(true_tag)
        y_pred.extend(predicted_tags)

    return y_true[:len(y_pred)], y_pred


# Function for Viterbi algorithm
def viterbi(sentence, pi, A, B):
    # Initialize variables
    num_tags = len(tag_to_number)
    num_words = len(word_to_number)
    delta = np.zeros((len(sentence), num_tags))
    psi = np.zeros((len(sentence), num_tags), dtype=int)

    # Initialization step
    for tag in range(num_tags):
        word_idx = word_to_number.get(sentence[0], None)
        if word_idx is None:
            word_idx = word_to_number['<UNK>']
        delta[0, tag] = pi[tag] * B[tag, word_idx]

    # Recursion step
    for t in range(1, len(sentence)):
        word_idx = word_to_number.get(sentence[t], None)
        if word_idx is None:
            word_idx = word_to_number['<UNK>']
        for curr_tag in range(num_tags):
```

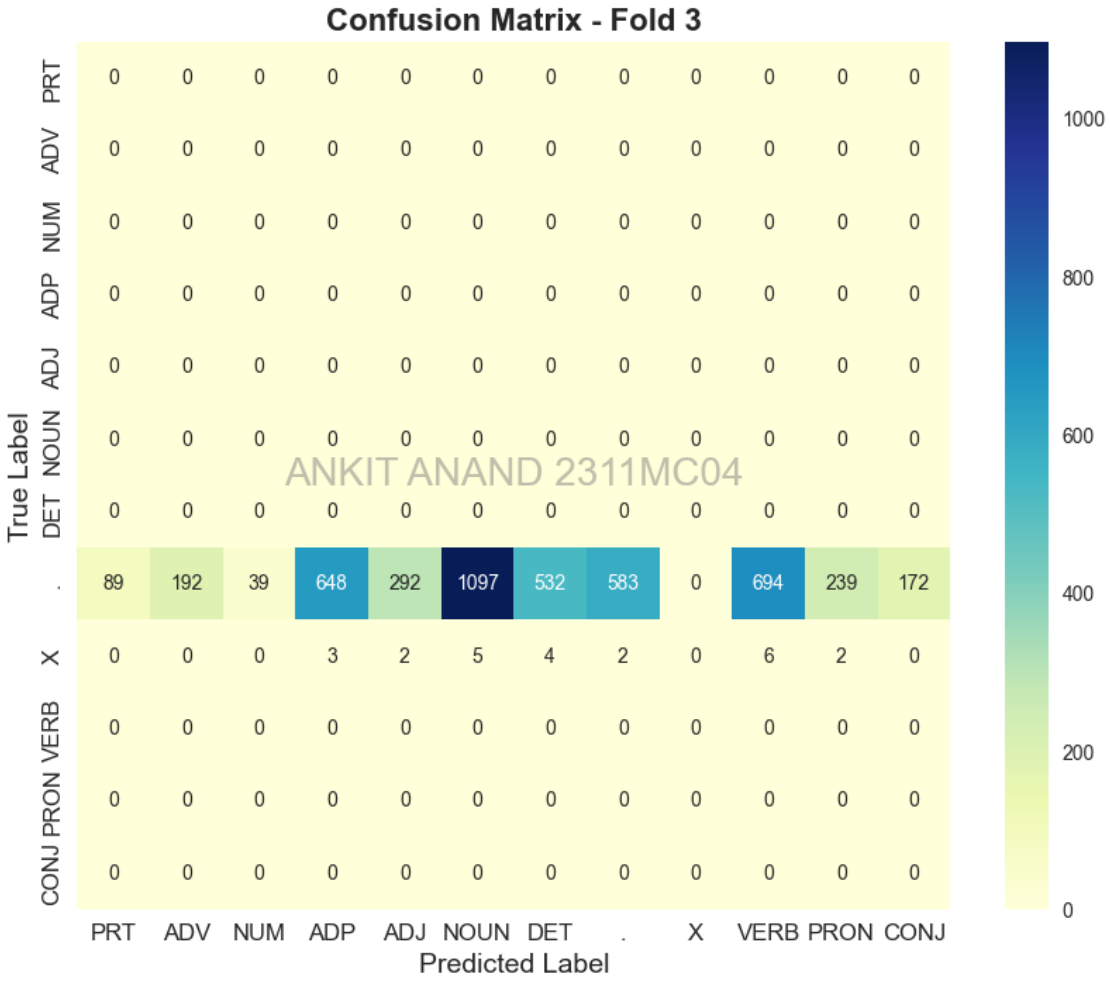
```
            delta[t, curr_tag] = np.max(delta[t-1] * A[:, curr_tag]) *␣
 ↪B[curr_tag, word_idx]
            psi[t, curr_tag] = np.argmax(delta[t-1] * A[:, curr_tag])

    # Backtracking step
    best_path = [np.argmax(delta[-1])]
    for t in range(len(sentence) - 1, 0, -1):
        best_path.append(psi[t, best_path[-1]])
    best_path.reverse()

    return [number_to_tag[tag] for tag in best_path]

# Read the content of the text file
file_path = "F:/IIT 2nd Semester/ASSIGNMENT/NLP/Brown_train.txt"
with open(file_path, 'r') as file:
    text_content = file.read()

# Split the content into lines
lines = text_content.strip().split('\n')

# Initialize a list to store word-tag pairs
pairs = []

# Parse each line to extract word-tag pairs
for line in lines:
    words_tags = line.strip().split()
    for word_tag in words_tags:
        parts = word_tag.split('/')
        if len(parts) == 2:
            word, tag = parts
            pairs.append([word, tag])

# Extract unique words and tags
unique_words = set(word for word, _ in pairs)
unique_tags = set(tag for _, tag in pairs)

# Mapping words and tags to numbers
word_to_number = {word: i for i, word in enumerate(unique_words)}
word_to_number['<UNK>'] = len(word_to_number)
tag_to_number = {tag: i for i, tag in enumerate(unique_tags)}
number_to_tag = {i: tag for tag, i in tag_to_number.items()}

# Initialize variables
num_tags = len(unique_tags)
num_words = len(unique_words)

# Perform 5-fold cross-validation
```

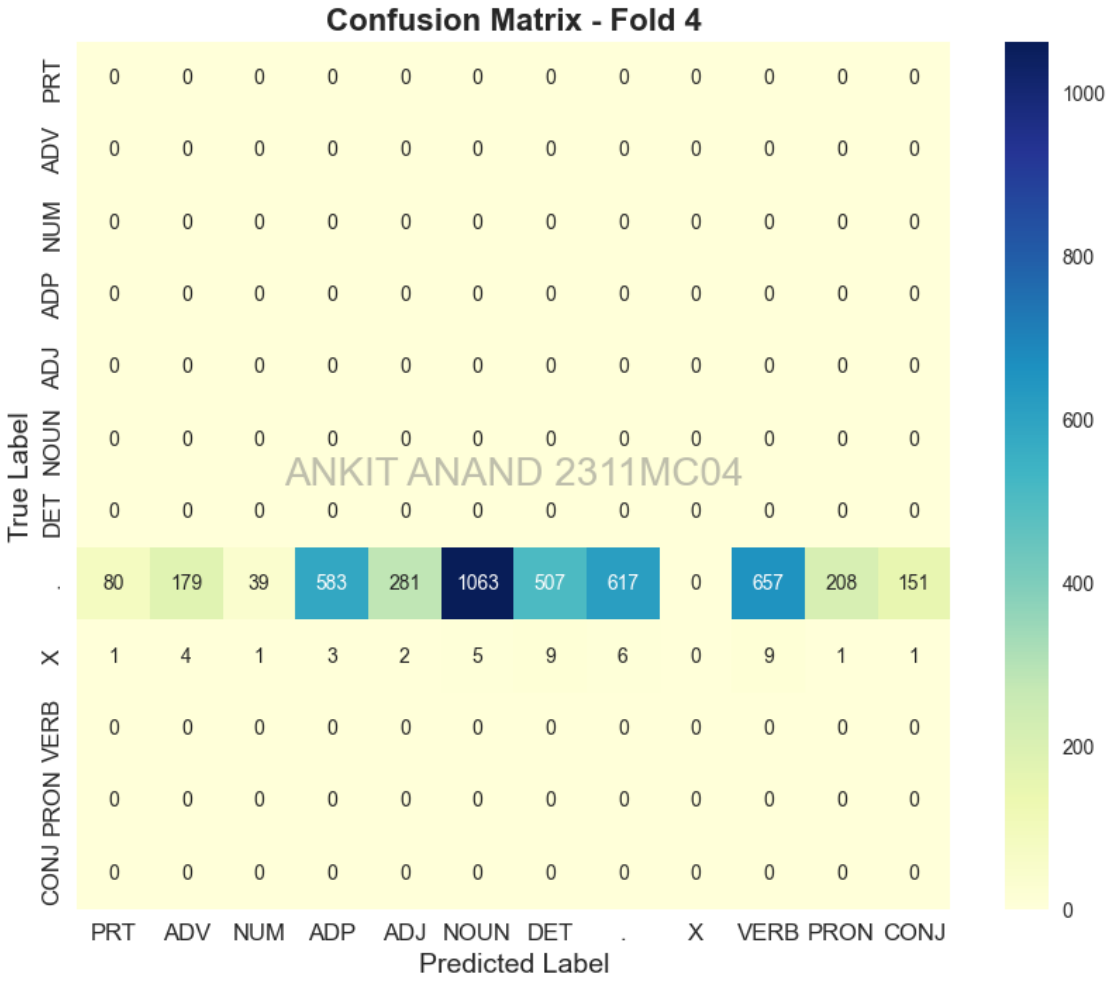
```
avg_f1_macro, avg_f1_micro, avg_accuracy, avg_precision, avg_recall,␣
 ↪avg_f_score, confusion_matrices = k_fold_cross_validation(pairs)


# Print average F1 scores and other metrics
print("Average F1 Score (Macro):", avg_f1_macro)
print("Average F1 Score (Micro):", avg_f1_micro)
print("Average Accuracy:", avg_accuracy)
print("Average Precision:", avg_precision)
print("Average Recall:", avg_recall)
print("Average F-Score:", avg_f_score)




# Plot confusion matrices with watermark
for i, cm in enumerate(confusion_matrices):
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, cmap='YlGnBu', fmt='d',␣
 ↪xticklabels=unique_tags, yticklabels=unique_tags)
    plt.title(f'Confusion Matrix - Fold {i+1}', fontsize=16, fontweight='bold')
    plt.xlabel('Predicted Label', fontsize=14)
    plt.ylabel('True Label', fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.text(0.5, 0.5, 'ANKIT ANAND 2311MC04', fontsize=20, color='gray',␣
 ↪alpha=0.5,
             ha='center', va='center', transform=plt.gca().transAxes)
    plt.show()
```

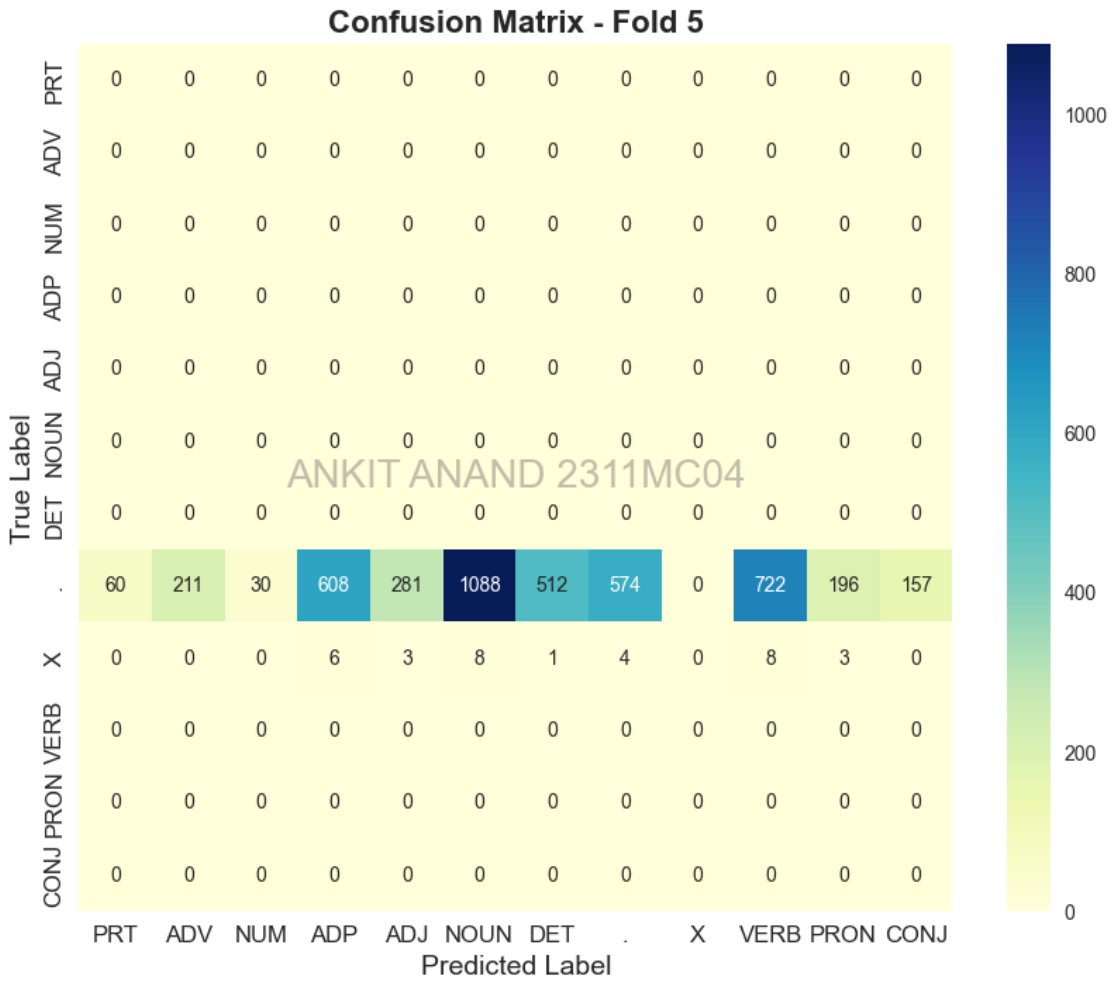
```
Average F1 Score (Macro): 0.0024159814087876207
Average F1 Score (Micro): 0.005437804990332381
Average Accuracy: 0.005437804990332381
Average Precision: 0.578509445354753
Average Recall: 0.3896808650507544
Average F-Score: 0.0024159814087876207
```


Confusion Matrix - Fold 1
True Label
PRT ADV NUM ADP ADJ NOUN DET . X VERB PRON CONJ
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
75 211 41 644 254 1060 483 585 0 745 221 157
0 1 1 5 0 9 2 11 0 8 2 2
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
ANKIT ANAND 2311MC04
1000
800
600
400
200
0
Predicted Label


Confusion Matrix - Fold 2
True Label
Predicted Label
PRT ADV NUM ADP ADJ NOUN DET . X VERB PRON CONJ
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
72 228 38 624 254 1106 575 594 0 740 198 148
4 0 0 4 2 8 3 7 0 5 0 1
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
ANKIT ANAND 2311MC04
1000
800
600
400
200
0


Confusion Matrix - Fold 3
True Label
Predicted Label
PRT ADV NUM ADP ADJ NOUN DET . X VERB PRON CONJ
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
89 192 39 648 292 1097 532 583 0 694 239 172
0 0 0 3 2 5 4 2 0 6 2 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
1000
800
600
400
200
0
ANKIT ANAND 2311MC04


Confusion Matrix - Fold 4
True Label
Predicted Label
PRT ADV NUM ADP ADJ NOUN DET . X VERB PRON CONJ
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
80 179 39 583 281 1063 507 617 0 657 208 151
1 4 1 3 2 5 9 6 0 9 1 1
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
1000
800
600
400
200
0
ANKIT ANAND 2311MC04


Confusion Matrix - Fold 5
True Label
Predicted Label
PRT ADV NUM ADP ADJ NOUN DET . X VERB PRON CONJ
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
60 211 30 608 281 1088 512 574 0 722 196 157
0 0 0 6 3 8 1 4 0 8 3 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
1000
800
600
400
200
0
ANKIT ANAND 2311MC04

[ ]: