model 1

----

imports

```
import tensorflow as tf
import keras
from keras.models import load_model
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras_tqdm import TQDMNotebookCallback
import numpy as np
np.random.seed(1337)
from keras_tqdm import TQDMNotebookCallback
import nltk
import xml.etree.ElementTree as ET
import pandas as pd
import os
import string
from nltk.tokenize import TreebankWordTokenizer
from numpy.random import random_sample
import re
import pickle
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

from keras.layers import Embedding, Flatten,LSTM, GRU
from keras.layers.convolutional import Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Activation,  Input, merge,Conv1D,MaxPooling1D,GlobalMaxPooling1D,Convolution1D
from keras import regularizers
from sklearn.metrics import precision_recall_fscore_support
from sklearn.model_selection import StratifiedKFold
import matplotlib.pyplot as plt
from keras.layers import Concatenate, concatenate
from keras import backend as K
from keras.layers import multiply
from keras.layers import merge
from keras.layers.core import *
from keras.layers.recurrent import LSTM
from keras.models import *
random_seed=1337
```

Define Callback functions to generate Measures

```
from keras import backend as K

def f1(y_true, y_pred):
    def recall(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall

    def precision(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision
    precision = precision(y_true, y_pred)
    recall = recall(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

Experiments to reproduce the results of Table 9

Load pre procssed Data

```
with open('../data/pickles/befree_3class_crawl-300d-2M.pickle', 'rb') as handle:
    gene_id_list = pickle.load(handle)
    gene_symbol_list = pickle.load(handle)
    disease_id_list = pickle.load(handle)
    X_train = pickle.load(handle)
    distance1_vectors = pickle.load(handle)
    distance2_vectors = pickle.load(handle)
    Y_train = pickle.load(handle)
    word_list = pickle.load(handle)
    word_vectors = pickle.load(handle)
    word_dict = pickle.load(handle)
    distance1_dict = pickle.load(handle)
    distance2_dict = pickle.load(handle)
    label_dict = pickle.load(handle)
    MAX_SEQUENCE_LENGTH = pickle.load(handle)
print ("word_vectors",len(word_vectors))
```

```
word_vectors 6766
```

position embedding

```
import keras
from keras_pos_embd import TrigPosEmbedding

model = keras.models.Sequential()
model.add(TrigPosEmbedding(
    input_shape=(None,),
    output_dim=20,                      # The dimension of embeddings.
    mode=TrigPosEmbedding.MODE_EXPAND,  # Use `expand` mode
    name='Pos-Embd',
))
model.compile('adam', keras.losses.mae, {})
```

```
d1_train_embedded=model.predict(distance1_vectors)
```

```
d1_train_embedded.shape
```

```
d2_train_embedded=model.predict(distance2_vectors)
```

```
d2_train_embedded.shape
```

```
(5330, 81, 20)
```

Prepare Word Embedding Layer

```
EMBEDDING_DIM=word_vectors.shape[1]
print("EMBEDDING_DIM=",EMBEDDING_DIM)
embedding_matrix=word_vectors

def create_embedding_layer(l2_reg=0.01,use_pretrained=True,is_trainable=False):

    if use_pretrained:
        return Embedding(len(word_dict) ,EMBEDDING_DIM,weights=[embedding_matrix],input_length=MAX_SEQUENCE_LENGTH,trainable=is_trainable,em

    else:
        return Embedding(len(word_dict) ,EMBEDDING_DIM,input_length=MAX_SEQUENCE_LENGTH)
```

```
EMBEDDING_DIM= 300
```

attention

```
INPUT_DIM = 2
TIME_STEPS = MAX_SEQUENCE_LENGTH
def attentionNew(inputs):
    inputs = Lambda(lambda x: tf.keras.backend.tanh(x))(inputs)
    input_dim = int(inputs.shape[2])
    a = Permute((2, 1))(inputs)
    a = Dense(TIME_STEPS, activation='softmax')(a)
    a_probs = Permute((2, 1))(a)
    output_attention_mul = multiply([inputs, a_probs])
    output_attention_mul = Lambda(lambda x: tf.keras.backend.tanh(x))(output_attention_mul)
    return output_attention_mul
```

Create the Model

```
# set parameter for metric calculation, 'macro' for multiclass classification
param='macro'
from keras.optimizers import Adam
def build_model():

    sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
    embedding_layer=create_embedding_layer(use_pretrained=True,is_trainable=False)
    embedded_sequences = embedding_layer(sequence_input)


    pos_embedd_1=Input(shape=(MAX_SEQUENCE_LENGTH,20), dtype='float32')
    pos_embedd_2=Input(shape=(MAX_SEQUENCE_LENGTH,20), dtype='float32')

    embedded_sequences = concatenate([embedded_sequences,pos_embedd_1,pos_embedd_2])


    x = Conv1D(64, 5, activation='relu')(embedded_sequences)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.4)(x)
    conv_sequence_w5=GlobalMaxPooling1D()(x)    #x = Flatten()(x)



    x = Conv1D(128, 3, activation='relu')(embedded_sequences)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.4)(x)
    conv_sequence_w4=GlobalMaxPooling1D()(x)    #x = Flatten()(x)



    x = Conv1D(256, 3, activation='relu')(embedded_sequences)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.4)(x)
    conv_sequence_w3=GlobalMaxPooling1D()(x)    #x = Flatten()(x)

    forward = GRU(100, recurrent_dropout=0.1, return_sequences=True)(embedded_sequences)
    backward = LSTM(100, go_backwards=True, recurrent_dropout=0.1, return_sequences=True)(embedded_sequences)
    lstm_gru_sequence = concatenate([forward, backward], axis=-1)
    # Apply attention mechanism
    attention_output = attentionNew(lstm_gru_sequence)  # Shape: (None, MAX_SEQUENCE_LENGTH, 200)
    attention_pooled = GlobalMaxPooling1D()(attention_output)  # Shape: (None, 200)

    merge = concatenate([conv_sequence_w5,conv_sequence_w4,conv_sequence_w3,attention_pooled])
    x = Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.05))(merge)
    x = Dropout(0.4)(x)
    preds = Dense(3, activation='softmax')(x)
    model = Model(inputs=[sequence_input, pos_embedd_1,pos_embedd_2],outputs=preds)
    opt=tf.keras.optimizers.Adam(learning_rate=0.001)
    model.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['acc',f1])
    return model
```

```
model = build_model()
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           (None, 81)           0
__________________________________________________________________________________________________
```

```
 embedding_1 (Embedding)         (None, 81, 300)      2029800     input_1[0][0]
 ____________________________________________________________________________________________________
 input_2 (InputLayer)            (None, 81, 20)       0
 ____________________________________________________________________________________________________
 input_3 (InputLayer)            (None, 81, 20)       0
 ____________________________________________________________________________________________________
 concatenate_1 (Concatenate)     (None, 81, 340)      0           embedding_1[0][0]
                                                                  input_2[0][0]
                                                                  input_3[0][0]
 ____________________________________________________________________________________________________
 gru_1 (GRU)                     (None, 81, 100)      132300      concatenate_1[0][0]
 ____________________________________________________________________________________________________
 lstm_1 (LSTM)                   (None, 81, 100)      176400      concatenate_1[0][0]
 ____________________________________________________________________________________________________
 concatenate_2 (Concatenate)     (None, 81, 200)      0           gru_1[0][0]
                                                                  lstm_1[0][0]
 ____________________________________________________________________________________________________
 lambda_1 (Lambda)               (None, 81, 200)      0           concatenate_2[0][0]
 ____________________________________________________________________________________________________
 permute_1 (Permute)             (None, 200, 81)      0           lambda_1[0][0]
 ____________________________________________________________________________________________________
 dense_1 (Dense)                 (None, 200, 81)      6642        permute_1[0][0]
 ____________________________________________________________________________________________________
 conv1d_1 (Conv1D)               (None, 77, 64)       108864      concatenate_1[0][0]
 ____________________________________________________________________________________________________
 conv1d_2 (Conv1D)               (None, 79, 128)      130688      concatenate_1[0][0]
 ____________________________________________________________________________________________________
 conv1d_3 (Conv1D)               (None, 79, 256)      261376      concatenate_1[0][0]
 ____________________________________________________________________________________________________
 permute_2 (Permute)             (None, 81, 200)      0           dense_1[0][0]
 ____________________________________________________________________________________________________
 max_pooling1d_1 (MaxPooling1D)  (None, 25, 64)       0           conv1d_1[0][0]
 ____________________________________________________________________________________________________
 max_pooling1d_2 (MaxPooling1D)  (None, 26, 128)      0           conv1d_2[0][0]
 ____________________________________________________________________________________________________
 max_pooling1d_3 (MaxPooling1D)  (None, 26, 256)      0           conv1d_3[0][0]
 ____________________________________________________________________________________________________
 multiply_1 (Multiply)           (None, 81, 200)      0           lambda_1[0][0]
                                                                  permute_2[0][0]
 ____________________________________________________________________________________________________
 dropout_1 (Dropout)             (None, 25, 64)       0           max_pooling1d_1[0][0]
 ____________________________________________________________________________________________________
 dropout_2 (Dropout)             (None, 26, 128)      0           max_pooling1d_2[0][0]
 ____________________________________________________________________________________________________
 dropout_3 (Dropout)             (None, 26, 256)      0           max_pooling1d_3[0][0]
 ____________________________________________________________________________________________________
 lambda_2 (Lambda)               (None, 81, 200)      0           multiply_1[0][0]
 ____________________________________________________________________________________________________
 global_max_pooling1d_1 (GlobalM (None, 64)           0           dropout_1[0][0]
 ____________________________________________________________________________________________________
 global_max_pooling1d_2 (GlobalM (None, 128)          0           dropout_2[0][0]
 ____________________________________________________________________________________________________
```

```
validation_split_rate = 0.1
skf = StratifiedKFold(n_splits=5,shuffle=True, random_state=42)
Y = [np.argmax(y, axis=None, out=None) for y in Y_train]
for tr_index, te_index in skf.split(X_train,Y):
    test_index = te_index
    train_index = tr_index

trainRate = (len(train_index)/len(Y))*100
testRate = (len(test_index)/len(Y))*100
print ("TrainRate:{:.2f}% testRate:{:.2f}% validation:{:.2f}%  ".format(trainRate,testRate, trainRate*validation_split_rate))
X_train, X_test = X_train[train_index], X_train[test_index]
pos_train1, pos_test1 = d1_train_embedded[train_index], d1_train_embedded[test_index]
pos_train2, pos_test2 = d2_train_embedded[train_index], d2_train_embedded[test_index]
y_train, y_test = Y_train[train_index], Y_train[test_index]



# # Saving the training data split as a pickle file
# training_data = {
#     'X_train': X_train,
#     'pos_train1': pos_train1,
#     'pos_train2': pos_train2,
#     'y_train': y_train
# }

# with open('training_data.pkl', 'wb') as f:
#     pickle.dump(training_data, f)
```

```
# # Saving the testing data split as a pickle file
# testing_data = {
#     'X_test': X_test,
#     'pos_test1': pos_test1,
#     'pos_test2': pos_test2,
#     'y_test': y_test
# }

# with open('testing_data.pkl', 'wb') as f:
#     pickle.dump(testing_data, f)
```

```
TrainRate:80.00% testRate:20.00% validation:8.00%
```

```
# Load the training data from the pickle file
with open('training_data.pkl', 'rb') as f:
    train_data = pickle.load(f)

# Load the testing data from the pickle file
with open('testing_data.pkl', 'rb') as f:
    test_data = pickle.load(f)

# Extract data from the loaded dictionaries
X_train = train_data['X_train']
pos_train1 = train_data['pos_train1']
pos_train2 = train_data['pos_train2']
y_train = train_data['y_train']


X_test = test_data['X_test']
pos_test1 = test_data['pos_test1']
pos_test2 = test_data['pos_test2']
y_test = test_data['y_test']
print(X_train.shape)
print(X_test.shape)
```

```
(4264, 81)
(1066, 81)
```

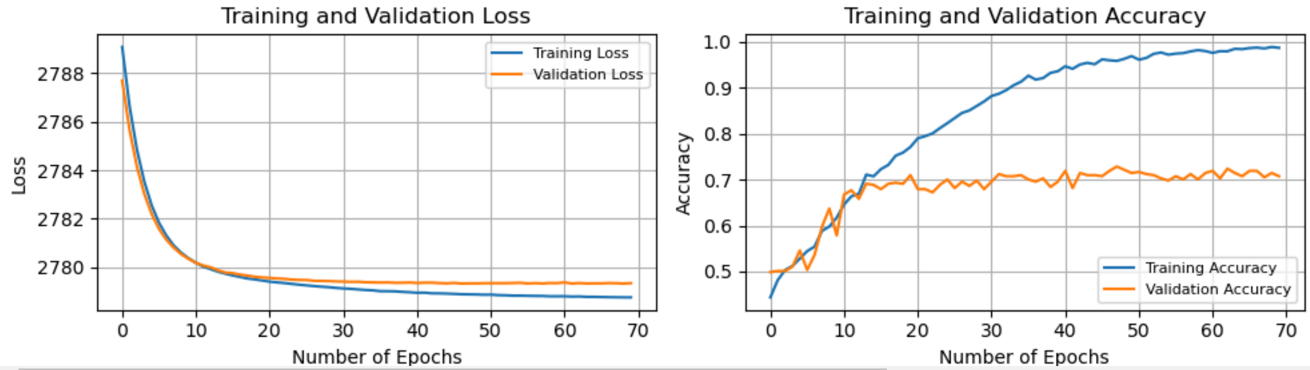
Run the Evaluation on the test dataset

```
MaxEpochs =70
batchsize =32
validation_split_rate = 0.1
history=model.fit([X_train,pos_train1,pos_train2], y_train,validation_split=validation_split_rate ,epochs=MaxEpochs, batch_size=batchsize,ve
```

```
Train on 3837 samples, validate on 427 samples
Epoch 1/70
3837/3837 [==============================] - 13s 3ms/step - loss: 2789.0856 - acc: 0.4438 - f1: 0.3066 - val_loss: 2787.7002 - val_ac
Epoch 2/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2786.6357 - acc: 0.4814 - f1: 0.2908 - val_loss: 2785.6618 - val_ac
Epoch 3/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2784.8455 - acc: 0.5033 - f1: 0.3376 - val_loss: 2784.1454 - val_ac
Epoch 4/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2783.5244 - acc: 0.5121 - f1: 0.3588 - val_loss: 2783.0171 - val_ac
Epoch 5/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2782.5451 - acc: 0.5283 - f1: 0.4026 - val_loss: 2782.1919 - val_ac
Epoch 6/70
3837/3837 [==============================] - 11s 3ms/step - loss: 2781.8217 - acc: 0.5444 - f1: 0.4170 - val_loss: 2781.5689 - val_ac
Epoch 7/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2781.2928 - acc: 0.5543 - f1: 0.4425 - val_loss: 2781.1235 - val_ac
Epoch 8/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2780.8872 - acc: 0.5882 - f1: 0.4786 - val_loss: 2780.7930 - val_ac
Epoch 9/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2780.5922 - acc: 0.5979 - f1: 0.5030 - val_loss: 2780.5321 - val_ac
Epoch 10/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2780.3610 - acc: 0.6171 - f1: 0.5306 - val_loss: 2780.3397 - val_ac
Epoch 11/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2780.1752 - acc: 0.6458 - f1: 0.5696 - val_loss: 2780.1740 - val_ac
Epoch 12/70
3837/3837 [==============================] - 13s 3ms/step - loss: 2780.0211 - acc: 0.6635 - f1: 0.6071 - val_loss: 2780.0620 - val_ac
Epoch 13/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.9138 - acc: 0.6690 - f1: 0.6268 - val_loss: 2779.9890 - val_ac
```

```
Epoch 14/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.8075 - acc: 0.7107 - f1: 0.6636 - val_loss: 2779.8629 - val_ac
Epoch 15/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.7264 - acc: 0.7071 - f1: 0.6789 - val_loss: 2779.7740 - val_ac
Epoch 16/70
3837/3837 [==============================] - 13s 3ms/step - loss: 2779.6514 - acc: 0.7227 - f1: 0.7003 - val_loss: 2779.7480 - val_ac
Epoch 17/70
3837/3837 [==============================] - 13s 3ms/step - loss: 2779.5906 - acc: 0.7318 - f1: 0.7127 - val_loss: 2779.6872 - val_ac
Epoch 18/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.5276 - acc: 0.7519 - f1: 0.7351 - val_loss: 2779.6462 - val_ac
Epoch 19/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.4892 - acc: 0.7587 - f1: 0.7358 - val_loss: 2779.5992 - val_ac
Epoch 20/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.4434 - acc: 0.7709 - f1: 0.7555 - val_loss: 2779.5709 - val_ac
Epoch 21/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.3967 - acc: 0.7897 - f1: 0.7761 - val_loss: 2779.5504 - val_ac
Epoch 22/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.3670 - acc: 0.7944 - f1: 0.7824 - val_loss: 2779.5271 - val_ac
Epoch 23/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.3390 - acc: 0.8004 - f1: 0.7905 - val_loss: 2779.5123 - val_ac
Epoch 24/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.3051 - acc: 0.8118 - f1: 0.8016 - val_loss: 2779.4807 - val_ac
Epoch 25/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.2698 - acc: 0.8225 - f1: 0.8135 - val_loss: 2779.4688 - val_ac
Epoch 26/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.2440 - acc: 0.8337 - f1: 0.8287 - val_loss: 2779.4643 - val_ac
Epoch 27/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.2144 - acc: 0.8449 - f1: 0.8394 - val_loss: 2779.4307 - val_ac
Epoch 28/70
3837/3837 [==============================] - 12s 3ms/step - loss: 2779.1906 - acc: 0.8504 - f1: 0.8444 - val_loss: 2779.4263 - val_ac
```

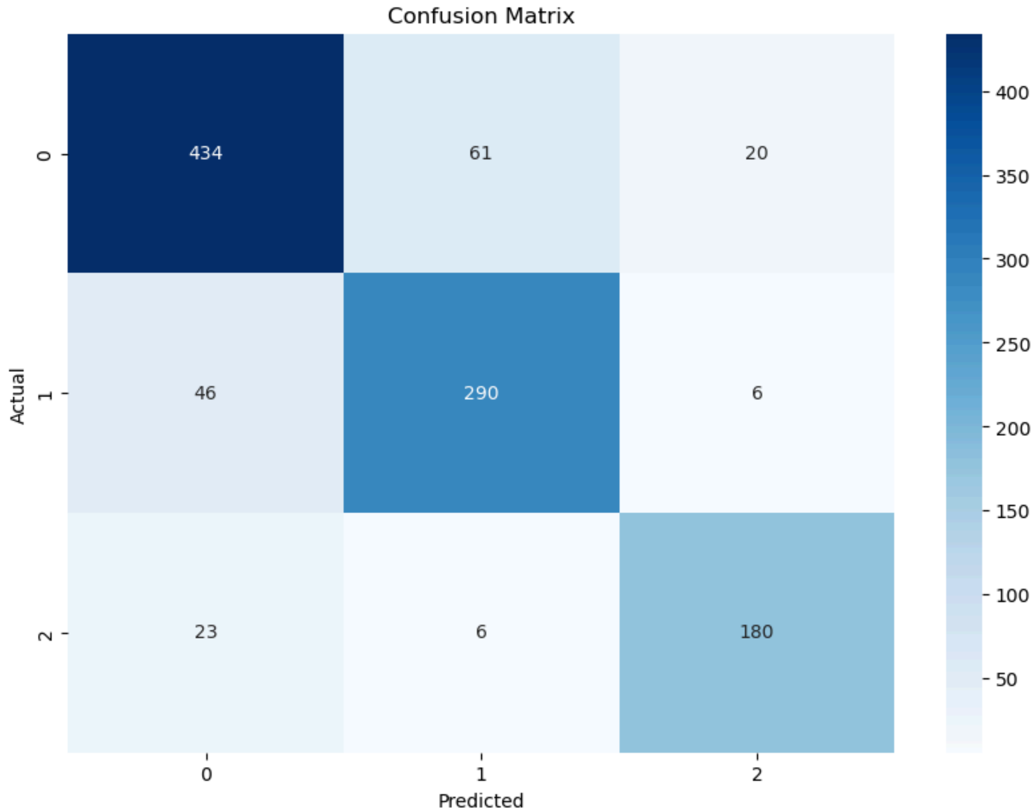
```python
import matplotlib.pyplot as plt

# Training & Validation accuracy
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['acc']
val_acc = history.history['val_acc']
epochs = len(train_loss)

xc = range(epochs)

plt.figure(figsize=(10, 3))

# Loss subplot
plt.subplot(1, 2, 1)
plt.plot(xc, train_loss, label='Training Loss')
plt.plot(xc, val_loss, label='Validation Loss')
plt.xlabel('Number of Epochs', fontsize=10)
plt.ylabel('Loss', fontsize=10)
plt.title('Training and Validation Loss', fontsize=12)
plt.legend(fontsize=8)
plt.grid(True)

# Accuracy subplot
plt.subplot(1, 2, 2)
plt.plot(xc, train_acc, label='Training Accuracy')
plt.plot(xc, val_acc, label='Validation Accuracy')
plt.xlabel('Number of Epochs', fontsize=10)
plt.ylabel('Accuracy', fontsize=10)
plt.title('Training and Validation Accuracy', fontsize=12)
plt.legend(fontsize=8, loc='lower right')  # Change position to lower right
plt.grid(True)

plt.tight_layout()
plt.show()
```




```
predicted = np.argmax(model.predict([X_test,pos_test1,pos_test2]), axis=1)
y_test_to_label = np.argmax(y_test, axis=1)
prec, reca, fscore, sup = precision_recall_fscore_support(y_test_to_label, predicted, average=param)
# Generate the classification report as a dictionary
report_dict = classification_report(y_test_to_label, predicted, output_dict=True)

# Create a new dictionary to hold the formatted values
formatted_report_dict = {}

# Iterate over the items in the report dictionary
for key, value in report_dict.items():
    if isinstance(value, dict):
        # Format the nested dictionary values
        formatted_report_dict[key] = {sub_key: f"{sub_value:.4f}" for sub_key, sub_value in value.items()}
    else:
        # Format the top-level dictionary values
        formatted_report_dict[key] = f"{value:.4f}"

# Create a string representation of the formatted dictionary
formatted_report_str = classification_report(y_test_to_label, predicted, digits=4)

# Print the formatted classification report
print(formatted_report_str)
print(" Precision:{:.2f}% Recall:{:.2f}% Fscore:{:.2f}% ".format(prec*100, reca*100, fscore*100))
```

```
              precision    recall  f1-score   support

           0     0.8628    0.8427    0.8527       515
           1     0.8123    0.8480    0.8298       342
           2     0.8738    0.8612    0.8675       209

    accuracy                         0.8480      1066
   macro avg     0.8496    0.8506    0.8500      1066
weighted avg     0.8488    0.8480    0.8482      1066

 Precision:84.96% Recall:85.06% Fscore:85.00%
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix, precision_recall_fscore_support
from sklearn.model_selection import StratifiedKFold
# Calculate and visualize the confusion matrix
cm = confusion_matrix(y_test_to_label, predicted)
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['0', '1','2'], yticklabels=['0', '1','2'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

# Print precision, recall, and f-score
prec, reca, fscore, sup = precision_recall_fscore_support(y_test_to_label, predicted, average=param)
print(" Precision:{:.2f}% Recall:{:.2f}% Fscore:{:.2f}% ".format(prec*100, reca*100, fscore*100))
```



Start coding or generate with AI.