

BeFree-2class_EUADR LSTM

September 17, 2024

1 Evaluation using the BeFree corpus

1.0.1 EUADR dataset

The EU-ADR dataset contains annotations on drugs, diseases, genes and proteins, and associations between them. In this study, we used only GDAs to evaluate the method. Each association is classified according to its level of certainty as positive association (PA), negative association (NA), speculative association (SA); or false association (FA). The EU-ADR corpus is based on 100 MEDLINE abstracts for each association set, and its annotation was conducted by three experts.

2

3 imports

```
[1]: import tensorflow as tf
from keras.models import load_model
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras_tqdm import TQDMNotebookCallback
import numpy as np
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
from keras.layers.convolutional import Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical
from 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 *

```

Using TensorFlow backend.

3.0.1 Define Callback functions to generate Measures

```

[2]: 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()))

```

4 Experiments to reproduce the results of Table 8

4.0.1 Load Prerocessed Data

```

[3]: with open('../data/pickles/befree_EUADR_2class_PubMed-and-PMC-w2v.pickle',
    'rb') as handle:

    W_train = pickle.load(handle)
    print("W_train", len(W_train))
    d1_train = pickle.load(handle)
    print("d1_train", len(d1_train))

```

```

d2_train = pickle.load(handle)
print("d2_train", len(d2_train))
Y_train = pickle.load(handle)
print("Y_train", len(d2_train))
Tr_word_list = pickle.load(handle)
print("Tr_word_list", len(d2_train))
word_vectors = pickle.load(handle)
print("word_vectors", len(word_vectors))
word_dict = pickle.load(handle)
print("word_dict", len(word_dict))
d1_dict = pickle.load(handle)
print("d1_dict", len(d1_dict))
d2_dict = pickle.load(handle)
print("d2_dict", len(d2_dict))
label_dict = pickle.load(handle)
print("label_dict", len(label_dict))
MAX_SEQUENCE_LENGTH = pickle.load(handle)
print("MAX_SEQUENCE_LENGTH", MAX_SEQUENCE_LENGTH)

```

```

W_train 355
d1_train 355
d2_train 355
Y_train 355
Tr_word_list 355
word_vectors 1355
word_dict 1355
d1_dict 169
d2_dict 171
label_dict 4
MAX_SEQUENCE_LENGTH 102

```

4.0.2 Create Position Embedding Vectors

```

[4]: import keras
from keras_pos_embd import TrigPosEmbedding

model = keras.models.Sequential()
model.add(TrigPosEmbedding(
    input_shape=(None,),
    output_dim=20,
    mode=TrigPosEmbedding.MODE_EXPAND,
    name='Pos-Embd',
))
model.compile('adam', keras.losses.mae, {})
model.summary()

d1_train_embedded=model.predict(d1_train)

```

```
d1_train_embedded.shape

d2_train_embedded=model.predict(d2_train)

d2_train_embedded.shape
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
Pos-Embd (TrigPosEmbedding)	(None, None, 20)	0

Total params: 0
 Trainable params: 0
 Non-trainable params: 0

[4]: (355, 102, 20)

4.0.3 Prepare Word Embedding Layer

```
[5]: EMBEDDING_DIM=word_vectors.shape[1]
embedding_matrix=word_vectors
print(EMBEDDING_DIM)
print(len(embedding_matrix))

param='binary'

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,
        ↪l2(l2_reg))

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

200
1355

4.0.4 Prepare Attention Mechanism

```
[6]: INPUT_DIM = 2
TIME_STEPS = MAX_SEQUENCE_LENGTH

[7]: def attentionNew(inputs):
    inputs = Lambda(lambda x: tf.keras.backend.sigmoid(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.
    ↪sigmoid(x))(output_attention_mul)
    return output_attention_mul
```

4.0.5 Create the Model

```
[8]: dropRate=0.1
param='binary'
def build_model_positionAttention():

    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(128, 7, activation='relu')(embedded_sequences)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.5)(x)

    x = Conv1D(64, 5, activation='relu')(x)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.5)(x)
    conv_sequence_7=GlobalMaxPooling1D()(x)    #x = Flatten()(x)

    forward = LSTM(100, recurrent_dropout=0.05)(embedded_sequences)
```

```

        backward = LSTM(100, go_backwards=True, recurrent_dropout=0.
↪05)(embedded_sequences)
        lstm_sequence = concatenate([forward, backward])

        merge = concatenate([conv_sequence_7, lstm_sequence])

        x = Dropout(dropRate)(x)
        x = Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.
↪05))(merge)
        x = Dropout(0.1)(x)
        preds = Dense(2, activation='softmax')(x)
        model = Model(inputs=[sequence_input, ↪
↪pos_embedd_1, pos_embedd_2], outputs=preds)
        model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc', f1])

        #model.summary()
        return model

```

```

[9]: model = build_model_positionAttention()
      model.summary()

```

Model: "model_1"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 102)	0	
embedding_1 (Embedding)	(None, 102, 200)	271000	input_1[0][0]
input_2 (InputLayer)	(None, 102, 20)	0	
input_3 (InputLayer)	(None, 102, 20)	0	
concatenate_1 (Concatenate)	(None, 102, 240)	0	input_2[0][0] input_3[0][0]

conv1d_1 (Conv1D)	(None, 96, 128)	215168	
concatenate_1[0][0]			
max_pooling1d_1 (MaxPooling1D)	(None, 32, 128)	0	conv1d_1[0][0]
dropout_1 (Dropout)	(None, 32, 128)	0	
max_pooling1d_1[0][0]			
conv1d_2 (Conv1D)	(None, 28, 64)	41024	dropout_1[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 9, 64)	0	conv1d_2[0][0]
dropout_2 (Dropout)	(None, 9, 64)	0	
max_pooling1d_2[0][0]			
lstm_1 (LSTM)	(None, 100)	136400	
concatenate_1[0][0]			
lstm_2 (LSTM)	(None, 100)	136400	
concatenate_1[0][0]			
global_max_pooling1d_1 (GlobalM	(None, 64)	0	dropout_2[0][0]
concatenate_2 (Concatenate)	(None, 200)	0	lstm_1[0][0] lstm_2[0][0]
concatenate_3 (Concatenate)	(None, 264)	0	
global_max_pooling1d_1[0][0]			
concatenate_2[0][0]			
dense_1 (Dense)	(None, 256)	67840	
concatenate_3[0][0]			
dropout_4 (Dropout)	(None, 256)	0	dense_1[0][0]

```
-----
dense_2 (Dense)                (None, 2)                514                dropout_4[0][0]
=====
```

```
=====
Total params: 868,346
Trainable params: 597,346
Non-trainable params: 271,000
-----
-----
```

```
[ ]:
```

4.0.6 Run the Evaluation using 10 fold Cross Validation

```
[10]: validation_split_rate=0.1
skf = StratifiedKFold(n_splits=5, random_state=None)
Y = [np.argmax(y, axis=None, out=None) for y in Y_train]
#print(len(Y))
all_histories=[]
for trI, teI in skf.split(W_train,Y):
    train_index =trI
    test_index  =teI
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))
#print(train_index, test_index)
X_train, X_test = W_train[train_index], W_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%

```

[11]: # 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)

```

(284, 102)

```

[12]: epochs =50
batch_size =32
validation_split_rate=0.1
history=model.fit([X_train,pos_train1,pos_train2],
    ↪y_train,validation_split=validation_split_rate ,epochs=epochs,
    ↪batch_size=batch_size,verbose=1)

```

Train on 255 samples, validate on 29 samples

Epoch 1/50

255/255 [=====] - 522s 2s/step - loss: 586.8758 - acc:
0.6235 - f1: 0.6226 - val_loss: 585.3108 - val_acc: 0.0690 - val_f1: 0.0690

Epoch 2/50
255/255 [=====] - 18s 70ms/step - loss: 584.6866 - acc:
0.4510 - f1: 0.4512 - val_loss: 582.9426 - val_acc: 0.9310 - val_f1: 0.9310

Epoch 3/50
255/255 [=====] - 47s 186ms/step - loss: 582.7049 -
acc: 0.6314 - f1: 0.6307 - val_loss: 581.7048 - val_acc: 0.8621 - val_f1: 0.8621

Epoch 4/50
255/255 [=====] - 29s 115ms/step - loss: 581.2989 -
acc: 0.4706 - f1: 0.4714 - val_loss: 580.2935 - val_acc: 0.9310 - val_f1: 0.9310

Epoch 5/50
255/255 [=====] - 55s 215ms/step - loss: 580.0127 -
acc: 0.6588 - f1: 0.6589 - val_loss: 579.2651 - val_acc: 0.9310 - val_f1: 0.9310

Epoch 6/50
255/255 [=====] - 28s 110ms/step - loss: 579.0379 -
acc: 0.6157 - f1: 0.6154 - val_loss: 578.3752 - val_acc: 0.9310 - val_f1: 0.9310

Epoch 7/50
255/255 [=====] - 23s 89ms/step - loss: 578.1911 - acc:
0.6549 - f1: 0.6547 - val_loss: 577.6010 - val_acc: 0.9310 - val_f1: 0.9310

Epoch 8/50
255/255 [=====] - 5s 21ms/step - loss: 577.5770 - acc:
0.6314 - f1: 0.6307 - val_loss: 577.0852 - val_acc: 0.9310 - val_f1: 0.9310

Epoch 9/50
255/255 [=====] - 2s 8ms/step - loss: 577.0397 - acc:
0.6314 - f1: 0.6313 - val_loss: 576.6284 - val_acc: 0.9310 - val_f1: 0.9310

Epoch 10/50
255/255 [=====] - 2s 8ms/step - loss: 576.6443 - acc:
0.6745 - f1: 0.6738 - val_loss: 576.3329 - val_acc: 0.8621 - val_f1: 0.8621

Epoch 11/50
255/255 [=====] - 2s 8ms/step - loss: 576.2826 - acc:
0.6745 - f1: 0.6741 - val_loss: 575.8536 - val_acc: 0.9310 - val_f1: 0.9310

Epoch 12/50
255/255 [=====] - 2s 6ms/step - loss: 576.1141 - acc:
0.6392 - f1: 0.6394 - val_loss: 575.8679 - val_acc: 0.8621 - val_f1: 0.8621

Epoch 13/50
255/255 [=====] - 2s 7ms/step - loss: 575.8180 - acc:
0.7020 - f1: 0.7016 - val_loss: 575.5275 - val_acc: 0.8966 - val_f1: 0.8966

Epoch 14/50
255/255 [=====] - 2s 6ms/step - loss: 575.5907 - acc:
0.7137 - f1: 0.7140 - val_loss: 575.3867 - val_acc: 0.8966 - val_f1: 0.8966

Epoch 15/50
255/255 [=====] - 2s 9ms/step - loss: 575.4940 - acc:
0.7216 - f1: 0.7216 - val_loss: 575.1622 - val_acc: 0.9310 - val_f1: 0.9310

Epoch 16/50
255/255 [=====] - 3s 12ms/step - loss: 575.3880 - acc:
0.7020 - f1: 0.7031 - val_loss: 575.0663 - val_acc: 0.9310 - val_f1: 0.9310

Epoch 17/50
255/255 [=====] - 2s 9ms/step - loss: 575.3626 - acc:
0.7098 - f1: 0.7089 - val_loss: 575.3945 - val_acc: 0.5862 - val_f1: 0.5862

Epoch 18/50
255/255 [=====] - 2s 8ms/step - loss: 575.3889 - acc:
0.6510 - f1: 0.6512 - val_loss: 574.9982 - val_acc: 0.9310 - val_f1: 0.9310

Epoch 19/50
255/255 [=====] - 2s 6ms/step - loss: 575.3346 - acc:
0.7098 - f1: 0.7092 - val_loss: 575.1613 - val_acc: 0.6552 - val_f1: 0.6552

Epoch 20/50
255/255 [=====] - 2s 7ms/step - loss: 575.1409 - acc:
0.7137 - f1: 0.7136 - val_loss: 574.8128 - val_acc: 0.8966 - val_f1: 0.8966

Epoch 21/50
255/255 [=====] - 2s 7ms/step - loss: 575.0511 - acc:
0.7412 - f1: 0.7409 - val_loss: 574.9236 - val_acc: 0.7931 - val_f1: 0.7931

Epoch 22/50
255/255 [=====] - 2s 8ms/step - loss: 574.9288 - acc:
0.7725 - f1: 0.7729 - val_loss: 574.8499 - val_acc: 0.7931 - val_f1: 0.7931

Epoch 23/50
255/255 [=====] - 2s 7ms/step - loss: 574.8660 - acc:
0.7765 - f1: 0.7768 - val_loss: 574.7311 - val_acc: 0.8276 - val_f1: 0.8276

Epoch 24/50
255/255 [=====] - 2s 7ms/step - loss: 574.7966 - acc:
0.7922 - f1: 0.7923 - val_loss: 574.6774 - val_acc: 0.8621 - val_f1: 0.8621

Epoch 25/50
255/255 [=====] - 2s 7ms/step - loss: 574.6938 - acc:
0.8392 - f1: 0.8388 - val_loss: 574.6389 - val_acc: 0.7931 - val_f1: 0.7931

Epoch 26/50
255/255 [=====] - 2s 7ms/step - loss: 574.7284 - acc:
0.8196 - f1: 0.8202 - val_loss: 574.7734 - val_acc: 0.7586 - val_f1: 0.7586

Epoch 27/50
255/255 [=====] - 2s 7ms/step - loss: 574.6313 - acc:
0.8863 - f1: 0.8863 - val_loss: 574.7174 - val_acc: 0.7586 - val_f1: 0.7586

Epoch 28/50
255/255 [=====] - 13s 52ms/step - loss: 574.5854 - acc:
0.8627 - f1: 0.8630 - val_loss: 574.6096 - val_acc: 0.7931 - val_f1: 0.7931

Epoch 29/50
255/255 [=====] - 2s 7ms/step - loss: 574.5237 - acc:
0.8941 - f1: 0.8942 - val_loss: 574.6714 - val_acc: 0.7931 - val_f1: 0.7931

Epoch 30/50
255/255 [=====] - 7s 27ms/step - loss: 574.5090 - acc:
0.9020 - f1: 0.9017 - val_loss: 574.5762 - val_acc: 0.7931 - val_f1: 0.7931

Epoch 31/50
255/255 [=====] - 2s 7ms/step - loss: 574.4359 - acc:
0.9059 - f1: 0.9052 - val_loss: 574.6773 - val_acc: 0.7586 - val_f1: 0.7586

Epoch 32/50
255/255 [=====] - 4s 16ms/step - loss: 574.3591 - acc:
0.9373 - f1: 0.9372 - val_loss: 574.6636 - val_acc: 0.7931 - val_f1: 0.7931

Epoch 33/50
255/255 [=====] - 4s 14ms/step - loss: 574.3912 - acc:
0.9059 - f1: 0.9059 - val_loss: 574.5253 - val_acc: 0.7586 - val_f1: 0.7586

Epoch 34/50
255/255 [=====] - 2s 7ms/step - loss: 574.4087 - acc:
0.8784 - f1: 0.8785 - val_loss: 574.4338 - val_acc: 0.8276 - val_f1: 0.8276
Epoch 35/50
255/255 [=====] - 2s 9ms/step - loss: 574.2980 - acc:
0.9412 - f1: 0.9412 - val_loss: 574.4556 - val_acc: 0.8276 - val_f1: 0.8276
Epoch 36/50
255/255 [=====] - 2s 9ms/step - loss: 574.2887 - acc:
0.9176 - f1: 0.9177 - val_loss: 574.6126 - val_acc: 0.7931 - val_f1: 0.7931
Epoch 37/50
255/255 [=====] - 25s 98ms/step - loss: 574.2268 - acc:
0.9608 - f1: 0.9609 - val_loss: 574.7714 - val_acc: 0.7586 - val_f1: 0.7586
Epoch 38/50
255/255 [=====] - 25s 98ms/step - loss: 574.2284 - acc:
0.9569 - f1: 0.9570 - val_loss: 574.6733 - val_acc: 0.7241 - val_f1: 0.7241
Epoch 39/50
255/255 [=====] - 26s 103ms/step - loss: 574.2075 -
acc: 0.9529 - f1: 0.9529 - val_loss: 574.4665 - val_acc: 0.8276 - val_f1: 0.8276
Epoch 40/50
255/255 [=====] - 60s 236ms/step - loss: 574.1612 -
acc: 0.9569 - f1: 0.9568 - val_loss: 574.6920 - val_acc: 0.6897 - val_f1: 0.6897
Epoch 41/50
255/255 [=====] - 40s 158ms/step - loss: 574.1420 -
acc: 0.9608 - f1: 0.9603 - val_loss: 574.4649 - val_acc: 0.7931 - val_f1: 0.7931
Epoch 42/50
255/255 [=====] - 58s 229ms/step - loss: 574.1420 -
acc: 0.9647 - f1: 0.9648 - val_loss: 574.7181 - val_acc: 0.7241 - val_f1: 0.7241
Epoch 43/50
255/255 [=====] - 132s 519ms/step - loss: 574.1348 -
acc: 0.9529 - f1: 0.9530 - val_loss: 574.5671 - val_acc: 0.6897 - val_f1: 0.6897
Epoch 44/50
255/255 [=====] - 18s 70ms/step - loss: 574.2098 - acc:
0.9137 - f1: 0.9141 - val_loss: 574.4708 - val_acc: 0.7931 - val_f1: 0.7931
Epoch 45/50
255/255 [=====] - 2s 8ms/step - loss: 574.1206 - acc:
0.9529 - f1: 0.9530 - val_loss: 574.5026 - val_acc: 0.8276 - val_f1: 0.8276
Epoch 46/50
255/255 [=====] - 2s 8ms/step - loss: 574.0492 - acc:
0.9804 - f1: 0.9805 - val_loss: 574.4952 - val_acc: 0.7931 - val_f1: 0.7931
Epoch 47/50
255/255 [=====] - 2s 8ms/step - loss: 574.0374 - acc:
0.9725 - f1: 0.9727 - val_loss: 574.4174 - val_acc: 0.8276 - val_f1: 0.8276
Epoch 48/50
255/255 [=====] - 2s 9ms/step - loss: 574.1528 - acc:
0.9216 - f1: 0.9219 - val_loss: 574.3728 - val_acc: 0.7931 - val_f1: 0.7931
Epoch 49/50
255/255 [=====] - 2s 9ms/step - loss: 574.0725 - acc:

```
0.9490 - f1: 0.9488 - val_loss: 574.6471 - val_acc: 0.6897 - val_f1: 0.6897
Epoch 50/50
255/255 [=====] - 2s 8ms/step - loss: 574.1212 - acc:
0.9255 - f1: 0.9255 - val_loss: 574.5488 - val_acc: 0.7241 - val_f1: 0.7241
```

```
[ ]:
```

```
[13]: 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)

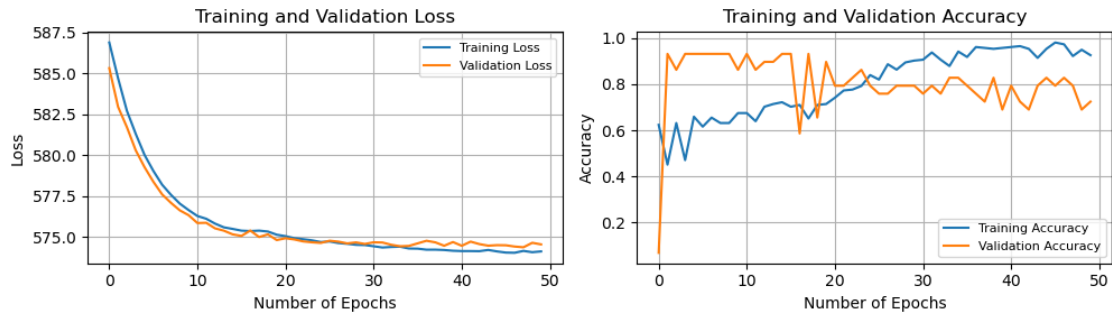
xc = range(epochs)

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()
```



```
[14]: from sklearn.metrics import accuracy_score, classification_report, \
      ↪ precision_recall_fscore_support

      # Predict on the training dataset
      train_predicted = np.argmax(model.predict([X_train, pos_train1, pos_train2]), \
      ↪ axis=1)
      y_train_to_label = np.argmax(y_train, axis=1)

      # Calculate accuracy, precision, recall, and F1-score for the training data
      train_accuracy = accuracy_score(y_train_to_label, train_predicted)
      train_prec, train_reca, train_fscore, _ = \
      ↪ precision_recall_fscore_support(y_train_to_label, train_predicted, \
      ↪ average=param)

      # Print the classification report for the training data
      print("Training Classification Report:")
      print(classification_report(y_train_to_label, train_predicted))

      # Print the precision, recall, and F1-score for the training data
      print("Training Accuracy: {:.2f}%".format(train_accuracy * 100))
      print("Training Precision: {:.2f}%".format(train_prec * 100))
      print("Training Recall: {:.2f}%".format(train_reca * 100))
      print("Training F1 Score: {:.2f}%".format(train_fscore * 100))
```

Training Classification Report:

	precision	recall	f1-score	support
0	0.91	0.96	0.93	89
1	0.98	0.96	0.97	195
accuracy			0.96	284
macro avg	0.95	0.96	0.95	284
weighted avg	0.96	0.96	0.96	284

Training Accuracy: 95.77%

Training Precision: 97.91%
Training Recall: 95.90%
Training F1 Score: 96.89%

[]:

[]:

```
[17]: 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))
```

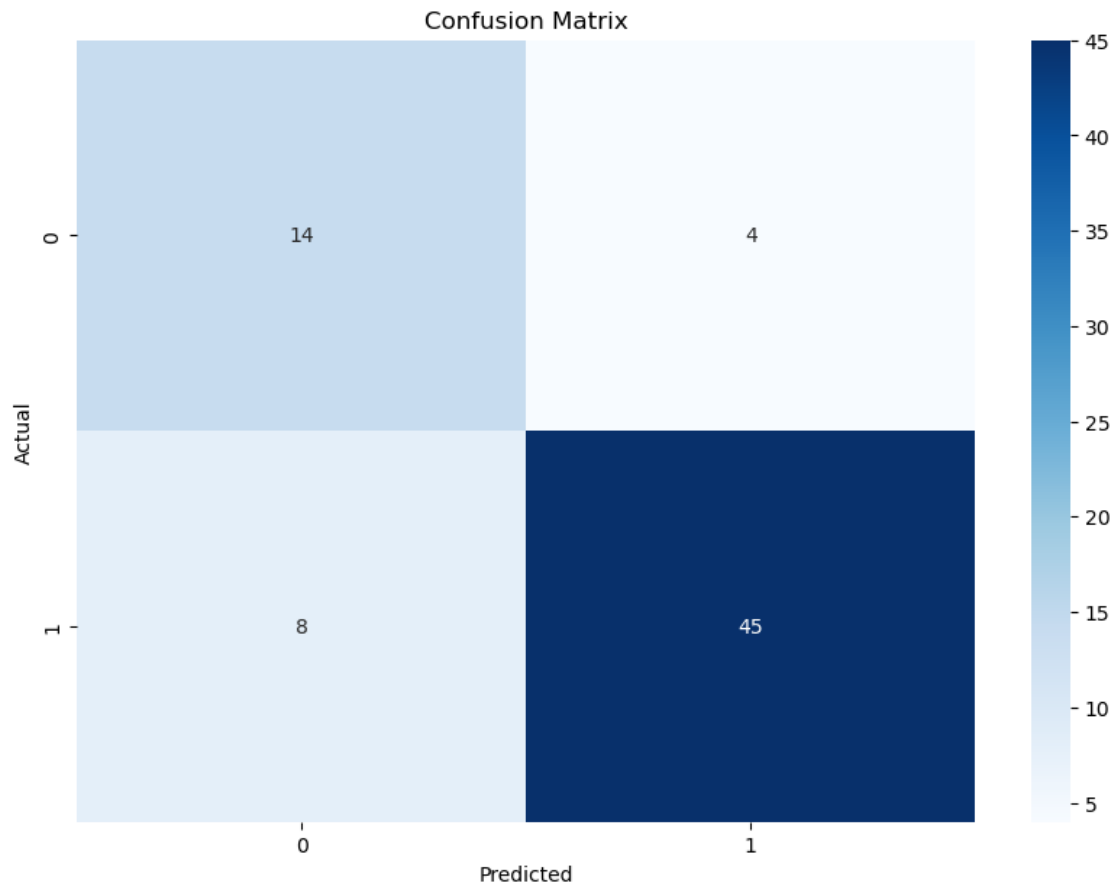
	precision	recall	f1-score	support
0	0.6087	0.7778	0.6829	18
1	0.9167	0.8302	0.8713	53
accuracy			0.8169	71

macro avg	0.7627	0.8040	0.7771	71
weighted avg	0.8386	0.8169	0.8235	71

Precision:91.67% Recall:83.02% Fscore:87.13%

```
[16]: 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'],
    yticklabels=['0', '1'])
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))
```

Precision:91.84% Recall:84.91% Fscore:88.24%

[]: