modelgu-gru-lstm

September 18, 2024

1 model 1

sara CNN + GRU +LSTM

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,GRU
from keras.layers.convolutional import Conv2D, MaxPooling2D
from 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
```

```python
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
```

```
Using TensorFlow backend.
```

3.0.1 Define Callback functions to generate Measures

[2]:
```python
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 9

4.0.1 Load pre procssed Data

[3]:
```python
with open('../data/pickles/train_and_test_data_sentences_snp_2class.pickle',␣
 ↪'rb') as handle:

    W_train = pickle.load(handle)
    d1_train = pickle.load(handle)
    d2_train = pickle.load(handle)
    Y_train = pickle.load(handle)
    Tr_word_list = pickle.load(handle)

    W_test = pickle.load(handle)
    d1_test = pickle.load(handle)
```

```
    d2_test = pickle.load(handle)
    Y_test = pickle.load(handle)
    Te_word_list = pickle.load(handle)


    word_vectors = pickle.load(handle)
    word_dict = pickle.load(handle)
    d1_dict = pickle.load(handle)
    d2_dict = pickle.load(handle)
    label_dict = pickle.load(handle)
    MAX_SEQUENCE_LENGTH = pickle.load(handle)
```

4.0.2 Prepare Word Embedding Layer

[4]:

```
EMBEDDING_DIM=word_vectors.shape[1]
embedding_matrix=word_vectors

def create_embedding_layer(l2_reg=0.1,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_tra
 ↪l2(l2_reg))
    else:
        return Embedding(len(word_dict)␣
 ↪,EMBEDDING_DIM,input_length=MAX_SEQUENCE_LENGTH)
```

[5]:

```
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
```

4.0.3 Create the Model

[6]:

```
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)

    x = Conv1D(256, 7, activation='relu')(embedded_sequences)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.5)(x)

    x = Conv1D(128, 5, activation='relu')(x)
    x = MaxPooling1D(3)(x)
    x = Dropout(0.5)(x)



    conv_sequence=GlobalMaxPooling1D()(x)    #x = Flatten()(x)


    forward = GRU (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,lstm_sequence])
    x = Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.
↪05))(merge)
    x = Dropout(0.5)(x)
    preds = Dense(2, activation='softmax')(x)
    model = Model(sequence_input, preds)
    model.
↪compile(loss='binary_crossentropy',optimizer='adam',metrics=['acc',f1])
    #model.summary()
    return model
```

[7]:
```
model = build_model()
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
==================
input_1 (InputLayer)            (None, 91)           0
__________________________________________________________________________________
__________________
```

```
embedding_1 (Embedding)          (None, 91, 200)      555000      input_1[0][0]
________________________________________________________________________________
__________________
conv1d_1 (Conv1D)                (None, 85, 256)      358656
embedding_1[0][0]
________________________________________________________________________________
__________________
max_pooling1d_1 (MaxPooling1D)   (None, 28, 256)      0           conv1d_1[0][0]
________________________________________________________________________________
__________________
dropout_1 (Dropout)              (None, 28, 256)      0
max_pooling1d_1[0][0]
________________________________________________________________________________
__________________
conv1d_2 (Conv1D)                (None, 24, 128)      163968      dropout_1[0][0]
________________________________________________________________________________
__________________
max_pooling1d_2 (MaxPooling1D)   (None, 8, 128)       0           conv1d_2[0][0]
________________________________________________________________________________
__________________
dropout_2 (Dropout)              (None, 8, 128)       0
max_pooling1d_2[0][0]
________________________________________________________________________________
__________________
gru_1 (GRU)                      (None, 100)          90300
embedding_1[0][0]
________________________________________________________________________________
__________________
lstm_1 (LSTM)                    (None, 100)          120400
embedding_1[0][0]
________________________________________________________________________________
__________________
global_max_pooling1d_1 (GlobalM (None, 128)           0           dropout_2[0][0]
________________________________________________________________________________
__________________
concatenate_1 (Concatenate)      (None, 200)          0           gru_1[0][0]
                                                                  lstm_1[0][0]
________________________________________________________________________________
__________________
concatenate_2 (Concatenate)      (None, 328)          0
global_max_pooling1d_1[0][0]
concatenate_1[0][0]
________________________________________________________________________________
__________________
dense_1 (Dense)                  (None, 256)          84224
concatenate_2[0][0]
________________________________________________________________________________
__________________
```

```
dropout_3 (Dropout)             (None, 256)          0           dense_1[0][0]
________________________________________________________________________________
__________________
dense_2 (Dense)                 (None, 2)            514         dropout_3[0][0]
================================================================================
==================
Total params: 1,373,062
Trainable params: 818,062
Non-trainable params: 555,000
________________________________________________________________________________
__________________
```

4.0.4 Run the Evaluation on the test dataset

[8]:
```
param='macro'
epochs =50
batch_size = 32
history=model.fit(W_train,
 ↪Y_train,epochs=epochs,validation_data=(W_test,Y_test),
 ↪batch_size=batch_size,verbose=1,callbacks=[TQDMNotebookCallback()])
predicted = np.argmax(model.predict(W_test), 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)
print("Precision:{:.2f}% Recall:{:.2f}% Fscore:{:.2f}% ".format(prec*100,
 ↪reca*100, fscore*100))
```

```
Train on 935 samples, validate on 365 samples

Training:   0%|          | 0/50 [00:00<?, ?it/s]

Epoch 1/50

Epoch 0:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 299s 320ms/step - loss: 5020.3105 -
acc: 0.6984 - f1: 0.6988 - val_loss: 5016.8132 - val_acc: 0.6712 - val_f1:
0.6799
Epoch 2/50

Epoch 1:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 7ms/step - loss: 5014.9901 - acc:
0.7144 - f1: 0.7182 - val_loss: 5013.3845 - val_acc: 0.6712 - val_f1: 0.6799
Epoch 3/50

Epoch 2:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 6ms/step - loss: 5012.5435 - acc:
0.7401 - f1: 0.7469 - val_loss: 5011.8206 - val_acc: 0.7589 - val_f1: 0.7632
Epoch 4/50
```

```
Epoch 3:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 6ms/step - loss: 5011.4094 - acc:
0.7733 - f1: 0.7792 - val_loss: 5011.0972 - val_acc: 0.7342 - val_f1: 0.7322
Epoch 5/50

Epoch 4:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 6ms/step - loss: 5010.8605 - acc:
0.7893 - f1: 0.7874 - val_loss: 5010.7052 - val_acc: 0.7452 - val_f1: 0.7388
Epoch 6/50

Epoch 5:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 6ms/step - loss: 5010.5180 - acc:
0.8214 - f1: 0.8260 - val_loss: 5010.4151 - val_acc: 0.7726 - val_f1: 0.7724
Epoch 7/50

Epoch 6:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5010.2949 - acc:
0.8257 - f1: 0.8302 - val_loss: 5010.1901 - val_acc: 0.8137 - val_f1: 0.8077
Epoch 8/50

Epoch 7:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 8s 8ms/step - loss: 5010.0759 - acc:
0.8727 - f1: 0.8760 - val_loss: 5010.0012 - val_acc: 0.8548 - val_f1: 0.8582
Epoch 9/50

Epoch 8:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.9394 - acc:
0.8834 - f1: 0.8827 - val_loss: 5009.8980 - val_acc: 0.8521 - val_f1: 0.8594
Epoch 10/50

Epoch 9:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 7ms/step - loss: 5009.7648 - acc:
0.9198 - f1: 0.9219 - val_loss: 5009.7924 - val_acc: 0.8301 - val_f1: 0.8385
Epoch 11/50

Epoch 10:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 7ms/step - loss: 5009.6717 - acc:
0.9144 - f1: 0.9167 - val_loss: 5009.6979 - val_acc: 0.8685 - val_f1: 0.8750
Epoch 12/50

Epoch 11:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.6770 - acc:
0.8920 - f1: 0.8911 - val_loss: 5009.6611 - val_acc: 0.8466 - val_f1: 0.8542
Epoch 13/50

Epoch 12:   0%|          | 0/935 [00:00<?, ?it/s]
```

```
935/935 [==============================] - 6s 6ms/step - loss: 5009.6458 - acc:
0.8802 - f1: 0.8833 - val_loss: 5009.7560 - val_acc: 0.8301 - val_f1: 0.8385
Epoch 14/50

Epoch 13:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 6ms/step - loss: 5009.5969 - acc:
0.8995 - f1: 0.9021 - val_loss: 5009.6727 - val_acc: 0.8274 - val_f1: 0.8359
Epoch 15/50

Epoch 14:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 7ms/step - loss: 5009.4688 - acc:
0.9187 - f1: 0.9171 - val_loss: 5009.5269 - val_acc: 0.8767 - val_f1: 0.8828
Epoch 16/50

Epoch 15:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.5265 - acc:
0.9016 - f1: 0.9042 - val_loss: 5009.5483 - val_acc: 0.8795 - val_f1: 0.8854
Epoch 17/50

Epoch 16:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 6ms/step - loss: 5009.4000 - acc:
0.9433 - f1: 0.9448 - val_loss: 5009.6463 - val_acc: 0.8548 - val_f1: 0.8620
Epoch 18/50

Epoch 17:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 7ms/step - loss: 5009.4078 - acc:
0.9305 - f1: 0.9286 - val_loss: 5009.4692 - val_acc: 0.8740 - val_f1: 0.8802
Epoch 19/50

Epoch 18:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 7ms/step - loss: 5009.3849 - acc:
0.9390 - f1: 0.9406 - val_loss: 5009.5298 - val_acc: 0.8685 - val_f1: 0.8750
Epoch 20/50

Epoch 19:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.3509 - acc:
0.9337 - f1: 0.9317 - val_loss: 5009.6227 - val_acc: 0.8411 - val_f1: 0.8490
Epoch 21/50

Epoch 20:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.3435 - acc:
0.9390 - f1: 0.9406 - val_loss: 5009.6191 - val_acc: 0.8329 - val_f1: 0.8411
Epoch 22/50

Epoch 21:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 8ms/step - loss: 5009.3583 - acc:
0.9433 - f1: 0.9448 - val_loss: 5009.4401 - val_acc: 0.8877 - val_f1: 0.8932
Epoch 23/50
```

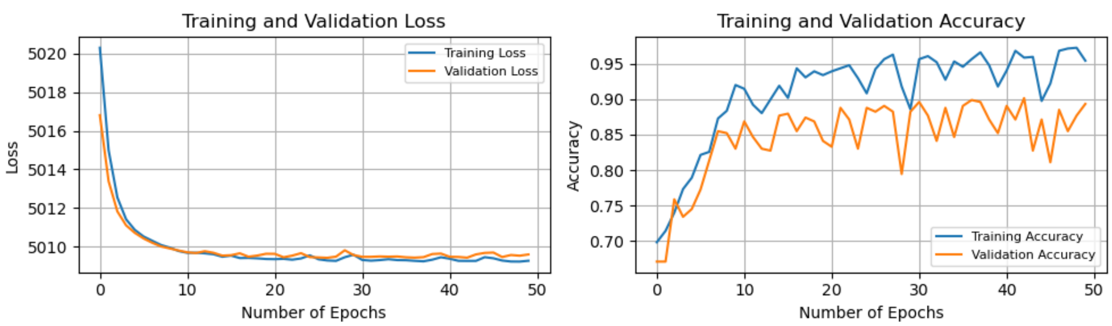
```
Epoch 22:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 8ms/step - loss: 5009.3161 - acc:
0.9476 - f1: 0.9490 - val_loss: 5009.5264 - val_acc: 0.8712 - val_f1: 0.8776
Epoch 24/50

Epoch 23:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.3795 - acc:
0.9294 - f1: 0.9312 - val_loss: 5009.6534 - val_acc: 0.8301 - val_f1: 0.8385
Epoch 25/50

Epoch 24:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.5398 - acc:
0.9080 - f1: 0.9067 - val_loss: 5009.4506 - val_acc: 0.8877 - val_f1: 0.8932
Epoch 26/50

Epoch 25:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.3350 - acc:
0.9422 - f1: 0.9437 - val_loss: 5009.4321 - val_acc: 0.8822 - val_f1: 0.8880
Epoch 27/50

Epoch 26:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 6ms/step - loss: 5009.2754 - acc:
0.9561 - f1: 0.9573 - val_loss: 5009.4134 - val_acc: 0.8904 - val_f1: 0.8958
Epoch 28/50

Epoch 27:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 5s 6ms/step - loss: 5009.2541 - acc:
0.9626 - f1: 0.9598 - val_loss: 5009.4728 - val_acc: 0.8822 - val_f1: 0.8880
Epoch 29/50

Epoch 28:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 6ms/step - loss: 5009.4423 - acc:
0.9176 - f1: 0.9161 - val_loss: 5009.8030 - val_acc: 0.7945 - val_f1: 0.7933
Epoch 30/50

Epoch 29:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 7ms/step - loss: 5009.5737 - acc:
0.8856 - f1: 0.8885 - val_loss: 5009.5568 - val_acc: 0.8822 - val_f1: 0.8880
Epoch 31/50

Epoch 30:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 6ms/step - loss: 5009.3013 - acc:
0.9561 - f1: 0.9573 - val_loss: 5009.4633 - val_acc: 0.8959 - val_f1: 0.9010
Epoch 32/50

Epoch 31:   0%|          | 0/935 [00:00<?, ?it/s]
```

```
935/935 [==============================] - 6s 7ms/step - loss: 5009.2649 - acc:
0.9604 - f1: 0.9615 - val_loss: 5009.4643 - val_acc: 0.8767 - val_f1: 0.8828
Epoch 33/50

Epoch 32:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 6ms/step - loss: 5009.2943 - acc:
0.9519 - f1: 0.9494 - val_loss: 5009.4834 - val_acc: 0.8411 - val_f1: 0.8337
Epoch 34/50

Epoch 33:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 6ms/step - loss: 5009.3402 - acc:
0.9273 - f1: 0.9292 - val_loss: 5009.4710 - val_acc: 0.8877 - val_f1: 0.8932
Epoch 35/50

Epoch 34:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 7ms/step - loss: 5009.2945 - acc:
0.9529 - f1: 0.9542 - val_loss: 5009.4811 - val_acc: 0.8466 - val_f1: 0.8542
Epoch 36/50

Epoch 35:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 7ms/step - loss: 5009.2878 - acc:
0.9455 - f1: 0.9469 - val_loss: 5009.4397 - val_acc: 0.8904 - val_f1: 0.8958
Epoch 37/50

Epoch 36:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 7ms/step - loss: 5009.2560 - acc:
0.9561 - f1: 0.9536 - val_loss: 5009.4225 - val_acc: 0.8986 - val_f1: 0.9036
Epoch 38/50

Epoch 37:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 6s 7ms/step - loss: 5009.2304 - acc:
0.9658 - f1: 0.9629 - val_loss: 5009.4455 - val_acc: 0.8959 - val_f1: 0.9010
Epoch 39/50

Epoch 38:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.3152 - acc:
0.9476 - f1: 0.9490 - val_loss: 5009.6083 - val_acc: 0.8712 - val_f1: 0.8776
Epoch 40/50

Epoch 39:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.4409 - acc:
0.9176 - f1: 0.9198 - val_loss: 5009.6324 - val_acc: 0.8521 - val_f1: 0.8594
Epoch 41/50

Epoch 40:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.3660 - acc:
0.9401 - f1: 0.9417 - val_loss: 5009.4654 - val_acc: 0.8904 - val_f1: 0.8958
Epoch 42/50
```

```
Epoch 41:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 8ms/step - loss: 5009.2560 - acc:
0.9679 - f1: 0.9688 - val_loss: 5009.4567 - val_acc: 0.8712 - val_f1: 0.8776
Epoch 43/50

Epoch 42:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 8ms/step - loss: 5009.2522 - acc:
0.9583 - f1: 0.9519 - val_loss: 5009.4206 - val_acc: 0.9014 - val_f1: 0.9062
Epoch 44/50

Epoch 43:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.2532 - acc:
0.9594 - f1: 0.9604 - val_loss: 5009.5891 - val_acc: 0.8274 - val_f1: 0.8131
Epoch 45/50

Epoch 44:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.4454 - acc:
0.8973 - f1: 0.9000 - val_loss: 5009.6675 - val_acc: 0.8712 - val_f1: 0.8776
Epoch 46/50

Epoch 45:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 7s 7ms/step - loss: 5009.3885 - acc:
0.9219 - f1: 0.9240 - val_loss: 5009.6860 - val_acc: 0.8110 - val_f1: 0.8203
Epoch 47/50

Epoch 46:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 8s 9ms/step - loss: 5009.2715 - acc:
0.9679 - f1: 0.9688 - val_loss: 5009.4536 - val_acc: 0.8849 - val_f1: 0.8906
Epoch 48/50

Epoch 47:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 8s 8ms/step - loss: 5009.2205 - acc:
0.9711 - f1: 0.9719 - val_loss: 5009.5583 - val_acc: 0.8548 - val_f1: 0.8620
Epoch 49/50

Epoch 48:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 10s 10ms/step - loss: 5009.2168 -
acc: 0.9722 - f1: 0.9729 - val_loss: 5009.5223 - val_acc: 0.8767 - val_f1:
0.8828
Epoch 50/50

Epoch 49:   0%|          | 0/935 [00:00<?, ?it/s]

935/935 [==============================] - 8s 9ms/step - loss: 5009.2599 - acc:
0.9540 - f1: 0.9515 - val_loss: 5009.5874 - val_acc: 0.8932 - val_f1: 0.8984
Precision:89.12% Recall:85.04% Fscore:86.63%
```

[9]:

```
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()
```



[10]:
```python
# Predict on the training dataset
train_predictions = np.argmax(model.predict(W_train), axis=1)

# Calculate accuracy
train_accuracy = accuracy_score(np.argmax(Y_train, axis=1), train_predictions)

# Print accuracy
print("Accuracy on Training Data: {:.3f}".format(train_accuracy))

# Print classification report
print("Classification Report on Training Data:")
print(classification_report(np.argmax(Y_train, axis=1), train_predictions))
```

```
Accuracy on Training Data: 0.961
Classification Report on Training Data:
              precision    recall  f1-score   support

           0       0.97      0.87      0.92       233
           1       0.96      0.99      0.97       702

    accuracy                           0.96       935
   macro avg       0.96      0.93      0.95       935
weighted avg       0.96      0.96      0.96       935
```

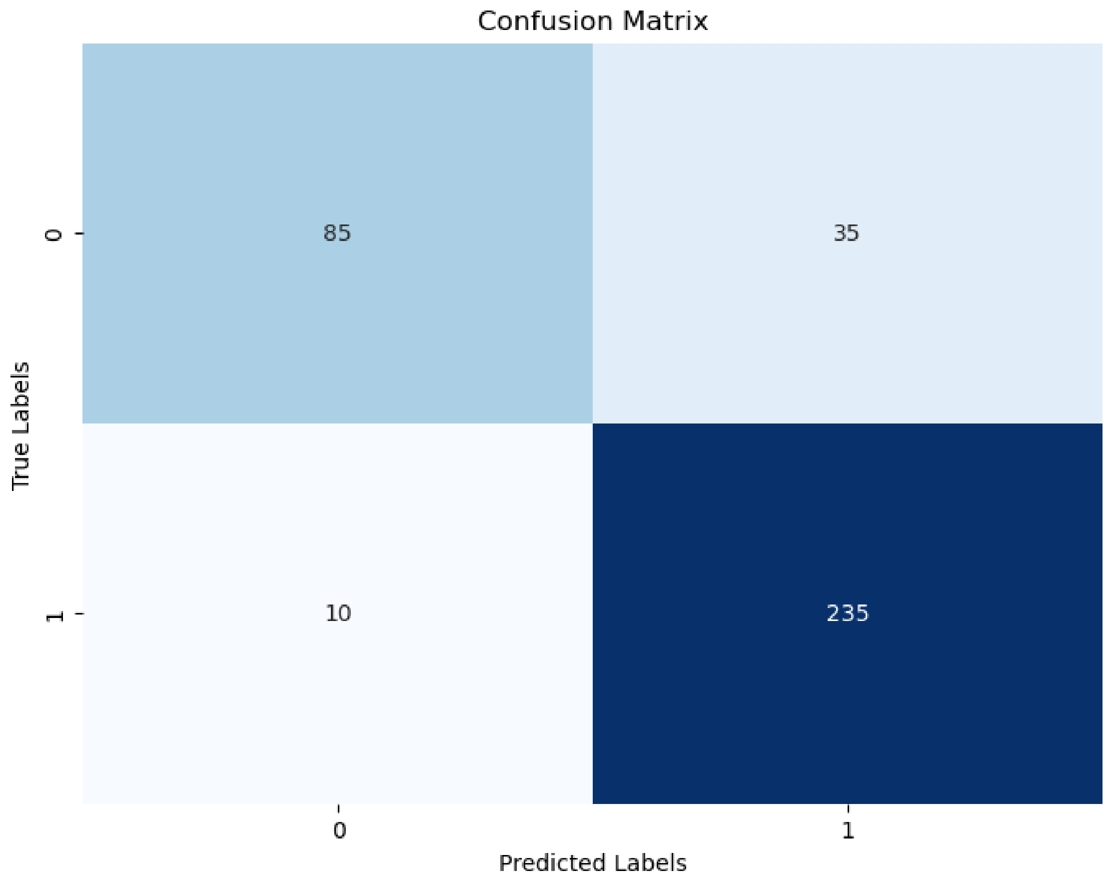
[11]:
```python
print('Running predictions...')
all_predictions, all_labels = [], []
labels = np.argmax(Y_test, axis=1)
y_pred = np.argmax(model.predict(W_test), axis=1)
all_predictions.extend(y_pred.astype('int32'))
all_labels.extend(labels.astype('int32'))
all_labels = np.array(all_labels)
all_predictions = np.array(all_predictions)

correct_pred_count = (all_labels == all_predictions).sum()
test_acc = correct_pred_count / len(all_labels)
# show the the accuracy of testing data
print('We got %d of %d correct (or %.3f accuracy)' % (correct_pred_count,␣
 ↪len(all_labels), test_acc))
print('Accuracy:', accuracy_score(y_true=all_labels, y_pred=all_predictions))

# 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))
```

```
Running predictions…
We got 320 of 365 correct (or 0.877 accuracy)
Accuracy: 0.8767123287671232
              precision    recall  f1-score   support

           0     0.8990    0.7417    0.8128       120
           1     0.8835    0.9592    0.9198       245

    accuracy                         0.8877       365
   macro avg     0.8912    0.8504    0.8663       365
weighted avg     0.8886    0.8877    0.8846       365

 Precision:89.12% Recall:85.04% Fscore:86.63%
```

[13]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Get the predicted labels
predicted_labels = np.argmax(model.predict(W_test), axis=1)

# Create the confusion matrix
cm = confusion_matrix(np.argmax(Y_test, axis=1), predicted_labels)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Labels')
```

```python
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```




```
[ ]:
```