```
# all the necessary imports
import os
import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
from torch import optim
import torch
import pandas as pd
import numpy as np
import spacy
import altair as alt
import gdown
nlp = spacy.load("en_core_web_sm")
alt.data_transformers.enable("vegafusion")
```

```
DataTransformerRegistry.enable('vegafusion')
```

```
# set the seed
manual_seed = 572
torch.manual_seed(manual_seed)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
n_gpu = torch.cuda.device_count()
print(device)
```

```
cuda
```

You can adap these two functions for your model.

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

def train(loader):
    model.train()
    total_loss = 0.0
    # iterate throught the data loader
    num_sample = 0
    for batch in loader:
        # load the current batch
        batch_input = batch.review
        batch_output = batch.label

        batch_input = batch_input.to(device)
        batch_output = batch_output.to(device)
        # forward propagation
        # pass the data through the model
        model_outputs = model(batch_input)
        # compute the loss
        cur_loss = criterion(model_outputs, batch_output)
        total_loss += cur_loss.cpu().item()

        # backward propagation (compute the gradients and update the model)
        # clear the buffer
        optimizer.zero_grad()
        # compute the gradients
        cur_loss.backward()
        # update the weights
        optimizer.step()

        num_sample += batch_output.shape[0]

    return total_loss/num_sample

# evaluation logic based on classification accuracy
def evaluate(loader):
    model.eval()
    all_pred=[]
```

```
    all_label = []
    with torch.no_grad(): # impacts the autograd engine and deactivate it. reduces memory usa
        for batch in loader:
             # load the current batch
            batch_input = batch.review
            batch_output = batch.label

            batch_input = batch_input.to(device)
            # forward propagation
            # pass the data through the model
            model_outputs = model(batch_input)
            # identify the predicted class for each example in the batch
            probabilities, predicted = torch.max(model_outputs.cpu().data, 1)
            # put all the true labels and predictions to two lists
            all_pred.extend(predicted)
            all_label.extend(batch_output.cpu())

    accuracy = accuracy_score(all_label, all_pred)
    f1score = f1_score(all_label, all_pred, average='macro')
    return accuracy,f1score
```

```
# funtion for save prediction
def out_prediction(first_name, last_name, prediction_list):
    """
    out_prediction takes three input varibles: first_name, last_name, prediction_list
    <first_name>, string, your first name, e.g., Tom
    <last_name>, string, your last name, e.g., Smith
    <prediction_list>, list of string which includes all your predications of TEST samples
                        e.g., ['1star','5star','3star']

    Generate a file is named with <yourfirstname>_<yourlastname>_PRED.txt in current director
    """
    output_file = open("{}_{}_PRED.txt".format(first_name,last_name),'w')
    for item in prediction_list:
        output_file.write(item+"\n")
    output_file.close()
```

Please write code to develop you system. More details are in `Lab4.ipynb`.

```
# Data download to colab working directory
url = f'https://drive.google.com/drive/folders/1zF5s4KRMxpr3OvUY8o_d0xv_YlaZSCsj?usp=drive_li
gdown.download_folder(url, quiet = False, use_cookies = False)
```

```
Processing file 1Vr4eJVM9kIK38ZONzamuIrQKf_gi3pw- .Rhistory
Processing file 19QypIq9BjPB_WhGIsT8Zrr658VV6uQHC readme.md
Retrieving folder contents completed
Building directory structure
Building directory structure completed
Downloading...
From: https://drive.google.com/uc?id=1FrYOgXiu-pVX_TQ57pSUtpwHeu1R4J0k
To: /content/data/yelp_review/.ipynb_checkpoints/EXAMPLE_GOLD-checkpoint.txt
100%|██████████| 21.0k/21.0k [00:00<00:00, 29.8MB/s]
Downloading...
From: https://drive.google.com/uc?id=1oCDq8R-xBdwcJBr_2KLpzg6WVgQN_Re9
To: /content/data/yelp_review/.ipynb_checkpoints/EXAMPLE_PRED_result-checkpoint.txt
```

```
From (redirected): https://drive.google.com/uc?id=1jjiGoOLJLlucSlJtb_EJ0Vo2xIUaMN2r&confirm=t
To: /content/data/yelp_review/Scorer.py
100%|██████████| 3.35k/3.35k [00:00<00:00, 10.2MB/s]
Downloading...
From: https://drive.google.com/uc?id=1crLuR8fhJ89RKEVut-UjgIyOz6-y5eh4
To: /content/data/yelp_review/test.tsv
100%|██████████| 2.63M/2.63M [00:00<00:00, 13.1MB/s]
Downloading...
From: https://drive.google.com/uc?id=10qVnIg3rvEXlcEA9hynHTKoYDURLQN4v
To: /content/data/yelp_review/train.tsv
100%|██████████| 21.3M/21.3M [00:00<00:00, 68.6MB/s]
Downloading...
From: https://drive.google.com/uc?id=1zozCGPs0XiJAPl1Il5cSWl9p0Aqv8YgO
To: /content/data/yelp_review/val.tsv
100%|██████████| 2.67M/2.67M [00:00<00:00, 20.1MB/s]
Downloading...
From: https://drive.google.com/uc?id=1Vr4eJVM9kIK38ZONzamuIrQKf_gi3pw-
To: /content/data/.Rhistory
0.00B [00:00, ?B/s]
Downloading...
From: https://drive.google.com/uc?id=19QypIq9BjPB_WhGIsT8Zrr658VV6uQHC
To: /content/data/readme.md
100%|██████████| 98.0/98.0 [00:00<00:00, 418kB/s]
Download completed
['/content/data/yelp_review/.ipynb_checkpoints/EXAMPLE_GOLD-checkpoint.txt',
 '/content/data/yelp_review/.ipynb_checkpoints/EXAMPLE_PRED_result-checkpoint.txt',
 '/content/data/yelp_review/EXAMPLE_GOLD.txt',
 '/content/data/yelp_review/EXAMPLE_PRED_result.txt',
 '/content/data/yelp_review/Scorer.py',
 '/content/data/yelp_review/test.tsv',
 '/content/data/yelp_review/train.tsv',
 '/content/data/yelp_review/val.tsv',
 '/content/data/.Rhistory',
 '/content/data/readme.md']
```

```
### Import for EDA
train_set = pd.read_csv('data/yelp_review/train.tsv', sep = '\t')
dev_set = pd.read_csv('data/yelp_review/val.tsv', sep = '\t')
test_set = pd.read_csv('data/yelp_review/test.tsv', sep = '\t')

train_set.head()
```

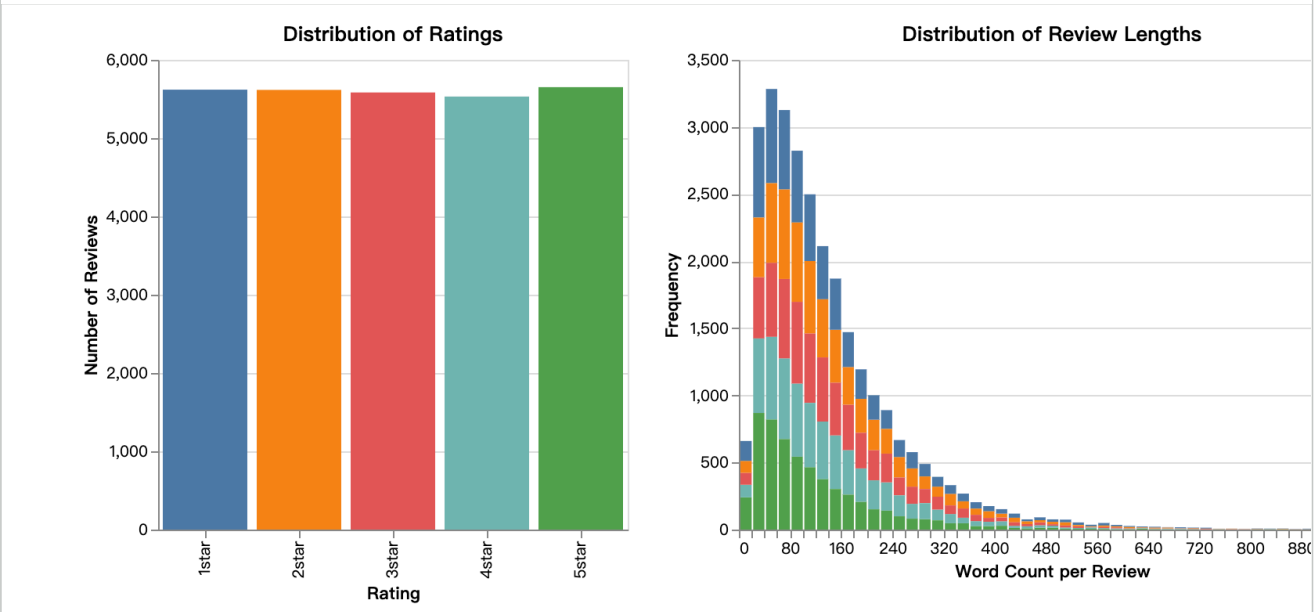
	content	rating
0	There are some restaurants that you don't want...	4star
1	Lucky for us there was no wait unlike other ti...	4star
2	Worst ever Michelin restaurant I have ever bee...	1star
3	Came here today to celebrate my birthdays with...	4star
4	This, is where hipsters go to get Caribbean fo...	2star

后续步骤： 使用 train_set 生成代码 New interactive sheet

EDA

```
### EDA training set
import copy

train_eda = train_set.copy()
train_eda['review_length'] = train_set['content'].apply(lambda x: len(str(x).split()))

# Rating Plot
ratings = alt.Chart(train_eda).mark_bar().encode(
    x = alt.X('rating').sort(['1star', '2star', '3star', '4star', '5star']).title('Rating'),
    y = alt.Y('count()').title('Number of Reviews'),
    color = alt.Color('rating').legend(None),
    tooltip = ['rating', 'count()']
).properties(
```

```
    title = 'Distribution of Ratings',
    width = 300,
    height = 300
)

# Length Plot
length = alt.Chart(train_eda).mark_bar().encode(
    x=alt.X('review_length', bin=alt.Bin(maxbins=60), title='Word Count per Review'),
    y=alt.Y('count()', title='Frequency'),
    color = alt.Color('rating').legend(None),
    tooltip=['count()']
).properties(
    title='Distribution of Review Lengths',
    width=400,
    height=300
)

final_chart = ratings | length
final_chart
```



From EDA, we know that the classes are balanced, and it seems like all the ratings are in proportion to the length of the comments.

Baseline

For baseline, I will use tfidf vectorizer and logistic regression, it is fast and provide a decent standard baseline score for my further models.

The baseline Macro F1 Score = 59.36%

```
### baseline

## import
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, f1_score
from sklearn.pipeline import make_pipeline
```

```
## data split

X_train = train_set['content']
```

```
y_train = train_set['rating']

X_dev = dev_set['content']
y_dev = dev_set['rating']

print(len(X_train), len(X_dev))
```

```
28000 3500
```

```
## Pipeline
baseline = make_pipeline(
    TfidfVectorizer(
        stop_words = 'english',
        ngram_range = (1, 2)
    ),

    LogisticRegression(
        solver = 'liblinear',
        C = 1.0
    )
)

# baseline train & pred
baseline.fit(X_train, y_train)
y_pred_baseline = baseline.predict(X_dev)

# calculate macro f1
f1_baseline = f1_score(y_dev, y_pred_baseline, average = 'macro')
```

```
print(f'The Macro F1 score of baseline with tfidf and logistic is: {f1_baseline}')
```

```
The Macro F1 score of baseline with tfidf and logistic is: 0.5936185962662195
```

CBOW

```
# Preprocessing
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
label_encoder.fit(train_set.rating)

train_y = label_encoder.transform(train_set.rating)
dev_y = label_encoder.transform(dev_set.rating)
```

```
# Vocabulary
from collections import defaultdict, Counter

vocabulary = Counter()
for sentence in train_set.content:
    tokens = sentence.lower().split()
    vocabulary.update(tokens)
```

```
from collections import defaultdict, Counter

## Create a w2i
def BuildWord2i(contents):
    counter = Counter()

    for content in contents:
        tokens = str(content).lower().split()
        counter.update(tokens)

    word2i = {}
    word2i['<PAD>'] = 0
```

```
        word2i['<UNK>'] = 1

        idx = 2
        for word, count in counter.items():
            word2i[word] = idx
            idx += 1

        return word2i

    ## get w2i
    w2i = BuildWord2i(train_set['content'])
```

```
    ## Get the embedding matrix from spacy
    def BuildEmbeddingMatrix(word2i, vocab_size, emb_dim = 300):
        weights_matrix = np.random.normal(scale = 0.6, size = (vocab_size, emb_dim))

        weights_matrix[w2i['<PAD>']] = np.zeros((emb_dim,))
        count = 0
        for word, i in w2i.items():
            if word in nlp.vocab and nlp.vocab[word].has_vector:
                weights_matrix[i] = nlp.vocab[word].vector
                found_count += 1
        return torch.tensor(weights_matrix, dtype = torch.float32)

    ## create embedding weights from spacy
    EMBEDDING_DIM = 300
    embedding_weights = BuildEmbeddingMatrix(w2i, len(w2i), EMBEDDING_DIM)
    embedding_weights.shape
```

```
    torch.Size([106121, 300])
```

```
    from torch.utils.data import TensorDataset, DataLoader
    from torch.nn.utils.rnn import pad_sequence
    def create_data_loader(df, y, w2i, batch_size=32, shuffle=True):
        pad_token_id = w2i.get('<PAD>', 0)
        unk_token_id = w2i.get('<UNK>', 0)

        def text_to_indices(text):
            tokens = text.split()
            return [w2i.get(token, unk_token_id) for token in tokens]

        indices_list = [torch.tensor(text_to_indices(text)) for text in df['content']]
        padded_inputs = pad_sequence(indices_list, batch_first=True, padding_value=pad_token_id)

        labels = torch.tensor(y, dtype=torch.long)

        dataset = TensorDataset(padded_inputs, labels)
        loader = DataLoader(dataset, batch_size=batch_size, shuffle=shuffle)

        return loader

    BATCH_SIZE = 64
    train_loader = create_data_loader(train_set, train_y, w2i, batch_size=BATCH_SIZE, shuffle=Tru
    dev_loader = create_data_loader(dev_set, dev_y,  w2i, batch_size=BATCH_SIZE, shuffle=False)
```

```
    # CBOW originated from lab 3
    HIDDEN_SIZE = 100
    class CBOW(nn.Module):
        def __init__(self, weights_matrix, num_classes, dropout_prob, padding_idx):
            super().__init__()
            self.embedding = nn.Embedding.from_pretrained(weights_matrix, padding_idx = padding_i
            self.embedding_dim = weights_matrix.shape[1]

            self.linear1 = nn.Linear(self.embedding_dim, HIDDEN_SIZE)
            self.dropout = nn.Dropout(p = dropout_prob)
            self.relu = nn.ReLU()
            self.linear2 = nn.Linear(HIDDEN_SIZE, num_classes)
```

```
        self.padding_idx = padding_idx

    def forward(self, x):
            non_pad_mask = (x != self.padding_idx)
            lengths = non_pad_mask.sum(dim=1).float().clamp(min=1).unsqueeze(1)

            embedded = self.embedding(x)
            x = torch.sum(embedded, dim=1) / lengths

            x = self.linear1(x)
            x = self.dropout(x)
            x = self.relu(x)
            x = self.linear2(x)
            return x
```

```
pad_idx = w2i['<PAD>']

model = CBOW(
    weights_matrix = embedding_weights,
    num_classes = 5,
    dropout_prob = 0.5,
    padding_idx = pad_idx
).to(device)
```

```
optimizer = torch.optim.Adam(model.parameters(), lr = 0.01)
criterion = nn.CrossEntropyLoss()
```

```
NUM_EPOCHS = 3
for epoch in range(NUM_EPOCHS):
    model.train()
    total_loss = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = model(inputs)

        loss = criterion(outputs, labels)

        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    avg_train_loss = total_loss / len(train_loader)

    model.eval()
    correct = 0
    total = 0
    val_loss = 0

    with torch.no_grad():
        for inputs, labels in dev_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()

            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
```

```
        avg_val_loss = val_loss / len(dev_loader)
        val_accuracy = 100 * correct / total

        print(f'Epoch [{epoch+1}/{NUM_EPOCHS}], '
              f'Train Loss: {avg_train_loss:.4f}, '
              f'Val Loss: {avg_val_loss:.4f}, '
              f'Val Acc: {val_accuracy:.2f}%')

    print("Finish processing")
```

```
Epoch [1/3], Train Loss: 1.4408, Val Loss: 1.3469, Val Acc: 41.74%
Epoch [2/3], Train Loss: 1.3760, Val Loss: 1.3383, Val Acc: 42.23%
Epoch [3/3], Train Loss: 1.3639, Val Loss: 1.3268, Val Acc: 42.54%
Finish processing
```

LSTM

```
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence
class LSTM(nn.Module):
    def __init__(self, weights_matrix, num_classes, hidden_size = 256, num_layers = 1, dropou
        super().__init__()

        weights_tensor = torch.tensor(weights_matrix, dtype=torch.float)
        self.embedding = nn.Embedding.from_pretrained(
            weights_tensor,
            padding_idx=padding_idx
        )
        self.emb_dim = weights_matrix.shape[1]

        self.lstm = nn.LSTM(
            input_size=self.emb_dim,
            hidden_size=hidden_size,
            num_layers=num_layers,
            batch_first=True,
            bidirectional=True,
            dropout=dropout_prob if num_layers > 1 else 0
        )

        self.fc = nn.Linear(hidden_size * 2, num_classes)

        self.dropout = nn.Dropout(dropout_prob)
        self.padding_idx = padding_idx

    def forward(self, x):
        lengths = (x != self.padding_idx).sum(dim=1).cpu()

        embeds = self.embedding(x) # [Batch, Seq, Emb]
        packed_input = pack_padded_sequence(embeds, lengths, batch_first=True, enforce_sorted
        packed_output, (hidden, cell) = self.lstm(packed_input)

        cat_hidden = torch.cat((hidden[-2,:,:], hidden[-1,:,:]), dim=1)

        output = self.dropout(cat_hidden)
        output = self.fc(output)

        return output
```

```
HIDDEN_SIZE = 128
NUM_LAYERS = 2
DROPOUT = 0.5
PAD_IDX = w2i['<PAD>']
NUM_CLASSES = 5

model = LSTM(
    weights_matrix=embedding_weights,
    num_classes=NUM_CLASSES,
    hidden_size=HIDDEN_SIZE,
```

```
    num_layers=NUM_LAYERS,
    dropout_prob=DROPOUT,
    padding_idx=PAD_IDX
).to(device)

optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)
```

```
/tmp/ipython-input-2886251409.py:6: UserWarning: To copy construct from a tensor, it is recomm
  weights_tensor = torch.tensor(weights_matrix, dtype=torch.float)
```

```
train_loss = float('inf')
best_val_loss = float('inf')
patience = 3
trigger_times = 0
epoch = 0
THRESHOLD = 0.01
MAX_EPOCHS = 100

while train_loss > THRESHOLD and epoch < MAX_EPOCHS:
    epoch += 1
    model.train()
    total_loss = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    avg_train_loss = total_loss / len(train_loader)
    train_loss = avg_train_loss

    model.eval()
    val_loss = 0

    all_preds = []
    all_labels = []

    with torch.no_grad():
        for inputs, labels in dev_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()

            _, predicted = torch.max(outputs.data, 1)

            all_preds.extend(predicted.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    avg_val_loss = val_loss / len(dev_loader)

    val_f1 = f1_score(all_labels, all_preds, average='macro')

    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        torch.save(model.state_dict(), 'best_model.pth')
        print(f"Save best model (F1: {val_f1:.4f})")
        trigger_times = 0
    else:
        trigger_times += 1
        print(f"Val Loss ({trigger_times}/{patience})")
```

```
            if trigger_times >= patience:
                print("Early stop triggers")
                break

        print(f'Epoch [{epoch}], '
              f'Train Loss: {avg_train_loss:.4f}, '
              f'Val Loss: {avg_val_loss:.4f}, '
              f'Val F1 (Macro): {val_f1:.4f}')

    print("Finish training")
    model.load_state_dict(torch.load('best_model.pth'))
```

```
Save best model (F1: 0.4521)
Epoch [1], Train Loss: 1.4031, Val Loss: 1.2327, Val F1 (Macro): 0.4521
Val Loss (1/3)
Epoch [2], Train Loss: 1.2334, Val Loss: 1.2392, Val F1 (Macro): 0.4346
Save best model (F1: 0.4686)
Epoch [3], Train Loss: 1.1421, Val Loss: 1.1874, Val F1 (Macro): 0.4686
Save best model (F1: 0.5289)
Epoch [4], Train Loss: 1.0361, Val Loss: 1.0636, Val F1 (Macro): 0.5289
Save best model (F1: 0.5310)
Epoch [5], Train Loss: 0.9297, Val Loss: 1.0446, Val F1 (Macro): 0.5310
Val Loss (1/3)
Epoch [6], Train Loss: 0.8368, Val Loss: 1.0540, Val F1 (Macro): 0.5501
Val Loss (2/3)
Epoch [7], Train Loss: 0.7459, Val Loss: 1.0646, Val F1 (Macro): 0.5570
Val Loss (3/3)
Early stop triggers
Finish training
<All keys matched successfully>
```

```
# Hyper parameter tunning

HIDDEN_SIZE = 256
NUM_LAYERS = 3
DROPOUT = 0.5
PAD_IDX = w2i['<PAD>']
NUM_CLASSES = 5

model_tuned = LSTM(
    weights_matrix=embedding_weights,
    num_classes=NUM_CLASSES,
    hidden_size=HIDDEN_SIZE,
    num_layers=NUM_LAYERS,
    dropout_prob=DROPOUT,
    padding_idx=PAD_IDX
).to(device)

optimizer = torch.optim.Adam(model_tuned.parameters(), lr = 0.001)
```

```
/tmp/ipython-input-2886251409.py:6: UserWarning: To copy construct from a tensor, it is recomm
  weights_tensor = torch.tensor(weights_matrix, dtype=torch.float)
```

```
train_loss = float('inf')
best_val_loss = float('inf')
patience = 3
trigger_times = 0
epoch = 0
THRESHOLD = 0.01
MAX_EPOCHS = 100

while train_loss > THRESHOLD and epoch < MAX_EPOCHS:
    epoch += 1
    model_tuned.train()
    total_loss = 0

    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
```

```
            outputs = model_tuned(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            total_loss += loss.item()

        avg_train_loss = total_loss / len(train_loader)
        train_loss = avg_train_loss

        model_tuned.eval()
        val_loss = 0

        all_preds = []
        all_labels = []

        with torch.no_grad():
            for inputs, labels in dev_loader:
                inputs, labels = inputs.to(device), labels.to(device)

                outputs = model_tuned(inputs)
                loss = criterion(outputs, labels)
                val_loss += loss.item()

                _, predicted = torch.max(outputs.data, 1)

                all_preds.extend(predicted.cpu().numpy())
                all_labels.extend(labels.cpu().numpy())

        avg_val_loss = val_loss / len(dev_loader)

        val_f1 = f1_score(all_labels, all_preds, average='macro')

        if avg_val_loss < best_val_loss:
            best_val_loss = avg_val_loss
            torch.save(model_tuned.state_dict(), 'best_model_tuned.pth')
            print(f"Save best model_tuned (F1: {val_f1:.4f})")
            trigger_times = 0
        else:
            trigger_times += 1
            print(f"Val Loss ({trigger_times}/{patience})")

            if trigger_times >= patience:
                print("Early stop triggers")
                break

        print(f'Epoch [{epoch}], '
              f'Train Loss: {avg_train_loss:.4f}, '
              f'Val Loss: {avg_val_loss:.4f}, '
              f'Val F1 (Macro): {val_f1:.4f}')

    print("Finish training")
    model_tuned.load_state_dict(torch.load('best_model_tuned.pth'))
```

```
Save best model_tuned (F1: 0.3612)
Epoch [1], Train Loss: 1.4040, Val Loss: 1.2835, Val F1 (Macro): 0.3612
Save best model_tuned (F1: 0.4822)
Epoch [2], Train Loss: 1.2019, Val Loss: 1.1576, Val F1 (Macro): 0.4822
Save best model_tuned (F1: 0.5128)
Epoch [3], Train Loss: 1.0639, Val Loss: 1.0784, Val F1 (Macro): 0.5128
Save best model_tuned (F1: 0.5407)
Epoch [4], Train Loss: 0.9518, Val Loss: 1.0268, Val F1 (Macro): 0.5407
Val Loss (1/3)
Epoch [5], Train Loss: 0.8555, Val Loss: 1.0524, Val F1 (Macro): 0.5452
Val Loss (2/3)
Epoch [6], Train Loss: 0.7450, Val Loss: 1.0955, Val F1 (Macro): 0.5398
Val Loss (3/3)
Early stop triggers
Finish training
<All keys matched successfully>
```

predict on the test set

```
test_set['rating'] = '1star'
test_loader = create_data_loader(test_set, w2i = w2i, y = label_encoder.transform(test_set.ra
```

```
def get_preds(model, loader):
    model.eval()
    all_preds = []

    with torch.no_grad():
        for inputs, labels in loader:
            inputs = inputs.to(device)
            outputs = model(inputs)

            _, predicted = torch.max(outputs.data, 1)

            preds = predicted.cpu().numpy()

            all_preds.extend(['star' + str(x) for x in preds])

    return all_preds

test_preds = get_preds(model_tuned, test_loader)
test_preds
```

```
['star0',
```

```
 'star0',
 'star2',
 'star2',
 'star1',
 'star4',
 'star2',
 'star1',
 'star3',
 'star0',
 'star1',
 'star3',
 ]
```

```
out_prediction('Tianhao', 'Cao', test_preds)
```

```
from google.colab import files

filenames = ['best_model.pth', 'best_model_tuned.pth', 'Tianhao_Cao_PRED.txt']

for fname in filenames:
    print(f"downloading {fname}...")
    files.download(fname)
```

```
downloading best_model.pth...
downloading best_model_tuned.pth...
downloading Tianhao_Cao_PRED.txt...
```

开始借助 AI 编写或生成代码。