notebook

November 29, 2024

0.1 Libraries

[4]:
```
%load_ext autoreload
%autoreload 2
```

[5]:
```
from collections import Counter
from collections import defaultdict
import itertools
import json
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
nltk.download('wordnet')  # downloads WordNet data
nltk.download('omw-1.4')  # downloads additional wordnet data for lemmatization
nltk.download('stopwords')  # downloads stopwords (if not already downloaded)
from sklearn.metrics import (
    accuracy_score,
    f1_score,
)
from wordcloud import WordCloud
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from datasets import Dataset
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    Trainer,
    TrainingArguments,
)

from emotion_dataset import EmotionDataset
from loader import loader
from train_model import train_model
from plot_scores import plot_scores
from predict import predict
from predict_on_fly import predict_on_fly
from metrics import compute_metrics, print_metrics, save_metrics
import transformer_model
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\difj6\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\difj6\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\difj6\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
c:\Users\difj6\OneDrive - Syddansk Universitet\Documents\Uni\7. semester\DM873
Deep learning\Project2\.venv\lib\site-packages\tqdm\auto.py:21: TqdmWarning:
IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

0.2 Device

[6]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using {device}")
```

```
Using cuda
```

0.3 Load and save dataset

[7]:
```
train_csv_path = "data/train.csv"
val_csv_path = "data/val.csv"
test_csv_path = "data/test.csv"

train_df, val_df, test_df = loader(train_csv_path, val_csv_path, test_csv_path)
```

1 Task 1: Data Preparation

1.1 Data set

[8]:
```
print(f"# train sentences: {len(train_df)}")
print(f"# validation sentences: {len(val_df)}")
print(f"# test sentences: {len(test_df)}")

print(train_df)
```

```
# train sentences: 16000
# validation sentences: 2000
# test sentences: 2000
                                                    text  label label_name
0                                i didnt feel humiliated      0    sadness
1      i can go from feeling so hopeless to so damned...      0    sadness
2       im grabbing a minute to post i feel greedy wrong      3      anger
3      i am ever feeling nostalgic about the fireplac...      2       love
4                                   i am feeling grouchy      3      anger
...                                                  ...    ...        ...
15995  i just had a very brief time in the beanbag an...      0    sadness
15996  i am now turning and i feel pathetic that i am...      0    sadness
15997                         i feel strong and good overall      1        joy
15998  i feel like this was such a rude comment and i...      3      anger
15999  i know a lot but i feel so stupid because i ca...      0    sadness

[16000 rows x 3 columns]
```

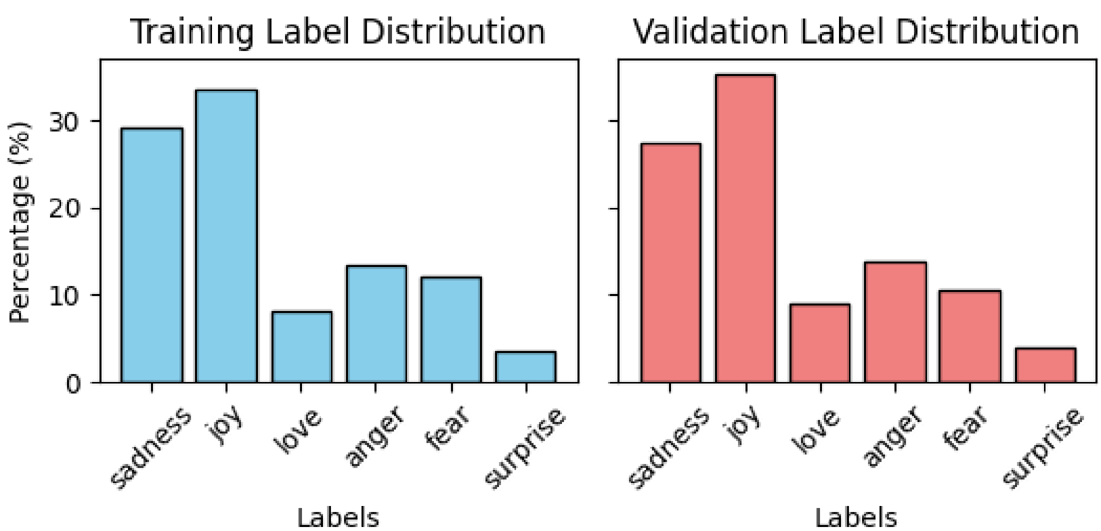
1.2 Step 1: Dataset Preparation

1.2.1 Label distribution

[9]:
```
# Calculate label distributions and percentages for training and validation sets
def calculate_distribution(df):
    label_distribution = df["label_name"].value_counts().reset_index()
    label_distribution.columns = ["label_name", "count"]
    label_distribution["percentage"] = round((label_distribution["count"] / label_distribution["count"].
↪sum()) * 100, 2)
    return label_distribution

# Calculate distributions
train_distribution = calculate_distribution(train_df)
val_distribution = calculate_distribution(val_df)

# Merge with label mapping for alignment
label_map_df = train_df[["label", "label_name"]].drop_duplicates().sort_values("label")
label_map = dict(zip(label_map_df["label"], label_map_df["label_name"]))
train_labels_df = label_map_df.merge(train_distribution, on="label_name", how="left")
val_labels_df = label_map_df.merge(val_distribution, on="label_name", how="left")

# Print training label mapping and distribution
num_classes = len(label_map_df)
print(f"Number of classes: {num_classes}")
```

```
print("Training Label Mapping and Distribution:")
print(train_labels_df.to_string(index=False))

# Plot side-by-side
fig, axes = plt.subplots(1, 2, figsize=(6, 3), sharey=True)

# Training set
axes[0].bar(train_labels_df["label_name"], train_labels_df["percentage"], color="skyblue",␣
↪edgecolor="black")
axes[0].set_title("Training Label Distribution")
axes[0].set_xlabel("Labels")
axes[0].set_ylabel("Percentage (%)")
axes[0].tick_params(axis='x', rotation=45)

# Validation set
axes[1].bar(val_labels_df["label_name"], val_labels_df["percentage"], color="lightcoral", edgecolor="black")
axes[1].set_title("Validation Label Distribution")
axes[1].set_xlabel("Labels")
axes[1].tick_params(axis='x', rotation=45)

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

```
Number of classes: 6
Training Label Mapping and Distribution:
 label label_name  count  percentage
     0    sadness   4666       29.16
     1        joy   5362       33.51
     2       love   1304        8.15
     3      anger   2159       13.49
     4       fear   1937       12.11
     5   surprise    572        3.58
```



1.3 Step 2: Tokenizing

1.3.1 Tokenizer

[10]:
```
tokenizer = RegexpTokenizer(r"[a-zA-Z]+|[!?'´`]+") # sequence that don't match the pattern act as␣
↪separators.
example_sentence = "This?.is,a:cu123stom;tokenization example!<"
example_tokens = tokenizer.tokenize(example_sentence)
print(example_tokens)
```

```
['This', '?', 'is', 'a', 'cu', 'stom', 'tokenization', 'example', '!']
```

1.3.2 Tokenize each split

[11]:
```
train_df["tokens"] = train_df["text"].str.lower().apply(tokenizer.tokenize)
val_df["tokens"] = val_df["text"].str.lower().apply(tokenizer.tokenize)
test_df["tokens"] = test_df["text"].str.lower().apply(tokenizer.tokenize)

train_vocab = set(token for tokens in train_df["tokens"] for token in tokens)

print(f"# words in train vocab: {len(train_vocab)}")
```

```
# words in train vocab: 15212
```

1.3.3 Word frequency

[12]:
```
def get_top_words_per_class(tokens_in, top_n=10):
    tokens_by_class = defaultdict(list)
    for tokens, label in zip(tokens_in, train_df["label_name"]):
        tokens_by_class[label].append(tokens)
    tokens_by_class = dict(tokens_by_class)
    results = []
    for label_name, tokens in tokens_by_class.items():
        flat_tokens = list(itertools.chain.from_iterable(tokens))
        most_common = Counter(flat_tokens).most_common(top_n)
        for word, count in most_common:
            results.append({"Label_name": label_name, "Word": word, "Count": count})
    return pd.DataFrame(results)

print(get_top_words_per_class(train_df["tokens"], top_n=10).to_string(index=False))
```

```
Label_name    Word  Count
   sadness       i   7635
   sadness    feel   3299
   sadness     and   2692
   sadness      to   2335
   sadness     the   2155
   sadness       a   1656
   sadness feeling   1523
   sadness      of   1422
   sadness    that   1299
   sadness      my   1245
     anger       i   3576
     anger    feel   1459
     anger     and   1258
     anger      to   1162
     anger     the   1109
     anger       a    791
     anger feeling    721
     anger    that    705
     anger      of    630
     anger      my    573
      love       i   2120
      love    feel    929
      love     and    902
      love      to    860
      love     the    780
      love       a    571
      love      of    482
      love    that    460
      love      my    399
      love feeling    378
  surprise       i    927
  surprise    feel    356
  surprise     and    354
  surprise     the    335
  surprise      to    267
  surprise       a    256
```

```
surprise    that    212
surprise feeling    209
surprise      of    191
surprise      my    163
    fear       i   3083
    fear    feel   1212
    fear      to   1116
    fear     and   1110
    fear     the   1000
    fear       a    806
    fear feeling    742
    fear      of    614
    fear    that    531
    fear      my    525
     joy       i   8518
     joy    feel   3928
     joy     and   3273
     joy      to   3232
     joy     the   2991
     joy       a   2120
     joy    that   1905
     joy      of   1651
     joy feeling   1539
     joy      my   1378
```

1.3.4 Data cleaning (remove stopwords and stem)

[13]:
```python
stemmer = PorterStemmer()
stop_words = set(stopwords.words("english"))

def preprocess_tokens(tokens):
    return [stemmer.stem(word) for word in tokens if word not in stop_words]

train_df["processed_tokens"] = train_df["tokens"].apply(preprocess_tokens)
val_df["processed_tokens"] = val_df["tokens"].apply(preprocess_tokens)
test_df["processed_tokens"] = test_df["tokens"].apply(preprocess_tokens)

print(get_top_words_per_class(train_df["processed_tokens"], top_n=10).to_string(index=False))
```

```
Label_name      Word  Count
   sadness      feel   4994
   sadness      like    881
   sadness        im    683
   sadness      know    297
   sadness       get    284
   sadness    realli    276
   sadness      time    271
   sadness      make    245
   sadness      want    244
   sadness        go    235
     anger      feel   2261
     anger      like    391
     anger        im    342
     anger       get    175
     anger      time    140
     anger      want    133
     anger     irrit    128
     anger    realli    124
     anger      know    122
     anger      hate    113
      love      feel   1406
      love      like    366
      love      love    277
      love        im    193
      love   support    103
      love    realli     92
      love      know     89
      love      want     89
```

```
    love      time     82
    love      care     82
surprise      feel    601
surprise      amaz    107
surprise      like     92
surprise        im     91
surprise   impress     63
surprise overwhelm     58
surprise     weird     57
surprise   surpris     56
surprise    curiou     54
surprise     funni     49
    fear      feel   2025
    fear        im    322
    fear      like    264
    fear     littl    149
    fear        go    139
    fear      know    136
    fear       bit    118
    fear      want    113
    fear      time    110
    fear       get    107
     joy      feel   5674
     joy      like   1023
     joy        im    799
     joy      make    381
     joy      time    334
     joy       get    322
     joy        go    315
     joy    realli    309
     joy      want    272
     joy      know    261
```

1.3.5 Frequency of words

[14]:

```
word_freq = Counter(itertools.chain.from_iterable(train_df["processed_tokens"]))
pd.DataFrame(word_freq.items(), columns=["Word", "Count"]).sort_values(by="Count", ascending=False).
↪to_csv("results/word_frequencies.csv", index=False)
print(pd.DataFrame(word_freq.most_common(100), columns=["Word", "Count"]).to_string(index=False))
```

```
  Word  Count
  feel  16961
  like   3017
    im   2430
   get    981
  time    979
realli    942
  know    938
  make    935
    go    882
  want    867
  love    805
 littl    736
 think    736
   day    675
 thing    672
 peopl    664
   one    647
 would    646
  even    600
 still    598
   ive    587
  life    555
   way    528
  need    521
   bit    521
someth    514
  much    496
```

dont	482
work	471
could	453
say	450
start	445
look	423
see	419
back	414
tri	410
good	408
pretti	392
right	357
alway	356
come	351
help	342
friend	340
also	337
year	336
today	332
use	326
take	317
around	315
person	303
cant	301
made	296
hate	285
well	279
though	274
happi	274
didnt	272
got	271
write	270
live	268
felt	266
lot	264
never	264
thought	263
hope	261
someon	259
find	259
everi	254
quit	250
read	246
less	246
sure	240
enough	238
week	236
give	234
mani	232
kind	230
home	227
away	226
support	224
long	222
ever	221
anyth	220
actual	220
talk	215
better	213
keep	212
left	211
let	210
everyth	210
without	209
rememb	209
last	207
care	205
tell	205

```
   world    205
  wonder    204
 sometim    201
     new    199
    http    199
```

1.3.6 Remove additional words

[15]:
```
additional_words_to_remove = ["feel", "realli", "im", "know", "also", "http"]

def remove_additional_words(tokens):
    return [word for word in tokens if word not in additional_words_to_remove]

train_df["processed_tokens"] = train_df["processed_tokens"].apply(remove_additional_words)
val_df["processed_tokens"] = val_df["processed_tokens"].apply(remove_additional_words)
test_df["processed_tokens"] = test_df["processed_tokens"].apply(remove_additional_words)

train_df["processed_text"] = train_df["processed_tokens"].apply(" ".join)
val_df["processed_text"] = val_df["processed_tokens"].apply(" ".join)
test_df["processed_text"] = test_df["processed_tokens"].apply(" ".join)

print(get_top_words_per_class(train_df["processed_tokens"], top_n=10).to_string(index=False))
```

```
Label_name      Word  Count
   sadness      like    881
   sadness       get    284
   sadness      time    271
   sadness      make    245
   sadness      want    244
   sadness        go    235
   sadness       day    224
   sadness     thing    221
   sadness       ive    217
   sadness     think    212
     anger      like    391
     anger       get    175
     anger      time    140
     anger      want    133
     anger     irrit    128
     anger      hate    113
     anger     thing    109
     anger      make    108
     anger        go    108
     anger     think    105
      love      like    366
      love      love    277
      love   support    103
      love      want     89
      love      time     82
      love      care     82
      love      long     72
      love       one     70
      love       get     70
      love     sweet     69
  surprise      amaz    107
  surprise      like     92
  surprise   impress     63
  surprise overwhelm     58
  surprise     weird     57
  surprise   surpris     56
  surprise    curiou     54
  surprise     funni     49
  surprise    strang     46
  surprise     shock     46
      fear      like    264
      fear     littl    149
      fear        go    139
      fear       bit    118
```

```
fear      want    113
fear      time    110
fear       get    107
fear      make    105
fear     think     94
fear     peopl     90
 joy      like   1023
 joy      make    381
 joy      time    334
 joy       get    322
 joy        go    315
 joy      want    272
 joy      love    257
 joy       day    241
 joy     think    233
 joy       one    211
```

1.3.7 WordCloud

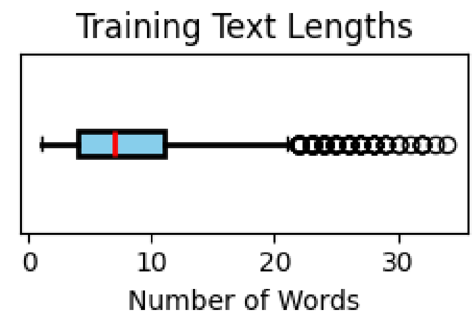
[16]:
```python
wordcloud = WordCloud(width=400, height=200, background_color="white").generate(" ".join(list(itertools.
↪chain.from_iterable(train_df["processed_tokens"]))))
plt.figure(figsize=(5, 3))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

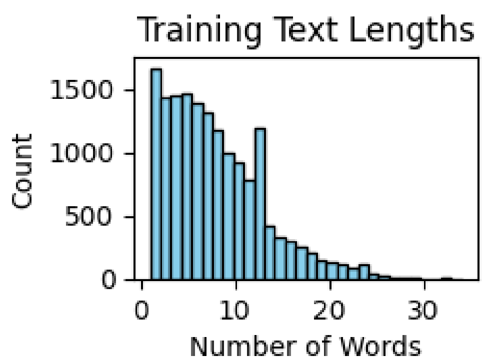
1.3.8 Sentence length distribution

[17]:
```python
train_lengths = [len(tokens) for tokens in train_df["processed_tokens"]]
mean_length = np.mean(train_lengths)
std_dev = np.std(train_lengths)

print(f"Length range for train: from {min(train_lengths)} to {max(train_lengths)} words")
print(f"Mean length for train: {mean_length:.0f} words")
print(f"Standard deviation for train: {std_dev:.0f}")

# Plot boxplot
plt.figure(figsize=(3, 1.2))
plt.boxplot(
    train_lengths,
    vert=False,
    patch_artist=True,
    boxprops=dict(facecolor="skyblue", linewidth=2), # larger, colored box
    whiskerprops=dict(linewidth=2),  # Thicker whiskers
    medianprops=dict(color="red", linewidth=2), # highlight the median
)
plt.title("Training Text Lengths")
```

```
plt.xlabel("Number of Words")
plt.yticks([])
plt.show()

# Plot distribution of lengths
plt.figure(figsize=(2.3, 1.5))
plt.hist(train_lengths, bins=30, color="skyblue", edgecolor="black")
plt.title("Training Text Lengths")
plt.xlabel("Number of Words")
plt.ylabel("Count")
plt.show()
```

```
Length range for train: from 1 to 34 words
Mean length for train: 8 words
Standard deviation for train: 5
```





1.3.9 Set max and min length

[18]:
```
# Set max length for the model. If sentence is longer, truncate it. If shorter, pad it.
# Set min length to remove very short sentences from the training set.
max_length = 10
min_length = 3

print(f"# train sentences before filtering: {len(train_df)}")
train_df = train_df[train_df["processed_tokens"].apply(len) >= min_length]
print(f"# train sentences after filtering: {len(train_df)}")
```

```
# train sentences before filtering: 16000
# train sentences after filtering: 14331
```

1.4 Step 3: Build a vocabulary

[19]:
```
vocab = {"<PAD>": 0, "<UNK>": 1}
for tokens in train_df["processed_tokens"]:
    for token in tokens:
        if token not in vocab:
            vocab[token] = len(vocab)

vocab_size = len(vocab)
print(f"Vocabulary size: {vocab_size}")

reverse_vocab = {v: k for k, v in vocab.items()}
```

```
Vocabulary size: 10336
```

1.5 Step 4: Encode all texts with the vocabulary

[20]:
```
def encode(tokens): # i.e. words to integers
    return [vocab[token] if token in vocab else 1 for token in tokens]

train_df["encoded"] = train_df["processed_tokens"].apply(encode)
val_df["encoded"] = val_df["processed_tokens"].apply(encode)
test_df["encoded"] = test_df["processed_tokens"].apply(encode)
```

1.6 Step 5: Maximum sequence length

[21]:
```
def pad(sequence):
    return sequence[:max_length] + [0] * (max_length - len(sequence))

train_df["padded"] = train_df["encoded"].apply(pad)
val_df["padded"] = val_df["encoded"].apply(pad)
test_df["padded"] = test_df["encoded"].apply(pad)
```

2 Task 2: RNN model

2.1 Model class

[72]:
```
class RNN_model(nn.Module):
    def __init__(self, type, vocab_size, embedding_dim, hidden_size, num_classes, padding_idx=0,␣
↪num_layers=1, dropout_rnn=0, dropout_fc=0):
        super(RNN_model, self).__init__()
        # embedding_dim: size of each embedding vector
        # hidden_size: number of features in the hidden state
        # num_layers: number of recurrent layers
        # bias: introduces a bias
        # batch_first: input and output tensors are provided as (batch, seq, feature)
        # dropout: if non-zero, introduces a dropout layer on the outputs of each RNN layer except the last␣
↪layer

        self.embedding = nn.Embedding(num_embeddings=vocab_size, embedding_dim=embedding_dim,␣
↪padding_idx=padding_idx)
        if type == "RNN":
            self.rnn = nn.RNN(input_size=embedding_dim, hidden_size=hidden_size, num_layers=num_layers,␣
↪bias=True, batch_first=True, dropout=dropout_rnn, nonlinearity="tanh")
        elif type == "GRU":
            self.rnn = nn.GRU(input_size=embedding_dim, hidden_size=hidden_size, num_layers=num_layers,␣
↪bias=True, batch_first=True, dropout=dropout_rnn)
        elif type == "LSTM":
            self.rnn = nn.LSTM(input_size=embedding_dim, hidden_size=hidden_size, num_layers=num_layers,␣
↪bias=True, batch_first=True, dropout=dropout_rnn)
        self.fc = nn.Linear(in_features=hidden_size, out_features=num_classes)
        self.dropout = nn.Dropout(p=dropout_fc)

    def forward(self, x):
        x = self.embedding(x)
        x, _ = self.rnn(x)
```

```
        x = x[:, -1, :]  # extract last hidden state for each sequence
        x = self.dropout(x) # apply dropout to the last hidden state
        x = self.fc(x) # pass last hidden state through the fully connected layer
        return x
```

2.2 Hyper parameter tuning

[73]:
```
train_dataset = EmotionDataset(train_df["padded"].tolist(), train_df["label"].tolist())
val_dataset = EmotionDataset(val_df["padded"].tolist(), val_df["label"].tolist())
test_dataset = EmotionDataset(test_df["padded"].tolist(), test_df["label"].tolist())
```

[ ]:
```
def grid_search(vocab_size, num_classes, train_dataset, val_dataset, label_map, device):
    param_grid = {
        # The best hyperparameters found is commented out
        "type": ["LSTM", "GRU", "RNN"], # "LSTM"
        "embedding_dim": [100, 75], # 75
        "hidden_size": [512, 256], # 256
        "layers": [1, 2], # 1
        "dropout_rnn": [0.0, 0.2], # 0.0
        "dropout_fc": [0.4, 0.6], # 0.4
        "learning_rate": [0.001, 0.0005], # 0.0001
        "reg_lambda": [0.0001, 0.00005], # 0.0001
        "batch_size": [16, 32], # 16
    }

    # Generate all combinations of hyperparameters
    keys, values = zip(*param_grid.items())
    configs = [dict(zip(keys, v)) for v in itertools.product(*values)]
    total_configs = len(configs)

    best_val_f1_score = 0
    results_list = []
    for i, config in enumerate(configs):
        label = f"model_{i}"
        print(f"Training model: {label} ({i+1}/{total_configs})")
        print(json.dumps(config, indent=4))

        train_loader = DataLoader(train_dataset, batch_size=config["batch_size"], shuffle=True)
        val_loader = DataLoader(val_dataset, batch_size=config["batch_size"], shuffle=False)

        # Model
        model = RNN_model(
            type=config["type"],
            vocab_size=vocab_size,
            embedding_dim=config["embedding_dim"],
            hidden_size=config["hidden_size"],
            num_classes=num_classes,
            num_layers=config["layers"],
            dropout_rnn=config["dropout_rnn"],
            dropout_fc=config["dropout_fc"],
        )

        # Train
        results = train_model(
            label=label,
            model=model,
            train_loader=train_loader,
            val_loader=val_loader,
            label_map=label_map,
            device=device,
            optimizer_type="AdamW",
            learning_rate=config["learning_rate"],
            reg_type="L2",
            reg_lambda=config["reg_lambda"],
            num_epochs=10
        )
```

```
        # Track the best model
        current_val_f1_score = max(results["val_f1_scores"])
        if current_val_f1_score > best_val_f1_score:
            best_val_f1_score = current_val_f1_score
            best_model_label = label
            print(f"New best model found: {best_model_label} with val f1 score: {best_val_f1_score:.4f}")

        # Add results to list
        results["config"] = config
        results_list.append(results)

    # Save results
    with open("results/grid_search_results.json", "w") as f:
        json.dump(results_list, f, indent=4)

    print(f"Best model found: {best_model_label}")

grid_search(vocab_size, num_classes, train_dataset, val_dataset, label_map, device)
```

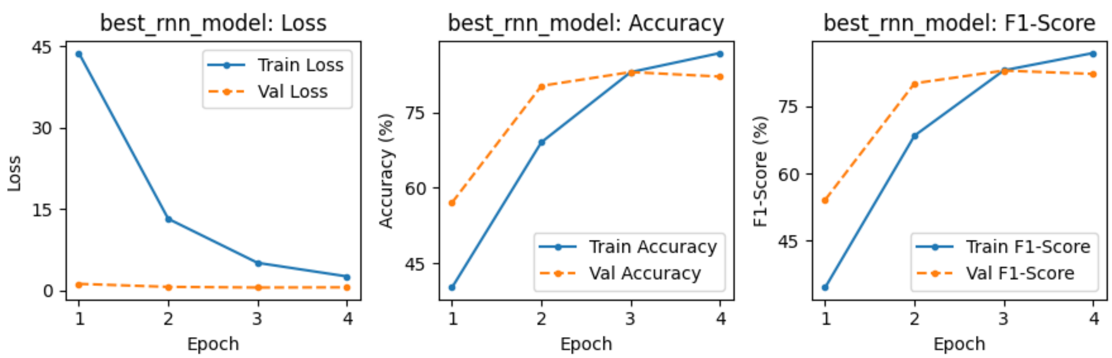
2.3 Best RNN model

[75]:

```
label = "best_rnn_model"
best_batch_size = 16

train_loader = DataLoader(train_dataset, batch_size=best_batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=best_batch_size, shuffle=False)

best_rnn_model = RNN_model(
    type="LSTM",
    vocab_size=vocab_size,
    embedding_dim=75,
    hidden_size=256,
    num_classes=num_classes,
    num_layers=1,
    dropout_rnn=0.0,
    dropout_fc=0.4,
)

best_rnn_results = train_model(
    label=label,
    model=best_rnn_model,
    train_loader=train_loader,
    val_loader=val_loader,
    label_map=label_map,
    device=device,
    optimizer_type="AdamW",
    learning_rate=0.001,
    reg_type="L2",
    reg_lambda=0.0001,
    num_epochs=4
)

plot_scores(best_rnn_results, label)
```

```
Epoch 1/4 (10s) | Train: loss 43.7021, acc 40.13%, f1 34.61% | Val: loss 1.1907,
acc 57.05%, f1 54.13%
Epoch 2/4 (7s) | Train: loss 13.1735, acc 69.10%, f1 68.43% | Val: loss 0.6395,
acc 80.35%, f1 80.08%
Epoch 3/4 (7s) | Train: loss 5.0673, acc 83.09%, f1 82.97% | Val: loss 0.5178,
acc 83.10%, f1 82.92%
Epoch 4/4 (7s) | Train: loss 2.5631, acc 86.90%, f1 86.83% | Val: loss 0.551,
acc 82.20%, f1 82.18%
Total Training Time: 32s

accuracy: 0.822
f1: 0.8218
precision: 0.8257
```

```
recall: 0.822
classification report:
              precision    recall  f1-score   support

     sadness       0.80      0.91      0.85       550
         joy       0.89      0.83      0.86       704
        love       0.69      0.69      0.69       178
       anger       0.86      0.76      0.80       275
        fear       0.79      0.80      0.79       212
    surprise       0.75      0.77      0.76        81

    accuracy                           0.82      2000
   macro avg       0.79      0.79      0.79      2000
weighted avg       0.83      0.82      0.82      2000

confusion matrix:
[502, 15, 7, 8, 16, 2]
[54, 581, 42, 10, 10, 7]
[15, 29, 122, 6, 3, 3]
[41, 14, 3, 208, 8, 1]
[16, 6, 4, 9, 169, 8]
[3, 5, 0, 2, 9, 62]
```



3 Task 3: Transformer model

[76]:
```
batch_size = 32
train_dataset = EmotionDataset(train_df["padded"].tolist(), train_df["label"].tolist())
val_dataset = EmotionDataset(val_df["padded"].tolist(), val_df["label"].tolist())
test_dataset = EmotionDataset(test_df["padded"].tolist(), test_df["label"].tolist())
train_labels = train_df["label"].tolist()
val_labels = val_df["label"].tolist()
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
n_embeds = 10336
num_epochs = 50
```

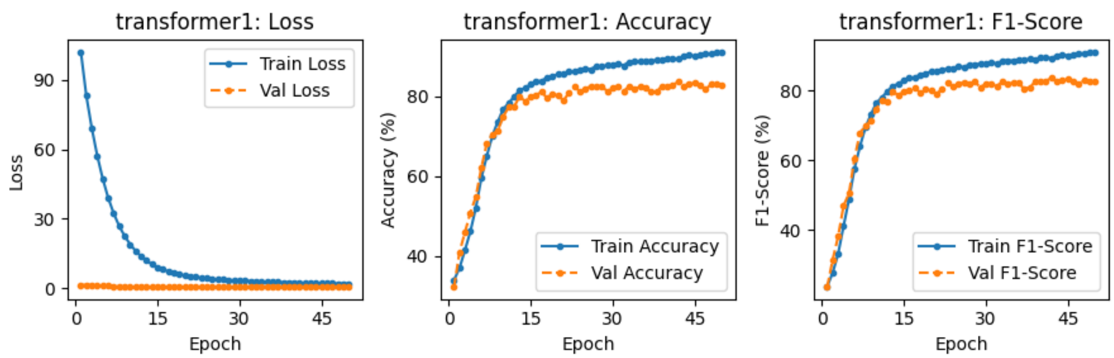
[81]:
```
config = {
        "label": "transformer1",
        "d_key": 64,
        "n_heads": 8,
        "mlp_factor": 4,
        "d_model": 128,
        "n_layers": 6,
        "dropout1": 0.5,
        "dropout2": 0.25,
        "optimizer_type": "Adam",
        "learning_rate": 0.0003,
        "weight_decay": 0,
```

```
        "reg_type": "L1",
        "reg_lambda": 1e-4,
    }
print(f"Training model: {config['label']}")
model = transformer_model.TransformerClassifier(n_embeds=n_embeds, n_classes=6, d_model=config["d_model"],
 ↪d_key=config["d_key"], n_heads=config["n_heads"], mlp_factor=config["mlp_factor"],
 ↪n_layers=config["n_layers"], device = device, dropout1=config["dropout1"], dropout2=config["dropout2"])
results = train_model(
    label=config["label"],
    model=model,
    train_loader=train_loader,
    val_loader=val_loader,
    label_map=label_map,
    device=device,
    optimizer_type=config["optimizer_type"],
    learning_rate=config["learning_rate"],
    weight_decay=config["weight_decay"],
    reg_type=config["reg_type"],
    reg_lambda=config["reg_lambda"],
    num_epochs=num_epochs,
)
plot_scores(results, config["label"])
```

```
Training model: transformer1
Epoch 1/50 (89s) | Train: loss 102.0006, acc 33.86%, f1 23.71% | Val: loss
1.5793, acc 32.10%, f1 23.49%
Epoch 2/50 (76s) | Train: loss 82.9869, acc 36.89%, f1 27.59% | Val: loss
1.5281, acc 40.85%, f1 31.31%
Epoch 3/50 (76s) | Train: loss 69.0236, acc 41.38%, f1 33.03% | Val: loss
1.4376, acc 45.90%, f1 38.03%
Epoch 4/50 (80s) | Train: loss 57.1575, acc 46.12%, f1 41.04% | Val: loss
1.3466, acc 50.80%, f1 46.88%
Epoch 5/50 (77s) | Train: loss 47.2545, acc 52.08%, f1 48.79% | Val: loss
1.1978, acc 54.85%, f1 50.44%
Epoch 6/50 (80s) | Train: loss 39.011, acc 59.49%, f1 57.57% | Val: loss 1.062,
acc 62.00%, f1 60.50%
Epoch 7/50 (79s) | Train: loss 32.3509, acc 65.00%, f1 63.95% | Val: loss
0.9193, acc 68.25%, f1 67.81%
Epoch 8/50 (77s) | Train: loss 26.9274, acc 70.02%, f1 69.34% | Val: loss
0.8478, acc 70.40%, f1 69.82%
Epoch 9/50 (76s) | Train: loss 22.5769, acc 73.58%, f1 73.13% | Val: loss
0.8064, acc 71.35%, f1 71.28%
Epoch 10/50 (76s) | Train: loss 19.0561, acc 76.75%, f1 76.42% | Val: loss
0.712, acc 74.90%, f1 74.65%
Epoch 11/50 (76s) | Train: loss 16.2046, acc 78.24%, f1 77.94% | Val: loss
0.6758, acc 77.20%, f1 77.05%
Epoch 12/50 (76s) | Train: loss 13.8662, acc 79.91%, f1 79.69% | Val: loss
0.6526, acc 77.35%, f1 76.76%
Epoch 13/50 (76s) | Train: loss 11.9736, acc 81.45%, f1 81.25% | Val: loss
0.6205, acc 79.95%, f1 79.66%
Epoch 14/50 (77s) | Train: loss 10.4343, acc 82.03%, f1 81.87% | Val: loss
0.6295, acc 78.65%, f1 78.57%
Epoch 15/50 (77s) | Train: loss 9.1926, acc 82.99%, f1 82.84% | Val: loss
0.5959, acc 79.95%, f1 79.67%
Epoch 16/50 (76s) | Train: loss 8.1612, acc 83.69%, f1 83.55% | Val: loss
0.5867, acc 80.30%, f1 80.02%
Epoch 17/50 (77s) | Train: loss 7.3331, acc 83.83%, f1 83.73% | Val: loss
0.5698, acc 81.00%, f1 80.82%
Epoch 18/50 (76s) | Train: loss 6.6518, acc 84.61%, f1 84.49% | Val: loss
0.6136, acc 79.55%, f1 79.36%
Epoch 19/50 (76s) | Train: loss 6.104, acc 84.82%, f1 84.73% | Val: loss 0.5767,
acc 80.55%, f1 80.34%
Epoch 20/50 (76s) | Train: loss 5.6224, acc 85.53%, f1 85.45% | Val: loss
0.5924, acc 80.15%, f1 79.88%
Epoch 21/50 (76s) | Train: loss 5.2473, acc 85.56%, f1 85.48% | Val: loss
0.6482, acc 78.85%, f1 78.85%
Epoch 22/50 (76s) | Train: loss 4.9044, acc 86.11%, f1 86.02% | Val: loss
0.5677, acc 80.80%, f1 80.52%
```

```
Epoch 23/50 (76s) | Train: loss 4.6268, acc 86.14%, f1 86.07% | Val: loss
0.5594, acc 82.25%, f1 82.19%
Epoch 24/50 (76s) | Train: loss 4.3677, acc 86.46%, f1 86.37% | Val: loss
0.5706, acc 81.20%, f1 80.95%
Epoch 25/50 (78s) | Train: loss 4.1486, acc 86.97%, f1 86.90% | Val: loss
0.5459, acc 81.90%, f1 81.75%
Epoch 26/50 (76s) | Train: loss 3.9806, acc 86.69%, f1 86.62% | Val: loss 0.542,
acc 82.40%, f1 82.38%
Epoch 27/50 (77s) | Train: loss 3.7892, acc 87.52%, f1 87.46% | Val: loss 0.542,
acc 82.30%, f1 81.97%
Epoch 28/50 (76s) | Train: loss 3.6444, acc 87.35%, f1 87.29% | Val: loss
0.5416, acc 82.55%, f1 82.39%
Epoch 29/50 (76s) | Train: loss 3.4955, acc 87.80%, f1 87.74% | Val: loss
0.5703, acc 81.15%, f1 80.88%
Epoch 30/50 (77s) | Train: loss 3.389, acc 87.71%, f1 87.65% | Val: loss 0.546,
acc 82.00%, f1 81.85%
Epoch 31/50 (76s) | Train: loss 3.2559, acc 88.16%, f1 88.10% | Val: loss
0.5646, acc 82.30%, f1 81.96%
Epoch 32/50 (76s) | Train: loss 3.1771, acc 87.64%, f1 87.59% | Val: loss
0.5844, acc 81.15%, f1 81.21%
Epoch 33/50 (76s) | Train: loss 3.0501, acc 88.59%, f1 88.54% | Val: loss
0.5427, acc 82.45%, f1 82.40%
Epoch 34/50 (76s) | Train: loss 2.9593, acc 88.60%, f1 88.55% | Val: loss
0.5736, acc 81.90%, f1 81.82%
Epoch 35/50 (76s) | Train: loss 2.8801, acc 88.61%, f1 88.56% | Val: loss
0.5598, acc 82.25%, f1 82.16%
Epoch 36/50 (77s) | Train: loss 2.8071, acc 88.83%, f1 88.78% | Val: loss
0.5595, acc 82.15%, f1 82.04%
Epoch 37/50 (77s) | Train: loss 2.7384, acc 88.84%, f1 88.79% | Val: loss
0.6097, acc 81.05%, f1 80.46%
Epoch 38/50 (76s) | Train: loss 2.659, acc 89.01%, f1 88.96% | Val: loss 0.5831,
acc 81.10%, f1 80.82%
Epoch 39/50 (77s) | Train: loss 2.5879, acc 88.94%, f1 88.89% | Val: loss
0.5631, acc 82.45%, f1 82.42%
Epoch 40/50 (77s) | Train: loss 2.5379, acc 89.44%, f1 89.40% | Val: loss 0.589,
acc 82.45%, f1 82.54%
Epoch 41/50 (77s) | Train: loss 2.4701, acc 89.35%, f1 89.31% | Val: loss
0.5726, acc 82.80%, f1 82.71%
Epoch 42/50 (76s) | Train: loss 2.4347, acc 89.32%, f1 89.28% | Val: loss
0.5533, acc 83.55%, f1 83.54%
Epoch 43/50 (77s) | Train: loss 2.3524, acc 89.97%, f1 89.93% | Val: loss
0.5814, acc 82.50%, f1 82.40%
Epoch 44/50 (76s) | Train: loss 2.3024, acc 90.38%, f1 90.34% | Val: loss
0.5854, acc 83.15%, f1 83.07%
Epoch 45/50 (77s) | Train: loss 2.2664, acc 89.99%, f1 89.95% | Val: loss
0.5644, acc 83.30%, f1 83.14%
Epoch 46/50 (76s) | Train: loss 2.2204, acc 90.38%, f1 90.34% | Val: loss
0.5628, acc 82.60%, f1 82.43%
Epoch 47/50 (76s) | Train: loss 2.167, acc 90.64%, f1 90.61% | Val: loss 0.606,
acc 81.65%, f1 81.72%
Epoch 48/50 (76s) | Train: loss 2.1392, acc 90.64%, f1 90.62% | Val: loss
0.5831, acc 83.00%, f1 82.94%
Epoch 49/50 (76s) | Train: loss 2.0786, acc 91.10%, f1 91.07% | Val: loss
0.5902, acc 82.90%, f1 82.65%
Epoch 50/50 (76s) | Train: loss 2.056, acc 90.95%, f1 90.92% | Val: loss 0.594,
acc 82.80%, f1 82.62%
Total Training Time: 3843s

accuracy: 0.828
f1: 0.8262
precision: 0.8267
recall: 0.828
classification report:
              precision    recall  f1-score   support

     sadness       0.88      0.85      0.87       550
         joy       0.83      0.89      0.86       704
        love       0.74      0.60      0.66       178
```

```
       anger       0.81      0.82      0.81       275
        fear       0.79      0.81      0.80       212
    surprise       0.74      0.68      0.71        81

    accuracy                           0.83      2000
   macro avg       0.80      0.78      0.79      2000
weighted avg       0.83      0.83      0.83      2000

confusion matrix:
[470, 34, 7, 18, 18, 3]
[23, 628, 24, 17, 5, 7]
[8, 51, 107, 8, 3, 1]
[16, 17, 4, 225, 12, 1]
[14, 12, 2, 6, 171, 7]
[2, 12, 0, 4, 8, 55]
```



Train multiple:

[ ]:
```
configurations = [
    #heads 16
    {
        "label": "transformer1",
        "d_key": 64,
        "n_heads": 16,
        "mlp_factor": 4,
        "d_model": 128,
        "n_layers": 6,
        "dropout1": 0.5,
        "dropout2": 0.25,
        "optimizer_type": "Adam",
        "learning_rate": 0.001,
        "weight_decay": 0,
        "reg_type": "L2",
        "reg_lambda": 1e-4,
    },
    # mlp_factor 2
    {
        "label": "transformer1",
        "d_key": 64,
        "n_heads": 8,
        "mlp_factor": 2,
        "d_model": 128,
        "n_layers": 6,
        "dropout1": 0.5,
        "dropout2": 0.25,
        "optimizer_type": "Adam",
        "learning_rate": 0.001,
        "weight_decay": 0,
        "reg_type": "L2",
```

```
        "reg_lambda": 1e-4,
    },
    # 0,0003 learning rate
    {
        "label": "transformer1",
        "d_key": 64,
        "n_heads": 8,
        "mlp_factor": 4,
        "d_model": 128,
        "n_layers": 6,
        "dropout1": 0.5,
        "dropout2": 0.25,
        "optimizer_type": "Adam",
        "learning_rate": 0.0003,
        "weight_decay": 0,
        "reg_type": "L2",
        "reg_lambda": 1e-4,
    },
    # L1 reg
    {
        "label": "transformer1",
        "d_key": 64,
        "n_heads": 8,
        "mlp_factor": 4,
        "d_model": 128,
        "n_layers": 6,
        "dropout1": 0.5,
        "dropout2": 0.25,
        "optimizer_type": "Adam",
        "learning_rate": 0.001,
        "weight_decay": 0,
        "reg_type": "L1",
        "reg_lambda": 1e-4,
    },
    # More dropout
    {
        "label": "transformer1",
        "d_key": 64,
        "n_heads": 8,
        "mlp_factor": 4,
        "d_model": 128,
        "n_layers": 6,
        "dropout1": 0.6,
        "dropout2": 0.3,
        "optimizer_type": "Adam",
        "learning_rate": 0.001,
        "weight_decay": 0,
        "reg_type": "L2",
        "reg_lambda": 1e-4,
    },
]
for config in configurations:
    print(f"Training model: {config['label']}")
    model = transformer_model.TransformerClassifier(n_embeds=n_embeds, n_classes=6,␣
↪d_model=config["d_model"], d_key=config["d_key"], n_heads=config["n_heads"],␣
↪mlp_factor=config["mlp_factor"], n_layers=config["n_layers"], device = device,␣
↪dropout1=config["dropout1"], dropout2=config["dropout2"])
    results = train_model(
        label=config["label"],
        model=model,
        train_loader=train_loader,
        val_loader=val_loader,
        label_map=label_map,
        device=device,
        optimizer_type=config["optimizer_type"],
        learning_rate=config["learning_rate"],
        weight_decay=config["weight_decay"],
```

```
        reg_type=config["reg_type"],
        reg_lambda=config["reg_lambda"],
        num_epochs=num_epochs,
    )
    plot_scores(results, config["label"])
```

4 Task 4: Analysis

[ ]:
```
test_loader = DataLoader(test_dataset, batch_size=best_batch_size, shuffle=True)

predict("best_rnn_model", best_rnn_model, device, test_loader, label_map, reverse_vocab)
```

```
accuracy: 0.8245
f1: 0.8248
precision: 0.8261
recall: 0.8245
classification report:
              precision    recall  f1-score   support

     sadness       0.85      0.90      0.87       581
         joy       0.87      0.82      0.84       695
        love       0.64      0.70      0.67       159
       anger       0.83      0.81      0.82       275
        fear       0.81      0.81      0.81       224
    surprise       0.64      0.62      0.63        66

    accuracy                           0.82      2000
   macro avg       0.77      0.78      0.78      2000
weighted avg       0.83      0.82      0.82      2000

confusion matrix:
[520, 31, 4, 12, 12, 2]
[35, 571, 52, 20, 8, 9]
[16, 23, 112, 4, 2, 2]
[20, 20, 4, 223, 7, 1]
[17, 6, 2, 8, 182, 9]
[4, 7, 0, 1, 13, 41]
```

[82]:
```
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=True)

predict("best_rnn_model", model, device, test_loader, label_map, reverse_vocab)
```

```
accuracy: 0.813
f1: 0.8115
precision: 0.8126
recall: 0.813
classification report:
              precision    recall  f1-score   support

     sadness       0.89      0.82      0.85       581
         joy       0.81      0.88      0.85       695
        love       0.68      0.59      0.63       159
       anger       0.77      0.81      0.79       275
        fear       0.80      0.80      0.80       224
    surprise       0.67      0.56      0.61        66

    accuracy                           0.81      2000
   macro avg       0.77      0.75      0.76      2000
weighted avg       0.81      0.81      0.81      2000

confusion matrix:
[479, 52, 6, 28, 13, 3]
[21, 614, 29, 15, 9, 7]
[7, 46, 94, 11, 0, 1]
[18, 19, 7, 222, 9, 0]
[15, 10, 2, 10, 180, 7]
```

```
[0, 13, 0, 3, 13, 37]
```

[ ]:
```python
config = {
        "label": "transformer1",
        "d_key": 64,
        "n_heads": 8,
        "mlp_factor": 4,
        "d_model": 128,
        "n_layers": 6,
        "dropout1": 0.5,
        "dropout2": 0.25,
        "optimizer_type": "Adam",
        "learning_rate": 0.0003,
        "weight_decay": 0,
        "reg_type": "L1",
        "reg_lambda": 1e-4,
    }
model = transformer_model.TransformerClassifier(n_embeds=n_embeds, n_classes=6, d_model=config["d_model"],
↪d_key=config["d_key"], n_heads=config["n_heads"], mlp_factor=config["mlp_factor"],
↪n_layers=config["n_layers"], device = device, dropout1=config["dropout1"], dropout2=config["dropout2"])
model.load_state_dict(torch.load('models/transformer1.pth'))
predict_on_fly(model, tokenizer, vocab, device, label_map, max_length)
```

5 Task 5: Pre-trained model (transfer learning)

[ ]:
```python
model_name = "distilbert-base-uncased"
num_epochs = 5
learning_rate = 5e-6
max_length = 32 # different max length as we are using raw data to be tokenized using the pretrained
↪tokenizer
batch_size = 32
label2id = {v: k for k, v in label_map.items()}

# Metric function
def pretrained_evaluation(predictions):
    preds = predictions.predictions.argmax(-1) # get predicted labels
    labels = predictions.label_ids
    accuracy = accuracy_score(labels, preds)
    f1 = f1_score(labels, preds, average="weighted")
    return {"accuracy": accuracy, "f1": f1}

# Load tokenizer
tokenizer_pretrained = AutoTokenizer.from_pretrained(model_name)

# Preprocessing function
def tokenize(batch):
    return tokenizer_pretrained(batch["text"], padding="max_length", truncation=True, max_length=max_length)

# Convert pandas DataFrame to Hugging Face Dataset
train_dataset = Dataset.from_pandas(train_df)
val_dataset = Dataset.from_pandas(val_df)
test_dataset = Dataset.from_pandas(test_df)

# Tokenize datasets
train_dataset = train_dataset.map(tokenize, batched=True)
val_dataset = val_dataset.map(tokenize, batched=True)
test_dataset = test_dataset.map(tokenize, batched=True)

# Set format for PyTorch
train_dataset = train_dataset.rename_column("label", "labels")
val_dataset = val_dataset.rename_column("label", "labels")
test_dataset = test_dataset.rename_column("label", "labels")
train_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "labels"])
val_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "labels"])
test_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "labels"])
```

```python
# Load pre-trained model
model = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    num_labels=num_classes,
    id2label=label_map,
    label2id=label2id
)

# Training arguments
training_args = TrainingArguments(
    output_dir="results",
    eval_strategy="epoch", # evaluate at the end of each epoch
    save_strategy="epoch", # save checkpoints after every epoch
    logging_strategy="epoch",
    learning_rate=learning_rate,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=64,
    weight_decay=0.01,
    num_train_epochs=num_epochs,
    load_best_model_at_end=True, # load best model at the end of training
    metric_for_best_model="f1", # specify metric to monitor
    save_total_limit=1, # keep only the best checkpoint
)

# Define Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=pretrained_evaluation, # compute metrics during evaluation
)

# Train model
trainer.train()

# Evaluate best model
best_results = trainer.evaluate()
print(f"Best Model Evaluation Results:\n {best_results}")

# Save best fine-tuned model
fine_tuned_model_name = "distilbert_finetuned"
trainer.save_model(f"models/{fine_tuned_model_name}")
tokenizer_pretrained.save_pretrained(f"models/{fine_tuned_model_name}")
```

[32]:
```python
# Load fine-tuned model
tokenizer_new = AutoTokenizer.from_pretrained(f"models/{fine_tuned_model_name}")
model_new = AutoModelForSequenceClassification.from_pretrained(f"models/{fine_tuned_model_name}",␣
↪num_labels=num_classes)
trainer = Trainer(model=model_new)

# Predict on test set
predictions = trainer.predict(test_dataset)

# Extract logits and compute predicted labels
logits = torch.tensor(predictions.predictions) # convert logits to a PyTorch tensor
predicted_labels = torch.argmax(logits, dim=1).numpy() # convert to numpy array for sklearn metrics

# Evaluate predictions
metrics = compute_metrics(test_df["label"], predicted_labels, label_map.values())
print_metrics(metrics)
save_metrics(fine_tuned_model_name, metrics)
```

```
100%|| 250/250 [00:01<00:00, 152.01it/s]
```

```
accuracy: 0.8885
f1: 0.8871
```

```
precision: 0.8869
recall: 0.8885
classification report:
              precision    recall  f1-score   support

     sadness       0.92      0.93      0.93       581
         joy       0.89      0.92      0.91       695
        love       0.74      0.70      0.72       159
       anger       0.90      0.90      0.90       275
        fear       0.90      0.87      0.88       224
    surprise       0.80      0.61      0.69        66

    accuracy                           0.89      2000
   macro avg       0.86      0.82      0.84      2000
weighted avg       0.89      0.89      0.89      2000

confusion matrix:
[543, 20, 3, 11, 4, 0]
[14, 641, 32, 6, 1, 1]
[6, 39, 111, 3, 0, 0]
[14, 7, 2, 247, 5, 0]
[11, 2, 0, 7, 195, 9]
[4, 9, 1, 0, 12, 40]
```