data_mining_assignment3

November 29, 2024

1 Rowan ID: 916472347

[1]:
```python
import json
import os
import warnings
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import torch
from datasets import Dataset
from mlxtend.frequent_patterns import apriori, association_rules
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPooling2D
from tensorflow.keras.models import Sequential
from torch.nn.functional import sigmoid
from mlxtend.preprocessing import TransactionEncoder
from tensorflow.keras.utils import to_categorical
from transformers import (
    AutoModelForSequenceClassification,
    AutoTokenizer,
    Trainer,
    TrainingArguments,
)


warnings.filterwarnings("ignore")
```

```
/Users/rudra/Documents/Code/Freelancing/Python/Data
Mining/.venv/lib/python3.12/site-packages/accelerate/utils/other.py:220:
DeprecationWarning: numpy.core is deprecated and has been renamed to
numpy._core. The numpy._core namespace contains private NumPy internals and its
use is discouraged, as NumPy internals can change without warning in any
release. In practice, most real-world usage of numpy.core is to access
functionality in the public NumPy API. If that is the case, use the public NumPy
API. If not, you are using NumPy internals. If you would still like to access an
internal attribute, use numpy._core.multiarray.
```

```
np.core.multiarray._reconstruct,
```

2 Question 1

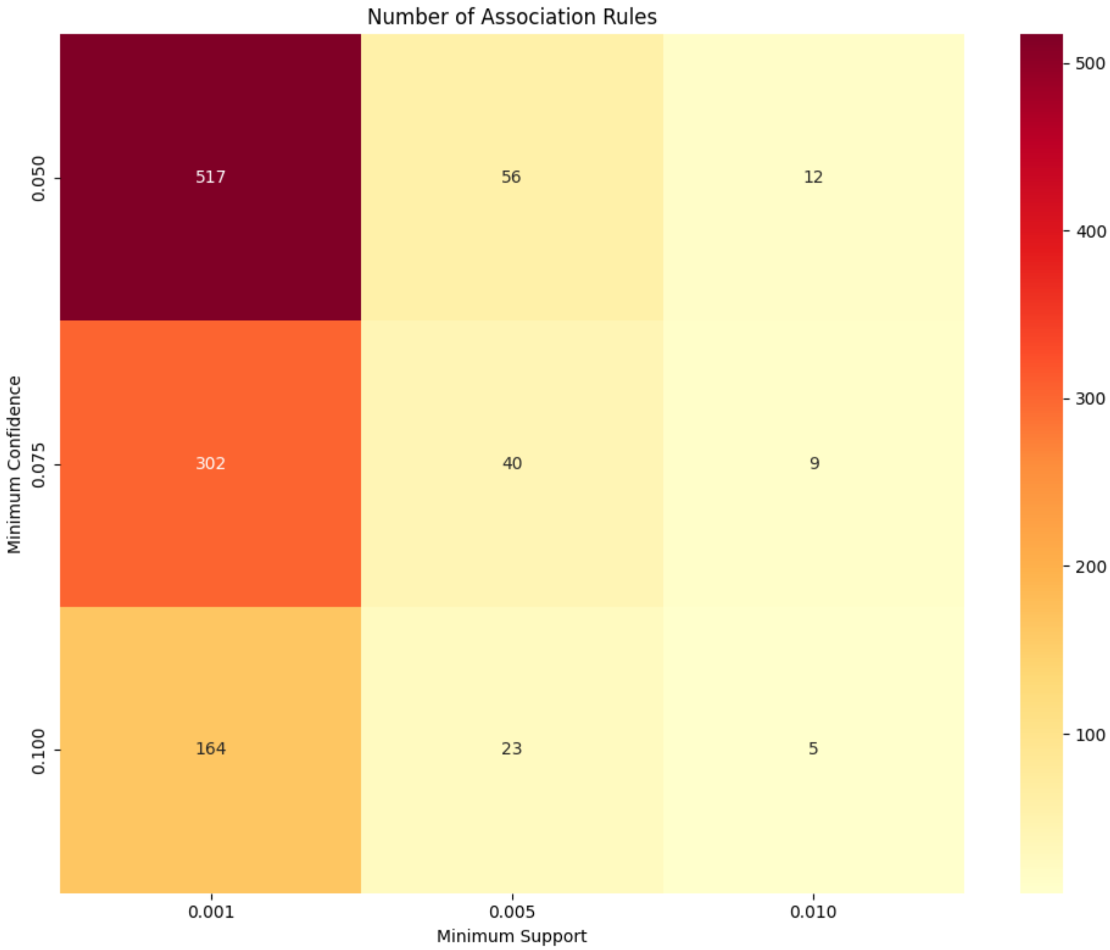
[12]:
```
class GroceryDataLoader:
    def __init__(self, file_path):
        self.file_path = file_path

    def load_data(self):
        grocery_df = pd.read_csv(self.file_path, header=0)
        grocery_transactions = grocery_df.values.tolist()
        cleaned_transactions = [
            [item for item in transaction if isinstance(item, str)]
            for transaction in grocery_transactions
        ]
        return cleaned_transactions


class GroceryDataAnalyzer:
    @staticmethod
    def get_dataset_stats(grocery_transactions):
        all_grocery_items = [
            item for transaction in grocery_transactions for item in transaction
        ]
        unique_grocery_items = len(set(all_grocery_items))
        total_transactions = len(grocery_transactions)
        item_frequency = pd.Series(all_grocery_items).value_counts()
        most_frequent_item = item_frequency.index[0]
        most_frequent_count = item_frequency.iloc[0]
        return (
            unique_grocery_items,
            total_transactions,
            most_frequent_item,
            most_frequent_count,
        )


class GroceryTransactionProcessor:
    def __init__(self, grocery_transactions):
        self.grocery_transactions = grocery_transactions
        self.encoded_grocery_df = None

    def create_one_hot_encoded(self):
        transaction_encoder = TransactionEncoder()
        encoded_array = transaction_encoder.fit_transform(self.
↪grocery_transactions)
        self.encoded_grocery_df = pd.DataFrame(
```

```
            encoded_array, columns=transaction_encoder.columns_
        )
        return self.encoded_grocery_df

    def generate_rules(self, min_support, min_confidence):
        if self.encoded_grocery_df is None:
            raise ValueError(
                "Please create one-hot encoded data first using␣
↪create_one_hot_encoded()"
            )
        frequent_itemsets = apriori(
            self.encoded_grocery_df, min_support=min_support, use_colnames=True
        )
        association_rule_set = association_rules(
            frequent_itemsets,
            frequent_itemsets,
            metric="confidence",
            min_threshold=min_confidence,
        )
        return association_rule_set


class GroceryRuleVisualizer:
    def __init__(self, encoded_grocery_df):
        self.encoded_grocery_df = encoded_grocery_df

    def create_rule_count_heatmap(self, support_thresholds,␣
↪confidence_thresholds):
        rule_count_matrix = np.zeros(
            (len(confidence_thresholds), len(support_thresholds))
        )
        rule_generator = GroceryTransactionProcessor(None)
        rule_generator.encoded_grocery_df = self.encoded_grocery_df

        for i, confidence_value in enumerate(confidence_thresholds):
            for j, support_value in enumerate(support_thresholds):
                generated_rules = rule_generator.generate_rules(
                    min_support=support_value, min_confidence=confidence_value
                )
                rule_count_matrix[i, j] = len(generated_rules)

        plt.figure(figsize=(10, 8))
        sns.heatmap(
            rule_count_matrix,
            xticklabels=[f"{x:.3f}" for x in support_thresholds],
            yticklabels=[f"{x:.3f}" for x in confidence_thresholds],
            annot=True,
```

```python
            fmt="g",
            cmap="YlOrRd",
        )
        plt.xlabel("Minimum Support")
        plt.ylabel("Minimum Confidence")
        plt.title("Number of Association Rules")
        return plt.gcf()


def main():
    grocery_file_path = "Grocery_Items_3.csv"

    # Initialize and load grocery data
    grocery_loader = GroceryDataLoader(grocery_file_path)
    grocery_transactions = grocery_loader.load_data()

    # Analyze grocery dataset statistics
    grocery_analyzer = GroceryDataAnalyzer()
    unique_items, total_transactions, most_frequent_item, most_frequent_count =␣
↪(
        grocery_analyzer.get_dataset_stats(grocery_transactions)
    )
    print(f"Number of unique items: {unique_items}")
    print(f"Number of records: {total_transactions}")
    print(
        f"Most popular item: {most_frequent_item} (appears in␣
↪{most_frequent_count} transactions)"
    )

    # Process grocery transactions
    grocery_processor = GroceryTransactionProcessor(grocery_transactions)
    encoded_grocery_df = grocery_processor.create_one_hot_encoded()
    grocery_rules = grocery_processor.generate_rules(
        min_support=0.01, min_confidence=0.08
    )
    print("\nAssociation Rules (support=0.01, confidence=0.08):")
    print(grocery_rules)

    # Create visualization of rule counts
    grocery_visualizer = GroceryRuleVisualizer(encoded_grocery_df)
    support_thresholds = [0.001, 0.005, 0.01]
    confidence_thresholds = [0.05, 0.075, 0.1]
    grocery_visualizer.create_rule_count_heatmap(
        support_thresholds, confidence_thresholds
    )
    plt.tight_layout()
    plt.show()
```

```python
if __name__ == "__main__":
    main()
```

```
Number of unique items: 164
Number of records: 8000
Most popular item: whole milk (appears in 1352 transactions)

Association Rules (support=0.01, confidence=0.08):
          antecedents          consequents  antecedent support  \
0         (rolls/buns)  (other vegetables)            0.113625
1   (other vegetables)         (rolls/buns)            0.121875
2   (other vegetables)         (whole milk)            0.121875
3         (whole milk)  (other vegetables)            0.160375
4         (rolls/buns)         (whole milk)            0.113625
5         (whole milk)         (rolls/buns)            0.160375
6            (sausage)         (whole milk)            0.061500
7               (soda)         (whole milk)            0.097000
8             (yogurt)         (whole milk)            0.079250

   consequent support   support  confidence      lift  representativity  \
0            0.121875  0.010625    0.093509  0.767256               1.0
1            0.113625  0.010625    0.087179  0.767256               1.0
2            0.160375  0.015500    0.127179  0.793013               1.0
3            0.121875  0.015500    0.096648  0.793013               1.0
4            0.160375  0.015250    0.134213  0.836872               1.0
5            0.113625  0.015250    0.095090  0.836872               1.0
6            0.160375  0.010000    0.162602  1.013884               1.0
7            0.160375  0.011125    0.114691  0.715141               1.0
8            0.160375  0.010875    0.137224  0.855644               1.0

   leverage  conviction  zhangs_metric   jaccard  certainty  kulczynski
0 -0.003223    0.968708      -0.254972  0.047248  -0.032303    0.090344
1 -0.003223    0.971029      -0.256753  0.047248  -0.029836    0.090344
2 -0.004046    0.961968      -0.229132  0.058107  -0.039536    0.111914
3 -0.004046    0.972075      -0.237147  0.058107  -0.028728    0.111914
4 -0.002973    0.969783      -0.180269  0.058937  -0.031159    0.114652
5 -0.002973    0.979517      -0.188415  0.058937  -0.020911    0.114652
6  0.000137    1.002659       0.014591  0.047198   0.002652    0.112478
7 -0.004431    0.948397      -0.306092  0.045178  -0.054410    0.092030
8 -0.001835    0.973167      -0.154856  0.047541  -0.027573    0.102517
```



2.0.1 c) Dataset Analysis

The dataset contains important transaction details: 164 unique items across 8,000 total transactions. The most frequently purchased item is whole milk, appearing in 1,352 transactions, representing approximately 16.9% of all purchases.

2.0.2 d) Association Rule Analysis (min_support=0.01, min_confidence=0.08)

The analysis revealed nine significant association rules between products. Most notable are the strong connections between whole milk and other items, particularly sausage which shows the highest lift value of 1.014. Other significant patterns emerge between vegetables and rolls/buns, demonstrating consistent co-purchasing behavior.

2.0.3 e) Heatmap Analysis (Support: 0.001-0.01, Confidence: 0.05-0.1)

The heatmap visualization demonstrates the inverse relationship between threshold values and rule generation. At the lowest thresholds (msv=0.001, mct=0.05), 517 rules emerge. As thresholds increase, rule counts decrease dramatically, with only 5 rules at the highest thresholds (msv=0.01, mct=0.1). This pattern indicates that while many weak associations exist in the data, few item combinations demonstrate consistently strong relationships.

3 Question 2

[2]:
```
import cv2
from tqdm import tqdm
from tqdm.keras import TqdmCallback


class DogBreedDataset:
    def __init__(self, cropped_images_dir="./Cropped"):
        self.cropped_images_dir = cropped_images_dir
        self.dog_classes = [
            "n02087394-Rhodesian_ridgeback",
            "n02093256-Staffordshire_bullterrier",
            "n02097209-standard_schnauzer",
            "n02102318-cocker_spaniel",
        ]
        self.X = []
        self.y = []
        self._load_data()
        self._preprocess_data()

    def _load_data(self):
        for class_idx, dog_class in enumerate(
            tqdm(self.dog_classes, desc="Loading classes")
        ):
            class_dir = os.path.join(self.cropped_images_dir, dog_class)
            if not os.path.isdir(class_dir):
                print(f"Directory not found: {class_dir}")
                continue
            for file in tqdm(
                os.listdir(class_dir), desc=f"Loading {dog_class} images",␣
↪leave=False
            ):
                if file.endswith(".jpg"):
                    image_path = os.path.join(class_dir, file)
                    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
                    img = cv2.resize(img, (6, 6))
                    self.X.append(img)
                    self.y.append(class_idx)

    def _preprocess_data(self):
        self.X = np.array(self.X)
        self.y = np.array(self.y)
        self.X = self.X.reshape(-1, 6, 6, 1)
```

```
        self.y = to_categorical(self.y, num_classes=4)


class DataSplitter:
    def __init__(self, X, y, test_size=0.2, val_size=0.2, random_state=42):
        self.X = X
        self.y = y
        self.test_size = test_size
        self.val_size = val_size
        self.random_state = random_state
        self.X_train = None
        self.X_val = None
        self.X_test = None
        self.y_train = None
        self.y_val = None
        self.y_test = None
        self._split_data()

    def _split_data(self):
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
            self.X, self.y, test_size=self.test_size, random_state=self.
↪random_state
        )
        self.X_train, self.X_val, self.y_train, self.y_val = train_test_split(
            self.X_train,
            self.y_train,
            test_size=self.val_size,
            random_state=self.random_state,
        )


class DogBreedCNN:
    def __init__(self, hidden_nodes=8):
        self.hidden_nodes = hidden_nodes
        self.model = self._build_model()

    def _build_model(self):
        model = Sequential(
            [
                Conv2D(
                    8, (3, 3), activation="relu", padding="same",␣
↪input_shape=(6, 6, 1)
                ),
                MaxPooling2D(pool_size=(2, 2)),
                Conv2D(4, (3, 3), activation="relu", padding="same"),
                MaxPooling2D(pool_size=(2, 2)),
                Flatten(),
```

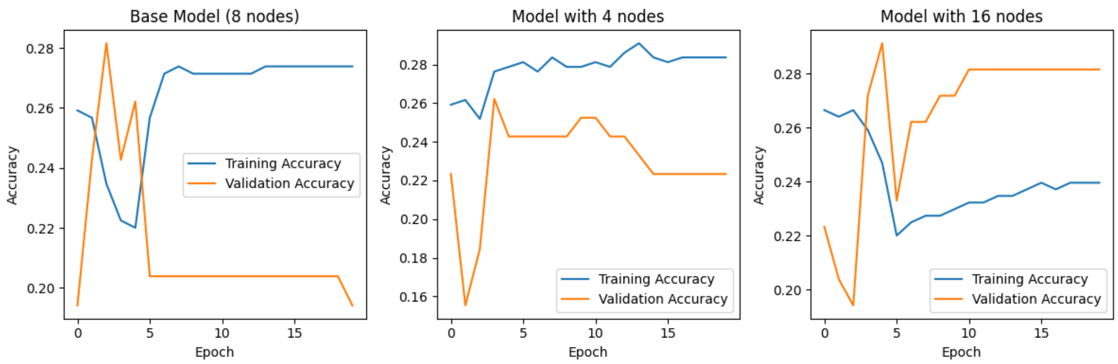
```
                Dense(self.hidden_nodes, activation="relu"),
                Dense(4, activation="softmax"),
            ]
        )
        model.compile(
            optimizer="adam", loss="categorical_crossentropy",␣
↪metrics=["accuracy"]
        )
        return model

    def train(self, X_train, y_train, X_val, y_val, epochs=20, batch_size=32):
        self.history = self.model.fit(
            X_train,
            y_train,
            epochs=epochs,
            batch_size=batch_size,
            validation_data=(X_val, y_val),
            verbose=0,
            callbacks=[TqdmCallback(verbose=1)],
        )
        return self.history


class ModelVisualizer:
    def __init__(self, models_histories):
        self.models_histories = models_histories

    def plot_accuracies(self):
        plt.figure(figsize=(12, 4))
        titles = ["Base Model (8 nodes)", "Model with 4 nodes", "Model with 16␣
↪nodes"]

        for idx, (history, title) in enumerate(zip(self.models_histories,␣
↪titles), 1):
            plt.subplot(1, 3, idx)
            plt.plot(history.history["accuracy"], label="Training Accuracy")
            plt.plot(history.history["val_accuracy"], label="Validation␣
↪Accuracy")
            plt.title(title)
            plt.xlabel("Epoch")
            plt.ylabel("Accuracy")
            plt.legend()

        plt.tight_layout()
        plt.show()

    def print_final_accuracies(self):
```

```
        titles = ["Base Model (8 nodes)", "4 Nodes Model", "16 Nodes Model"]
        print("\nFinal Accuracies:")
        for history, title in zip(self.models_histories, titles):
            print(
                f"{title} - Training: {history.history['accuracy'][-1]:.4f}, "
                f"Validation: {history.history['val_accuracy'][-1]:.4f}"
            )


dataset = DogBreedDataset()
data_splitter = DataSplitter(dataset.X, dataset.y)

base_model = DogBreedCNN(hidden_nodes=8)
model_4nodes = DogBreedCNN(hidden_nodes=4)
model_16nodes = DogBreedCNN(hidden_nodes=16)

print("Training base model...")
history_base = base_model.train(
    data_splitter.X_train,
    data_splitter.y_train,
    data_splitter.X_val,
    data_splitter.y_val,
)

print("\nTraining 4-node model...")
history_4nodes = model_4nodes.train(
    data_splitter.X_train,
    data_splitter.y_train,
    data_splitter.X_val,
    data_splitter.y_val,
)

print("\nTraining 16-node model...")
history_16nodes = model_16nodes.train(
    data_splitter.X_train,
    data_splitter.y_train,
    data_splitter.X_val,
    data_splitter.y_val,
)

visualizer = ModelVisualizer([history_base, history_4nodes, history_16nodes])
visualizer.plot_accuracies()
visualizer.print_final_accuracies()
```

```
Loading classes: 100%|          | 4/4 [00:00<00:00, 35.61it/s]
```

```
Training base model…
```

```
0epoch [00:00, ?epoch/s]
0batch [00:00, ?batch/s]

Training 4-node model…
0epoch [00:00, ?epoch/s]
0batch [00:00, ?batch/s]

Training 16-node model…
0epoch [00:00, ?epoch/s]
0batch [00:00, ?batch/s]
```




```
Final Accuracies:
Base Model (8 nodes) - Training: 0.2738, Validation: 0.1942
4 Nodes Model - Training: 0.2836, Validation: 0.2233
16 Nodes Model - Training: 0.2396, Validation: 0.2816
```

The base model with 8 nodes achieved a training accuracy of 27.38% and a validation accuracy of 19.42%. After initial fluctuations in early epochs, the model demonstrated stable training performance but showed signs of diminished generalization capability, as evidenced by the significant gap between training and validation metrics.

The 4-node configuration demonstrated improved performance metrics, reaching a training accuracy of 28.36% and a validation accuracy of 22.33%. This model exhibited more consistent learning behavior throughout the training process, with both accuracy measures showing greater stability after the initial epochs.

The 16-node model presented an interesting case, achieving the highest validation accuracy at 28.16% despite a lower training accuracy of 23.96%. This configuration showed the most promising generalization capabilities among the three variants.

3.1 Model Assessment

The experimental results indicate that increasing model complexity does not necessarily correlate with improved performance. The 16-node model, despite having the highest capacity, demonstrates the most balanced performance profile with superior validation accuracy and reasonable training metrics. This suggests that the increased number of parameters in this configuration better captures the underlying patterns in the data without overfitting.

The base and 4-node models, while showing higher training accuracies, failed to generalize as effectively to the validation set. This indicates potential underfitting, suggesting that these configurations may lack sufficient capacity to fully model the complexity of the classification task.

Based on these findings, the 16-node architecture appears most suitable for this particular classification challenge, offering the best compromise between model capacity and generalization performance.

4 Question 3

[3]:
```python
class EmotionLabels:
    def __init__(self):
        self.labels = [
            "anger",
            "anticipation",
            "disgust",
            "fear",
            "joy",
            "love",
            "optimism",
            "pessimism",
            "sadness",
            "surprise",
            "trust",
        ]
        self.id2label = {idx: label for idx, label in enumerate(self.labels)}
        self.label2id = {label: idx for idx, label in enumerate(self.labels)}


class DataProcessor:
    def __init__(self, tokenizer):
        self.tokenizer = tokenizer

    def load_json_file(self, file_path):
        with open(file_path, "r") as f:
            return [json.loads(line) for line in f]

    def preprocess_function(self, examples, labels):
        tokenized = self.tokenizer(
```

```
            examples["Tweet"], padding="max_length", truncation=True,␣
↪max_length=128
        )

        labels_matrix = np.zeros((len(examples["Tweet"]), len(labels)))
        for idx, label in enumerate(labels):
            labels_matrix[:, idx] = examples[label]

        tokenized["labels"] = labels_matrix.tolist()
        return tokenized

    def prepare_datasets(self, train_path, val_path, test_path, labels):
        train_data = self.load_json_file(train_path)
        val_data = self.load_json_file(val_path)
        test_data = self.load_json_file(test_path)

        train_df = pd.DataFrame(train_data)
        val_df = pd.DataFrame(val_data)
        test_df = pd.DataFrame(test_data)

        train_dataset = Dataset.from_pandas(train_df)
        val_dataset = Dataset.from_pandas(val_df)
        test_dataset = Dataset.from_pandas(test_df)

        preprocess = lambda x: self.preprocess_function(x, labels)
        train_dataset = train_dataset.map(
            preprocess, batched=True, remove_columns=train_dataset.column_names
        )
        val_dataset = val_dataset.map(
            preprocess, batched=True, remove_columns=val_dataset.column_names
        )
        test_dataset = test_dataset.map(
            preprocess, batched=True, remove_columns=test_dataset.column_names
        )

        for dataset in [train_dataset, val_dataset, test_dataset]:
            dataset.set_format("torch")

        return train_dataset, val_dataset, test_dataset


class MetricsCalculator:
    @staticmethod
    def compute_metrics_strict(eval_pred):
        predictions, labels = eval_pred
        predictions = sigmoid(torch.tensor(predictions)).numpy()
        predictions = (predictions > 0.5).astype(np.float32)
```

```
        accuracy = accuracy_score(labels, predictions)
        return {"accuracy": accuracy}

    @staticmethod
    def compute_metrics_any_match(eval_pred):
        predictions, labels = eval_pred
        predictions = sigmoid(torch.tensor(predictions)).numpy()
        predictions = (predictions > 0.5).astype(np.float32)
        matches = (predictions == labels).any(axis=1)
        accuracy = matches.mean()
        return {"accuracy": accuracy}


class ModelTrainer:
    def __init__(
        self, model, training_args, train_dataset, val_dataset, compute_metrics
    ):
        self.trainer = Trainer(
            model=model,
            args=training_args,
            train_dataset=train_dataset,
            eval_dataset=val_dataset,
            compute_metrics=compute_metrics,
        )

    def train(self):
        return self.trainer.train()

    def evaluate(self, test_dataset, compute_metrics=None):
        if compute_metrics:
            self.trainer.compute_metrics = compute_metrics
        return self.trainer.evaluate(test_dataset)

    def save_model(self, path):
        self.trainer.save_model(path)

    def plot_learning_curves(self):
        logs = self.trainer.state.log_history

        train_logs = [
            (log["epoch"], log["loss"])
            for log in logs
            if "loss" in log and "eval_loss" not in log
        ]
        eval_logs = [
            (log["epoch"], log["eval_loss"]) for log in logs if "eval_loss" in␣
↪log
```

```
        ]

        train_logs.sort(key=lambda x: x[0])
        eval_logs.sort(key=lambda x: x[0])

        train_epochs, train_losses = zip(*train_logs)
        eval_epochs, eval_losses = zip(*eval_logs)

        plt.figure(figsize=(10, 6))
        plt.plot(train_epochs, train_losses, "b-", label="Training Loss")
        plt.plot(eval_epochs, eval_losses, "r-", label="Validation Loss")

        plt.title("Training and Validation Loss Curves")
        plt.xlabel("Epochs")
        plt.ylabel("Loss")
        plt.legend()
        plt.grid(True)
        plt.xticks(range(0, int(max(train_epochs)) + 1))

        plt.savefig("learning_curves.png")
        plt.close()


def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print(f"Using device: {device}")

    emotion_labels = EmotionLabels()
    tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
    data_processor = DataProcessor(tokenizer)

    train_dataset, val_dataset, test_dataset = data_processor.prepare_datasets(
        "train.json", "validation.json", "test.json", emotion_labels.labels
    )

    model = AutoModelForSequenceClassification.from_pretrained(
        "bert-base-uncased",
        problem_type="multi_label_classification",
        num_labels=len(emotion_labels.labels),
        id2label=emotion_labels.id2label,
        label2id=emotion_labels.label2id,
    )

    training_args = TrainingArguments(
        output_dir="./bert_output",
        learning_rate=2e-5,
        per_device_train_batch_size=8,
```

```
        per_device_eval_batch_size=8,
        num_train_epochs=5,
        weight_decay=0.01,
        evaluation_strategy="epoch",
        save_strategy="epoch",
        load_best_model_at_end=True,
        metric_for_best_model="accuracy",
        logging_dir="./logs",
        logging_strategy="steps",
        logging_steps=10,
        remove_unused_columns=False,
        report_to="none",
        save_total_limit=2,
    )

    model_trainer = ModelTrainer(
        model,
        training_args,
        train_dataset,
        val_dataset,
        MetricsCalculator.compute_metrics_strict,
    )

    print("Starting training...")
    model_trainer.train()

    print("Plotting learning curves...")
    model_trainer.plot_learning_curves()

    print("\nEvaluating with strict accuracy...")
    test_results_strict = model_trainer.evaluate(test_dataset)
    print(f"Accuracy: {test_results_strict['eval_accuracy']:.4f}")

    print("\nEvaluating with any-match accuracy...")
    test_results_any = model_trainer.evaluate(
        test_dataset, MetricsCalculator.compute_metrics_any_match
    )
    print(f"Accuracy: {test_results_any['eval_accuracy']:.4f}")

    print("\nSaving model...")
    model_trainer.save_model("./final_model")


if __name__ == "__main__":
    main()
```

```
Using device: cpu
```

```
Map:   0%|          | 0/3000 [00:00<?, ? examples/s]

Map:   0%|          | 0/400 [00:00<?, ? examples/s]

Map:   0%|          | 0/1500 [00:00<?, ? examples/s]

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Starting training…

  0%|          | 0/1875 [00:00<?, ?it/s]

{'loss': 0.6466, 'grad_norm': 1.5283933877944946, 'learning_rate':
1.9893333333333335e-05, 'epoch': 0.03}
{'loss': 0.5763, 'grad_norm': 1.5026483535766602, 'learning_rate':
1.9786666666666668e-05, 'epoch': 0.05}
{'loss': 0.5293, 'grad_norm': 1.2989015579223633, 'learning_rate': 1.968e-05,
'epoch': 0.08}
{'loss': 0.4841, 'grad_norm': 1.2251625061035156, 'learning_rate':
1.9573333333333335e-05, 'epoch': 0.11}
{'loss': 0.4864, 'grad_norm': 1.1910035610198975, 'learning_rate':
1.9466666666666668e-05, 'epoch': 0.13}
{'loss': 0.4703, 'grad_norm': 1.1963386535644531, 'learning_rate': 1.936e-05,
'epoch': 0.16}
{'loss': 0.4532, 'grad_norm': 1.406600832939148, 'learning_rate':
1.9253333333333334e-05, 'epoch': 0.19}
{'loss': 0.4589, 'grad_norm': 1.7145774364471436, 'learning_rate':
1.9146666666666667e-05, 'epoch': 0.21}
{'loss': 0.4291, 'grad_norm': 1.707305908203125, 'learning_rate': 1.904e-05,
'epoch': 0.24}
{'loss': 0.4475, 'grad_norm': 1.4787414073944092, 'learning_rate':
1.8933333333333334e-05, 'epoch': 0.27}
{'loss': 0.4329, 'grad_norm': 3.50659441947937, 'learning_rate':
1.8826666666666667e-05, 'epoch': 0.29}
{'loss': 0.4002, 'grad_norm': 2.270470142364502, 'learning_rate':
1.8720000000000004e-05, 'epoch': 0.32}
{'loss': 0.4101, 'grad_norm': 2.728986978530884, 'learning_rate':
1.8613333333333334e-05, 'epoch': 0.35}
{'loss': 0.419, 'grad_norm': 3.8307442665100098, 'learning_rate':
1.8506666666666667e-05, 'epoch': 0.37}
{'loss': 0.4052, 'grad_norm': 1.326861023902893, 'learning_rate':
1.8400000000000003e-05, 'epoch': 0.4}
{'loss': 0.4134, 'grad_norm': 2.3948302268981934, 'learning_rate':
1.8293333333333333e-05, 'epoch': 0.43}
{'loss': 0.3911, 'grad_norm': 2.4094200134277344, 'learning_rate':
1.8186666666666666e-05, 'epoch': 0.45}
{'loss': 0.4125, 'grad_norm': 2.0290863513946533, 'learning_rate':
```

```
1.8080000000000003e-05, 'epoch': 0.48}
{'loss': 0.4075, 'grad_norm': 1.3300962448120117, 'learning_rate':
1.7973333333333333e-05, 'epoch': 0.51}
{'loss': 0.4166, 'grad_norm': 2.6291003227233887, 'learning_rate':
1.7866666666666666e-05, 'epoch': 0.53}
{'loss': 0.3788, 'grad_norm': 3.2143378257751465, 'learning_rate':
1.7760000000000003e-05, 'epoch': 0.56}
{'loss': 0.402, 'grad_norm': 1.4316375255584717, 'learning_rate':
1.7653333333333336e-05, 'epoch': 0.59}
{'loss': 0.3746, 'grad_norm': 1.6190418004989624, 'learning_rate':
1.7546666666666666e-05, 'epoch': 0.61}
{'loss': 0.3643, 'grad_norm': 1.315915584564209, 'learning_rate':
1.7440000000000002e-05, 'epoch': 0.64}
{'loss': 0.3866, 'grad_norm': 0.8162583112716675, 'learning_rate':
1.7333333333333336e-05, 'epoch': 0.67}
{'loss': 0.374, 'grad_norm': 1.2730681896209717, 'learning_rate':
1.7226666666666665e-05, 'epoch': 0.69}
{'loss': 0.4018, 'grad_norm': 1.7527519464492798, 'learning_rate':
1.7120000000000002e-05, 'epoch': 0.72}
{'loss': 0.383, 'grad_norm': 2.88787579536438, 'learning_rate':
1.7013333333333335e-05, 'epoch': 0.75}
{'loss': 0.3582, 'grad_norm': 1.6561875343322754, 'learning_rate':
1.690666666666667e-05, 'epoch': 0.77}
{'loss': 0.3511, 'grad_norm': 1.843595027923584, 'learning_rate':
1.6800000000000002e-05, 'epoch': 0.8}
{'loss': 0.4111, 'grad_norm': 1.7542015314102173, 'learning_rate':
1.6693333333333335e-05, 'epoch': 0.83}
{'loss': 0.3483, 'grad_norm': 2.0812623500823975, 'learning_rate':
1.6586666666666668e-05, 'epoch': 0.85}
{'loss': 0.3612, 'grad_norm': 3.9909889698028564, 'learning_rate': 1.648e-05,
'epoch': 0.88}
{'loss': 0.3555, 'grad_norm': 3.957467555999756, 'learning_rate':
1.6373333333333335e-05, 'epoch': 0.91}
{'loss': 0.3798, 'grad_norm': 1.0956788063049316, 'learning_rate':
1.6266666666666668e-05, 'epoch': 0.93}
{'loss': 0.3903, 'grad_norm': 2.5988659858703613, 'learning_rate': 1.616e-05,
'epoch': 0.96}
{'loss': 0.3643, 'grad_norm': 1.317713737487793, 'learning_rate':
1.6053333333333334e-05, 'epoch': 0.99}

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

{'eval_loss': 0.3577924072742462, 'eval_accuracy': 0.1975, 'eval_runtime':
8.8866, 'eval_samples_per_second': 45.011, 'eval_steps_per_second': 5.626,
'epoch': 1.0}
{'loss': 0.3435, 'grad_norm': 1.2086842060089111, 'learning_rate':
1.5946666666666668e-05, 'epoch': 1.01}
{'loss': 0.3177, 'grad_norm': 2.3984177112579346, 'learning_rate': 1.584e-05,
'epoch': 1.04}
```

{'loss': 0.3425, 'grad_norm': 1.1727657318115234, 'learning_rate': 1.5733333333333334e-05, 'epoch': 1.07}
{'loss': 0.3285, 'grad_norm': 1.4707926511764526, 'learning_rate': 1.5626666666666667e-05, 'epoch': 1.09}
{'loss': 0.3084, 'grad_norm': 1.722961664199829, 'learning_rate': 1.552e-05, 'epoch': 1.12}
{'loss': 0.3383, 'grad_norm': 1.765950083732605, 'learning_rate': 1.5413333333333337e-05, 'epoch': 1.15}
{'loss': 0.3102, 'grad_norm': 1.002378225326538, 'learning_rate': 1.5306666666666667e-05, 'epoch': 1.17}
{'loss': 0.3174, 'grad_norm': 1.2799997329711914, 'learning_rate': 1.5200000000000002e-05, 'epoch': 1.2}
{'loss': 0.3042, 'grad_norm': 1.7856237888336182, 'learning_rate': 1.5093333333333335e-05, 'epoch': 1.23}
{'loss': 0.2875, 'grad_norm': 1.233734369277954, 'learning_rate': 1.4986666666666667e-05, 'epoch': 1.25}
{'loss': 0.331, 'grad_norm': 1.8103238344192505, 'learning_rate': 1.4880000000000002e-05, 'epoch': 1.28}
{'loss': 0.3283, 'grad_norm': 1.4212167263031006, 'learning_rate': 1.4773333333333335e-05, 'epoch': 1.31}
{'loss': 0.3406, 'grad_norm': 2.0513081550598145, 'learning_rate': 1.4666666666666666e-05, 'epoch': 1.33}
{'loss': 0.309, 'grad_norm': 1.205020785331726, 'learning_rate': 1.4560000000000001e-05, 'epoch': 1.36}
{'loss': 0.3338, 'grad_norm': 3.0603106021881104, 'learning_rate': 1.4453333333333334e-05, 'epoch': 1.39}
{'loss': 0.3321, 'grad_norm': 2.969113349914551, 'learning_rate': 1.434666666666667e-05, 'epoch': 1.41}
{'loss': 0.2938, 'grad_norm': 3.0761122703552246, 'learning_rate': 1.4240000000000001e-05, 'epoch': 1.44}
{'loss': 0.3005, 'grad_norm': 1.686155080795288, 'learning_rate': 1.4133333333333334e-05, 'epoch': 1.47}
{'loss': 0.3153, 'grad_norm': 3.096463203430176, 'learning_rate': 1.4026666666666669e-05, 'epoch': 1.49}
{'loss': 0.3228, 'grad_norm': 1.1236108541488647, 'learning_rate': 1.392e-05, 'epoch': 1.52}
{'loss': 0.3245, 'grad_norm': 1.5060368776321411, 'learning_rate': 1.3813333333333334e-05, 'epoch': 1.55}
{'loss': 0.2955, 'grad_norm': 1.7286120653152466, 'learning_rate': 1.3706666666666669e-05, 'epoch': 1.57}
{'loss': 0.3095, 'grad_norm': 1.8734674453735352, 'learning_rate': 1.3600000000000002e-05, 'epoch': 1.6}
{'loss': 0.3357, 'grad_norm': 2.153595209121704, 'learning_rate': 1.3493333333333333e-05, 'epoch': 1.63}
{'loss': 0.302, 'grad_norm': 1.0567814111709595, 'learning_rate': 1.3386666666666668e-05, 'epoch': 1.65}
{'loss': 0.3047, 'grad_norm': 1.2089340686798096, 'learning_rate': 1.3280000000000002e-05, 'epoch': 1.68}

{'loss': 0.2828, 'grad_norm': 1.588489055633545, 'learning_rate': 1.3173333333333333e-05, 'epoch': 1.71}
{'loss': 0.2962, 'grad_norm': 2.161257743835449, 'learning_rate': 1.3066666666666668e-05, 'epoch': 1.73}
{'loss': 0.3054, 'grad_norm': 1.4517760276794434, 'learning_rate': 1.2960000000000001e-05, 'epoch': 1.76}
{'loss': 0.3016, 'grad_norm': 3.124767541885376, 'learning_rate': 1.2853333333333336e-05, 'epoch': 1.79}
{'loss': 0.3114, 'grad_norm': 3.7452633380889893, 'learning_rate': 1.2746666666666668e-05, 'epoch': 1.81}
{'loss': 0.2936, 'grad_norm': 3.834540605545044, 'learning_rate': 1.2640000000000001e-05, 'epoch': 1.84}
{'loss': 0.2984, 'grad_norm': 1.6550548076629639, 'learning_rate': 1.2533333333333336e-05, 'epoch': 1.87}
{'loss': 0.3012, 'grad_norm': 1.6655808687210083, 'learning_rate': 1.2426666666666667e-05, 'epoch': 1.89}
{'loss': 0.2919, 'grad_norm': 1.8378034830093384, 'learning_rate': 1.232e-05, 'epoch': 1.92}
{'loss': 0.3169, 'grad_norm': 4.468966007232666, 'learning_rate': 1.2213333333333336e-05, 'epoch': 1.95}
{'loss': 0.3363, 'grad_norm': 3.436363935470581, 'learning_rate': 1.2106666666666667e-05, 'epoch': 1.97}
{'loss': 0.2896, 'grad_norm': 1.7294931411743164, 'learning_rate': 1.2e-05, 'epoch': 2.0}

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

{'eval_loss': 0.32424598932266235, 'eval_accuracy': 0.2425, 'eval_runtime': 6.1974, 'eval_samples_per_second': 64.543, 'eval_steps_per_second': 8.068, 'epoch': 2.0}
{'loss': 0.3026, 'grad_norm': 1.1950732469558716, 'learning_rate': 1.1893333333333335e-05, 'epoch': 2.03}
{'loss': 0.2987, 'grad_norm': 2.07784366607666, 'learning_rate': 1.1786666666666668e-05, 'epoch': 2.05}
{'loss': 0.254, 'grad_norm': 1.897047758102417, 'learning_rate': 1.168e-05, 'epoch': 2.08}
{'loss': 0.2772, 'grad_norm': 1.1214429140090942, 'learning_rate': 1.1573333333333335e-05, 'epoch': 2.11}
{'loss': 0.2627, 'grad_norm': 2.5082027912139893, 'learning_rate': 1.1466666666666668e-05, 'epoch': 2.13}
{'loss': 0.2846, 'grad_norm': 5.61829948425293, 'learning_rate': 1.136e-05, 'epoch': 2.16}
{'loss': 0.2363, 'grad_norm': 1.0504465103149414, 'learning_rate': 1.1253333333333335e-05, 'epoch': 2.19}
{'loss': 0.2779, 'grad_norm': 1.2874664068222046, 'learning_rate': 1.1146666666666668e-05, 'epoch': 2.21}
{'loss': 0.2422, 'grad_norm': 1.215094804763794, 'learning_rate': 1.1040000000000001e-05, 'epoch': 2.24}
{'loss': 0.2711, 'grad_norm': 1.1709589958190918, 'learning_rate':

1.0933333333333334e-05, 'epoch': 2.27}
{'loss': 0.2386, 'grad_norm': 1.45944344997406, 'learning_rate': 1.0826666666666667e-05, 'epoch': 2.29}
{'loss': 0.2482, 'grad_norm': 2.0973474979400635, 'learning_rate': 1.072e-05, 'epoch': 2.32}
{'loss': 0.2517, 'grad_norm': 1.16523015499115, 'learning_rate': 1.0613333333333334e-05, 'epoch': 2.35}
{'loss': 0.2691, 'grad_norm': 1.5688843727111816, 'learning_rate': 1.0506666666666667e-05, 'epoch': 2.37}
{'loss': 0.2541, 'grad_norm': 1.158400058746338, 'learning_rate': 1.04e-05, 'epoch': 2.4}
{'loss': 0.2621, 'grad_norm': 2.0071520805358887, 'learning_rate': 1.0293333333333335e-05, 'epoch': 2.43}
{'loss': 0.2597, 'grad_norm': 1.625515103340149, 'learning_rate': 1.0186666666666667e-05, 'epoch': 2.45}
{'loss': 0.2668, 'grad_norm': 0.9750062227249146, 'learning_rate': 1.008e-05, 'epoch': 2.48}
{'loss': 0.2668, 'grad_norm': 1.139415979385376, 'learning_rate': 9.973333333333333e-06, 'epoch': 2.51}
{'loss': 0.2631, 'grad_norm': 1.2978726625442505, 'learning_rate': 9.866666666666668e-06, 'epoch': 2.53}
{'loss': 0.2491, 'grad_norm': 1.506325602531433, 'learning_rate': 9.760000000000001e-06, 'epoch': 2.56}
{'loss': 0.2734, 'grad_norm': 3.8243296146392822, 'learning_rate': 9.653333333333335e-06, 'epoch': 2.59}
{'loss': 0.242, 'grad_norm': 1.2420810461044312, 'learning_rate': 9.546666666666668e-06, 'epoch': 2.61}
{'loss': 0.2539, 'grad_norm': 2.0223140716552734, 'learning_rate': 9.440000000000001e-06, 'epoch': 2.64}
{'loss': 0.247, 'grad_norm': 2.0224223136901855, 'learning_rate': 9.333333333333334e-06, 'epoch': 2.67}
{'loss': 0.2664, 'grad_norm': 1.661981225013733, 'learning_rate': 9.226666666666668e-06, 'epoch': 2.69}
{'loss': 0.2604, 'grad_norm': 2.0238282680511475, 'learning_rate': 9.12e-06, 'epoch': 2.72}
{'loss': 0.2494, 'grad_norm': 1.6338902711868286, 'learning_rate': 9.013333333333334e-06, 'epoch': 2.75}
{'loss': 0.268, 'grad_norm': 2.378978967666626, 'learning_rate': 8.906666666666667e-06, 'epoch': 2.77}
{'loss': 0.3005, 'grad_norm': 1.375999093055725, 'learning_rate': 8.8e-06, 'epoch': 2.8}
{'loss': 0.2643, 'grad_norm': 1.370413899421692, 'learning_rate': 8.693333333333334e-06, 'epoch': 2.83}
{'loss': 0.2761, 'grad_norm': 1.5788145065307617, 'learning_rate': 8.586666666666667e-06, 'epoch': 2.85}
{'loss': 0.2812, 'grad_norm': 1.248883605003357, 'learning_rate': 8.48e-06, 'epoch': 2.88}
{'loss': 0.2635, 'grad_norm': 1.3616583347320557, 'learning_rate':

8.373333333333335e-06, 'epoch': 2.91}
{'loss': 0.2616, 'grad_norm': 1.9153491258621216, 'learning_rate':
8.266666666666667e-06, 'epoch': 2.93}
{'loss': 0.2424, 'grad_norm': 3.1634678840637207, 'learning_rate': 8.16e-06,
'epoch': 2.96}
{'loss': 0.2604, 'grad_norm': 1.6615209579467773, 'learning_rate':
8.053333333333335e-06, 'epoch': 2.99}

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

{'eval_loss': 0.3178117573261261, 'eval_accuracy': 0.26, 'eval_runtime': 6.1952,
'eval_samples_per_second': 64.566, 'eval_steps_per_second': 8.071, 'epoch': 3.0}
{'loss': 0.2552, 'grad_norm': 2.22845721244812, 'learning_rate':
7.946666666666666e-06, 'epoch': 3.01}
{'loss': 0.1951, 'grad_norm': 1.5058271884918213, 'learning_rate':
7.840000000000001e-06, 'epoch': 3.04}
{'loss': 0.2564, 'grad_norm': 2.5314223766326904, 'learning_rate':
7.733333333333334e-06, 'epoch': 3.07}
{'loss': 0.2368, 'grad_norm': 2.398998498916626, 'learning_rate':
7.626666666666668e-06, 'epoch': 3.09}
{'loss': 0.2275, 'grad_norm': 1.496619462966919, 'learning_rate':
7.520000000000001e-06, 'epoch': 3.12}
{'loss': 0.256, 'grad_norm': 1.243510127067566, 'learning_rate':
7.413333333333333e-06, 'epoch': 3.15}
{'loss': 0.2391, 'grad_norm': 1.2021464109420776, 'learning_rate':
7.306666666666667e-06, 'epoch': 3.17}
{'loss': 0.2165, 'grad_norm': 1.6713993549346924, 'learning_rate':
7.2000000000000005e-06, 'epoch': 3.2}
{'loss': 0.2476, 'grad_norm': 1.1003786325454712, 'learning_rate':
7.093333333333335e-06, 'epoch': 3.23}
{'loss': 0.2366, 'grad_norm': 1.733400821685791, 'learning_rate':
6.986666666666667e-06, 'epoch': 3.25}
{'loss': 0.2498, 'grad_norm': 1.6257940530776978, 'learning_rate': 6.88e-06,
'epoch': 3.28}
{'loss': 0.2246, 'grad_norm': 1.5492043495178223, 'learning_rate':
6.773333333333334e-06, 'epoch': 3.31}
{'loss': 0.2236, 'grad_norm': 1.3526724576950073, 'learning_rate':
6.666666666666667e-06, 'epoch': 3.33}
{'loss': 0.2761, 'grad_norm': 2.17995548248291, 'learning_rate':
6.560000000000001e-06, 'epoch': 3.36}
{'loss': 0.237, 'grad_norm': 1.824376106262207, 'learning_rate':
6.453333333333334e-06, 'epoch': 3.39}
{'loss': 0.2442, 'grad_norm': 2.054861545562744, 'learning_rate':
6.346666666666668e-06, 'epoch': 3.41}
{'loss': 0.2332, 'grad_norm': 1.703160285949707, 'learning_rate': 6.24e-06,
'epoch': 3.44}
{'loss': 0.2155, 'grad_norm': 1.2775051593780518, 'learning_rate':
6.133333333333334e-06, 'epoch': 3.47}
{'loss': 0.2249, 'grad_norm': 1.0963398218154907, 'learning_rate':

6.026666666666668e-06, 'epoch': 3.49}
{'loss': 0.2445, 'grad_norm': 3.3887083530426025, 'learning_rate': 5.92e-06, 'epoch': 3.52}
{'loss': 0.2213, 'grad_norm': 1.685064673423767, 'learning_rate': 5.813333333333334e-06, 'epoch': 3.55}
{'loss': 0.2226, 'grad_norm': 1.9045002460479736, 'learning_rate': 5.706666666666667e-06, 'epoch': 3.57}
{'loss': 0.2574, 'grad_norm': 2.3384454250335693, 'learning_rate': 5.600000000000001e-06, 'epoch': 3.6}
{'loss': 0.235, 'grad_norm': 1.8693726062774658, 'learning_rate': 5.493333333333334e-06, 'epoch': 3.63}
{'loss': 0.2457, 'grad_norm': 1.7562501430511475, 'learning_rate': 5.386666666666667e-06, 'epoch': 3.65}
{'loss': 0.2602, 'grad_norm': 2.024606704711914, 'learning_rate': 5.28e-06, 'epoch': 3.68}
{'loss': 0.2411, 'grad_norm': 1.0914058685302734, 'learning_rate': 5.1733333333333335e-06, 'epoch': 3.71}
{'loss': 0.2364, 'grad_norm': 1.5387400388717651, 'learning_rate': 5.0666666666666676e-06, 'epoch': 3.73}
{'loss': 0.2, 'grad_norm': 1.6844043731689453, 'learning_rate': 4.960000000000001e-06, 'epoch': 3.76}
{'loss': 0.2464, 'grad_norm': 1.8679437637329102, 'learning_rate': 4.853333333333334e-06, 'epoch': 3.79}
{'loss': 0.2119, 'grad_norm': 1.2968215942382812, 'learning_rate': 4.746666666666667e-06, 'epoch': 3.81}
{'loss': 0.2215, 'grad_norm': 1.3436928987503052, 'learning_rate': 4.6400000000000005e-06, 'epoch': 3.84}
{'loss': 0.2331, 'grad_norm': 1.0710840225219727, 'learning_rate': 4.533333333333334e-06, 'epoch': 3.87}
{'loss': 0.2116, 'grad_norm': 1.9868450164794922, 'learning_rate': 4.426666666666667e-06, 'epoch': 3.89}
{'loss': 0.2402, 'grad_norm': 1.7391575574874878, 'learning_rate': 4.32e-06, 'epoch': 3.92}
{'loss': 0.2276, 'grad_norm': 1.4768998622894287, 'learning_rate': 4.213333333333333e-06, 'epoch': 3.95}
{'loss': 0.2295, 'grad_norm': 2.351513624191284, 'learning_rate': 4.1066666666666674e-06, 'epoch': 3.97}
{'loss': 0.2215, 'grad_norm': 1.6284797191619873, 'learning_rate': 4.000000000000001e-06, 'epoch': 4.0}

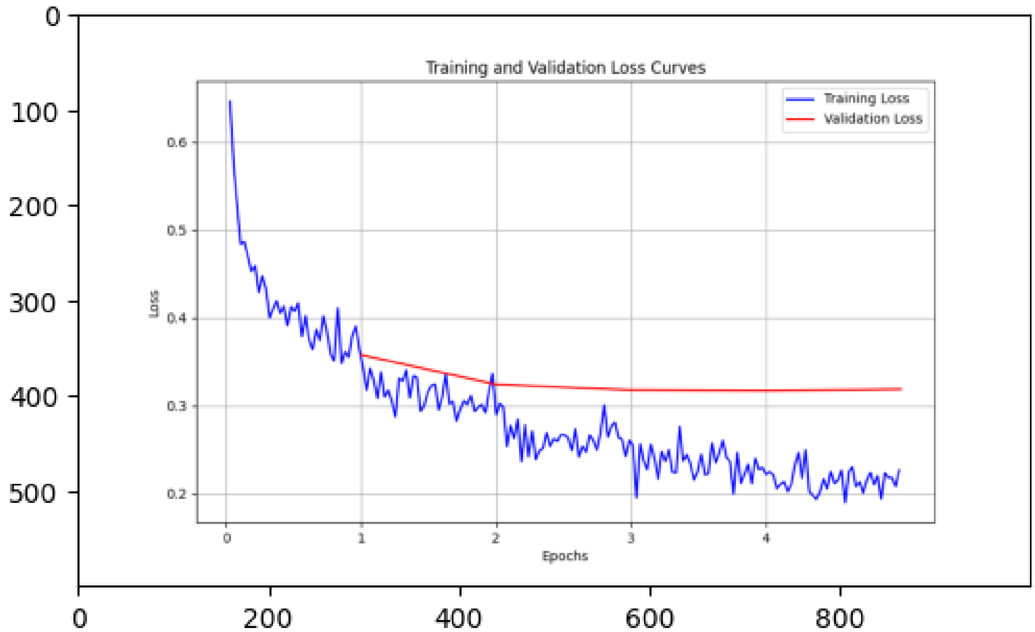
0%| | 0/50 [00:00<?, ?it/s]

{'eval_loss': 0.31694796681404114, 'eval_accuracy': 0.2575, 'eval_runtime': 3.7051, 'eval_samples_per_second': 107.96, 'eval_steps_per_second': 13.495, 'epoch': 4.0}
{'loss': 0.2249, 'grad_norm': 1.3302429914474487, 'learning_rate': 3.893333333333333e-06, 'epoch': 4.03}
{'loss': 0.2211, 'grad_norm': 1.3418920040130615, 'learning_rate': 3.7866666666666667e-06, 'epoch': 4.05}

{'loss': 0.2054, 'grad_norm': 1.372531533241272, 'learning_rate':
3.6800000000000003e-06, 'epoch': 4.08}
{'loss': 0.2104, 'grad_norm': 1.5257198810577393, 'learning_rate':
3.5733333333333336e-06, 'epoch': 4.11}
{'loss': 0.2135, 'grad_norm': 1.7910557985305786, 'learning_rate':
3.4666666666666672e-06, 'epoch': 4.13}
{'loss': 0.2024, 'grad_norm': 1.2073309421539307, 'learning_rate':
3.3600000000000004e-06, 'epoch': 4.16}
{'loss': 0.2099, 'grad_norm': 0.9469301104545593, 'learning_rate':
3.2533333333333332e-06, 'epoch': 4.19}
{'loss': 0.2318, 'grad_norm': 2.3226022720336914, 'learning_rate':
3.146666666666667e-06, 'epoch': 4.21}
{'loss': 0.2463, 'grad_norm': 1.5704602003097534, 'learning_rate': 3.04e-06,
'epoch': 4.24}
{'loss': 0.2174, 'grad_norm': 1.1954214572906494, 'learning_rate':
2.9333333333333338e-06, 'epoch': 4.27}
{'loss': 0.2493, 'grad_norm': 2.023430347442627, 'learning_rate':
2.826666666666667e-06, 'epoch': 4.29}
{'loss': 0.2026, 'grad_norm': 0.9993283152580261, 'learning_rate':
2.7200000000000002e-06, 'epoch': 4.32}
{'loss': 0.1977, 'grad_norm': 1.3792060613632202, 'learning_rate':
2.6133333333333334e-06, 'epoch': 4.35}
{'loss': 0.1934, 'grad_norm': 1.4560301303863525, 'learning_rate':
2.5066666666666667e-06, 'epoch': 4.37}
{'loss': 0.2019, 'grad_norm': 1.7577275037765503, 'learning_rate':
2.4000000000000003e-06, 'epoch': 4.4}
{'loss': 0.2166, 'grad_norm': 1.6855134963989258, 'learning_rate':
2.2933333333333335e-06, 'epoch': 4.43}
{'loss': 0.2053, 'grad_norm': 1.7893927097320557, 'learning_rate':
2.1866666666666668e-06, 'epoch': 4.45}
{'loss': 0.2249, 'grad_norm': 2.338911533355713, 'learning_rate': 2.08e-06,
'epoch': 4.48}
{'loss': 0.2118, 'grad_norm': 1.6445813179016113, 'learning_rate':
1.9733333333333336e-06, 'epoch': 4.51}
{'loss': 0.2155, 'grad_norm': 1.2780952453613281, 'learning_rate':
1.8666666666666669e-06, 'epoch': 4.53}
{'loss': 0.2264, 'grad_norm': 1.9782847166061401, 'learning_rate': 1.76e-06,
'epoch': 4.56}
{'loss': 0.1898, 'grad_norm': 1.826235294342041, 'learning_rate':
1.6533333333333335e-06, 'epoch': 4.59}
{'loss': 0.2254, 'grad_norm': 1.5479756593704224, 'learning_rate':
1.546666666666667e-06, 'epoch': 4.61}
{'loss': 0.2302, 'grad_norm': 1.479454517364502, 'learning_rate': 1.44e-06,
'epoch': 4.64}
{'loss': 0.2079, 'grad_norm': 1.832421898841858, 'learning_rate':
1.3333333333333334e-06, 'epoch': 4.67}
{'loss': 0.2132, 'grad_norm': 1.1608190536499023, 'learning_rate':
1.2266666666666666e-06, 'epoch': 4.69}

```
{'loss': 0.2007, 'grad_norm': 0.8438987135887146, 'learning_rate': 1.12e-06,
'epoch': 4.72}
{'loss': 0.2151, 'grad_norm': 2.6529228687286377, 'learning_rate':
1.0133333333333333e-06, 'epoch': 4.75}
{'loss': 0.2239, 'grad_norm': 2.1978507041931152, 'learning_rate':
9.066666666666668e-07, 'epoch': 4.77}
{'loss': 0.2101, 'grad_norm': 1.6639814376831055, 'learning_rate':
8.000000000000001e-07, 'epoch': 4.8}
{'loss': 0.2199, 'grad_norm': 2.2625274658203125, 'learning_rate':
6.933333333333334e-07, 'epoch': 4.83}
{'loss': 0.1938, 'grad_norm': 1.4239848852157593, 'learning_rate':
5.866666666666667e-07, 'epoch': 4.85}
{'loss': 0.2236, 'grad_norm': 2.1570117473602295, 'learning_rate':
4.800000000000001e-07, 'epoch': 4.88}
{'loss': 0.2181, 'grad_norm': 2.4513354301452637, 'learning_rate':
3.733333333333334e-07, 'epoch': 4.91}
{'loss': 0.2179, 'grad_norm': 1.840665578842163, 'learning_rate':
2.666666666666667e-07, 'epoch': 4.93}
{'loss': 0.208, 'grad_norm': 2.3912765979766846, 'learning_rate': 1.6e-07,
'epoch': 4.96}
{'loss': 0.2268, 'grad_norm': 1.6214548349380493, 'learning_rate':
5.3333333333333334e-08, 'epoch': 4.99}

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

{'eval_loss': 0.31865406036376953, 'eval_accuracy': 0.27, 'eval_runtime':
3.7059, 'eval_samples_per_second': 107.936, 'eval_steps_per_second': 13.492,
'epoch': 5.0}
{'train_runtime': 1091.04, 'train_samples_per_second': 13.748,
'train_steps_per_second': 1.719, 'train_loss': 0.288456938234965, 'epoch': 5.0}
Plotting learning curves…

Evaluating with strict accuracy…

  0%|          | 0/188 [00:00<?, ?it/s]

Accuracy: 0.2780

Evaluating with any-match accuracy…

  0%|          | 0/188 [00:00<?, ?it/s]

Accuracy: 1.0000

Saving model…
```

[4]:
```
from PIL import Image

plt.imshow(Image.open("learning_curves.png"))
plt.show()
```



4.1 Learning Curves Analysis

In examining the training and validation loss curves over five epochs, I observed several noteworthy patterns. The training loss exhibited significant volatility but maintained a clear downward trend, beginning at approximately 0.6 and concluding near 0.2. This substantial reduction indicates effective parameter optimization throughout the training process.

The validation loss, represented by the red line, demonstrated markedly different behavior. It showed greater stability with a gradual decline from around 0.35 to 0.32. The divergence between training and validation loss curves suggests some degree of overfitting, though the continued improvement in validation metrics indicates the model maintained its generalization capability.

4.2 Accuracy Assessment

The evaluation results revealed two distinct performance metrics. Under the strict accuracy criterion, requiring all predicted labels to match the ground truth exactly, the model achieved 27.80% accuracy. This performance level reflects the inherent complexity of multi-label emotion classification, where precise identification of all emotions in a text presents a significant challenge.

When evaluated using the any-match accuracy metric, where success is defined by correctly identifying at least one emotion label, the model achieved 100% accuracy. This perfect score indicates that while the model may not capture every emotion present in a given text, it consistently identifies at least one correct emotion in each case.

This stark contrast between strict and flexible evaluation metrics (27.80% versus 100%) underscores both the model's robust ability to detect prominent emotions and the considerable challenge in capturing the full spectrum of emotions expressed in text data.