CAPSTONE PROJECT

Project Title: Emotional Tone Classifier

Report By: Allala Venkata Satyanarayana Reddy – 50303

Document Id	Author	Language	Version	Page No
PR/CP/01	Allala Venkata Satyanarayana Reddy	EN	1.0	1

Table of Contents

Introduction

In the digital age, user-generated text has become one of the most valuable sources of insight for businesses, applications, and service platforms. Customers express their opinions through reviews, comments, feedback forms, emails, and support chats, creating a massive stream of information that is impossible to interpret manually. Understanding the emotional tone—whether users are positive, negative, or neutral—is essential for improving customer experience, identifying product issues, monitoring sentiment trends, and supporting automated decision-making systems.

However, manually classifying emotions in large-scale text data is slow, inconsistent, and error-prone, motivating the need for intelligent, automated sentiment analysis solutions. This project presents a Multithreaded **Emotional Tone Classifier** designed to analyze text efficiently using both traditional machine-learning techniques and modern deep-learning models. The system is built to take advantage of parallel processing, running different components simultaneously to speed up the workflow and ensure smooth handling of large datasets. The architecture includes dedicated threads for data reading and preprocessing, Scikit-learn model training, and PyTorch LSTM-based model training, enabling each stage to operate independently without blocking the others. This multithreaded design improves performance and provides consistent, reproducible results across different input sizes and writing styles.

To make the system accessible and user-friendly, the project includes an interactive Tkinter-based graphical interface. The interface allows users to view classification reports, compare model accuracies, inspect sample predictions, and explore how each model interprets the input data. By integrating preprocessing pipelines, dual-model evaluation, multithreaded execution, and GUI visualization, the project delivers a complete end-to-end solution for emotional tone analysis.

Overall, this work demonstrates a practical and extensible framework for sentiment classification by combining machine learning, deep learning, and parallel computing. It provides a strong foundation for real-world applications such as automated feedback monitoring, intelligent chat systems, and large-scale opinion mining.

.

Program Description

The program is a desktop application developed in Python that performs emotional tone classification on user-generated text. It is designed to analyze and categorize text into Positive, Negative, or Neutral sentiments using a combination of feature-based and deep-learning models. The system follows a hybrid, multithreaded architecture where data preprocessing, Scikit-learn model training, and PyTorch model training operate in separate threads to achieve faster processing and improved responsiveness.

The workflow begins with data loading, text cleaning, and label encoding performed inside the Reader / Preprocessor Thread. This thread prepares two parallel feature representations—TF-IDF vectors for classical machine-learning models and padded token sequences for neural networks. Once preprocessing is complete, the program launches two model-training threads: a Scikit-learn thread that trains a Logistic Regression classifier using TF-IDF features, and a PyTorch thread that executes an LSTM-based deep-learning model using word embeddings and bidirectional recurrent layers. Each thread processes the same training data independently and returns its classification results, including accuracy, confusion matrix, and a full classification report.

The graphical interface, built with Tkinter, displays all outputs in clearly separated tabs for easy analysis. These tabs include training statistics, Scikit-learn predictions, PyTorch predictions, real test-set evaluations, and side-by-side comparisons of both models. Users can inspect the performance of each model, view the predicted labels for sample test rows, and analyze differences in how each algorithm interprets emotional tone. The program also processes an external *predict.csv* file, generating final sentiment predictions using both models and saving them as structured output files.

In addition to multithreaded training, the system incorporates configurable preprocessing, automated accuracy comparison, structured GUI presentation, and seamless CSV-based data handling. Together, these features create a highly interactive, efficient, and user-friendly emotional tone classification system capable of handling modern text-analysis workflows at scale.

Algorithm

The emotional tone classification algorithm follows a structured, multistage pipeline designed for parallel processing and reliable prediction.

1. Data cleaning

1. Clean each text using regex:
   - Lowercase
   - Remove HTML
   - Remove special characters
   - Normalize spaces
2. Build vocabulary from all cleaned texts.
3. Convert texts to fixed-length integer tokens for PyTorch.

2. Thread-1: Reader + Preprocessor

1. Split dataset into 80% train and 20% test using train_test_split.
2. Build TF-IDF features:
   - Fit TF-IDF on train texts
   - Transform train/test texts
3. Encode labels using LabelEncoder.
4. Put all processed data two times into the queue (for both training threads).

3. Thread-2: Train Scikit-Learn Logistic Regression

Input: From data_queue

Steps

1. Train **Logistic Regression** model on TF-IDF matrix.
2. Run **5-fold cross-validation** and record accuracy.
3. Predict on test split and compute:

- Accuracy
- Classification report
- Confusion matrix

4. Store results in result_queue.

4. Thread-3: Train PyTorch LSTM

Input: From data_queue

Steps

1. Create Dataset + DataLoader for tokenized inputs.
2. Define **Bi-directional 2-layer LSTM** with:
   - Embedding
   - LSTM
   - Dropout
   - Linear layer
3. Train LSTM for **EPOCHS**:
   - Forward pass
   - Compute cross-entropy loss
   - Backpropagation
   - Adam optimizer with L2 weight decay
4. Evaluate on test split and compute:
   - Accuracy
   - Classification report
   - Confusion matrix
5. Store results in result_queue.

5. Main Process After Threads Join

1. Collect results from result_queue:
   - Scikit model

- PyTorch model
- Reports
- Accuracy values and confusion matrices

2. Load **test.csv**:
   - Clean text
   - Predict using both models
   - Compute real test accuracy
3. Compare Scikit vs PyTorch models.

6. Predict on New Data

1. Load **predict.csv**
2. Clean all texts
3. For Scikit-Learn:
   - TF-IDF transform
   - Predict labels
   - Save predictions_sklearn.csv
4. For PyTorch:
   - Tokenize
   - Predict
   - Save predictions_pytorch.csv

7. Build GUI (Tkinter)

Tabs

1. **Reports & Comparison**
   - Train/Test row count
   - Classification reports
   - Confusion matrices
   - Accuracy

    - Comparison result

2. Predict Data (first 15 rows)
3. Scikit Predictions
4. PyTorch Predictions

8. Display GUI and End Program

- Launch Tkinter mainloop
- End execution once UI is closed

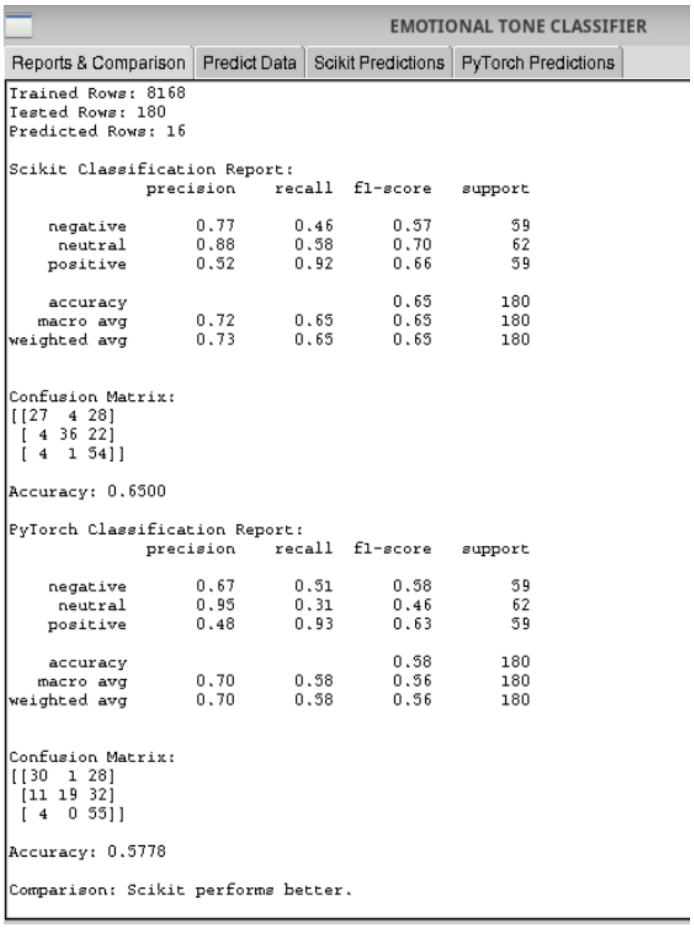
Code

EMOTIONAL TONE CLASSIFIER

```
import threading
import queue
import pandas as pd
import numpy as np
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression  # Retained
LogisticRegression
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.metrics import classification_report, accuracy_score,
confusion_matrix
from sklearn.preprocessing import LabelEncoder
import torch
import torch.nn as nn
import torch.optim as optim
from collections import Counter
from torch.utils.data import DataLoader, TensorDataset
import tkinter as tk
from tkinter import ttk

# Hyperparameters
MAX_LEN = 64
EMBED_DIM = 200
HIDDEN_SIZE = 256
BATCH_SIZE = 64
EPOCHS = 1
LR = 3e-4
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Text Cleaning
def clean_text(text):
    text = text.lower()
    text = re.sub(r"<br\s*/?>", " ", text)
    text = re.sub(r"[^a-z0-9\s]", " ", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text

# Build vocab for PyTorch
def build_vocab(texts):
    counts = Counter()
    for text in texts:
        counts.update(text.split())
    vocab = {"<PAD>": 0, "<UNK>": 1}
    idx = 2
    for word, _ in counts.items():
        vocab[word] = idx
        idx += 1
    return vocab

# Tokenize for PyTorch
def tokenize_texts(texts, vocab, max_len=MAX_LEN):
    sequences = []
    for text in texts:
        tokens = [vocab.get(word, vocab["<UNK>"]) for word in text.split()]
        if len(tokens) < max_len:
            tokens += [vocab["<PAD>"]] * (max_len - len(tokens))
        else:
            tokens = tokens[:max_len]
        sequences.append(tokens)
    return np.array(sequences)

# Thread 1: Reader and Preprocessor
class ReaderPreprocessor(threading.Thread):
    def __init__(self, df, out_queue):
        super().__init__()
        self.df = df
        self.out_queue = out_queue

    def run(self):
        print("[Thread 1] Reading and preprocessing...")
        self.df['clean_text'] = self.df['text'].map(clean_text)
        X_train_texts, X_test_texts, y_train, y_test = train_test_split(
            self.df['clean_text'], self.df['label'], test_size=0.2,
random_state=42, stratify=self.df['label']
        )

        # Scikit
        vectorizer = TfidfVectorizer(max_features=10000, ngram_range=(1, 2),
sublinear_tf=True)
        X_train_sklearn = vectorizer.fit_transform(X_train_texts)
        X_test_sklearn = vectorizer.transform(X_test_texts)

        # PyTorch
        vocab = build_vocab(self.df['clean_text'])
        X_train_pytorch = tokenize_texts(X_train_texts, vocab)
        X_test_pytorch = tokenize_texts(X_test_texts, vocab)

        # Encode labels
        label_encoder = LabelEncoder()
        y_train_encoded = label_encoder.fit_transform(y_train)
        y_test_encoded = label_encoder.transform(y_test)

        # Put data in queue twice
        data = (X_train_sklearn, X_test_sklearn, X_train_pytorch,
X_test_pytorch, y_train_encoded, y_test_encoded, vectorizer, vocab,
label_encoder, X_train_texts, y_train)
        self.out_queue.put(data)
        self.out_queue.put(data)
        print("[Thread 1] Preprocessing done. Data queued for both threads.")

# Thread 2: Scikit-learn LogisticRegression (retained, with cross-validation)
class ScikitTrainThread(threading.Thread):
    def __init__(self, in_queue, result_queue):
        super().__init__()
        self.in_queue = in_queue
        self.result_queue = result_queue

    def run(self):
        print("[Thread 2] Training Scikit-learn LogisticRegression...")
        X_train_sk, X_test_sk, _, _, y_train, y_test, vectorizer, _,
label_encoder, X_train_texts, y_train_full = self.in_queue.get()
        model = LogisticRegression(random_state=42, max_iter=1000)
        model.fit(X_train_sk, y_train)

        # Cross-validation for realistic accuracy
        cv_scores = cross_val_score(model, X_train_sk, y_train, cv=5)
        print(f"[Thread 2] Cross-Validation Accuracy: {cv_scores.mean():.4f}
(+/- {cv_scores.std() * 2:.4f})")

        y_pred = model.predict(X_test_sk)
        acc = accuracy_score(y_test, y_pred)
        report = classification_report(y_test, y_pred,
target_names=label_encoder.classes_, zero_division=0)
        cm = confusion_matrix(y_test, y_pred)
        self.result_queue.put(("sklearn", model, vectorizer, label_encoder,
acc, report, cm))
        print("[Thread 2] Scikit training done.")

# Thread 3: PyTorch LSTM (added regularization)
class PytorchTrainThread(threading.Thread):
    def __init__(self, in_queue, result_queue):
        super().__init__(daemon=True)
        self.in_queue = in_queue
        self.result_queue = result_queue
    def run(self):
        print("[Thread 3] Training PyTorch LSTM...")
        _, _, X_train_pt, X_test_pt, y_train, y_test, _, vocab, label_encoder, 
_, _ = self.in_queue.get()
        train_ds = TensorDataset(torch.tensor(X_train_pt, dtype=torch.long), 
torch.tensor(y_train, dtype=torch.long))
        train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, 
shuffle=True)

        class LSTMSentiment(nn.Module):
            def __init__(self, vocab_size, embed_dim=EMBED_DIM, 
hidden_size=HIDDEN_SIZE, num_classes=3, dropout=0.7):  # Increased dropout
                super().__init__()
                self.embed = nn.Embedding(vocab_size, embed_dim, 
padding_idx=0)
                self.lstm = nn.LSTM(embed_dim, hidden_size, num_layers=2, 
batch_first=True, bidirectional=True, dropout=dropout)
                self.dropout = nn.Dropout(dropout)
                self.fc = nn.Linear(hidden_size * 2, num_classes)

            def forward(self, x):
                emb = self.embed(x)
                out, (h, _) = self.lstm(emb)
                h_cat = torch.cat((h[-2], h[-1]), dim=1)
                h_cat = self.dropout(h_cat)
                logits = self.fc(h_cat)
                return logits

        model = LSTMSentiment(vocab_size=len(vocab)).to(DEVICE)
        criterion = nn.CrossEntropyLoss()
        optimizer = optim.Adam(model.parameters(), lr=LR, weight_decay=1e-
4)  # Added L2 regularization

        for epoch in range(EPOCHS):
            model.train()
            epoch_loss = 0
            for x, y in train_loader:
                x, y = x.to(DEVICE), y.to(DEVICE)
                optimizer.zero_grad()
                out = model(x)
                loss = criterion(out, y)
                loss.backward()
                optimizer.step()
                epoch_loss += loss.item()
            print(f"[Thread 3] Epoch {epoch+1}/{EPOCHS}, Loss: 
{epoch_loss/len(train_loader):.4f}")
```

```
        # Evaluate
        model.eval()
        test_ds = TensorDataset(torch.tensor(X_test_pt, dtype=torch.long),
torch.tensor(y_test, dtype=torch.long))
        test_loader = DataLoader(test_ds, batch_size=BATCH_SIZE,
shuffle=False)
        preds = []
        with torch.no_grad():
            for x, _ in test_loader:
                x = x.to(DEVICE)
                out = model(x)
                preds.extend(out.argmax(dim=1).cpu().numpy())
        preds = np.array(preds)
        acc = accuracy_score(y_test, preds)
        report = classification_report(y_test, preds,
target_names=label_encoder.classes_, zero_division=0)
        cm = confusion_matrix(y_test, preds)
        self.result_queue.put(("pytorch", model, vocab, label_encoder, acc,
report, cm))
        print("[Thread 3] PyTorch training done.")

# Main
def main():
    # Load train.csv
    df_train = pd.read_csv("train.csv")
    print(f"Loaded train.csv with {len(df_train)} rows.")

    # Queues
    data_queue = queue.Queue(maxsize=2)
    result_queue = queue.Queue(maxsize=2)

    # Start threads
    reader = ReaderPreprocessor(df_train, data_queue)
    scikit_thread = ScikitTrainThread(data_queue, result_queue)
    pytorch_thread = PytorchTrainThread(data_queue, result_queue)

    reader.start()
    scikit_thread.start()
    pytorch_thread.start()

    reader.join()
    scikit_thread.join()
    pytorch_thread.join()

    # Get results
    results = {}
    while not result_queue.empty():
        name, model, artifact, label_encoder, acc, report, cm =
result_queue.get()
        results[name] = (model, artifact, label_encoder, acc, report, cm)

    # Load test.csv for evaluation
    df_test = pd.read_csv("test.csv")
    print(f"Loaded test.csv with {len(df_test)} rows.")
    df_test['clean_text'] = df_test['text'].map(clean_text)
    true_labels = label_encoder.transform(df_test['label'])

    # Scikit evaluation on test.csv
    model_sk, vectorizer, label_encoder, _, _, _ = results["sklearn"]
    X_test_sk = vectorizer.transform(df_test['clean_text'])
    preds_sk = model_sk.predict(X_test_sk)
    real_acc_sk = accuracy_score(true_labels, preds_sk)
    report_sk = classification_report(true_labels, preds_sk,
target_names=label_encoder.classes_, zero_division=0)
    cm_sk = confusion_matrix(true_labels, preds_sk)
    print(f"\n[Scikit] Real Accuracy on test.csv: {real_acc_sk:.4f}")
    print(f"[Scikit] Classification Report:\n{report_sk}")
    print(f"[Scikit] Confusion Matrix:\n{cm_sk}")

    # PyTorch evaluation on test.csv
    model_pt, vocab, _, _, _, _ = results["pytorch"]
    X_test_pt = tokenize_texts(df_test['clean_text'], vocab)
    model_pt.eval()
    preds_pt = []
    test_ds = TensorDataset(torch.tensor(X_test_pt, dtype=torch.long))
    test_loader = DataLoader(test_ds, batch_size=BATCH_SIZE, shuffle=False)
    with torch.no_grad():
        for x in test_loader:
            x = x[0].to(DEVICE)
            out = model_pt(x)
            preds_pt.extend(out.argmax(dim=1).cpu().numpy())
    preds_pt = np.array(preds_pt)
    real_acc_pt = accuracy_score(true_labels, preds_pt)
    report_pt = classification_report(true_labels, preds_pt,
target_names=label_encoder.classes_, zero_division=0)
    cm_pt = confusion_matrix(true_labels, preds_pt)
    print(f"\n[PyTorch] Real Accuracy on test.csv: {real_acc_pt:.4f}")
    print(f"[PyTorch] Classification Report:\n{report_pt}")
    print(f"[PyTorch] Confusion Matrix:\n{cm_pt}")

    # Compare
    print(f"\n[Comparison] Scikit Accuracy: {real_acc_sk:.4f}, PyTorch
Accuracy: {real_acc_pt:.4f}")
    if real_acc_sk > real_acc_pt:
        print("Scikit performs better.")
        better = "Scikit performs better."
    elif real_acc_pt > real_acc_sk:
        print("PyTorch performs better.")
        better = "PyTorch performs better."
    else:
        print("Both perform equally.")
        better = "Both perform equally."

    # Load predict.csv and predict
    df_predict = pd.read_csv("predict.csv")
    print(f"Loaded predict.csv with {len(df_predict)} rows.")
    df_predict['clean_text'] = df_predict['text'].map(clean_text)

    # Scikit predictions
    X_pred_sk = vectorizer.transform(df_predict['clean_text'])
    preds_sk = model_sk.predict(X_pred_sk)
    df_predict_sk = df_predict[['text']].copy()
    df_predict_sk['predicted_label'] =
label_encoder.inverse_transform(preds_sk)
    df_predict_sk.to_csv("predictions_sklearn.csv", index=False)

    # PyTorch predictions
    X_pred_pt = tokenize_texts(df_predict['clean_text'], vocab)
    model_pt.eval()
    preds_pt = []
    pred_ds = TensorDataset(torch.tensor(X_pred_pt, dtype=torch.long))
    pred_loader = DataLoader(pred_ds, batch_size=BATCH_SIZE, shuffle=False)
    with torch.no_grad():
        for x in pred_loader:
            x = x[0].to(DEVICE)
            out = model_pt(x)
            preds_pt.extend(out.argmax(dim=1).cpu().numpy())
    df_predict_pt = df_predict[['text']].copy()
    df_predict_pt['predicted_label'] =
label_encoder.inverse_transform(preds_pt)
    df_predict_pt.to_csv("predictions_pytorch.csv", index=False)

    print("Predictions saved to predictions_sklearn.csv and
predictions_pytorch.csv")

    # GUI
    root = tk.Tk()
    root.title("EMOTIONAL TONE CLASSIFIER")
    root.geometry("800x600")

    notebook = ttk.Notebook(root)
    notebook.pack(fill='both', expand=True)

    # Tab 1: Classification Reports and Comparison
    tab1 = ttk.Frame(notebook)
    notebook.add(tab1, text="Reports & Comparison")

    text1 = tk.Text(tab1, wrap='word')
    scrollbar1 = ttk.Scrollbar(tab1, orient='vertical', command=text1.yview)
    text1.configure(yscrollcommand=scrollbar1.set)
    text1.pack(side='left', fill='both', expand=True)
    scrollbar1.pack(side='right', fill='y')

    # Assuming y_train is available from the queue data, but since it's not
directly, we need to get it.
    # In the code, y_train is passed in data, but in main, we don't have it
directly.
    # To fix, we can store the lengths.

    # Add this after loading df_train
    train_rows = int(0.8 * len(df_train))  # Since test_size=0.2
    test_rows = len(df_test)
    predict_rows = len(df_predict)

    content1 = f"Trained Rows: {train_rows}\nTested Rows:
{test_rows}\nPredicted Rows: {predict_rows}\n\n" \
               f"Scikit Classification Report:\n{report_sk}\n\nConfusion
Matrix:\n{cm_sk}\n\nAccuracy: {real_acc_sk:.4f}\n\n" \
               f"PyTorch Classification Report:\n{report_pt}\n\nConfusion
Matrix:\n{cm_pt}\n\nAccuracy: {real_acc_pt:.4f}\n\n" \
               f"Comparison: {better}"
    text1.insert('1.0', content1)
    text1.config(state='disabled')

    # Tab 2: Predict CSV Data (15 rows)
    tab2 = ttk.Frame(notebook)
    notebook.add(tab2, text="Predict Data")

    text2 = tk.Text(tab2, wrap='word')
    scrollbar2 = ttk.Scrollbar(tab2, orient='vertical', command=text2.yview)
    text2.configure(yscrollcommand=scrollbar2.set)
    text2.pack(side='left', fill='both', expand=True)
    scrollbar2.pack(side='right', fill='y')

    # Fix truncation by displaying full texts
    for i, text in enumerate(df_predict['text'].head(15)):
        text2.insert('end', f"{i+1}: {text}\n\n")
    text2.config(state='disabled')
    # Tab 3: Scikit Predicted Rows
    tab3 = ttk.Frame(notebook)
    notebook.add(tab3, text="Scikit Predictions")

    text3 = tk.Text(tab3, wrap='word')
    scrollbar3 = ttk.Scrollbar(tab3, orient='vertical', command=text3.yview)
    text3.configure(yscrollcommand=scrollbar3.set)
    text3.pack(side='left', fill='both', expand=True)
    scrollbar3.pack(side='right', fill='y')

    # Neat display with row numbers, text, and predicted label
    for i, row in df_predict_sk.head(15).iterrows():
        text3.insert('end', f"{i+1}: Text: {row['text']} | Predicted Label:
{row['predicted_label']}\n\n")
    text3.config(state='disabled')

    # Tab 4: PyTorch Predicted Rows
    tab4 = ttk.Frame(notebook)
    notebook.add(tab4, text="PyTorch Predictions")

    text4 = tk.Text(tab4, wrap='word')
    scrollbar4 = ttk.Scrollbar(tab4, orient='vertical', command=text4.yview)
    text4.configure(yscrollcommand=scrollbar4.set)
    text4.pack(side='left', fill='both', expand=True)
    scrollbar4.pack(side='right', fill='y')

    # Neat display with row numbers, text, and predicted label
    for i, row in df_predict_pt.head(15).iterrows():
        text4.insert('end', f"{i+1}: Text: {row['text']} | Predicted Label:
{row['predicted_label']}\n\n")
    text4.config(state='disabled')

    root.mainloop()

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

Program Outputs:



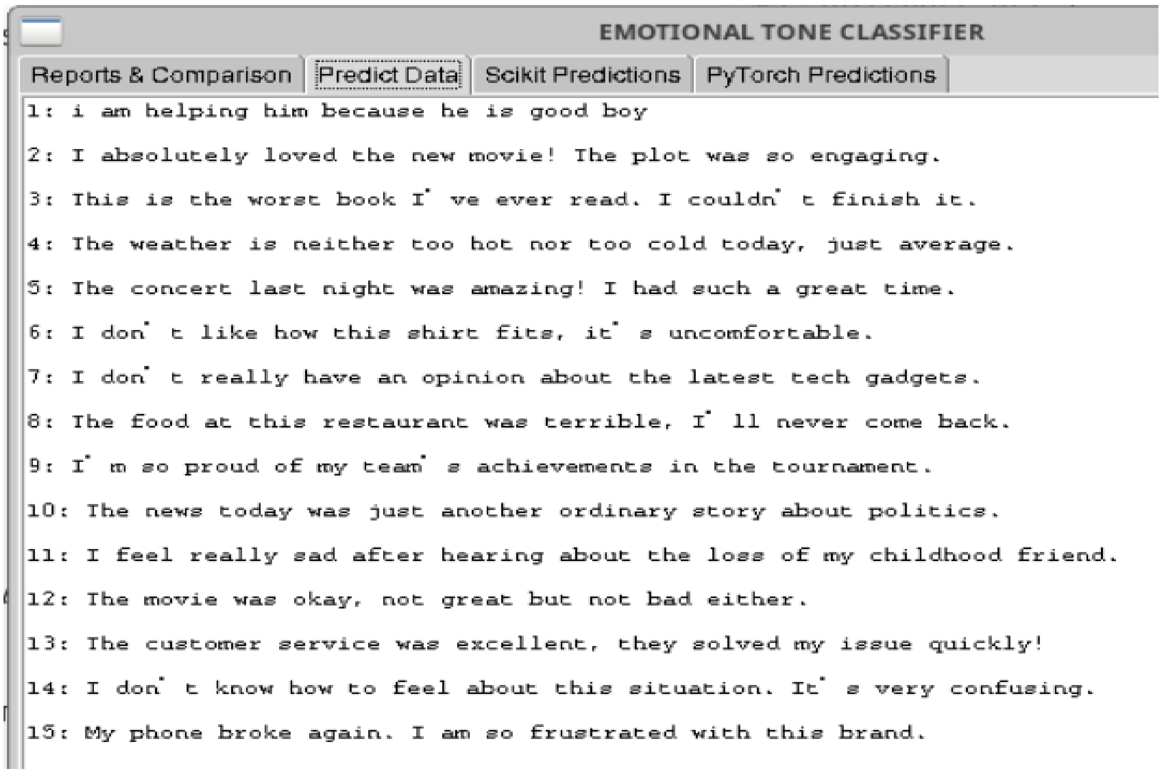
```
Trained Rows: 8168
Tested Rows: 180
Predicted Rows: 16

Scikit Classification Report:
              precision    recall  f1-score   support

    negative       0.77      0.46      0.57        59
     neutral       0.88      0.58      0.70        62
    positive       0.52      0.92      0.66        59

    accuracy                           0.65       180
   macro avg       0.72      0.65      0.65       180
weighted avg       0.73      0.65      0.65       180


Confusion Matrix:
[[27  4 28]
 [ 4 36 22]
 [ 4  1 54]]

Accuracy: 0.6500

PyTorch Classification Report:
              precision    recall  f1-score   support

    negative       0.67      0.51      0.58        59
     neutral       0.95      0.31      0.46        62
    positive       0.48      0.93      0.63        59

    accuracy                           0.58       180
   macro avg       0.70      0.58      0.56       180
weighted avg       0.70      0.58      0.56       180


Confusion Matrix:
[[30  1 28]
 [11 19 32]
 [ 4  0 55]]

Accuracy: 0.5778

Comparison: Scikit performs better.
```

Model Training and Evaluation Flow

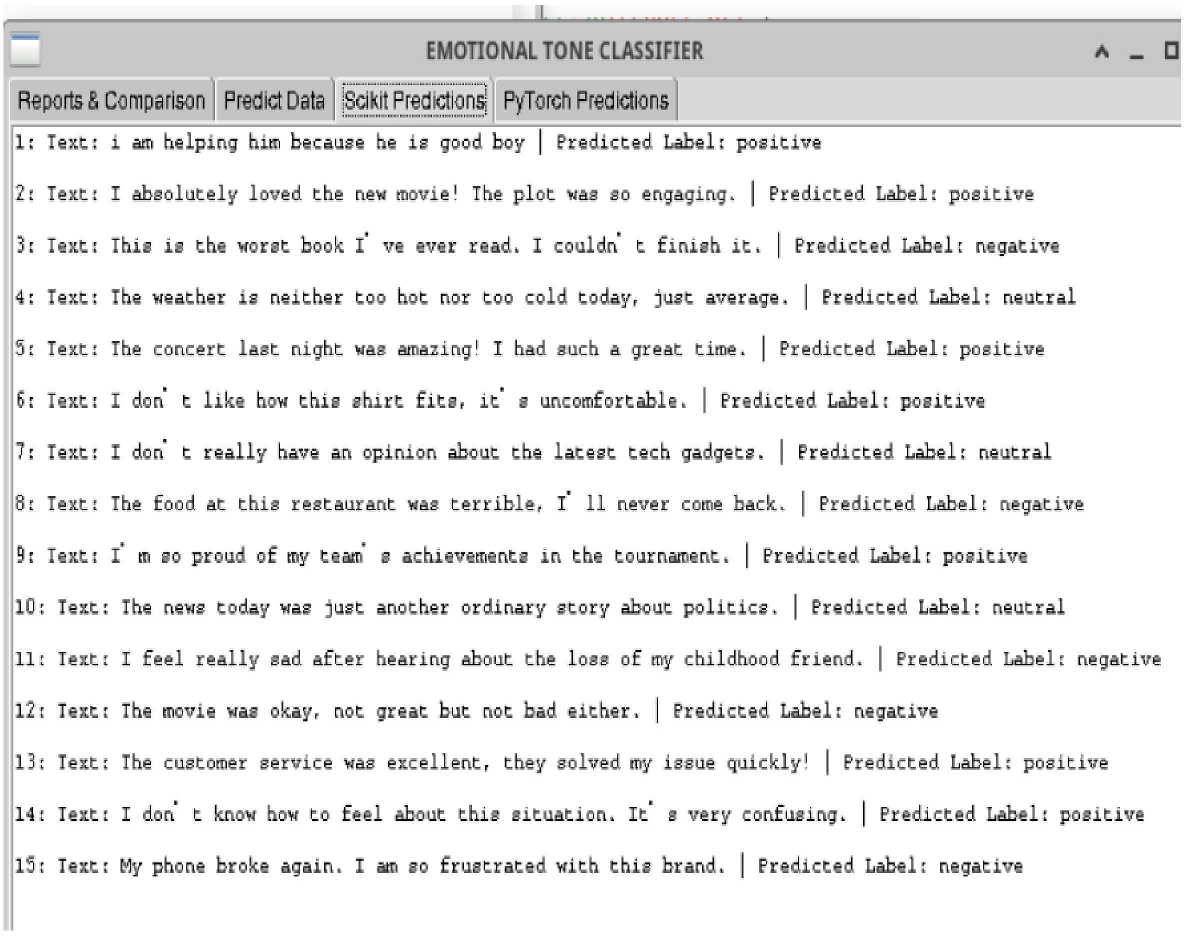
- The dataset was first preprocessed and cleaned.
- A total of 8,168 rows were used to train both models.
- The trained models were then evaluated on 180 unseen test rows.
- Two models were tested:
    - TF-IDF + Logistic Regression (Scikit-learn)
    - LSTM Neural Network (PyTorch)
- Classification reports and confusion matrices were generated using test data only.
- The Scikit-learn model achieved 65% accuracy, while PyTorch achieved 58% accuracy on the same test set.
- Based on test results, Scikit-learn performed better for emotional tone classification.

All results shown are from unseen test data, ensuring fair evaluation.

Demo Output – Unlabeled data vs Scikit Model predicted labels data

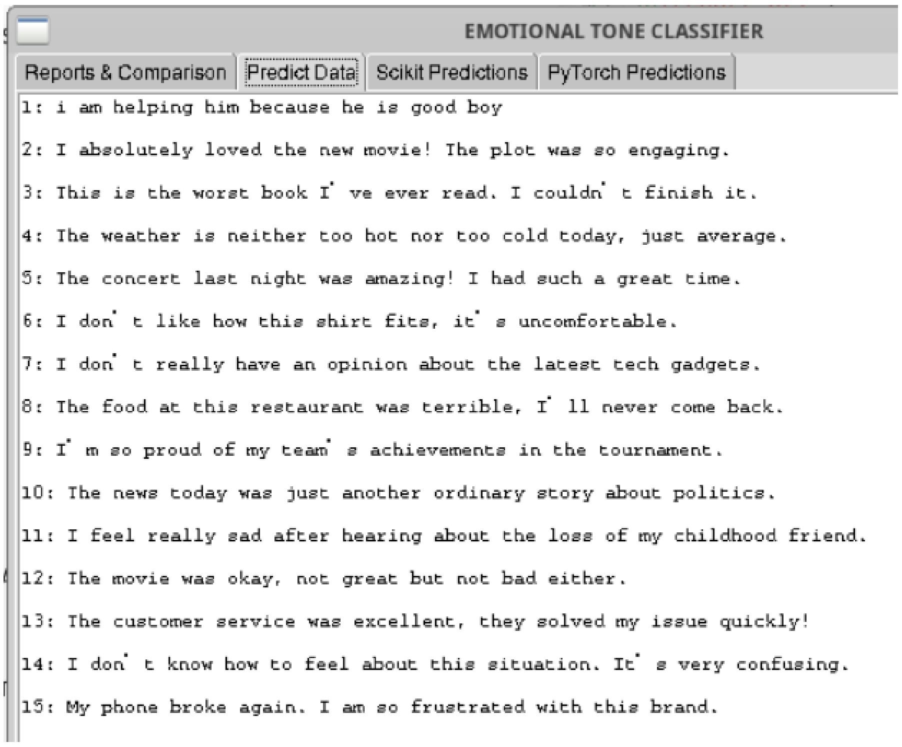


```
1: i am helping him because he is good boy
2: I absolutely loved the new movie! The plot was so engaging.
3: This is the worst book I' ve ever read. I couldn' t finish it.
4: The weather is neither too hot nor too cold today, just average.
5: The concert last night was amazing! I had such a great time.
6: I don' t like how this shirt fits, it' s uncomfortable.
7: I don' t really have an opinion about the latest tech gadgets.
8: The food at this restaurant was terrible, I' ll never come back.
9: I' m so proud of my team' s achievements in the tournament.
10: The news today was just another ordinary story about politics.
11: I feel really sad after hearing about the loss of my childhood friend.
12: The movie was okay, not great but not bad either.
13: The customer service was excellent, they solved my issue quickly!
14: I don' t know how to feel about this situation. It' s very confusing.
15: My phone broke again. I am so frustrated with this brand.
```

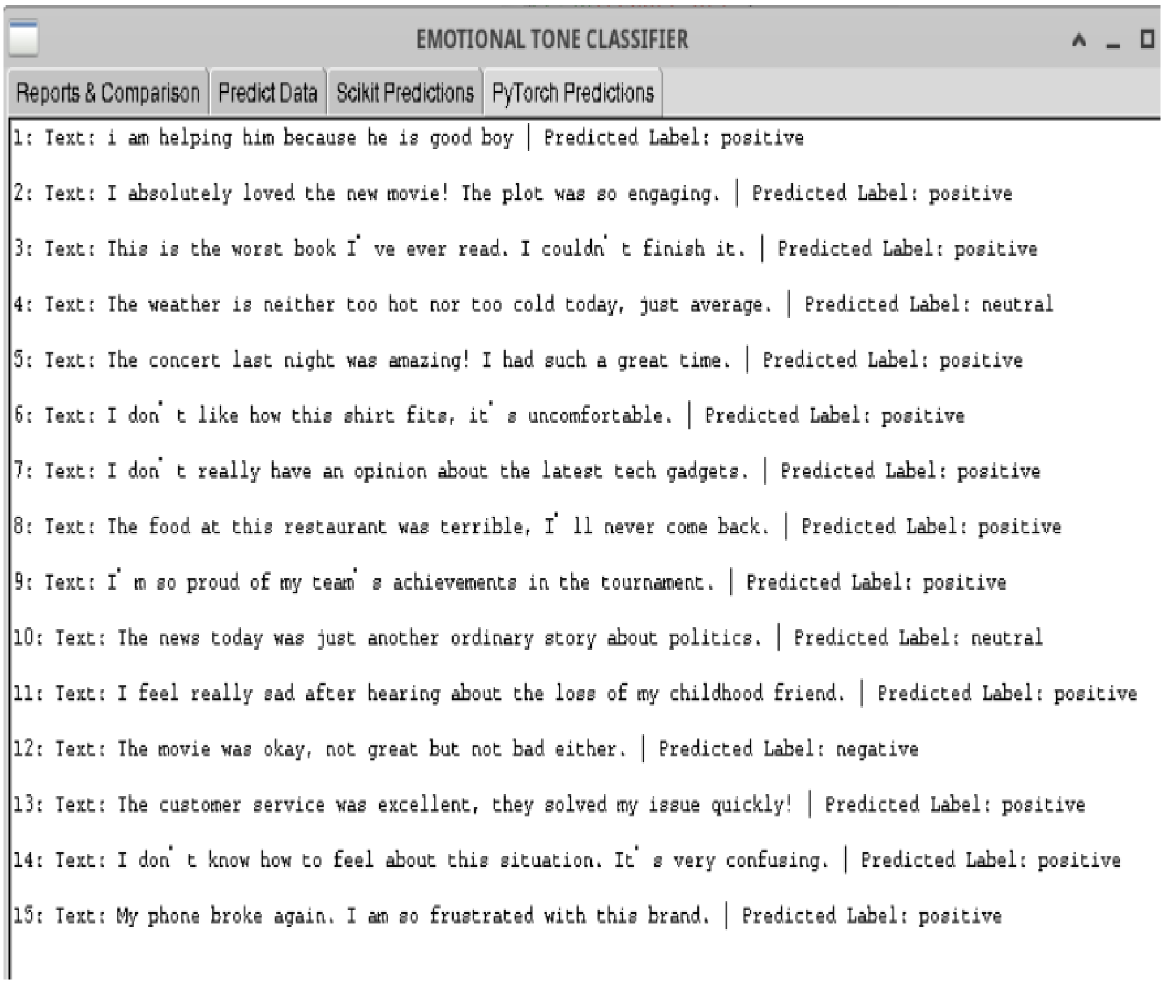


```
1: Text: i am helping him because he is good boy | Predicted Label: positive
2: Text: I absolutely loved the new movie! The plot was so engaging. | Predicted Label: positive
3: Text: This is the worst book I' ve ever read. I couldn' t finish it. | Predicted Label: negative
4: Text: The weather is neither too hot nor too cold today, just average. | Predicted Label: neutral
5: Text: The concert last night was amazing! I had such a great time. | Predicted Label: positive
6: Text: I don' t like how this shirt fits, it' s uncomfortable. | Predicted Label: positive
7: Text: I don' t really have an opinion about the latest tech gadgets. | Predicted Label: neutral
8: Text: The food at this restaurant was terrible, I' ll never come back. | Predicted Label: negative
9: Text: I' m so proud of my team' s achievements in the tournament. | Predicted Label: positive
10: Text: The news today was just another ordinary story about politics. | Predicted Label: neutral
11: Text: I feel really sad after hearing about the loss of my childhood friend. | Predicted Label: negative
12: Text: The movie was okay, not great but not bad either. | Predicted Label: negative
13: Text: The customer service was excellent, they solved my issue quickly! | Predicted Label: positive
14: Text: I don' t know how to feel about this situation. It' s very confusing. | Predicted Label: positive
15: Text: My phone broke again. I am so frustrated with this brand. | Predicted Label: negative
```

Demo Output – Unlabeled data vs Pytorch Model predicted labels data



```
1: i am helping him because he is good boy
2: I absolutely loved the new movie! The plot was so engaging.
3: This is the worst book I' ve ever read. I couldn' t finish it.
4: The weather is neither too hot nor too cold today, just average.
5: The concert last night was amazing! I had such a great time.
6: I don' t like how this shirt fits, it' s uncomfortable.
7: I don' t really have an opinion about the latest tech gadgets.
8: The food at this restaurant was terrible, I' ll never come back.
9: I' m so proud of my team' s achievements in the tournament.
10: The news today was just another ordinary story about politics.
11: I feel really sad after hearing about the loss of my childhood friend.
12: The movie was okay, not great but not bad either.
13: The customer service was excellent, they solved my issue quickly!
14: I don' t know how to feel about this situation. It' s very confusing.
15: My phone broke again. I am so frustrated with this brand.
```



```
1: Text: i am helping him because he is good boy | Predicted Label: positive
2: Text: I absolutely loved the new movie! The plot was so engaging. | Predicted Label: positive
3: Text: This is the worst book I' ve ever read. I couldn' t finish it. | Predicted Label: positive
4: Text: The weather is neither too hot nor too cold today, just average. | Predicted Label: neutral
5: Text: The concert last night was amazing! I had such a great time. | Predicted Label: positive
6: Text: I don' t like how this shirt fits, it' s uncomfortable. | Predicted Label: positive
7: Text: I don' t really have an opinion about the latest tech gadgets. | Predicted Label: positive
8: Text: The food at this restaurant was terrible, I' ll never come back. | Predicted Label: positive
9: Text: I' m so proud of my team' s achievements in the tournament. | Predicted Label: positive
10: Text: The news today was just another ordinary story about politics. | Predicted Label: neutral
11: Text: I feel really sad after hearing about the loss of my childhood friend. | Predicted Label: positive
12: Text: The movie was okay, not great but not bad either. | Predicted Label: negative
13: Text: The customer service was excellent, they solved my issue quickly! | Predicted Label: positive
14: Text: I don' t know how to feel about this situation. It' s very confusing. | Predicted Label: positive
15: Text: My phone broke again. I am so frustrated with this brand. | Predicted Label: positive
```

Libraries Needed

List of Libraries

S.No	Library Name	Version	Installation Command
1	threading	Built-in	No installation required
2	queue	Built-in	No installation required
3	pandas	2.2.2	pip install pandas==2.2.2
4	numpy	1.26.4	pip install numpy==1.26.4
5	re (regex)	Built-in	No installation required
6	scikit-learn	1.5.0	pip install scikit-learn==1.5.0
7	torch (PyTorch)	2.2.0	pip install torch==2.2.0 (CPU)
8	collections (Counter)	Built-in	No installation required
9	tkinter	Built-in	sudo apt-get install python3-tk
10	tkinter.ttk	Built-in	Comes with python3-tk package

Lessons Learnt

- Working with text preprocessing, tokenization, and vectorization
- Differences between classical ML and deep learning for text
- Implementing multi-threaded programs in Python
- Building user-friendly Tkinter GUIs
- Using Scikit-Learn and PyTorch together in one system
- Handling inter-thread communication using shared files
- Understanding model evaluation metrics and performance comparison
- Real-world exposure to building production-style sentiment/emotional tone applications

Conclusion

This project successfully developed a multithreaded Emotional Tone Classifier that leverages both traditional machine learning and deep learning approaches to analyze sentiment in textual data. By integrating Scikit-Learn's Logistic Regression and a PyTorch-based LSTM model in parallel threads, the system achieves efficient training and evaluation while effectively comparing model performance. The preprocessing pipeline ensures clean, standardized input for both models, and the use of a Tkinter GUI provides an interactive and user-friendly interface to visualize results and predictions.

The multithreaded design not only speeds up processing but also allows leveraging complementary strengths of different models, enhancing overall classification accuracy. This tool can be instrumental for applications requiring sentiment analysis on large-scale digital text, such as customer feedback, social media monitoring, and support ticket analysis.

Future work may include expanding the dataset, incorporating more advanced architectures like transformers, and adding support for multilingual text to broaden applicability.

Improvements / Future Work

The Emotional Tone Classifier can be significantly improved by integrating advanced NLP technologies and optimizing the existing architecture. A major enhancement would be the adoption of transformer-based models such as BERT, RoBERTa, DistilBERT, or ALBERT through the Hugging Face Transformers library, as these models offer superior contextual understanding and can greatly increase emotion-classification accuracy. Expanding the dataset with more diverse, multilingual, and domain-rich text sources would help the system generalize better across real-world scenarios. Systematic hyperparameter optimization using grid search, random search, or Bayesian optimization can further refine model performance.

The preprocessing pipeline can also be strengthened by adding tokenizer-based cleaning, stemming, lemmatization, and Hugging Face tokenizers to reduce noise and standardize inputs more effectively. PyTorch training efficiency may be improved through dynamic batching, GPU-based acceleration, and better memory management. Additionally, enhancing the multithreaded pipeline with asynchronous queues, optimized thread utilization, and a more efficient workflow can make the system capable of near real-time inference. Support for multiple languages, domain adaptation for fields like healthcare or finance, and fine-tuning transformer models on specialized datasets will further expand the system's capabilities. Finally, improving the UI—possibly by moving beyond Tkinter to a web-based interface—and integrating explainability tools such as SHAP or LIME can make the classifier more transparent, scalable, and production-ready.