Homework 3

Text classification using LSTM and Logistic Regression on AG News Dataset

Name: Zinia Sultana Joti

NJIT ID: 31730019

Introduction

In this report, we will perform a text classification task using classical machine learning and deep neural network algorithm on a publicly available data: AG News dataset. The goal is to

- classify news articles into predefined categories.
- Apply two different algorithms:
  - classical ml model - Logistic Regression (LR) with TF-IDF features
  - deep learning model- Long Short-Term Memory (LSTM).
- Compare their performance.
- Understand differences in their modeling approaches.

Dataset

The AG's corpus contains 496,835 categorized news articles from more than 2000 news sources. The AG's news topic classification dataset is used as a text classification benchmark. It contains 4 largest classes from the original corpus, each containing 30,000 training samples and 1,900 test samples. The total number of training samples is 120,000 and testing samples is 7,600. The file contains a list of classes with corresponding labels. Dataset was split into training, validation and test sets.
The classes in the dataset are World, Sports, Business, Sci/Tech.

```
[zj98@login02 Code]$ python prepare_data.py
{'text': "Wall St. Bears Claw Back Into the Black (Reuters) Reuters - Short-sellers,
 Wall Street's dwindling\\band of ultra-cynics, are seeing green again.", 'label': 2
}
Train size: 120000
Test size: 7600
Saved train/val/test in data folder.
Train: 108000
Validation: 12000
Test: 7600
Label distribution:
Business: 30000 samples
Sci/Tech: 30000 samples
Sports: 30000 samples
World: 30000 samples
```

Figure 1: Dataset information

Preprocessing

Logistic regression

Logistic regression is a traditional machine learning algorithm for text classification. It uses feature extraction techniques.

Bag-of-words TFIDF: BoG is constructed by using most frequent words from training subset. Here, the count of each word is used as the features, is also defined as 'term-frequency'. The inverse document frequency (IDF) is computed by taking the logarithm of the ratio of total number of documents and number of documents with the word in the training subset. The features are normalized by the largest feature value.

The following preprocessing steps are applied:

1. TF-IDF vectorization
2. Lowercasing
3. Stop words removal (using default "English" stop words)
4. Feature limit to 20,000 terms

LSTM

LSTM is designed to capture sequential dependencies in text. It processes text as ordered sequences rather than independent features.

The pipeline for LSTM focuses on preparing text sequences:

1. Tokenization: converting text into words/subwords
2. Vocabulary building based on training data
3. Padding sequences to fixed length for batch processing
4. No stop word removal required to preserve sentence structure and context for sequence modeling.

Models

Logistic regression

Model: logistic regression
Input: TF-IDF features

Activation function: Sigmoid for binary, Softmax for multi-class classification

The model learns a linear decision boundary that separates classes based on importance of input features.

LSTM

Long Short-Term Memory is a version of recurrent neural network which addresses the vanishing gradient problem faced in RNN. It uses gates activation layers) to forget irrelevant data and remember past knowledge that the network has seen so far. A LSTM consists of three different gates for different purposes.

1. Forget Gate
2. Input Gate
3. Output Gate

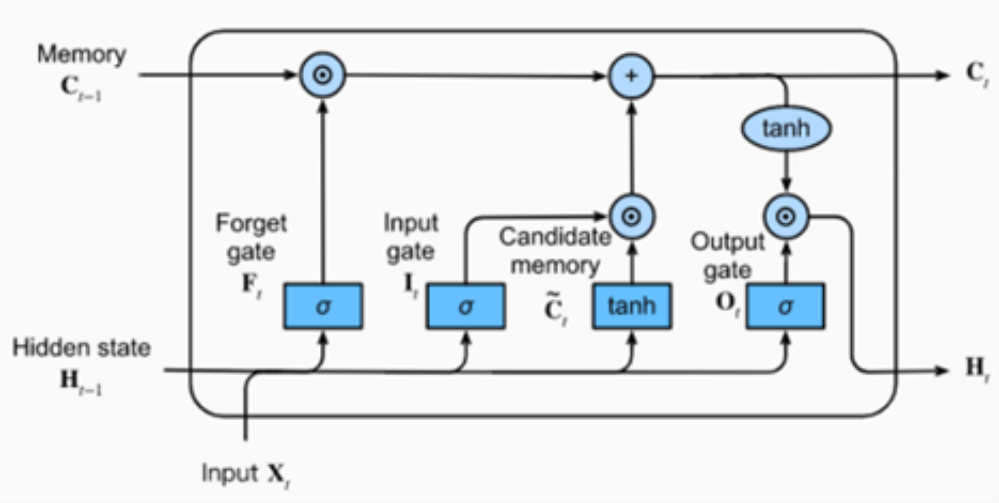

Figure 2: LSTM architecture

Training setup

Batch size: 32
Embedding dimensions: 100
Hidden dimensions: 128
Learning rate: 0.001
Epochs: 10

Loss function: Cross Entropy Loss
Optimizer: Adam
Maximum sequence length: 100
Minimum word frequency: 2
Vocab size: 52162
Number of classes: 4

Evaluation Metrics

There are few metrics to evaluate a model's performance. We will use accuracy, precision, recall, F1 score. These metrics are derived from the confusion matrix.

Confusion Matrix

	Actually positive	Actually negative
Predicted positive	True positives (TP)	False positives (FP)
Predicted negative	False Negatives (FN)	True Negatives (TN)

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

$$Precision, P = \frac{TP}{TP + FP}$$

$$Recall, R = \frac{TP}{TP + FN}$$

$$F1 = \frac{2PR}{P + R}$$

Results

Both Logistic Regression (LR) and LSTM models achieved strong performance on the AG News classification task, with test accuracy more than 90%. However, LR performed slightly better than LSTM across all the evaluation metrics.
Table 1 below illustrates the evaluation metrics where logistic regression consistently shows better performance than LSTM.

Table 1: Evaluation metrics for logistic regression and LSTM model

Model	Loss	Accuracy	Precision	Recall	F1 score
Logistic Regression	0.285912354	0.915263158	0.915083334	0.915263158	0.915071952
LSTM	0.353681161	0.908289474	0.908631597	0.908289474	0.908389392

The performance differences are quite small, Figure 3 demonstrates the variation in values of the metrics.

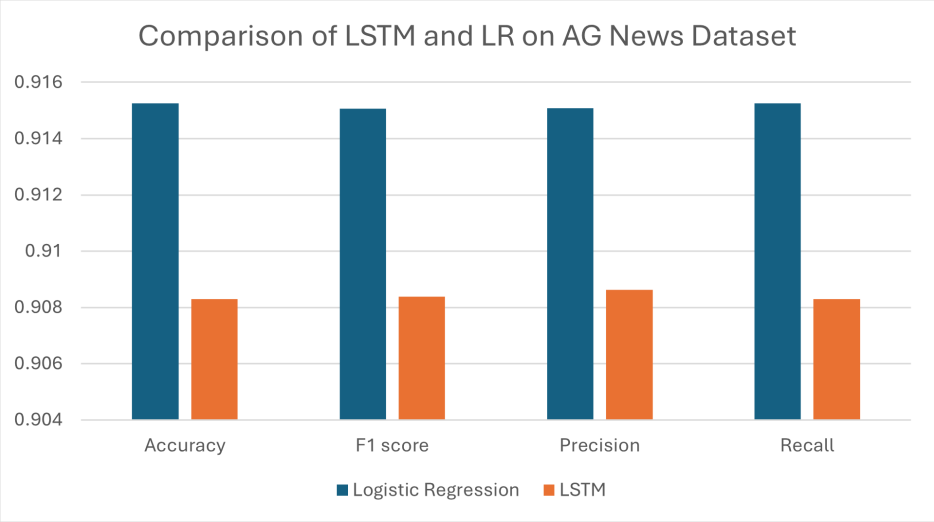

Figure 3: Final comparison of all metrics between Logistic regression and LSTM

Logistic regression output:

Logistic Regression shows strong and stable performance on both validation and test sets, indicating good generalization.

Table 2: Performance of Logistic Regression on validation set and test set

Dataset	Accuracy	Precision	Recall	F1 score
Validation set	0.9215	0.921276	0.9215	0.921301
Test set	0.915263158	0.915083334	0.915263158	0.915071952

LSTM training curves:

Performance of LSTM training improved rapidly after first few epochs.

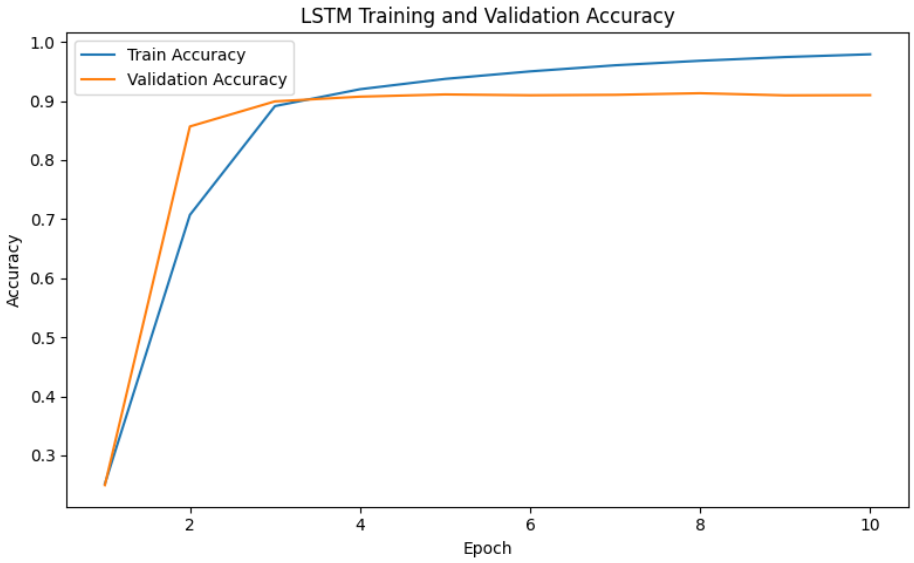

Figure 4: Training and Validation Accuracy of LSTM

Both training and validation accuracy increased significantly, showing fast learning. The validation accuracy stabilizes around 91%, while the training accuracy continues to increase. This indicates mild overfitting on training data.

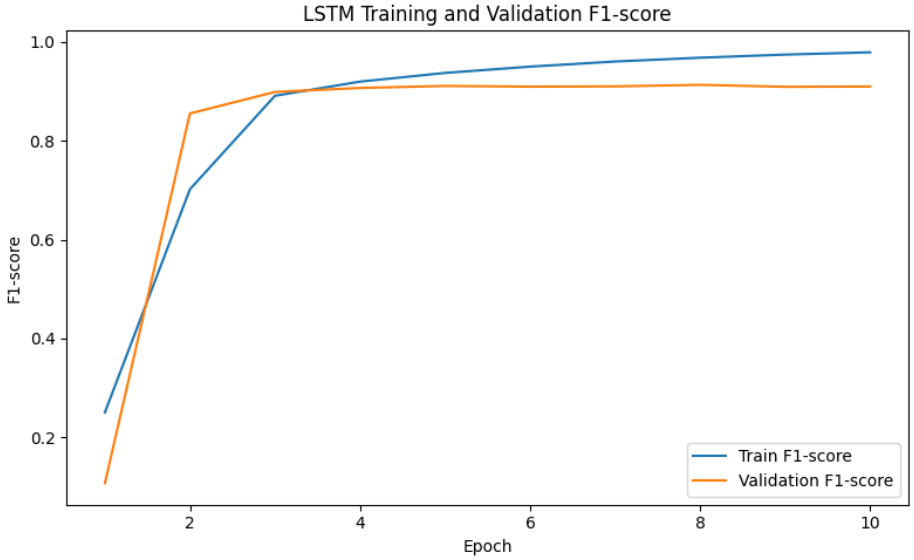

Figure 5: Training and Validation F1 Score of LSTM

Figure 5 shows a similar trend for F1 score. Validation performance stabilizing after around epoch 3–5.

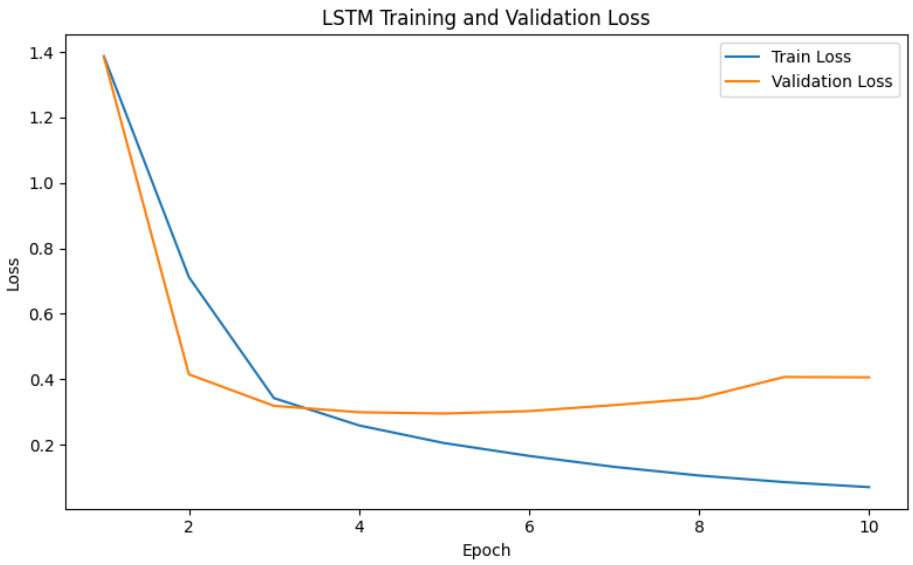

Figure 6: Training and Validation loss of LSTM

The training loss decreases steadily across epochs, indicating models are learning effectively. The validation loss initially decreases but after about 4–5 epochs, it begins to increase. The model starts to overfit in later epochs, as a result it performs worse on unseen data.

Discussion

Logistic Regression combined with TF-IDF features, treats text as bag-of-words representation and based on word frequency it identifies the importance of words. In the AG News structured dataset, articles are categorized by topic-specific keywords, this approach is highly effective and computationally efficient.

On the contrary, LSTM is a sequence-based deep learning model that captures the word order and context in a sentence. LSTM can learn complex linguistic patterns; however, it requires more data and longer training to perform effectively. The number of epochs for this project was only 10 to reduce computational costs. Additionally, LSTM models can improve by tuning hyperparameters such as embedding size, hidden dimensions and sequence length. Moreover, LR ran significantly faster (only 15s), making it more practical choice for this task. By contrast, LSTM required 31 minutes as it involves iterative training and higher computational costs.

Overall, the results demonstrate that traditional machine learning model (in this case, Logistic regression), can remain highly competitive for text classification tasks. While LSTM provides more expressive frameworks.

Conclusion

In brief, both models showed remarkable accuracy on the AG News dataset, while Logistic Regression outperformed LSTM in all metrics. We can imply that simple models with TF-IDF features can be highly effective on structural datasets and on text classification tasks. On the other hand, LSTM can be used where contextual data is required, however, it may require fine tuning its hyperparameters and large dataset.

Appendix: Code

1. prepare_data.py

```python
from datasets import load_dataset
from collections import Counter
from sklearn.model_selection import train_test_split
import pandas as pd
import os

os.system("cls")
# load dataset
dataset = load_dataset("ag_news")

train_data = dataset["train"]
test_data = dataset["test"]

print(train_data[0])
print("Train size:", len(train_data))
print("Test size:", len(test_data))

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

# split train data into train and validation sets
train_df, val_df = train_test_split(
    train_df, test_size=0.1,
    random_state=42,
    stratify=train_df["label"]
    )

os.makedirs("data", exist_ok=True)

train_df.to_csv("data/train.csv", index=False)
val_df.to_csv("data/val.csv", index=False)
test_df.to_csv("data/test.csv", index=False)

print("Saved train/val/test in data folder.")
```

```
print("Train:", len(train_df))
print("Validation:", len(val_df))
print("Test:", len(test_df))

label_counter = Counter(train_data["label"])
# print("Label counts:", label_counter)
label_names = {
    0: "World",
    1: "Sports",
    2: "Business",
    3: "Sci/Tech"
}
print("Label distribution:")
for label, count in label_counter.items():
    print(f"{label_names[label]}: {count} samples")
```

Listing 1: prepare_data.py

2. data_helper.py

```
import re
from collections import Counter
import pandas as pd
import torch
from torch.utils.data import Dataset, DataLoader

def load_data(train_path, val_path, test_path):
    train_df = pd.read_csv(train_path)
    val_df = pd.read_csv(val_path)
    test_df = pd.read_csv(test_path)

    return train_df, val_df, test_df

def clean_text(text):
    text = str(text).lower().strip()
    text = re.sub(r'<.*?>', '', text)        # Remove HTML tags
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)      # Remove non-
       alphanumeric characters (except spaces)
    return text
```

```
19
20 def tokenize(text):
21     return clean_text(text).split()
22
23 def build_vocab(texts, min_freq=2):
24     counter = Counter()
25     for text in texts:
26         tokens = tokenize(text)
27         counter.update(tokens)
28
29     vocab ={
30         "<PAD>": 0,
31         "<UNK>": 1
32     }
33
34     for token, freq in counter.items():
35         if freq >= min_freq:
36             vocab[token] = len(vocab)
37
38     return vocab
39
40 def text_to_sequence(text, vocab):
41     tokens = tokenize(text)
42     return [vocab.get(token, vocab["<UNK>"]) for token in tokens]
43
44 def pad_sequence(seq, max_len, pad_value=0):
45     if len(seq) < max_len:
46         seq = seq+ [pad_value] * (max_len - len(seq))
47     else:
48         seq = seq[:max_len]
49     return seq
50
51 class AGNewsDataset(Dataset):
52     def __init__(self, dataframe, vocab, max_len=100):
53         self.texts = dataframe["text"].tolist()
54         self.labels = dataframe["label"].tolist()
55         self.vocab = vocab
56         self.max_len = max_len
```

```
57
58      def __len__(self):
59          return len(self.texts)
60
61      def __getitem__(self, idx):
62          text = self.texts[idx]
63          label = self.labels[idx]
64
65          seq = text_to_sequence(text, self.vocab)
66          seq = pad_sequence(seq, self.max_len)
67
68          return torch.tensor(seq), torch.tensor(label)
69
70  def create_dataloader(train_df, val_df, test_df, vocab, batch_size
        =32, max_len=100):
71      train_dataset = AGNewsDataset(train_df, vocab, max_len)
72      val_dataset = AGNewsDataset(val_df, vocab, max_len)
73      test_dataset = AGNewsDataset(test_df, vocab, max_len)
74
75      train_loader = DataLoader(train_dataset, batch_size=batch_size,
            shuffle=True)
76      val_loader = DataLoader(val_dataset, batch_size=batch_size,
            shuffle=False)
77      test_loader = DataLoader(test_dataset, batch_size=batch_size,
            shuffle=False)
78
79      return train_loader, val_loader, test_loader
```

Listing 2: data_helper.py

3. lstm.py

```
1  import torch
2  import torch.nn as nn
3
4  class LSTMClassifier(nn.Module):
5      def __init__(self, vocab_size, embed_dim, hidden_dim, output_dim
           , pad_idx, dropout=0.5):
6          super().__init__()
```

```
7
8           self.embedding = nn.Embedding(
9               num_embeddings=vocab_size,
10              embedding_dim=embed_dim,
11              padding_idx=pad_idx
12          )
13
14          self.lstm = nn.LSTM(
15              input_size=embed_dim,
16              hidden_size=hidden_dim,
17              batch_first=True
18          )
19
20          self.fc = nn.Linear(hidden_dim, output_dim)
21          self.dropout = nn.Dropout(dropout)
22
23      def forward(self, text):
24          x = self.embedding(text)
25          _, (hidden, _) = self.lstm(x)
26          hidden = hidden[-1]
27          hidden = self.dropout(hidden)
28          output = self.fc(hidden)
29          return output
```

Listing 3: lstm.py

4. train_lstm.py

```
1  import argparse
2  import torch
3  import torch.nn as nn
4  from sklearn.metrics import accuracy_score,
       precision_recall_fscore_support
5  import pandas as pd
6  import os
7
8  from data_helper import load_data, build_vocab, create_dataloader
9  from lstm import LSTMClassifier
10
```

```python
def train_one_epoch(model, dataloader, optimizer, criterion, device):
    model.train()
    total_loss = 0
    all_preds = []
    all_labels = []

    for texts, labels in dataloader:
        texts, labels = texts.to(device), labels.to(device)

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

        total_loss += loss.item()
        preds = torch.argmax(outputs, dim=1).cpu().numpy()
        all_preds.extend(preds)
        all_labels.extend(labels.cpu().numpy())

    avg_loss = total_loss / len(dataloader)
    acc = accuracy_score(all_labels, all_preds)
    precision, recall, f1, _ = precision_recall_fscore_support(
        all_labels, all_preds, average="weighted", zero_division=0
        )

    return avg_loss, acc, precision, recall, f1

def evaluate(model, dataloader, criterion, device):
    model.eval()
    total_loss = 0
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for texts, labels in dataloader:
            texts, labels = texts.to(device), labels.to(device)
```

```
            outputs = model(texts)
            loss = criterion(outputs, labels)
            total_loss += loss.item()

            preds = torch.argmax(outputs, dim=1).cpu().numpy()
            all_preds.extend(preds)
            all_labels.extend(labels.cpu().numpy())
    avg_loss = total_loss / len(dataloader)
    acc = accuracy_score(all_labels, all_preds)
    precision, recall, f1, _ = precision_recall_fscore_support(
        all_labels, all_preds, average="weighted", zero_division=0
        )
    return avg_loss, acc, precision, recall, f1

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--batch_size", type=int, default=32)
    parser.add_argument("--embed_dim", type=int, default=100)
    parser.add_argument("--hidden_dim", type=int, default=128)
    parser.add_argument("--epochs", type=int, default=10)
    parser.add_argument("--lr", type=float, default=0.001)
    parser.add_argument("--max_len", type=int, default=100)
    parser.add_argument("--min_freq", type=int, default=2)
    args = parser.parse_args()

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

    train_df, val_df, test_df = load_data(
        "data/train.csv",
        "data/val.csv",
        "data/test.csv"
        )
    vocab = build_vocab(train_df["text"], min_freq=args.min_freq)
    pad_idx = vocab["<PAD>"]
    vocab_size = len(vocab)
    num_classes = len(train_df["label"].unique())
```

```
85
86      print("Train size:", len(train_df))
87      print("Validation size:", len(val_df))
88      print("Test size:", len(test_df))
89      print("Vocab size:", vocab_size)
90      print("Number of classes:", num_classes)
91
92      train_loader, val_loader, test_loader = create_dataloader(
93          train_df, val_df, test_df,
94          vocab=vocab,
95          batch_size=args.batch_size,
96          max_len=args.max_len
97      )
98
99      model = LSTMClassifier(
100         vocab_size=vocab_size,
101         embed_dim=args.embed_dim,
102         hidden_dim=args.hidden_dim,
103         output_dim=num_classes,
104         pad_idx=pad_idx,
105         dropout=0.5
106     )
107     model = model.to(device)
108
109     criterion = nn.CrossEntropyLoss()
110     optimizer = torch.optim.Adam(model.parameters(), lr=args.lr)
111
112     best_val_f1 = 0.0
113     best_model_path = f"best_lstm.pt"
114
115     train_losses =[]
116     val_losses = []
117     train_accs = []
118     val_accs = []
119     train_precs = []
120     val_precs = []
121     train_f1s = []
122     val_f1s = []
```

```
123
124     # Training loop
125     for epoch in range( args.epochs):
126         train_loss, train_acc, train_prec, train_rec, train_f1 =
               train_one_epoch(
127             model, train_loader, optimizer, criterion, device
128         )
129         train_losses.append(train_loss)
130         train_accs.append(train_acc)
131         train_precs.append(train_prec)
132         train_f1s.append(train_f1)
133
134         val_loss, val_acc, val_prec, val_rec, val_f1 = evaluate(
135             model, val_loader, criterion, device
136         )
137         val_losses.append(val_loss)
138         val_accs.append(val_acc)
139         val_precs.append(val_prec)
140         val_f1s.append(val_f1)
141
142         print(f"Epoch {epoch+1}/{args.epochs}")
143         print(f"Train Loss: {train_loss:.4f} | Acc: {train_acc:.4f}
               | Prec: {train_prec:.4f} | Rec: {train_rec:.4f} | F1: {
               train_f1:.4f}")
144         print(f"Val Loss: {val_loss:.4f} | Acc: {val_acc:.4f} | Prec
               : {val_prec:.4f} | Rec: {val_rec:.4f} | F1: {val_f1:.4f}"
               )
145
146         if val_f1>best_val_f1:
147             best_val_f1 = val_f1
148             torch.save(model.state_dict(), best_model_path)
149             print(f"New best model saved with F1: {best_val_f1:.4f}"
                   )
150
151     history_df = pd.DataFrame({
152         "epoch": list(range(1, args.epochs + 1)),
153         "train_loss": train_losses,
154         "val_loss": val_losses,
```

```
        "train_acc": train_accs,
        "val_acc": val_accs,
        "train_prec": train_precs,
        "val_prec": val_precs,
        "train_f1": train_f1s,
        "val_f1": val_f1s
    })
    history_df.to_csv(f"lstm_training_history.csv", index=False)

    # Load best model and evaluate on test set
    model.load_state_dict(torch.load(best_model_path))
    test_loss, test_acc, test_prec, test_rec, test_f1 = evaluate(
        model, test_loader, criterion, device
    )

    os.makedirs("results", exist_ok=True)

    summary_df = pd.DataFrame([{
    "model": "lstm",
    "test_loss": test_loss,
    "test_accuracy": test_acc,
    "test_precision": test_prec,
    "test_recall": test_rec,
    "test_f1": test_f1
    }])

    summary_df.to_csv(f"results/lstm_test_results.csv", index=False)

    print("\nFinal Test Results:")
    print(f"Test Loss: {test_loss:.4f} | Acc: {test_acc:.4f} | Prec:
        {test_prec:.4f} | Rec: {test_rec:.4f} | F1: {test_f1:.4f}")

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

Listing 4: train_lstm.py

5. train_logreg.py

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,
   precision_recall_fscore_support
import os
from sklearn.metrics import log_loss



def load_data():
    train_df = pd.read_csv("data/train.csv")
    val_df = pd.read_csv("data/val.csv")
    test_df = pd.read_csv("data/test.csv")
    return train_df, val_df, test_df


def evaluate_model(model, X, y):
    preds = model.predict(X)

    acc = accuracy_score(y, preds)
    precision, recall, f1, _ = precision_recall_fscore_support(
        y, preds, average="weighted", zero_division=0
    )

    return acc, precision, recall, f1


def main():
    os.makedirs("results", exist_ok=True)

    # Load data
    train_df, val_df, test_df = load_data()

    X_train = train_df["text"]
    y_train = train_df["label"]
```

```
36
37      X_val = val_df["text"]
38      y_val = val_df["label"]
39
40      X_test = test_df["text"]
41      y_test = test_df["label"]
42
43      # TF-IDF vectorization
44      vectorizer = TfidfVectorizer(
45          lowercase=True,
46          stop_words="english",
47          max_features=20000
48      )
49
50      X_train_tfidf = vectorizer.fit_transform(X_train)
51      X_val_tfidf = vectorizer.transform(X_val)
52      X_test_tfidf = vectorizer.transform(X_test)
53
54      # Logistic Regression model
55      model = LogisticRegression(
56          max_iter=1000,
57          random_state=42
58      )
59
60      model.fit(X_train_tfidf, y_train)
61
62      # Validation metrics
63      val_acc, val_prec, val_rec, val_f1 = evaluate_model(model,
            X_val_tfidf, y_val)
64
65      print("Validation Results")
66      print(f"Accuracy : {val_acc:.4f}")
67      print(f"Precision: {val_prec:.4f}")
68      print(f"Recall   : {val_rec:.4f}")
69      print(f"F1-score : {val_f1:.4f}")
70
71      # Test metrics
```

```
    test_acc, test_prec, test_rec, test_f1 = evaluate_model(model,
        X_test_tfidf, y_test)

    print("\nTest Results")
    print(f"Accuracy : {test_acc:.4f}")
    print(f"Precision: {test_prec:.4f}")
    print(f"Recall   : {test_rec:.4f}")
    print(f"F1-score : {test_f1:.4f}")

    probs = model.predict_proba(X_test_tfidf)
    test_loss = log_loss(y_test, probs)

    # Save summary
    summary_df = pd.DataFrame([{
        "model": "logistic_regression",
        "val_accuracy": val_acc,
        "val_precision": val_prec,
        "val_recall": val_rec,
        "val_f1": val_f1,
        "test_accuracy": test_acc,
        "test_precision": test_prec,
        "test_recall": test_rec,
        "test_f1": test_f1,
        "test_loss":test_loss
    }])

    summary_df.to_csv("results/logreg_test_results.csv", index=False
        )
    print("\nSaved results to results/logreg_test_results.csv")


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

Listing 5: train_logreg.py

6. visual.py

```python
import pandas as pd
import matplotlib.pyplot as plt
import os

def plot_lstm_loss(lstm_history):
    plt.figure(figsize=(10, 5))
    plt.plot(lstm_history["epoch"], lstm_history["train_loss"],
        label="LSTM Train Loss")
    plt.plot(lstm_history["epoch"], lstm_history["val_loss"], label=
        "LSTM Val Loss")
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.title("LSTM Training and Validation Loss")
    plt.legend()
    plt.savefig(os.path.join("plots", "lstm_loss_plot.png"))
    plt.show()

def plot_lstm_accuracy(lstm_history):
    plt.figure(figsize=(10, 5))
    plt.plot(lstm_history["epoch"], lstm_history["train_acc"], label
        ="LSTM Train Accuracy")
    plt.plot(lstm_history["epoch"], lstm_history["val_acc"], label="
        LSTM Val Accuracy")
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
    plt.title("LSTM Training and Validation Accuracy")
    plt.legend()
    plt.savefig(os.path.join("plots", "lstm_accuracy_plot.png"))
    plt.show()

def plot_lstm_f1(lstm_history):
    plt.figure(figsize=(10, 5))
    plt.plot(lstm_history["epoch"], lstm_history["train_f1"], label=
        "LSTM Train F1")
    plt.plot(lstm_history["epoch"], lstm_history["val_f1"], label="
        LSTM Val F1")
    plt.xlabel("Epoch")
    plt.ylabel("F1 Score")
```

```
    plt.title("LSTM Training and Validation F1 Score")
    plt.legend()
    plt.savefig(os.path.join("plots", "lstm_f1_plot.png"))
    plt.show()

def plot_test_comparison(lstm_test, logreg_test):
    metrics = ["test_accuracy", "test_precision", "test_recall", "
        test_f1"]
    metric_names = ["Accuracy", "Precision", "Recall", "F1 Score"]

    lstm_values = [lstm_test[m].iloc[0] for m in metrics]
    logreg_values = [logreg_test[m].iloc[0] for m in metrics]

    x = range(len(metrics))
    width = 0.35

    plt.figure(figsize=(10, 5))
    plt.bar([i - width/2 for i in x], lstm_values, width=width,
        label="LSTM")
    plt.bar([i + width/2 for i in x], logreg_values, width=width,
        label="Logistic Regression")
    plt.xlabel("Metrics")
    plt.ylabel("Score")
    plt.title("Final Test Results Comparison")
    plt.xticks(list(x), metric_names)
    plt.legend()
    plt.savefig(os.path.join("plots", "test_comparison.png"))
    plt.show()

def main():
    os.makedirs("plots",exist_ok=True)

    lstm_history = pd.read_csv("lstm_training_history.csv")
    logreg_history = pd.read_csv("results/logreg_training_history.
        csv")

    lstm_test = pd.read_csv("results/lstm_test_results.csv")
    logreg_test = pd.read_csv("results/logreg_test_results.csv")
```

```
67
68      plot_lstm_loss(lstm_history)
69      plot_lstm_accuracy(lstm_history)
70      plot_lstm_f1(lstm_history)
71      plot_test_comparison(lstm_test, logreg_test)
72
73  if __name__ == "__main__":
74      main()
```

Listing 6: visual.py