```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import pandas as pd
import nltk
import re
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from torch.nn.utils.rnn import pad_sequence
from collections import Counter

nltk.download('punkt')
# nltk.download('punkt_tab') # Download the missing resource - removed
from nltk.tokenize import RegexpTokenizer # Use a different tokenizer

# 1. Load Data
df = pd.read_csv("/content/tweet_sentiment_3000.csv")  # CSV should have 'tweet' and 'label' columns

# 2. Preprocessing
def clean_text(text):
    text = re.sub(r"http\S+", "", text)
    text = re.sub(r"[^A-Za-z0-9 ]+", "", text)
    text = text.lower()
    return text

df["tweet"] = df["tweet"].apply(clean_text)

# 3. Tokenization and Vocabulary
tokenizer = RegexpTokenizer(r'\w+') # Initialize RegexpTokenizer
tokenized = [tokenizer.tokenize(t) for t in df["tweet"]] # Use RegexpTokenizer
word_counts = Counter(word for sentence in tokenized for word in sentence)
vocab = {word: i + 2 for i, (word, _) in enumerate(word_counts.items())}
vocab["<PAD>"] = 0
vocab["<UNK>"] = 1

def encode(text):
    # Use the same tokenizer for encoding
    return [vocab.get(word, vocab["<UNK>"]) for word in tokenizer.tokenize(text)]

encoded = [torch.tensor(encode(t)) for t in df["tweet"]]
padded = pad_sequence(encoded, batch_first=True, padding_value=0)
labels = torch.tensor(df["label"].values)

# 4. Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(padded, labels, test_size=0.2)

# 5. Dataset and Dataloader
class TweetDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

    def __len__(self):
        return len(self.X)

train_ds = TweetDataset(X_train, y_train)
test_ds = TweetDataset(X_test, y_test)

train_loader = DataLoader(train_ds, batch_size=32, shuffle=True)
test_loader = DataLoader(test_ds, batch_size=32)

# 6. LSTM Model
class LSTMClassifier(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=0)
        self.lstm = nn.LSTM(embed_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, 1)
        self.sigmoid = nn.Sigmoid()

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

```
            _, (hn, _) = self.lstm(x)
            out = self.fc(hn[-1])
            return self.sigmoid(out).squeeze()

    model = LSTMClassifier(vocab_size=len(vocab), embed_dim=64, hidden_dim=128)
    criterion = nn.BCELoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    # 7. Training
    for epoch in range(15):
        model.train()
        for X_batch, y_batch in train_loader:
            preds = model(X_batch)
            loss = criterion(preds, y_batch.float())
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        print(f"Epoch {epoch+1}, Loss: {loss.item():.4f}")

    # 8. Evaluation
    model.eval()
    all_preds = []
    all_labels = []
    with torch.no_grad():
        for X_batch, y_batch in test_loader:
            preds = model(X_batch)
            all_preds.extend(preds.round().numpy())
            all_labels.extend(y_batch.numpy())

    acc = accuracy_score(all_labels, all_preds)
    print(f"Test Accuracy: {acc:.4f}")
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Epoch 1, Loss: 0.0015
Epoch 2, Loss: 0.0006
Epoch 3, Loss: 0.0003
Epoch 4, Loss: 0.0002
Epoch 5, Loss: 0.0002
Epoch 6, Loss: 0.0001
Epoch 7, Loss: 0.0001
Epoch 8, Loss: 0.0001
Epoch 9, Loss: 0.0001
Epoch 10, Loss: 0.0001
Epoch 11, Loss: 0.0000
Epoch 12, Loss: 0.0000
Epoch 13, Loss: 0.0000
Epoch 14, Loss: 0.0000
Epoch 15, Loss: 0.0000
Test Accuracy: 1.0000
```