```
import numpy as np
import pandas as pd
import re
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
```

```
df = pd.read_csv("SMSSpamCollection", sep='\t', names=["label","text"])
print(df.head())
print(df['label'].value_counts())
```

```
  label                                               text
0   ham  Go until jurong point, crazy.. Available only ...
1   ham                      Ok lar... Joking wif u oni...
2  spam  Free entry in 2 a wkly comp to win FA Cup fina...
3   ham  U dun say so early hor... U c already then say...
4   ham  Nah I don't think he goes to usf, he lives aro...
label
ham     4825
spam     747
Name: count, dtype: int64
```

```
df['label'] = df['label'].map({'ham':0, 'spam':1})
```

```
def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    return text
```

```
df['text'] = df['text'].apply(clean_text)
```

```
tokenized = df['text'].apply(lambda x: x.split())
```

```
from collections import Counter

counter = Counter()
for tokens in tokenized:
    counter.update(tokens)

vocab = {word:i+2 for i,(word,_) in enumerate(counter.items())}
vocab['<PAD>'] = 0
vocab['<UNK>'] = 1
```

```
# http://nlp.stanford.edu/data/glove.6B.zip
```

```
!wget https://nlp.stanford.edu/data/glove.6B.zip
!unzip -q glove.6B.zip

embedding_dim = 100
embeddings_index = {}

with open('glove.6B.100d.txt', encoding='utf8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = vector
```

```
--2026-02-19 03:48:41--  https://nlp.stanford.edu/data/glove.6B.zip
Resolving nlp.stanford.edu (nlp.stanford.edu)... 171.64.67.140
Connecting to nlp.stanford.edu (nlp.stanford.edu)|171.64.67.140|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip [following]
--2026-02-19 03:48:42--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
```

```
Length: 862182613 (822M) [application/zip]
Saving to: ‘glove.6B.zip’

glove.6B.zip        100%[===================>] 822.24M  5.04MB/s    in 2m 40s

2026-02-19 03:51:22 (5.14 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]
```

```
embedding_matrix = np.zeros((len(vocab), embedding_dim))

for word, i in vocab.items():
    vector = embeddings_index.get(word)
    if vector is not None:
        embedding_matrix[i] = vector
```

```
def encode(text):
    return [vocab.get(word,1) for word in text.split()]
```

```
max_len = 50

def pad(seq):
    if len(seq) < max_len:
        seq += [0]*(max_len-len(seq))
    else:
        seq = seq[:max_len]
    return seq
```

```
X = df['text'].apply(lambda x: pad(encode(x))).tolist()
y = df['label'].values

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=42)
```

```
class TextCNN(nn.Module):
    def __init__(self, vocab_size, embedding_dim, embedding_matrix):
        super(TextCNN, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.embedding.weight.data.copy_(torch.from_numpy(embedding_matrix))
        self.embedding.weight.requires_grad = False

        self.conv = nn.Conv1d(embedding_dim, 100, kernel_size=5)
        self.pool = nn.MaxPool1d(2)
        self.fc = nn.Linear(100, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.embedding(x)
        x = x.permute(0,2,1)
        x = self.conv(x)
        x = self.pool(x)
        x = torch.mean(x, dim=2)
        x = self.fc(x)
        return self.sigmoid(x)
```

```
model = TextCNN(len(vocab), embedding_dim, embedding_matrix)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
for epoch in range(5):
    model.train()
    inputs = torch.LongTensor(X_train)
    labels = torch.FloatTensor(y_train)

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

    print(f"Epoch {epoch+1}, Loss: {loss.item()}")
```

```
Epoch 1, Loss: 0.6734747886657715
Epoch 2, Loss: 0.6268223524093628
Epoch 3, Loss: 0.589587390422821
Epoch 4, Loss: 0.5607481002807617
Epoch 5, Loss: 0.5391108989715576
```

```
model.eval()
with torch.no_grad():
    test_inputs = torch.LongTensor(X_test)
    predictions = model(test_inputs).squeeze()
    preds = (predictions > 0.5).int().numpy()
```

```
print("Accuracy:", accuracy_score(y_test, preds))
print("Precision:", precision_score(y_test, preds))
print("Recall:", recall_score(y_test, preds))
print("F1-score:", f1_score(y_test, preds))
print("Confusion Matrix:\n", confusion_matrix(y_test, preds))
```

```
Accuracy: 0.8663677130044843
Precision: 0.0
Recall: 0.0
F1-score: 0.0
Confusion Matrix:
 [[966   0]
 [149   0]]
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
print(set(y_train))
```

```
{np.int64(0), np.int64(1)}
```

```
labels = labels.float()
```

```
train_dataset = torch.utils.data.TensorDataset(
    torch.LongTensor(X_train),
    torch.FloatTensor(y_train)
)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)

pos_weight = torch.tensor([len(y_train)/sum(y_train)])

criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

for epoch in range(10):
    model.train()
    total_loss = 0

    for inputs, labels in train_loader:
        labels = labels.float()

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

        total_loss += loss.item()

    print(f"Epoch {epoch+1}, Loss: {total_loss/len(train_loader)}")
```

```
Epoch 1, Loss: 0.9587992949145181
Epoch 2, Loss: 0.9551339166504996
Epoch 3, Loss: 0.9537365938935961
Epoch 4, Loss: 0.9529187389782496
Epoch 5, Loss: 0.9495579711028508
Epoch 6, Loss: 0.9490008464881352
Epoch 7, Loss: 0.9464917685304369
Epoch 8, Loss: 0.9452875605651311
Epoch 9, Loss: 0.9465128540992737
Epoch 10, Loss: 0.9448722447667803
```

```
model.eval()

with torch.no_grad():
    test_inputs = torch.LongTensor(X_test)
    outputs = model(test_inputs)
    probs = torch.sigmoid(outputs).squeeze()
    preds = (probs > 0.5).int().numpy()
```

```
print("Predicted class counts:", np.bincount(preds))
```

```
Predicted class counts: [224 891]
```

```
criterion = nn.BCEWithLogitsLoss()
```

```
preds = (probs > 0.6).int().numpy()
```

```
pos_weight = torch.tensor([2.0])
criterion = nn.BCEWithLogitsLoss(pos_weight=pos_weight)
```

```
print(confusion_matrix(y_test, preds))
```

```
[[946  20]
 [ 15 134]]
```

```
cm = confusion_matrix(y_test, preds)

accuracy = (cm[0][0] + cm[1][1]) / cm.sum()
print("Accuracy:", accuracy)
```

```
Accuracy: 0.968609865470852
```

```
Start coding or generate with AI.
```