```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import re
import matplotlib.pyplot as plt
```

```
df = pd.read_csv("SMSSpamCollection", sep='\t', header=None)
df.columns = ['label', 'text']

df.head()
df['label'].value_counts()
```

label	count
ham	4825
spam	747

dtype: int64

```
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)
```

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

```
tokenized_texts = [text.split() for text in df['text']]
```

```
vocab = {}
for sentence in tokenized_texts:
    for word in sentence:
        if word not in vocab:
            vocab[word] = len(vocab) + 1

vocab_size = len(vocab) + 1
print("Vocabulary Size:", vocab_size)
```

```
Vocabulary Size: 8630
```

```
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip
```

```
--2026-02-19 03:54:39--  http://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|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://nlp.stanford.edu/data/glove.6B.zip [following]
--2026-02-19 03:54:39--  https://nlp.stanford.edu/data/glove.6B.zip
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 03:54:39--  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  5.01MB/s    in 2m 42s

2026-02-19 03:57:21 (5.08 MB/s) - ‘glove.6B.zip’ saved [862182613/862182613]

Archive:  glove.6B.zip
```

```
  inflating: glove.6B.50d.txt
  inflating: glove.6B.100d.txt
  inflating: glove.6B.200d.txt
  inflating: glove.6B.300d.txt
```

```
embeddings_index = {}
embedding_dim = 100 # Using 100d embeddings as an example
with open('glove.6B.100d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

embedding_matrix = np.zeros((vocab_size, embedding_dim))

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

```
max_len = 50

def encode_text(tokens):
    encoded = [vocab.get(word, 0) for word in tokens]
    if len(encoded) < max_len:
        encoded += [0] * (max_len - len(encoded))
    else:
        encoded = encoded[:max_len]
    return encoded

X = np.array([encode_text(tokens) for tokens in tokenized_texts])
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.from_pretrained(
            torch.tensor(embedding_matrix, dtype=torch.float32),
            freeze=False
        )

        self.conv = nn.Conv1d(
            in_channels=embedding_dim,
            out_channels=100,
            kernel_size=3
        )

        self.pool = nn.AdaptiveMaxPool1d(1)
        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).squeeze(2)
        x = self.fc(x)
        return self.sigmoid(x)
```

```
model = TextCNN(vocab_size, embedding_dim, embedding_matrix)

criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

X_train_tensor = torch.tensor(X_train, dtype=torch.long)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)

for epoch in range(5):
    model.train()
    optimizer.zero_grad()
```

```
    outputs = model(X_train_tensor).squeeze()
    loss = criterion(outputs, y_train_tensor)

    loss.backward()
    optimizer.step()

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

```
Epoch 1, Loss: 0.6645227074623108
Epoch 2, Loss: 0.5612035393714905
Epoch 3, Loss: 0.4860960841178894
Epoch 4, Loss: 0.43546655774116516
Epoch 5, Loss: 0.4042167365550995
```

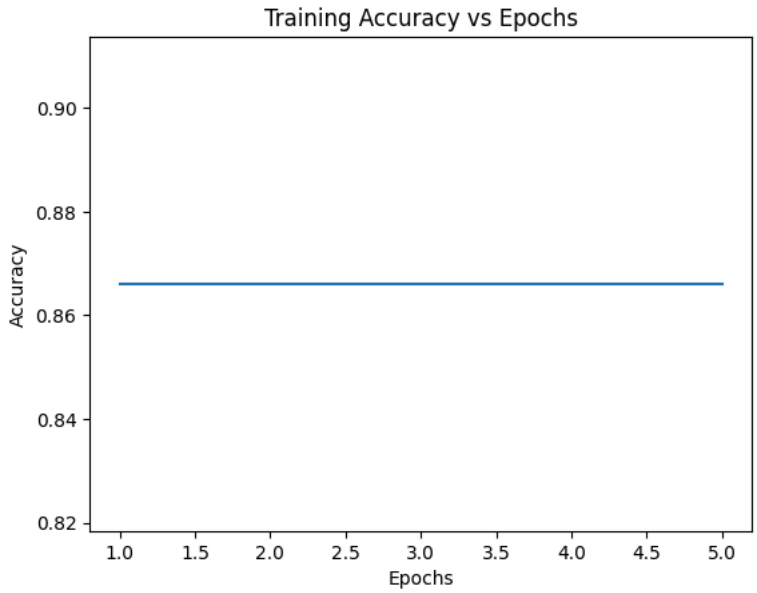
```
losses = []

for epoch in range(5):
    model.train()
    optimizer.zero_grad()

    outputs = model(X_train_tensor).squeeze()
    loss = criterion(outputs, y_train_tensor)

    loss.backward()
    optimizer.step()

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

```
Epoch 1, Loss: 0.38710859417915344
Epoch 2, Loss: 0.3794078230857849
Epoch 3, Loss: 0.37721553444862366
Epoch 4, Loss: 0.3775363862514496
Epoch 5, Loss: 0.37822797894477844
```

```
accuracies = []

for epoch in range(5):
    model.train()
    optimizer.zero_grad()

    outputs = model(X_train_tensor).squeeze()
    loss = criterion(outputs, y_train_tensor)

    loss.backward()
    optimizer.step()

    preds = (outputs > 0.5).float()
    accuracy = (preds == y_train_tensor).float().mean()

    accuracies.append(accuracy.item())
    losses.append(loss.item())

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

```
Epoch 1, Loss: 0.3779047429561615, Accuracy: 0.8660534024238586
Epoch 2, Loss: 0.3758465647697449, Accuracy: 0.8660534024238586
Epoch 3, Loss: 0.3717876076698303, Accuracy: 0.8660534024238586
Epoch 4, Loss: 0.3657819926738739, Accuracy: 0.8660534024238586
Epoch 5, Loss: 0.35809552669525146, Accuracy: 0.8660534024238586
```

```
plt.figure()
plt.plot(range(1, len(accuracies)+1), accuracies)
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training Accuracy vs Epochs")
plt.show()
```




```
from sklearn.metrics import confusion_matrix
import numpy as np
import torch

# Convert X_test to a PyTorch tensor
X_test_tensor = torch.tensor(X_test, dtype=torch.long)

# Set the model to evaluation mode
model.eval()

# Get predictions
with torch.no_grad():
    outputs = model(X_test_tensor).squeeze()
    predictions = (outputs > 0.5).float().numpy() # Apply threshold and convert to numpy

cm = confusion_matrix(y_test, predictions)

plt.figure()
plt.imshow(cm)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.colorbar()
plt.show()
```