Below is a Complete Python Script for Text Classification Using Tensorflow and your own CSV file, along with a sample format for the CSV data.

1. Sample CSV File (data.csv)

Your CSV file should look like this (2 columns: text, label):

text,label

"This movie was fantastic! I loved it.",1

"Terrible movie. It was a waste of time.",0

"Great performances and story.",1

"Awful script and bad acting.",0

"Best film of the year!",1

"I don't recommend it at all.",0

"Amazing experience!",1

"Completely boring and predictable.",0

Save this file as data.csv in the same folder as your Python script.

2. Full Python Code e (text_classifier.py):-

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Dense

# Optional: Suppress TensorFlow info logs
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

```
# Step 1: Load CSV
df = pd.read_csv('data.csv')

# Step 2: Clean Data (drop any missing values)
df.dropna(inplace=True)

# Step 3: Split Data
texts = df['text'].astype(str).tolist()
labels = df['label'].tolist()

x_train_texts, x_test_texts, y_train, y_test = train_test_split(
    texts, labels, test_size=0.2, random_state=42
)

# Step 4: Tokenization and Padding
vocab_size = 1000
maxlen = 50
tokenizer = Tokenizer(num_words=vocab_size, oov_token='<OOV>')
tokenizer.fit_on_texts(x_train_texts)

x_train_seq = tokenizer.texts_to_sequences(x_train_texts)
x_test_seq = tokenizer.texts_to_sequences(x_test_texts)

x_train_pad = pad_sequences(x_train_seq, maxlen=maxlen, padding='post',
truncating='post')
x_test_pad = pad_sequences(x_test_seq, maxlen=maxlen, padding='post',
truncating='post')

# Convert labels to NumPy arrays
y_train = np.array(y_train)
y_test = np.array(y_test)

# Step 5: Build the Model
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=32),
    GlobalAveragePooling1D(),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification
])

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
```

```
)

# Step 6: Train the Model
history = model.fit(
    x_train_pad, y_train,
    epochs=10,
    batch_size=4,
    validation_split=0.2
)

# Step 7: Evaluate the Model
loss, acc = model.evaluate(x_test_pad, y_test)
print(f"Test Accuracy: {acc:.2f}")

# Step 8: Visualize Accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Step 9: Predict on New Samples
sample_texts = ["I really enjoyed the movie!", "It was terrible and boring."]
sample_seq = tokenizer.texts_to_sequences(sample_texts)
sample_pad = pad_sequences(sample_seq, maxlen=maxlen, padding='post')

predictions = model.predict(sample_pad)

for i, text in enumerate(sample_texts):
    sentiment = 'Positive' if predictions[i] > 0.5 else 'Negative'
    print(f"Text: {text} => Prediction: {sentiment} ({predictions[i][0]:.2f})")
```

Output:-

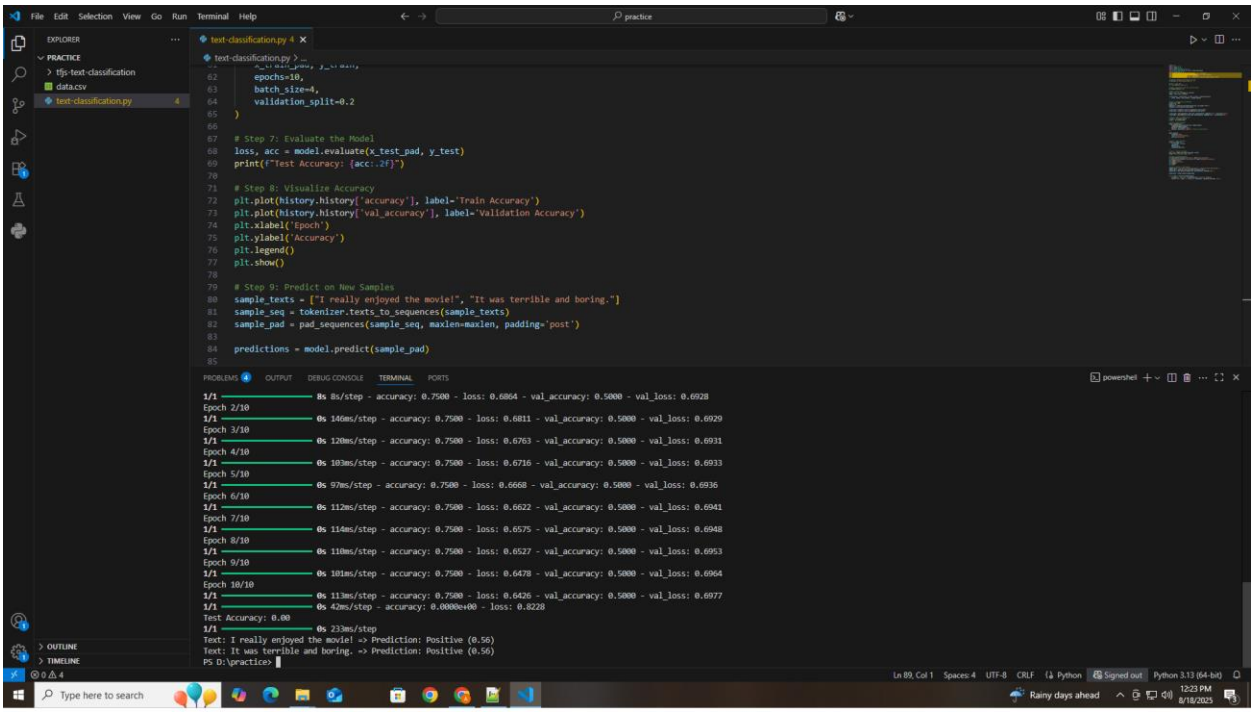
Console will show training/validation accuracy

- Final test accuracy
- A plot of training performance
- Sample predictions like:

Text: I really enjoyed the movie! => Prediction: Positive (0.92)

Text: It was terrible and boring. => Prediction: Negative (0.12)

As shown below.



Step-by-Step Code Explanation

□ Step 0: Import Libraries and Suppress Logs

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Dense

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

- Imports all necessary libraries.
- The `os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'` line hides TensorFlow info/warning logs to clean up the output.

□ Step 1: Load CSV Data

```
df = pd.read_csv('data.csv')
```

- Reads a CSV file named `data.csv` into a pandas DataFrame.
- Assumes the file contains at least two columns: `text` and `label`.

□ Step 2: Clean Data

```
df.dropna(inplace=True)
```

- Removes any rows with missing values (NaNs) to prevent errors during training.

□ Step 3: Split Data into Training and Test Sets

```
texts = df['text'].astype(str).tolist()
labels = df['label'].tolist()

x_train_texts, x_test_texts, y_train, y_test = train_test_split(
    texts, labels, test_size=0.2, random_state=42
)
```

- Extracts the `text` and `label` columns from the DataFrame.
- Converts them to Python lists.
- Uses `train_test_split()` to split into 80% training and 20% test sets.

□ Step 4: Tokenization and Padding

```
vocab_size = 1000
maxlen = 50
tokenizer = Tokenizer(num_words=vocab_size, oov_token='<OOV>')
tokenizer.fit_on_texts(x_train_texts)

x_train_seq = tokenizer.texts_to_sequences(x_train_texts)
x_test_seq = tokenizer.texts_to_sequences(x_test_texts)
```

```
x_train_pad = pad_sequences(x_train_seq, maxlen=maxlen, padding='post',
truncating='post')
x_test_pad = pad_sequences(x_test_seq, maxlen=maxlen, padding='post',
truncating='post')
```

- **Tokenizer**: Converts words to integer sequences. `vocab_size` limits vocabulary to the top 1000 most frequent words.
- `<OOV>` is used to handle out-of-vocabulary words.
- **Padding**: Ensures all sequences are the same length (`maxlen=50`) by padding/truncating at the end.

□ Step 5: Convert Labels to NumPy Arrays

```
y_train = np.array(y_train)
y_test = np.array(y_test)
```

- Keras `model.fit()` expects labels as NumPy arrays or tensors, not plain lists.

□ Step 6: Build the Neural Network

```
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=32),
    GlobalAveragePooling1D(),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification
])
```

- **Embedding Layer**: Turns word indices into dense vectors of fixed size (32).
- **GlobalAveragePooling1D**: Averages the sequence of word vectors into a single vector.
- **Dense Layer**: Fully connected layer with 16 units and ReLU activation.
- **Output Layer**: Single neuron with sigmoid for binary classification (`0` or `1`).

□ Step 7: Compile the Model

```
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

- **Optimizer**: `adam` is a popular choice for deep learning.
- **Loss**: `binary_crossentropy` is appropriate for binary classification.
- **Metric**: Track model accuracy during training.

□ Step 8: Train the Model

```
history = model.fit(
    x_train_pad, y_train,
    epochs=10,
    batch_size=4,
    validation_split=0.2
)
```

- Trains the model using the padded training data and labels.
- `epochs=10`: Number of times the model sees the full dataset.
- `batch_size=4`: Updates weights after every 4 samples.
- `validation_split=0.2`: 20% of training data used to validate the model during training.

□ Step 9: Evaluate the Model

```
loss, acc = model.evaluate(x_test_pad, y_test)
print(f"Test Accuracy: {acc:.2f}")
```

- Tests the model on the unseen test data.
- Prints out the test accuracy.

□ Step 10: Visualize Accuracy

```
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

- Plots training and validation accuracy per epoch.
- Helps identify underfitting or overfitting visually.

□ Step 11: Make Predictions

```
sample_texts = ["I really enjoyed the movie!", "It was terrible and boring."]
sample_seq = tokenizer.texts_to_sequences(sample_texts)
sample_pad = pad_sequences(sample_seq, maxlen=maxlen, padding='post')

predictions = model.predict(sample_pad)
```

```
for i, text in enumerate(sample_texts):
    sentiment = 'Positive' if predictions[i] > 0.5 else 'Negative'
    print(f"Text: {text} => Prediction: {sentiment}
({predictions[i][0]:.2f})")
```

- Demonstrates model prediction on new text samples.
- Texts are tokenized, padded, and passed into the model.
- Output is a probability between 0 and 1.
- If the probability is > 0.5, the sentiment is considered "Positive"; otherwise, "Negative".

□ Summary of Workflow

1. **Data Load & Clean**
2. **Text Preprocessing (Tokenize + Pad)**
3. **Model Definition (Embedding → Pooling → Dense layers)**
4. **Training & Validation**
5. **Evaluation**
6. **Prediction on New Data**