Sentiment Analysis using Deep Learning

Introduction

This code performs sentiment analysis using a deep learning model. Sentiment analysis, also known as opinion mining, is the process of determining the sentiment or emotion expressed in text data. In this project, we analyze sentiment in text reviews.

```
# Importing necessary libraries
import pandas as pd
import re
import string

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

import tensorflow as tf
from tensorflow.keras.layers import Embedding, Bidirectional, LSTM, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Data Preprocessing

Text Cleaning: The text in the "Review Text" column is cleaned by removing emojis and converting the text to lowercase.
Punctuation Removal: Punctuation is removed from the cleaned text.
Tokenization: The text is tokenized for further analysis.
Label Encoding: The "Rating" column is label encoded to represent sentiment labels

```
# Defining a class for text cleaning and preprocessing
class CleanText():
    def __init__(self, clean_pattern=r"[^A-ZĞÜŞİÖÇIa-zğüı'şöç0-9.\"',()]"):
        self.clean_pattern = clean_pattern

    def __call__(self, text):
        if isinstance(text, str):
            docs = [[text]]

        if isinstance(text, list):
            docs = text

        text = [[re.sub(self.clean_pattern, " ", sent) for sent in sents] for sents in docs]

        return text


# Defining a function to remove emojis
def remove_emoji(data):
    emoj = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
        u"\U00002500-\U00002BEF"
        u"\U00002702-\U000027B0"
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        u"\U0001f926-\U0001f937"
        u"\U00010000-\U0010ffff"
        u"\u2640-\u2642"
        u"\u2600-\u2B55"
        u"\u200d"
        u"\u23cf"
        u"\u23e9"
        u"\u231a"
        u"\ufe0f"  # dingbats
        u"\u3030"
                      "]+", re.UNICODE)
    return re.sub(emoj, '', data)

# Defining a function to tokenize text
def tokenize(text):
    text = re.sub(r" +", " ", str(text))
```

```
    text = re.split(r"(\d+|[a-zA-ZğüşıöçĞÜŞİÖÇ]+|\W)", text)
    text = list(filter(lambda x: x != '' and x != ' ', text))
    sent_tokenized = ' '.join(text)
    return sent_tokenized

regex = re.compile('[%s]' % re.escape(string.punctuation))

# Defining a regular expression for punctuation removal
def remove_punct(text):
    text = regex.sub(" ", text)
    return text

clean = CleanText()

# Defining a function to remove punctuation
def label_encode(x):
    if x == 1 or x == 2:
        return 0
    if x == 3:
        return 1
    if x == 5 or x == 4:
        return 2

# Defining a function to label encode sentiment
def label2name(x):
    if x == 0:
        return "Negative"
    if x == 1:
        return "Neutral"
    if x == 2:
        return "Positive"



# Loading the dataset (replace with your dataset path)
df = pd.read_csv("/content/Womens Clothing E-Commerce Reviews.csv")
df.head()
```

	Unnamed: 0	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Divisi Na
0	0	767	33	NaN	Absolutely wonderful - silky and sexy and comf...	4	1	0	Initmat

```
# Label encode the 'Rating' column to convert it to sentiment labels
df["label"] = df["Rating"].apply(lambda x: label_encode(x))
# Map label values to sentiment names
df["label_name"] = df["label"].apply(lambda x: label2name(x))

# Preprocess the 'Review Text' column: lowercase, remove punctuation, and remove emojis
df["Review Text"] = df["Review Text"].apply(str)
df["Review Text"] = df["Review Text"].apply(lambda x: remove_punct(clean(remove_emoji(x).lower())[0][0]))

# Tokenize and pad sequences for model input
tokenizer = Tokenizer()
tokenizer.fit_on_texts(df["Review Text"])
word_index = tokenizer.word_index
sequences = tokenizer.texts_to_sequences(df["Review Text"])
padded_sequences = pad_sequences(sequences, maxlen=100, padding='post', truncating='post')
```

Model Description:

The sentiment analysis deep learning model is designed to classify text reviews into three sentiment categories: "Negative," "Neutral," and "Positive." The model is built using the TensorFlow framework and consists of several layers, including embedding layers, Bidirectional LSTM layers, and dense layers.

Embedding Layer:

Input Dimension: Vocabulary size (number of unique words in the text corpus) + 1
Output Dimension: 128
Input Length: 100
Purpose: The embedding layer is responsible for converting words into dense vectors of fixed size for model input.

Bidirectional LSTM Layers:

Two Bidirectional LSTM layers are used to capture the contextual information in the text. Each LSTM layer has 64 units. The first LSTM layer returns sequences, and the second one does not.

Dense Layers:

The model includes a dense layer with 64 units and a ReLU activation function. The final dense layer has 3 units with a softmax activation function, representing the three sentiment categories ("Negative," "Neutral," and "Positive").

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, df["label"], test_size=0.2, random_state=42)

# Build a deep learning model for sentiment analysis
model = tf.keras.Sequential([
    Embedding(input_dim=len(word_index) + 1, output_dim=128, input_length=100),
    Bidirectional(LSTM(64, return_sequences=True)),
    Bidirectional(LSTM(64)),
    Dense(64, activation='relu'),
    Dense(3, activation='softmax')
])
```

Model Compilation

Optimizer: Adam
Loss Function: Sparse Categorical Cross-Entropy
Metrics: Accuracy

Model Training

The model is trained using the training data with six training epochs.

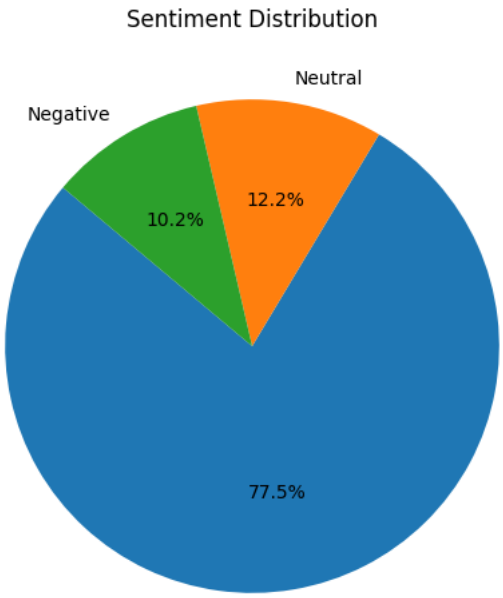
Model Evaluation

The model is evaluated on the testing data to assess its performance. The evaluation includes calculating the test loss and test accuracy.

```
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, np.array(y_train), epochs=6, validation_data=(X_test, np.array(y_test)))

# Evaluate the model on the testing data
test_loss, test_accuracy = model.evaluate(X_test, np.array(y_test))
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
Epoch 1/6
588/588 [==============================] - 187s 303ms/step - loss: 0.5126 - accuracy: 0.7982 - val_loss: 0.4847 - val_accuracy: 0.8(
Epoch 2/6
588/588 [==============================] - 175s 298ms/step - loss: 0.3881 - accuracy: 0.8369 - val_loss: 0.4278 - val_accuracy: 0.8:
Epoch 3/6
588/588 [==============================] - 175s 298ms/step - loss: 0.3316 - accuracy: 0.8605 - val_loss: 0.4817 - val_accuracy: 0.8(
Epoch 4/6
588/588 [==============================] - 174s 296ms/step - loss: 0.2979 - accuracy: 0.8817 - val_loss: 0.4769 - val_accuracy: 0.8:
Epoch 5/6
588/588 [==============================] - 186s 317ms/step - loss: 0.2588 - accuracy: 0.8987 - val_loss: 0.5245 - val_accuracy: 0.8(
Epoch 6/6
588/588 [==============================] - 176s 299ms/step - loss: 0.2253 - accuracy: 0.9169 - val_loss: 0.5519 - val_accuracy: 0.8(
147/147 [==============================] - 11s 76ms/step - loss: 0.5519 - accuracy: 0.8082
Test Loss: 0.5518839359283447
Test Accuracy: 0.8082162737846375
```

```
#Sentiment Distribution
sentiment_counts = df["label_name"].value_counts()
labels = sentiment_counts.index
sizes = sentiment_counts.values

plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.title('Sentiment Distribution')
plt.show()
```




```
plt.figure(figsize=(20, 5))

# Sentiment distribution by age group
sns.countplot(data=df, x='Age', hue='label_name')
plt.title('Sentiment by Age Group')
plt.xticks(rotation=45)

plt.show()
```