Sentiment_Analysis_Using_RNN_in_PyTorch_with_WV

January 29, 2025

1 Load the dataset

[83]:
```python
import pandas as pd
```

[84]:
```python
df=pd.read_csv("/content/IMDB_Small_Dataset.csv")
```

[85]:
```python
df.head(2)
```

[85]:
```
                                              review sentiment
0  Dick Clement and Ian La Frenais have a solid h…  positive
1  When I checked out the review for this film af…  negative
```

2 Check Duplicates

[86]:
```python
df.drop_duplicates(inplace=True)
df.shape
```

[86]:
```
(4999, 2)
```

3 2) Data Preprocessing

4 1)Lower Case

[87]:
```python
df["review"]=df["review"].str.lower()
```

[88]:
```python
# REMOVE URL's.
import re
def remove_urls(text):
    return re.sub(r'http\S+', '', text)
```

[89]:
```python
df["review"] = df["review"].apply(remove_urls)
```

[90]:
```python
#REMOVE PUNCTUATIONS AND EMOJI
import re
```

```
def remove_punctuations(text):
    text=re.sub(r"[^A-Za-z0-9\s]","",text)
    return text
```

[91]:

```
df["review"] = df["review"].apply(remove_punctuations)
```

[92]:

```
#REMOVE HTML
import re

def remove_html(text):
    text=re.sub(r'<.*?>', '', text)
    return text
```

[93]:

```
df["review"] = df["review"].apply(remove_html)
```

[94]:

```
#REMOVE STOPWORDS
import nltk

nltk.download('stopwords')
nltk.download('punkt')
nltk.download('punkt_tab')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

def remove_stopword(text):
    stop_words = stopwords.words('english')  # Specify 'english' for English␣
 ↪stopwords
    temp_text = word_tokenize(text)

    for word in temp_text:
        if word in stop_words:
            text=text.replace(word,"")
    return text
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data…
[nltk_data]   Package punkt_tab is already up-to-date!
```

[95]:

```
df["review"] = df["review"].apply(remove_stopword)
```

[96]:

```
from nltk.stem import PorterStemmer
def Stemming(text):
    ps = PorterStemmer()
    tokens = word_tokenize(text)
    stemmed_words = []
    for token in tokens:
        stemmed_token = ps.stem(token)
        stemmed_words.append(stemmed_token)
    return ' '.join(stemmed_words)
```

[97]:

```
df["review"] = df["review"].apply(Stemming)
```

[98]:

```
df.head(3)
```

[98]:

```
                                              review sentiment
0  dck clement n l fren sold h rte s fr s tv work…  positive
1  ccked revew th flm d wtcd t surprs re peopl gv…  negative
2  nmeli sequel chrtm sri orglli entl summer sri …  negative
```

5 Changing the Target values to categorical value

[99]:

```
df["sentiment"].replace({"positive": 0, "negative": 1}, inplace=True)
df["sentiment"] = df["sentiment"].astype(int)  # Explicitly cast to int
```

```
<ipython-input-99-4e45398487db>:1: FutureWarning: A value is trying to be set on
a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df["sentiment"].replace({"positive": 0, "negative": 1}, inplace=True)
<ipython-input-99-4e45398487db>:1: FutureWarning: Downcasting behavior in
`replace` is deprecated and will be removed in a future version. To retain the
old behavior, explicitly call `result.infer_objects(copy=False)`. To opt-in to
the future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df["sentiment"].replace({"positive": 0, "negative": 1}, inplace=True)
```

[100]:

```
Y=df["sentiment"]
```

6 Text Vectorization

[101]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tf = TfidfVectorizer()
X =tf.fit_transform(df['review']).toarray()
```

[102]:
```
import pandas as pd
from gensim.models import Word2Vec
import numpy as np


# Tokenize the reviews (split into words)
tokenized_reviews = df['review'].apply(lambda x: x.split())

# Train Word2Vec model on tokenized data
model = Word2Vec(sentences=tokenized_reviews, vector_size=100, window=5, min_count=1, workers=4)

# Convert each document into a vector by averaging word embeddings
def get_document_vector(tokens):
    vectors = [model.wv[word] for word in tokens if word in model.wv]
    if len(vectors) == 0:
        return np.zeros(model.vector_size)  # Handle empty sentences
    return np.mean(vectors, axis=0)

# Apply to the tokenized reviews
df['vector'] = tokenized_reviews.apply(get_document_vector)
```

[103]:
```
print(df['vector'][:5])
```

```
0    [-0.19649187, 0.48427805, 0.1041212, -0.129814…
1    [-0.09659416, 0.43620813, -0.041145463, 0.1952…
2    [-0.120750986, 0.40673116, -0.011033958, -0.04…
3    [-0.14329562, 0.45249522, 0.05360219, -0.00329…
4    [-0.0956648, 0.50861186, 0.081866406, 0.001185…
Name: vector, dtype: object
```

[104]:
```
# Convert the list of vectors into a 2D NumPy array
X = np.stack(df['vector'].values)
```

[105]:
```
X.shape
```

[105]:
```
(4999, 100)
```

7 Split the dataset

[106]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.
 ↪20,random_state=24)
```

[107]:
```
X_train.shape
```

[107]:
```
(3999, 100)
```

[108]:
```
shape=X_train.shape
```

[109]:
```
shape[1]
```

[109]:
```
100
```

[110]:
```
X_test.shape
```

[110]:
```
(1000, 100)
```

[111]:
```
type(X_train)
```

[111]:
```
numpy.ndarray
```

[112]:
```
type(Y_train)
```

[112]:
```
pandas.core.series.Series
```

[114]:
```
Y_train = Y_train.to_numpy()
Y_test = Y_test.to_numpy()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-114-31b54da38624> in <cell line: 0>()
----> 1 Y_train = Y_train.to_numpy()
      2 Y_test = Y_test.to_numpy()

AttributeError: 'numpy.ndarray' object has no attribute 'to_numpy'
```

[115]:
```
X_train.ndim
```

[115]:
```
2
```

8 Create Tensor Datasets

[116]:

```
import torch
from torch.utils.data import DataLoader,TensorDataset
```

[117]:

```
train_set = TensorDataset(torch.from_numpy(X_train).float(), torch.
 ↪from_numpy(Y_train).float())
test_set = TensorDataset(torch.from_numpy(X_test).float(), torch.
 ↪from_numpy(Y_test).float())
```

9 Data Loader (Load Data in Batches)

[118]:

```
train_loader = DataLoader(train_set, shuffle=True, batch_size=64)
test_loader = DataLoader(test_set, shuffle=True, batch_size=64)
```

10 RNN

[119]:

```
import torch.nn as nn
import torch.optim as optim
```

11 Setting up the device

[120]:

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

12 RNN

[121]:

```
import torch
import torch.nn as nn
import torch.optim as optim

class Rnn(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers):
        super().__init__()
        self.num_layers = num_layers
        self.hidden_size = hidden_size

        # RNN Layer
        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True)

        # Fully Connected Layer
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        # Initialize hidden state with zeros
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.
 ↪device)

        # RNN forward pass
        out, _ = self.rnn(x, h0)

        # Pass through fully connected layer
        out = self.fc(out[:, -1, :])
        return out
```

13 Hyperparameters

[122]:

```
input_dim = shape[1] # Updated to match TF-IDF feature size
hidden_dim = 128
output_dim = 1  # Binary classification (positive or negative sentiment)
num_layers = 1
num_epochs = 10
batch_size = 64
learning_rate = 0.001
```

14 Initialize model, criterion, and optimizer

[123]:

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = Rnn(input_dim, hidden_dim, output_dim, num_layers).to(device)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

15 Training

[124]:

```
for epoch in range(num_epochs):
    model.train()
    for X_batch, Y_batch in train_loader:
        X_batch, Y_batch = X_batch.to(device), Y_batch.to(device)

        # Add an additional dimension for the sequence length
        X_batch = X_batch.unsqueeze(1)

        outputs = model(X_batch)

        # Apply sigmoid activation to get probabilities
        outputs = torch.sigmoid(outputs.squeeze())

        # Compute the loss
        loss = criterion(outputs, Y_batch)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

```
Epoch [1/10], Loss: 0.6597
Epoch [2/10], Loss: 0.6717
Epoch [3/10], Loss: 0.6515
Epoch [4/10], Loss: 0.6824
Epoch [5/10], Loss: 0.6147
Epoch [6/10], Loss: 0.6926
Epoch [7/10], Loss: 0.6477
Epoch [8/10], Loss: 0.6154
Epoch [9/10], Loss: 0.6918
Epoch [10/10], Loss: 0.7172
```

16 EVALUATION

[125]:

```
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for X_batch, Y_batch in test_loader:
        X_batch, Y_batch = X_batch.to(device), Y_batch.to(device)

        # Add an additional dimension for the sequence length
        X_batch = X_batch.unsqueeze(1)

        outputs = model(X_batch)
        predicted = (torch.sigmoid(outputs.squeeze()) > 0.5).float()
        total += Y_batch.size(0)
        correct += (predicted == Y_batch).sum().item()

    accuracy = correct / total
    print(f'Accuracy: {accuracy * 100:.2f}%')
```

```
Accuracy: 62.80%
```