```
Name : Harshada Mhaske
Div : B
```

Name : Harshada Mhaske Div : B

```
# The IMDB sentiment classification dataset consists of 50,000 movie reviews from I
# The reviews are preprocessed and each one is encoded as a sequence of word indexe
# The words within the reviews are indexed by their overall frequency within the da
# The 50,000 reviews are split into 25,000 for training and 25,000 for testing.
# Text Process word by word at diffrent timestamp ( You may use RNN LSTM GRU )
# convert input text to vector reprent input text
# DOMAIN: Digital content and entertainment industry
# CONTEXT: The objective of this project is to build a text classification model th
# DATA DESCRIPTION: The Dataset of 50,000 movie reviews from IMDB, labelled by sent
# Reviews have been preprocessed, and each review is encoded as a sequence of word
# For convenience, the words are indexed by their frequency in the dataset, meaning
# Use the first 20 words from each review to speed up training, using a max vocabul
# As a convention, "0" does not stand for a specific word, but instead is used to e
# PROJECT OBJECTIVE: Build a sequential NLP classifier which can use input text par
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
#loading imdb data with most frequent 10000 words
from keras.datasets import imdb
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000) # you may ta
```

```
data = np.concatenate((X_train, X_test), axis=0)    # Combines rows
label = np.concatenate((y_train, y_test), axis=0)

# Check shapes
print("X_train shape:", X_train.shape)  # (25000,)
print("X_test shape:", X_test.shape)    # (25000,)
print("y_train shape:", y_train.shape)  # (25000,)
print("y_test shape:", y_test.shape)    # (25000,)

# Print first review and label
print("Review is:", X_train[0])
print("Review is:", y_train[0])
```

```
X_train shape: (25000,)
X_test shape: (25000,)
y_train shape: (25000,)
y_test shape: (25000,)
Review is: [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173,
Review is: 1
```

```
vocab=imdb.get_word_index() # Retrieve the word index file mapping words to indices
print(vocab)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imc
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
{'fawn': 34701, 'tsukino': 52006, 'nunnery': 52007, 'sonja': 16816, 'vani': 63951, 'w
```

```
y_train
```

```
array([1, 0, 0, ..., 0, 1, 0])
```

```
y_test
```

```
array([0, 1, 1, ..., 0, 0, 0])
```

```
# Function to perform relevant sequence adding on the data
# Now it is time to prepare our data. We will vectorize every review and fill it with zer
# That means we fill every review that is shorter than 500 with zeros.
# We do this because the biggest review is nearly that long and every input for our neura
# We also transform the targets into floats.
# sequences is name of method the review less than 10000 we perform padding overthere # b
# VECTORIZE as one cannot feed integers into a NN
# Encoding the integer sequences into a binary matrix - one hot encoder basically
# From integers representing words, at various lengths - to a normalized one hot encoded


def vectorize(sequences, dimension = 10000): # We will vectorize every review and fill it
# Create an all-zero matrix of shape (len(sequences), dimension)
  results = np.zeros((len(sequences), dimension))
  for i, sequence in enumerate(sequences):
    results[i, sequence] = 1
  return results


# Now we split our data into a training and a testing set.
# The training set will contain reviews and the testing set
# # Set a VALIDATION set


# Split the data manually
test_x = data[:10000]
test_y = label[:10000]
train_x = data[10000:]
train_y = label[10000:]

# Check shapes (use .shape without parentheses)
print("test_x shape:", test_x.shape)      # (10000,)
print("test_y shape:", test_y.shape)      # (10000,)
print("train_x shape:", train_x.shape)    # (40000,)
print("train_y shape:", train_y.shape)    # (40000,)

# Unique categories (0 or 1)
```

```
print("Categories:", np.unique(label))

# Total number of unique word indices used in the dataset
print("Number of unique words:", len(np.unique(np.hstack(data))))
```

```
test_x shape: (10000,)
test_y shape: (10000,)
train_x shape: (40000,)
train_y shape: (40000,)
Categories: [0 1]
Number of unique words: 9998
```

```
length = [len(i) for i in data]
print("Average Review length:", np.mean(length))
print("Standard Deviation:", round(np.std(length)))
```

```
Average Review length: 234.75892
Standard Deviation: 173
```

```
# If you look at the data you will realize it has been already pre-processed.
# All words have been mapped to integers and the integers represent the words sorted by t
# This is very common in text analysis to represent a dataset like this.
# So 4 represents the 4th most used word,
# 5 the 5th most used word and so on...
# The integer 1 is reserved for the start marker,
# the integer 2 for an unknown word and 0 for padding.
# Let's look at a single training example:
print("Label:", label[0])
print("Label:", label[1])
print(data[0])
```

```
Label: 1
Label: 0
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5
```

```
# Retrieves a dict mapping words to their index in the IMDB dataset.
index = imdb.get_word_index() # word to index
# Create inverted index from a dictionary with document ids as keys and a list of terms a
reverse_index = dict([(value, key) for (key, value) in index.items()]) # id to word
decoded = " ".join( [reverse_index.get(i - 3, "#") for i in data[0]] )
# The indices are offset by 3 because 0, 1 and 2 are reserved indices for "padding", "sta
print(decoded)
```

```
# this film was just brilliant casting location scenery story direction everyone's re
```

```
from keras.preprocessing.sequence import pad_sequences
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

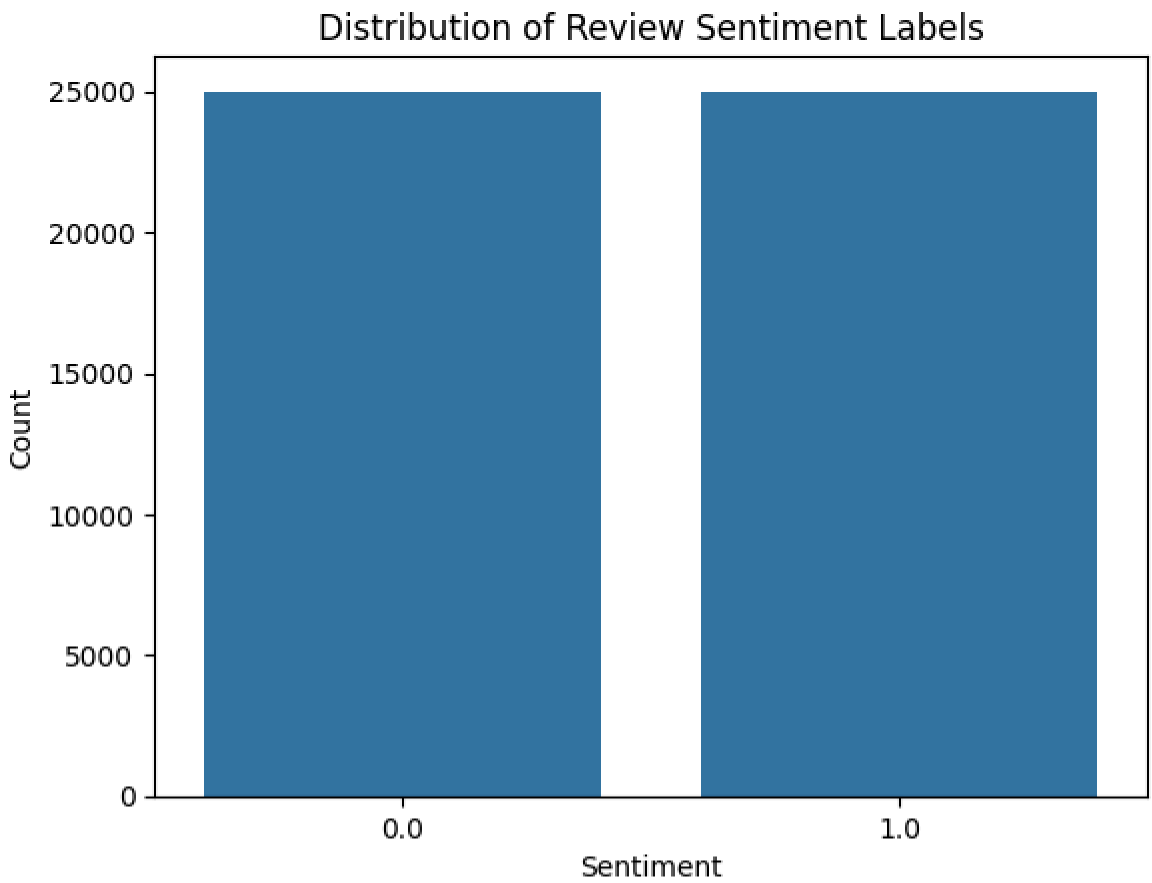
```
import numpy as np

# Vectorization = padding reviews to same length (say 500)
data = pad_sequences(data, maxlen=500)

# Convert labels to float32
label = np.array(label).astype("float32")

# Create DataFrame for labels
labelDF = pd.DataFrame({'label': label})

# Visualize class distribution
sns.countplot(x='label', data=labelDF)
plt.title("Distribution of Review Sentiment Labels")
plt.xlabel("Sentiment")
plt.ylabel("Count")
plt.show()
```




```
from sklearn.model_selection import train_test_split

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(data, label, test_size=0.20, random_s

# Print the shapes of the splits
print("X_train shape:", X_train.shape)  # Should output (40000, 500) after padding
print("X_test shape:", X_test.shape)    # Should output (10000, 500)
```

```
X_train shape: (40000, 500)
X_test shape: (10000, 500)
```

```
# Let's create sequential model
from keras.utils import to_categorical
from keras import models
from keras import layers
model = models.Sequential()
# Input - Layer
# Note that we set the input-shape to 10,000 at the input-layer because our reviews are 1
# The input-layer takes 10,000 as input and outputs it with a shape of 50.
model.add(layers.Dense(50, activation = "relu", input_shape=(10000, )))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarnir
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
from keras import models, layers

# Define the Sequential model
model = models.Sequential()

# Input Layer
model.add(layers.Dense(50, activation="relu", input_shape=(10000,)))  # Input layer with

# Add dropout and dense layers
model.add(layers.Dropout(0.3))  # Dropout with 30% rate to prevent overfitting
model.add(layers.Dense(50, activation="relu"))  # Dense layer with 50 neurons and ReLU ac
model.add(layers.Dropout(0.2))  # Dropout with 20% rate to prevent overfitting
model.add(layers.Dense(50, activation="relu"))  # Another Dense layer with 50 neurons and

# Output Layer
model.add(layers.Dense(1, activation="sigmoid"))  # Output layer with sigmoid activation

# Model summary to view architecture
model.summary()
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 50)	500,050
dropout_2 (Dropout)	(None, 50)	0
dense_5 (Dense)	(None, 50)	2,550
dropout_3 (Dropout)	(None, 50)	0
dense_6 (Dense)	(None, 50)	2,550
dense_7 (Dense)	(None, 1)	51

Total params: 505,201 (1.93 MB)
Trainable params: 505,201 (1.93 MB)

```
# For early stopping
# Stop training when a monitored metric has stopped improving.
# monitor: Quantity to be monitored.
# patience: Number of epochs with no improvement after which training will be stopped.
import tensorflow as tf
import numpy as np
from sklearn.model_selection import train_test_split

# Early stopping callback
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

# Compile the model
# We use the “adam” optimizer, an algorithm that changes the weights and biases during tr
# We also choose binary-crossentropy as loss (because we deal with binary classification)
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)

# Train the model
results = model.fit(
    X_train, y_train,
    epochs=2,
    batch_size=500,
    validation_data=(X_test, y_test),
    callbacks=[callback]
)

# Let's check mean validation accuracy of our model
print("Mean validation accuracy:", np.mean(results.history["val_accuracy"]))
```

```
Epoch 1/2
80/80 ━━━━━━━━━━━━━━━━━━━━ 8s 77ms/step - accuracy: 0.5494 - loss: 0.6686 - val_accur
Epoch 2/2
80/80 ━━━━━━━━━━━━━━━━━━━━ 7s 85ms/step - accuracy: 0.5549 - loss: 0.6637 - val_accur
```

```
Mean validation accuracy: 0.5426500141620636
```

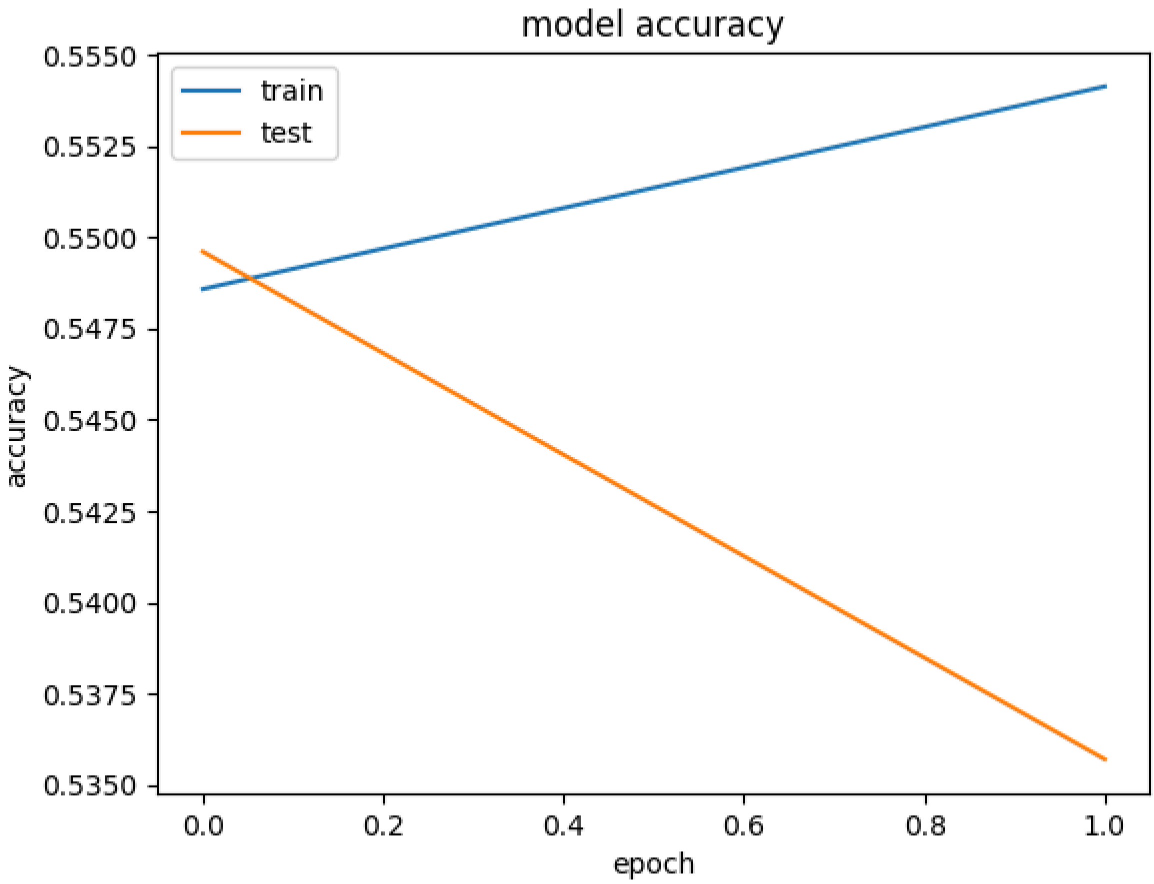
```
# Evaluate the model
score = model.evaluate(X_test, y_test, batch_size=500)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step - accuracy: 0.5331 - loss: 0.6827
Test loss: 0.6809059381484985
Test accuracy: 0.5357000231742859
```

```
# list all data in history
print(results.history.keys())
```

```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

```
# summarize history for accuracy
plt.plot(results.history['accuracy'])
plt.plot(results.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```




```
# summarize history for loss
plt.plot(results.history['loss'])
```

```
plt.plot(results.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```