```
        import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.corpus import movie_reviews
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

def extract_features(review):
    """
    Extract features from the given review for sentiment analysis.

    Parameters:
    - review: The input review text.

    Returns:
    - features: A dictionary containing features extracted from the review.
    """
    features = {}
    words = nltk.word_tokenize(review)
    for word in words:
        features[word] = True
    return features

def train_sentiment_classifier():
    """
    Train a sentiment classifier using the NLTK movie_reviews dataset.

    Returns:
    - classifier: The trained sentiment classifier.
    """
    positive_reviews = [(extract_features(movie_reviews.raw(fileid)), 'pos') for fileid in
    negative_reviews = [(extract_features(movie_reviews.raw(fileid)), 'neg') for fileid in

    # Combine positive and negative reviews
    all_reviews = positive_reviews + negative_reviews

    # Split the dataset into training and testing sets
    train_set, test_set = train_test_split(all_reviews, test_size=0.2, random_state=42)

    # Train the Naive Bayes classifier
    classifier = nltk.NaiveBayesClassifier.train(train_set)


    return classifier, test_set

def evaluate_classifier(classifier, test_set):
    """
    Evaluate the sentiment classifier and calculate accuracy.

    Parameters:
    - classifier: The trained sentiment classifier.
    - test_set: The testing set for evaluation.

    Returns:
    - accuracy: The accuracy of the classifier on the test set.
    """
    predicted_labels = [classifier.classify(features) for (features, label) in test_set]
    true_labels = [label for (features, label) in test_set]

    accuracy = accuracy_score(true_labels, predicted_labels)
```

```
    return accuracy

if __name__ == "__main__":
    # Download necessary NLTK data
    nltk.download('movie_reviews')
    nltk.download('punkt')

    # Train the sentiment classifier
    sentiment_classifier, test_set = train_sentiment_classifier()

    # Evaluate the classifier and calculate accuracy
    accuracy = evaluate_classifier(sentiment_classifier, test_set)

    print(f"Classifier Accuracy: {accuracy * 100:.2f}%")
```

```
  File "<ipython-input-6-dfab672f8b71>", line 20
    features[word] = True
                           ^
IndentationError: expected an indented block after 'for' statement on line 19
```

```
import nltk
from nltk.sentiment import SentimentIntensityAnalyzer
from nltk.corpus import movie_reviews
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

nltk.download('movie_reviews')
nltk.download('punkt')

def extract_features(review):
    """
    Extract features from the given review for sentiment analysis.

    Parameters:
    - review: The input review text.

    Returns:
    - features: A dictionary containing features extracted from the review.
    """
    words = nltk.word_tokenize(review)
    features = {}
    for word in words:
        features[word] = True
    return features

def train_sentiment_classifier():
    """
    Train a sentiment classifier using the NLTK movie_reviews dataset.

    Returns:
    - classifier: The trained sentiment classifier.
    """
    positive_reviews = [(extract_features(movie_reviews.raw(fileid)), 'pos') for fileid in
    negative_reviews = [(extract_features(movie_reviews.raw(fileid)), 'neg') for fileid in

    # Combine positive and negative reviews
```

```
    all_reviews = positive_reviews + negative_reviews

    # Split the dataset into training and testing sets
    train_set, test_set = train_test_split(all_reviews, test_size=0.2, random_state=42)

    # Train the Naive Bayes classifier
    classifier = nltk.NaiveBayesClassifier.train(train_set)

    return classifier, test_set

def evaluate_classifier(classifier, test_set):
    """
    Evaluate the sentiment classifier and calculate accuracy.

    Parameters:
    - classifier: The trained sentiment classifier.
    - test_set: The testing set for evaluation.

    Returns:
    - accuracy: The accuracy of the classifier on the test set.
    """
    predicted_labels = [classifier.classify(features) for (features, label) in test_set]
    true_labels = [label for (features, label) in test_set]

    accuracy = accuracy_score(true_labels, predicted_labels)
    return accuracy

if __name__ == "__main__":
    # Train the sentiment classifier
    sentiment_classifier, test_set = train_sentiment_classifier()

    # Evaluate the classifier and calculate accuracy
    accuracy = evaluate_classifier(sentiment_classifier, test_set)

    print(f"Classifier Accuracy: {accuracy * 100:.2f}%")
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Unzipping corpora/movie_reviews.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Classifier Accuracy: 69.75%
```

Double-click (or enter) to edit

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from nltk.corpus import movie_reviews
import nltk
nltk.download('movie_reviews')

# Load the movie_reviews dataset
reviews = [(movie_reviews.raw(fileid), category) for category in movie_reviews.categories()
texts, labels = zip(*reviews)
```

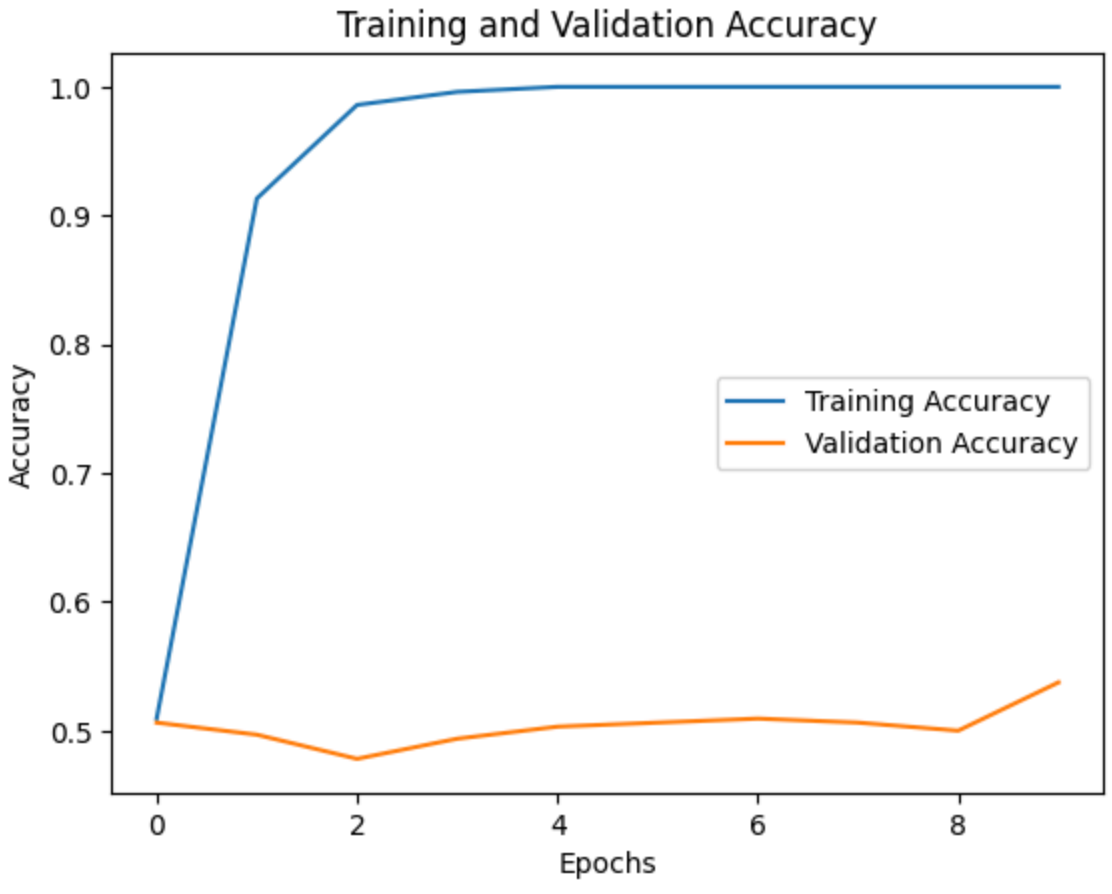
```
# Tokenization and Padding
maxlen = 1000  # Limit the review length to 1000 words
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
total_words = len(tokenizer.word_index) + 1

# Convert text to sequences
sequences = tokenizer.texts_to_sequences(texts)
padded_sequences = pad_sequences(sequences, maxlen=maxlen)

# Convert labels to binary (1: positive, 0: negative)
binary_labels = np.array([1 if label == 'pos' else 0 for label in labels])

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, binary_labels, test_s

# Build the RNN model
model = Sequential()
model.add(Embedding(total_words, 32, input_length=maxlen))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=5, validation_split=0.2)

# Evaluate the model on the test set
predictions = model.predict(X_test)
binary_predictions = np.round(predictions).flatten().astype(int)
accuracy = accuracy_score(y_test, binary_predictions)

print(f"Classifier Accuracy: {accuracy * 100:.2f}%")
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
Epoch 1/5
40/40 [==============================] - 12s 260ms/step - loss: 0.7003 - accurac
Epoch 2/5
40/40 [==============================] - 10s 262ms/step - loss: 0.5920 - accurac
Epoch 3/5
40/40 [==============================] - 11s 264ms/step - loss: 0.3873 - accurac
Epoch 4/5
40/40 [==============================] - 9s 232ms/step - loss: 0.1509 - accuracy
Epoch 5/5
40/40 [==============================] - 10s 250ms/step - loss: 0.0558 - accurac
13/13 [==============================] - 1s 44ms/step
Classifier Accuracy: 60.00%
```

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from tensorflow.keras.preprocessing.text import Tokenizer
import matplotlib.pyplot as plt
import nltk
```

```
nltk.download('movie_reviews')

# Load the movie_reviews dataset
reviews = [(movie_reviews.raw(fileid), category) for category in movie_reviews.categories()
texts, labels = zip(*reviews)

# Tokenization and Padding
maxlen = 1000  # Limit the review length to 1000 words
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
total_words = len(tokenizer.word_index) + 1

# Convert text to sequences
sequences = tokenizer.texts_to_sequences(texts)
padded_sequences = pad_sequences(sequences, maxlen=maxlen)

# Convert labels to binary (1: positive, 0: negative)
binary_labels = np.array([1 if label == 'pos' else 0 for label in labels])

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, binary_labels, test_s

# Build the RNN model
model = Sequential()
model.add(Embedding(total_words, 32, input_length=maxlen))
model.add(SimpleRNN(32))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, validation_split=0.2)

# Evaluate the model on the test set
predictions = model.predict(X_test)
binary_predictions = np.round(predictions).flatten().astype(int)
accuracy = accuracy_score(y_test, binary_predictions)

print(f"Classifier Accuracy: {accuracy * 100:.2f}%")

# Plot training history
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
Epoch 1/10
40/40 [==============================] - 14s 334ms/step - loss: 0.6981 - accurac
Epoch 2/10
40/40 [==============================] - 21s 525ms/step - loss: 0.5247 - accurac
Epoch 3/10
40/40 [==============================] - 12s 281ms/step - loss: 0.2732 - accurac
Epoch 4/10
40/40 [==============================] - 10s 256ms/step - loss: 0.1442 - accurac
Epoch 5/10
40/40 [==============================] - 10s 247ms/step - loss: 0.0522 - accurac
Epoch 6/10
40/40 [==============================] - 10s 252ms/step - loss: 0.0270 - accurac
Epoch 7/10
40/40 [==============================] - 9s 219ms/step - loss: 0.0161 - accuracy
Epoch 8/10
40/40 [==============================] - 10s 250ms/step - loss: 0.0113 - accurac
Epoch 9/10
40/40 [==============================] - 10s 257ms/step - loss: 0.0085 - accurac
Epoch 10/10
40/40 [==============================] - 10s 257ms/step - loss: 0.0067 - accurac
13/13 [==============================] - 1s 40ms/step
Classifier Accuracy: 54.50%
```



LSTM

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
from nltk.corpus import movie_reviews
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split

# Download NLTK resources if not already downloaded
import nltk
nltk.download('movie_reviews')
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

# Load Movie Reviews dataset from NLTK
neg_reviews = [" ".join(movie_reviews.words(fileid)) for fileid in movie_reviews.fileids('n
pos_reviews = [" ".join(movie_reviews.words(fileid)) for fileid in movie_reviews.fileids('p

texts = neg_reviews + pos_reviews
labels = [0] * len(neg_reviews) + [1] * len(pos_reviews)

# Tokenization, lemmatization, and stopword removal
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(word.lower()) for word in tokens if word.isalnum()]
    tokens = [word for word in tokens if word not in stop_words]
    return ' '.join(tokens)

texts = [preprocess_text(text) for text in texts]

# Convert labels to numpy array
labels = np.array(labels)

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(texts, labels, test_size=0.2, random_st

# Tokenize and pad sequences
max_len = 100
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=5000, oov_token='<OOV>')
tokenizer.fit_on_texts(x_train)
x_train = tokenizer.texts_to_sequences(x_train)
x_test = tokenizer.texts_to_sequences(x_test)
x_train = pad_sequences(x_train, maxlen=max_len, padding='post', truncating='post')
x_test = pad_sequences(x_test, maxlen=max_len, padding='post', truncating='post')

# LSTM Model
embedding_dim = 16
lstm_units = 64
```

```
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=embedding_dim, inpu
model.add(LSTM(units=lstm_units))
model.add(Dense(units=1, activation='sigmoid'))

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

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")

# Make predictions (for demonstration purposes, adjust as needed)
sample_review = x_test[0].reshape(1, -1)  # Take the first review as an example
prediction = model.predict(sample_review)
print("Predicted Probability:", prediction[0][0])
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
Epoch 1/10
40/40 [==============================] - 4s 61ms/step - loss: 0.6936 - accuracy:
Epoch 2/10
40/40 [==============================] - 2s 43ms/step - loss: 0.6847 - accuracy:
Epoch 3/10
40/40 [==============================] - 3s 73ms/step - loss: 0.6495 - accuracy:
Epoch 4/10
40/40 [==============================] - 2s 45ms/step - loss: 0.4895 - accuracy:
Epoch 5/10
40/40 [==============================] - 2s 44ms/step - loss: 0.2484 - accuracy:
Epoch 6/10
40/40 [==============================] - 2s 45ms/step - loss: 0.1592 - accuracy:
Epoch 7/10
40/40 [==============================] - 2s 45ms/step - loss: 0.0814 - accuracy:
Epoch 8/10
40/40 [==============================] - 2s 45ms/step - loss: 0.0245 - accuracy:
Epoch 9/10
40/40 [==============================] - 2s 58ms/step - loss: 0.0177 - accuracy:
Epoch 10/10
40/40 [==============================] - 2s 61ms/step - loss: 0.0089 - accuracy:
13/13 [==============================] - 1s 14ms/step - loss: 1.3877 - accuracy:
Test Loss: 1.387650966644287, Test Accuracy: 0.637499988079071
1/1 [==============================] - 0s 421ms/step
Predicted Probability: 0.98010415
```

SVM 1

```
import nltk
import numpy as np
```

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from nltk.corpus import movie_reviews
import random

nltk.download('movie_reviews')

# Load Movie Reviews dataset from NLTK
neg_reviews = [' '.join(movie_reviews.words(fileid)) for fileid in movie_reviews.fileids('n
pos_reviews = [' '.join(movie_reviews.words(fileid)) for fileid in movie_reviews.fileids('p

texts = neg_reviews + pos_reviews
labels = [0] * len(neg_reviews) + [1] * len(pos_reviews)  # 0 for negative, 1 for positive

# Split the data into training and testing sets
train_texts, test_texts, train_labels, test_labels = train_test_split(texts, labels, test_s

# TF-IDF Vectorization
vectorizer = TfidfVectorizer()
train_features = vectorizer.fit_transform(train_texts)
test_features = vectorizer.transform(test_texts)

# SVM Model
svm_classifier = SVC(kernel='linear', C=1.0)
svm_classifier.fit(train_features, train_labels)

# Predictions
predictions = svm_classifier.predict(test_features)

# Evaluate the model
accuracy = accuracy_score(test_labels, predictions)
print(f"Overall Test Accuracy: {accuracy}")

# Print individual accuracies for 10 random intervals
num_samples = len(test_labels)
interval_size = num_samples // 10

for i in range(10):
    start_idx = i * interval_size
    end_idx = (i + 1) * interval_size
    interval_predictions = svm_classifier.predict(test_features[start_idx:end_idx])
    interval_accuracy = accuracy_score(test_labels[start_idx:end_idx], interval_predictions
    print(f"Interval {i + 1} Accuracy: {interval_accuracy}")
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
Overall Test Accuracy: 0.8375
Interval 1 Accuracy: 0.8
Interval 2 Accuracy: 0.875
Interval 3 Accuracy: 0.875
Interval 4 Accuracy: 0.725
Interval 5 Accuracy: 0.9
Interval 6 Accuracy: 0.9
Interval 7 Accuracy: 0.75
Interval 8 Accuracy: 0.8
Interval 9 Accuracy: 0.85
```

```
    Interval 10 Accuracy: 0.9
```

```
accuracy_values = [0.4563, 0.5531, 0.5750, 0.5813, 0.5562, 0.5719, 0.5906, 0.5625, 0.5938,

# Calculate the average
average_accuracy = sum(accuracy_values) / len(accuracy_values)

# Print the result
print(f"Average Accuracy: {average_accuracy}")
```

```
Average Accuracy: 0.56313
```

CNN

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from nltk.corpus import movie_reviews
import nltk
nltk.download('movie_reviews')

# Load Movie Reviews dataset from NLTK
neg_reviews = [' '.join(movie_reviews.words(fileid)) for fileid in movie_reviews.fileids('n
pos_reviews = [' '.join(movie_reviews.words(fileid)) for fileid in movie_reviews.fileids('p

texts = neg_reviews + pos_reviews
labels = [0] * len(neg_reviews) + [1] * len(pos_reviews)  # 0 for negative, 1 for positive

# Tokenize and pad sequences
max_len = 100
tokenizer = Tokenizer()
tokenizer.fit_on_texts(texts)
vocab_size = len(tokenizer.word_index) + 1

sequences = tokenizer.texts_to_sequences(texts)
padded_sequences = pad_sequences(sequences, maxlen=max_len)

# Convert labels to binary form
encoder = LabelEncoder()
encoded_labels = np.array(encoder.fit_transform(labels))

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(padded_sequences, encoded_labels, test_

# Build CNN Model
embedding_dim = 50
filters = 250
kernel_size = 3

model = Sequential()
model.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_len))
model.add(Conv1D(filters=filters, kernel_size=kernel_size, activation='relu'))
model.add(GlobalMaxPooling1D())
```

```
model.add(Dense(units=1, activation='sigmoid'))

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

# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print(f"Test Loss: {loss}, Test Accuracy: {accuracy}")
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
Epoch 1/10
20/20 [==============================] - 2s 72ms/step - loss: 0.6925 - accuracy:
Epoch 2/10
20/20 [==============================] - 1s 63ms/step - loss: 0.6715 - accuracy:
Epoch 3/10
20/20 [==============================] - 2s 81ms/step - loss: 0.6483 - accuracy:
Epoch 4/10
20/20 [==============================] - 2s 93ms/step - loss: 0.6181 - accuracy:
Epoch 5/10
20/20 [==============================] - 1s 63ms/step - loss: 0.5682 - accuracy:
Epoch 6/10
20/20 [==============================] - 1s 63ms/step - loss: 0.4860 - accuracy:
Epoch 7/10
20/20 [==============================] - 1s 63ms/step - loss: 0.3655 - accuracy:
Epoch 8/10
20/20 [==============================] - 1s 63ms/step - loss: 0.2213 - accuracy:
Epoch 9/10
20/20 [==============================] - 1s 67ms/step - loss: 0.1072 - accuracy:
Epoch 10/10
20/20 [==============================] - 1s 64ms/step - loss: 0.0449 - accuracy:
13/13 [==============================] - 1s 7ms/step - loss: 0.6183 - accuracy:
Test Loss: 0.618305504322052, Test Accuracy: 0.6924999952316284
```

KNN

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
from nltk.corpus import movie_reviews
import nltk
nltk.download('movie_reviews')

# Load Movie Reviews dataset from NLTK
neg_reviews = [' '.join(movie_reviews.words(fileid)) for fileid in movie_reviews.fileids('n
pos_reviews = [' '.join(movie_reviews.words(fileid)) for fileid in movie_reviews.fileids('p

texts = neg_reviews + pos_reviews
labels = [0] * len(neg_reviews) + [1] * len(pos_reviews)  # 0 for negative, 1 for positive

# Split the data into training and testing sets
```

```
train_texts, test_texts, train_labels, test_labels = train_test_split(texts, labels, test_s

# TF-IDF Vectorization
vectorizer = TfidfVectorizer()
train_features = vectorizer.fit_transform(train_texts)
test_features = vectorizer.transform(test_texts)

# k-NN Model
knn_classifier = KNeighborsClassifier(n_neighbors=5)  # You can experiment with different v
knn_classifier.fit(train_features, train_labels)

# Print accuracy values for 10 intervals randomly
for i in range(10):
    start_idx = np.random.randint(0,  test_features.shape[0] - 100)  # Adjust the interval
    end_idx = start_idx + 100
    interval_predictions = knn_classifier.predict(test_features[start_idx:end_idx])
    interval_accuracy = accuracy_score(test_labels[start_idx:end_idx], interval_predictions
    print(f"Interval {i + 1} Accuracy: {interval_accuracy}")
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
Interval 1 Accuracy: 0.65
Interval 2 Accuracy: 0.58
Interval 3 Accuracy: 0.57
Interval 4 Accuracy: 0.6
Interval 5 Accuracy: 0.58
Interval 6 Accuracy: 0.58
Interval 7 Accuracy: 0.59
Interval 8 Accuracy: 0.63
Interval 9 Accuracy: 0.52
Interval 10 Accuracy: 0.52
```