Project Program Book

Name of the student: Kota MadhaviLatha

Name of the college: K L University

Department: MCA

Specialization: ARTIFICAL INTELLIGENCE

1. Sentiment Analysis of Movie Review

Description:

Build a sentiment analysis model to classify movie reviews as positive or negative.

About Dataset

IMDB dataset having 50K movie reviews for natural language processing or Text analytics.
This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. We provide a set of 25,000 highly polar movie reviews for training and 25,000 for testing. So, predict the number of positive and negative reviews using either classification or deep learning algorithms.

Large Movie Review Dataset

This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. We provide a set of 25,000 highly polar movie reviews for training, and 25,000 for testing. There is additional unlabeled data for use as well. Raw text and already processed bag of words formats are provided. See the README file contained in the release for more details.

Large Movie Review Dataset v1.0

When using this dataset, please cite our ACL 2011 paper [bib].

Contact

For comments or questions on the dataset please contact Andrew Maas. As you publish papers using the dataset please notify us so we can post a link on this page.

Publications Using the Dataset

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word

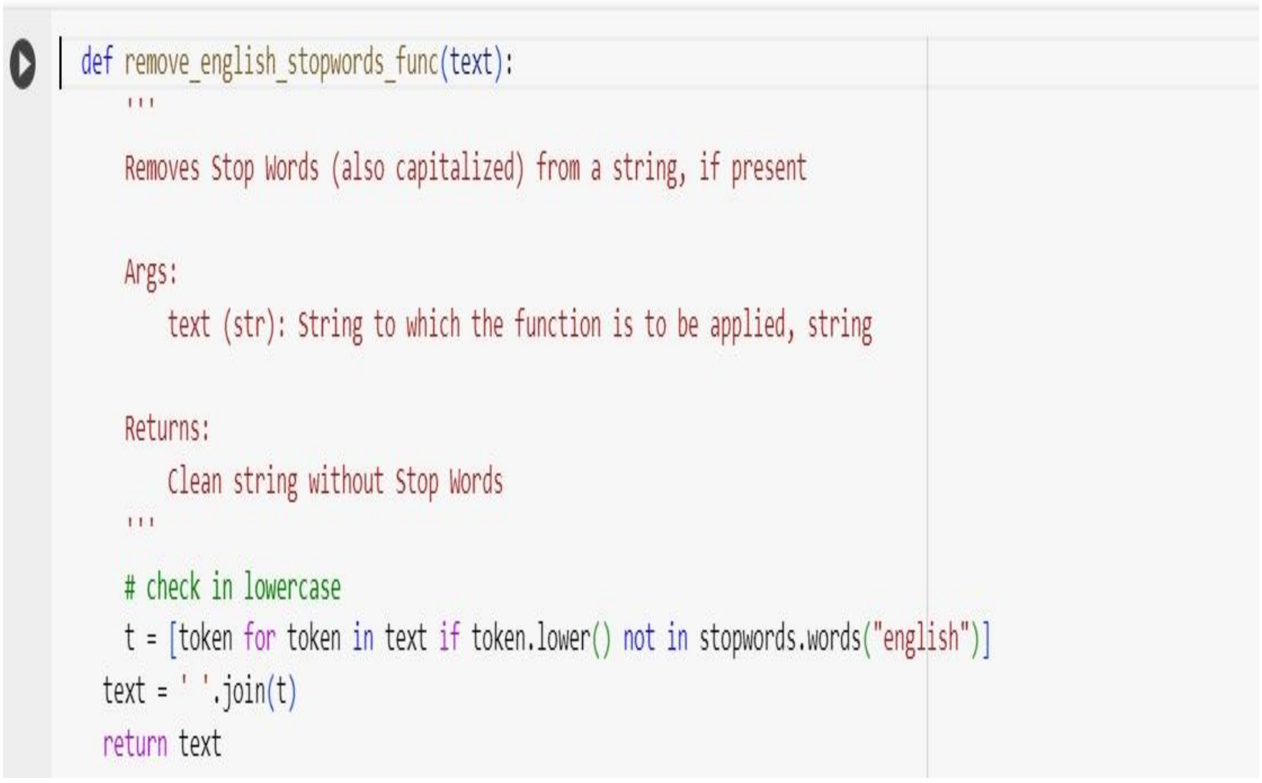
Vectors for Sentiment Analysis. *The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).*

Data Preprocessing: Cleaning Text Data

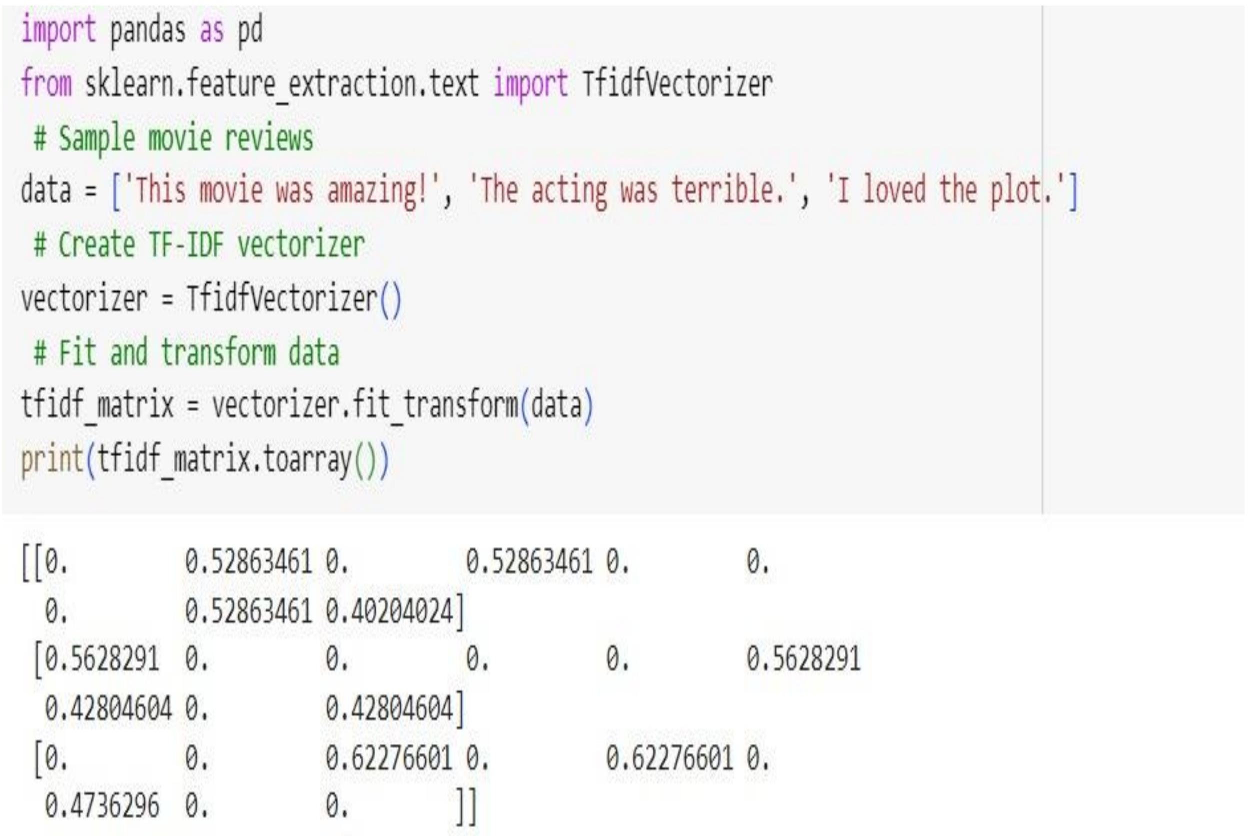
Understanding the Steps

Before we dive into the code, let's clarify the steps involved:

1. **Import necessary libraries:** We'll use NLTK for stopwords and tokenization.
2. **Load the text data:** This can be from a file, database, or any other source.
3. **Remove punctuation:** We'll use regular expressions for this.
4. **Tokenization:** Breaking text into individual words or tokens.
5. **Remove stopwords:** Filter out common words that don't carry significant meaning.



```
def remove_english_stopwords_func(text):
    '''
    Removes Stop Words (also capitalized) from a string, if present

    Args:
        text (str): String to which the function is to be applied, string

    Returns:
        Clean string without Stop Words
    '''
    # check in lowercase
    t = [token for token in text if token.lower() not in stopwords.words("english")]
    text = ' '.join(t)
    return text
```



```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
 # Sample movie reviews
data = ['This movie was amazing!', 'The acting was terrible.', 'I loved the plot.']
 # Create TF-IDF vectorizer
vectorizer = TfidfVectorizer()
 # Fit and transform data
tfidf_matrix = vectorizer.fit_transform(data)
print(tfidf_matrix.toarray())
```

```
[[0.         0.52863461 0.         0.52863461 0.         0.
  0.         0.52863461 0.40204024]
 [0.5628291  0.         0.         0.         0.         0.5628291
  0.42804604 0.         0.42804604]
 [0.         0.         0.62276601 0.         0.62276601 0.
  0.4736296  0.         0.        ]]
```

Tokenization:

Tokenisation is a technique for breaking down a piece of text into small units, called tokens. A token may be a word, part of a word or just characters like punctuation.

Tokenisation can therefore be roughly divided into three groups:

- Word Tokenization
- Character Tokenization and
- Partial Word Tokenization (n-gram characters)

In the following I will present two tokenizers:

- Word Tokenizer
- Sentence Tokenizer

Of course there are some more. Find the one on the NLTK Homepage which fits best to your data or to your problem solution.

text_for_tokenization = \

"Hi my name is Hemalatha. \

I am an Student. \

Currently I am pursuing my MCA in KL University about specilizationin Artifical Intelligence , more specifically about the Pre-Processing Steps."

text_for_tokenization.

Feature Extraction for Movie Reviews: TF-IDF and Word Embeddings:

Converting text data, like movie reviews, into numerical features is a crucial step in Natural Language Processing (NLP) for tasks such as sentiment analysis, recommendation systems, and topic modeling. This process is often referred to as feature extraction.

Popular Methods:

TF-IDF (Term Frequency-Inverse Document Frequency)

- How it works:

    - Calculates the importance of a word in a document relative to the entire corpus.
    - TF: Measures the frequency of a term in a document.
    - IDF: Downweights terms that appear frequently across documents.
- Advantages: Simple to implement, captures word importance based on frequency.
- Disadvantages: Ignores semantic and syntactic information, high dimensionality.

Word Embeddings

- How it works:
    - Represents words as dense vectors in a continuous space.
    - Captures semantic and syntactic relationships between words.
    - Popular methods: Word2Vec, GloVe, FastText.
- Advantages: Captures semantic and syntactic information, lower dimensionality compared to TF-IDF.
- Disadvantages: Requires large amounts of data for training, might not capture specific domain knowledge.

Implementation Steps

1. Data Preprocessing:
    - Clean the text data by removing stop words, punctuation, and handling stemming or lemmatization.
    - Tokenize the text into individual words.
2. Feature Extraction:
    - TF-IDF:
        - Use a TF-IDF vectorizer to convert text data into numerical features.
        - Experiment with different parameters like n-grams (unigrams, bigrams, trigrams) to improve performance.
    - Word Embeddings:

        - Load pre-trained word embeddings (e.g., from Gensim or spaCy).
        - Convert words in each review into their corresponding vectors.
        - Average or sum the word vectors to obtain a document-level representation.

3. Feature Transformation (Optional):
    - Dimensionality reduction techniques like Principal Component Analysis (PCA) or Truncated Singular Value Decomposition (TSVD) can be applied to reduce the feature space.

```
import pandas as pd
 from sklearn.feature_extraction.text import TfidfVectorizer
 # Sample movie reviews
data = ['This movie was amazing!', 'The acting was terrible.', 'I loved the plot.']
 # Create TF-IDF vectorizer
 vectorizer = TfidfVectorizer()
 # Fit and transform data
 tfidf_matrix = vectorizer.fit_transform(data)
 print(tfidf_matrix.toarray())
```

Model Building for Movie Reviews:

We aim to build a model that can accurately predict the sentiment of a movie review (positive or negative) based on its text content.

Model Selection: Logistic Regression vs. Neural Network

- Logistic Regression:
    - Simpler model, interpretable coefficients.
    - Suitable for smaller datasets and when computational resources are limited.
    - Effective for capturing linear relationships between features and the target variable.
- Neural Network:

    - More complex model, capable of learning non-linear relationships.
    - Often outperforms logistic regression on large datasets and complex problems.
    - Can capture intricate patterns in text data.

Model Building Steps:

```
from sklearn.feature_extraction.text


import TfidfVectorizer
 from sklearn.linear_model
 import LogisticRegression
 from sklearn.metrics
 import accuracy_score, confusion_matrix
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, LSTM, Dense
```

Load and preprocess data:

- Import your dataset containing movie reviews and their corresponding sentiment labels (positive or negative).
- Clean the text data by removing stop words, punctuation, and handling stemming or lemmatization.

Feature Extraction:

- Logistic Regression:

    - Convert text data into numerical features using TF-IDF.
- Neural Network:
    - Tokenize the text data into sequences of integers.
    - Pad sequences to a fixed length.

Model Training:

- Logistic Regression:
    - Split data into training and testing sets.
    - Create a Logistic Regression model and fit it to the training data.
- Neural Network:
    - Create an Embedding layer to represent words as dense vectors.
    - Add LSTM layers to capture sequential information.
    - Add a Dense layer with sigmoid activation for classification.
    - Compile the model with appropriate loss function (e.g., binary crossentropy) and optimizer.
    - Train the model on the training data.

Model Evaluation:

- Calculate accuracy, precision, recall, and F1-score on the testing set.
- Analyze confusion matrix to understand model performance.

Code Example (Logistic Regression):

```
# Assuming you have a DataFrame named 'df' with columns 'text' and
'sentiment'
 X = df['text']
 y = df['sentiment']
# Create TF-IDF features
 vectorizer = TfidfVectorizer()
 X = vectorizer.fit_transform(X)
# Split data
```

```
 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create and train Logistic Regression model

model = LogisticRegression()

model.fit(X_train, y_train)

# Make predictions and evaluate

 y_pred = model.predict(X_test)

 accuracy = accuracy_score(y_test, y_pred)

 print("Accuracy:",

accuracy)
```

Code Example (Neural Network):

```
# Assuming you have tokenized and padded text data

# Create the model

 model = Sequential([

Embedding(num_words, embedding_dim, input_length=max_length),
LSTM(64),

Dense(1, activation='sigmoid')

 ])

 # Compile the model

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

# Train the model

model.fit(X_train,   y_train, epochs=10, batch_size=32,
validation_data=(X_test,   y_test))
```

Model Selection and Improvement:

- Experiment with different hyperparameters for both models.
- Consider using techniques like cross-validation for robust evaluation.
- Explore more complex neural network architectures (e.g., CNN, RNN, BERT) for better performance.

- Analyze the errors made by the model to identify areas for improvement.

Evaluating Your Model: Accuracy, Precision, Recall, and F1-Score:

Before diving into the code, let's clarify the metrics:

- Accuracy: Overall correctness of the model (correct predictions / total predictions).
- Precision: Ability of the model to correctly predict positive cases (true positives / (true positives + false positives)).
- Recall: Ability of the model to identify all actual positive cases (true positives / (true positives + false negatives)).
- F1-score: Harmonic mean of precision and recall, providing a balance between the two.

Python Libraries:

We'll primarily use sklearn for this task.

```
# Assuming y_true is the ground truth and y_pred is the model's predictions
accuracy = accuracy_score(y_true, y_pred)
 precision = precision_score(y_true, y_pred)
 recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
print("Accuracy:",   accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

Additional Considerations:

- Confusion Matrix: This can provide a deeper understanding of model performance.

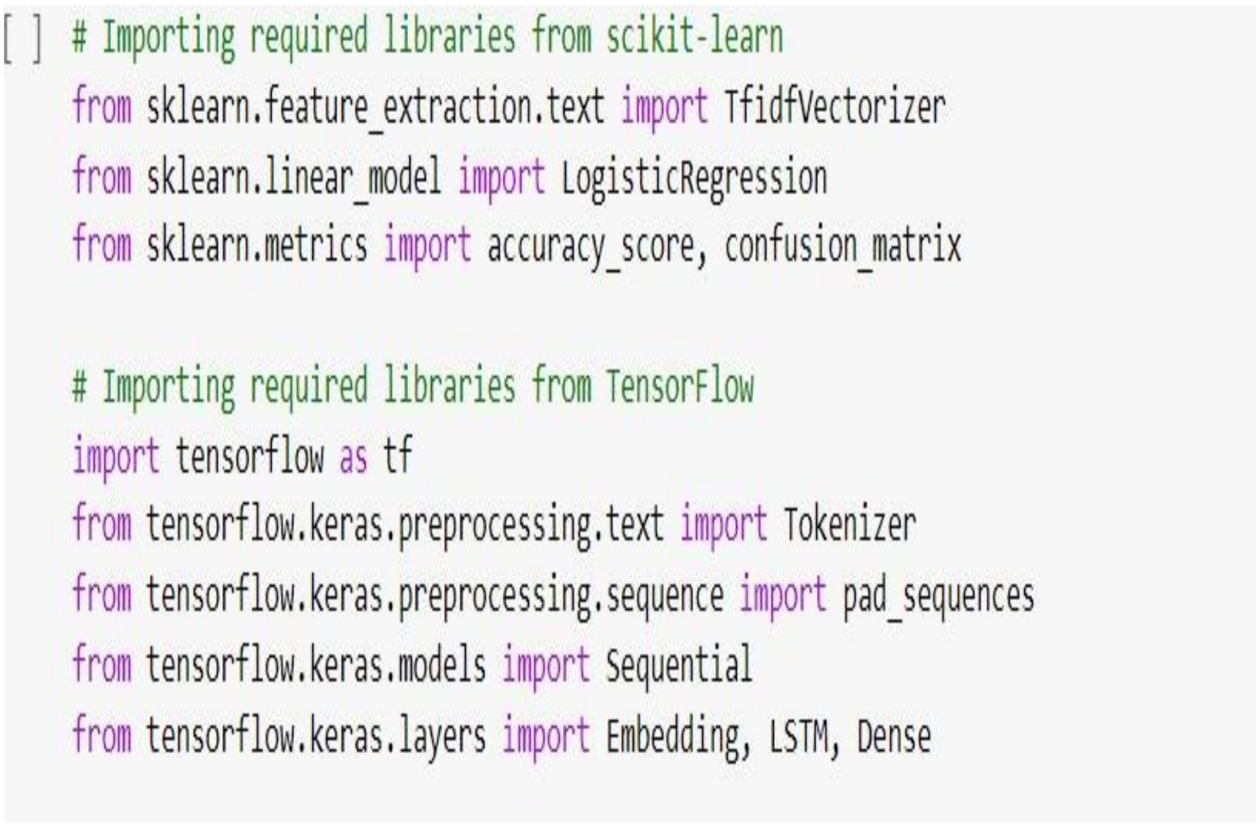
```
from sklearn.metrics
import confusion_matrix
cm = confusion_matrix(y_true, y_pred)
```

print(cm)

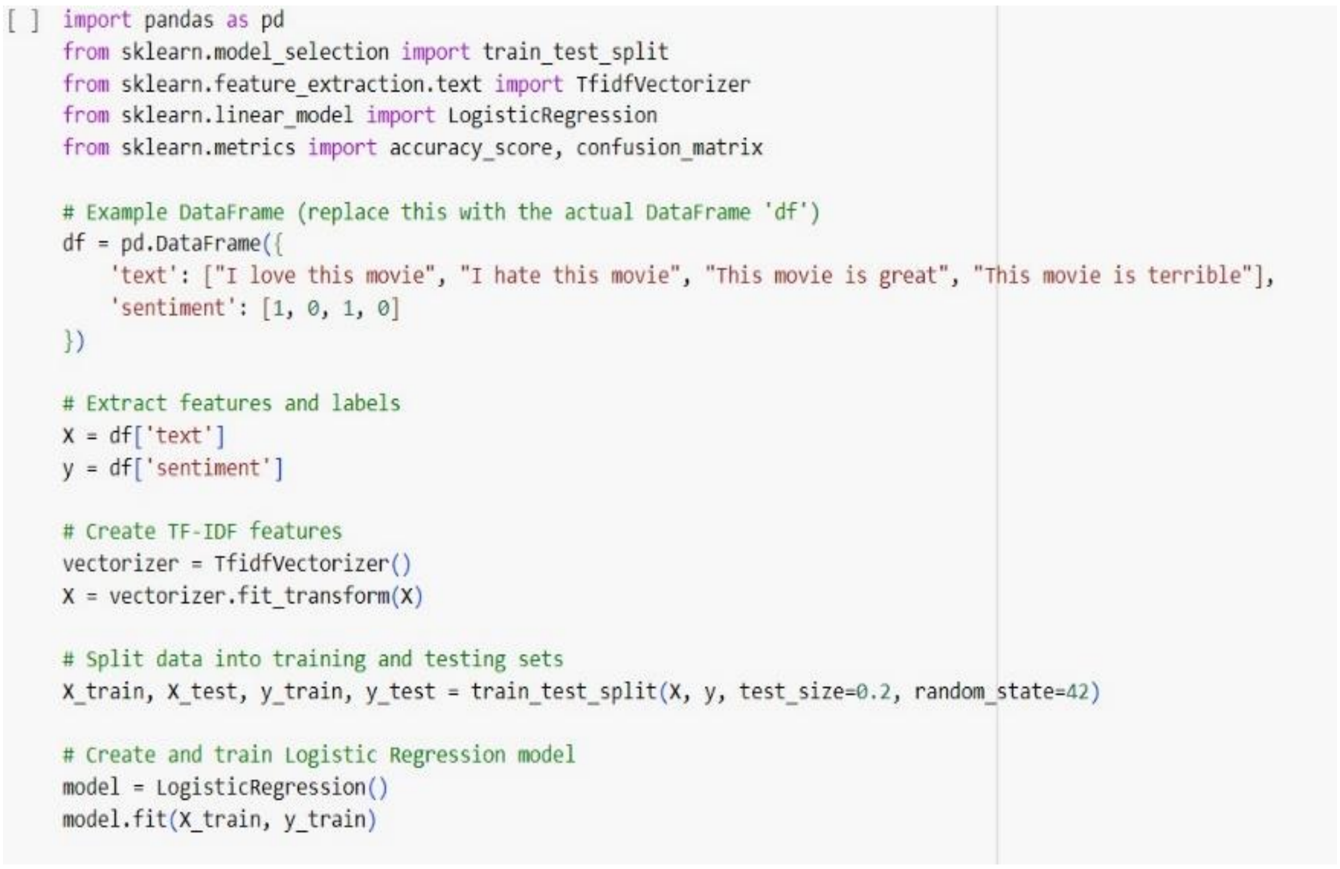
. Imbalanced Datasets: If your dataset is imbalanced, accuracy might not be a reliable metric. Precision, recall, and F1-score can provide a better picture.

. ROC Curve and AUC: For more complex evaluation, consider plotting the ROC curve and calculating the AUC (Area Under the Curve).

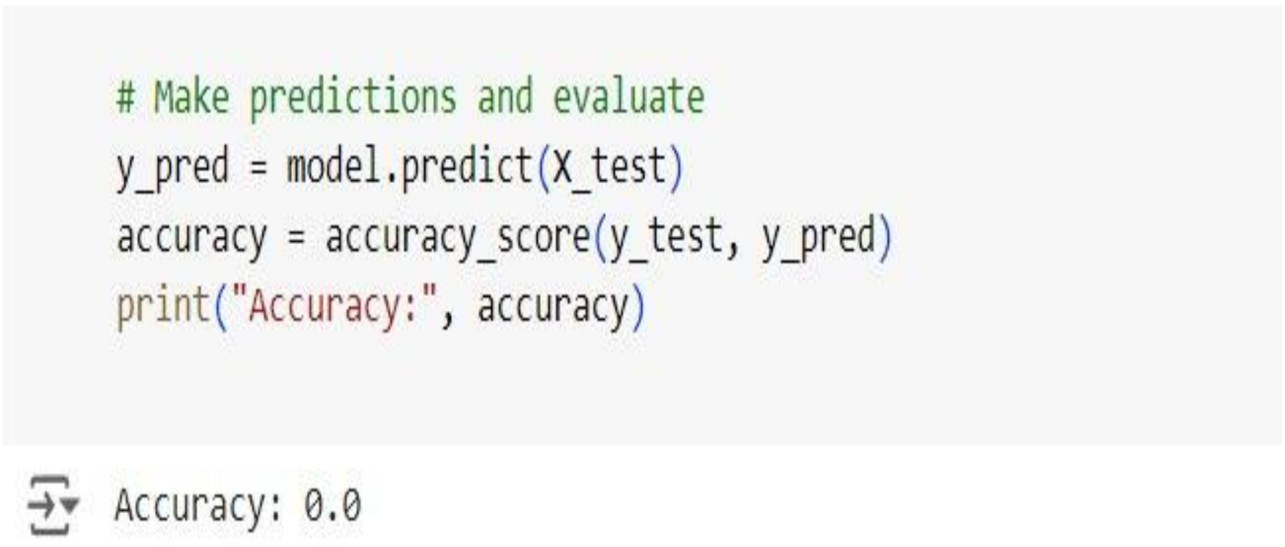
Example with TensorFlow/Keras:



```
[ ] # Importing required libraries from scikit-learn
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score, confusion_matrix

    # Importing required libraries from TensorFlow
    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, LSTM, Dense
```



```
[ ] import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score, confusion_matrix

    # Example DataFrame (replace this with the actual DataFrame 'df')
    df = pd.DataFrame({
        'text': ["I love this movie", "I hate this movie", "This movie is great", "This movie is terrible"],
        'sentiment': [1, 0, 1, 0]
    })

    # Extract features and labels
    X = df['text']
    y = df['sentiment']

    # Create TF-IDF features
    vectorizer = TfidfVectorizer()
    X = vectorizer.fit_transform(X)

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Create and train Logistic Regression model
    model = LogisticRegression()
    model.fit(X_train, y_train)
```



```
# Make predictions and evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.0
```