SENTIMENT ANALYSIS

VINOTHINI.V

LOADING DATASET:

- Collect a Dataset: First, you'll need a dataset of text documents with labeled sentiment (positive, negative, neutral). Common sentiment analysis datasets include movie reviews, social media comments, or product reviews.
- Preprocess the Data: Clean and preprocess the text data by removing punctuation, stop words, and any irrelevant information. You may also tokenize the text into words or subwords.
- Inference: Once your model is trained and evaluated, you can use it to analyze the sentiment of new text data.

- Label Encoding: Assign numerical labels to the sentiment classes, e.g., 0 for negative, 1 for neutral, and 2 for positive.
- Split the Data: Divide your dataset into training, validation, and test sets. This helps evaluate your model's performance.
- Vectorize Text: Convert the text data into numerical form using techniques like TF-IDF, Word Embeddings (Word2Vec, GloVe), or more advanced methods like BERT embeddings.

- Choose a Machine Learning or Deep Learning Model: Select a model for sentiment analysis. Common choices include Logistic Regression, Naive Bayes, Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), or transformer-based models like BERT.
- Train the Model: Fit your chosen model on the training data and tune hyperparameters using the validation set.
- Evaluate the Model: Use the test set to assess your model's performance by calculating metrics such as accuracy, precision, recall, and F1-score.

.Here’s a basic Python example using scikit-learn and a simple TF-IDF vectorizer along with a Logistic Regression classifier:

```
From sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

tfidf_vectorizer = TfidfVectorizer(max_features=5000)

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

X_test_tfidf = tfidf_vectorizer.transform(X_test).
```

- model = LogisticRegression()
- model.fit(X_train_tfidf, y_train)
- y_pred = model.predict(X_test_tfidf)
- accuracy = accuracy_score(y_test
- Remember, the choice of dataset, preprocessing steps, and model can vary based on your specific requirements and the nature of your text data. More advanced models like BERT or GPT-3 might yield better results for complex tasks.

PREPROCESSING THE DATA

- Data Preprocessing:
- Text Cleaning: Remove any special characters, punctuation, and HTML tags.
- Tokenization: Split text into words or subword tokens.
- Lowercasing: Convert text to lowercase to ensure consistency.
- Stopword Removal: Eliminate common words (e.g., "and," "the") that don't contribute much to sentiment.
- Stemming or Lemmatization: Reduce words to their base form (e.g., "running" to "

- Text Vectorization:
- Use techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or Word Embeddings (e.g., Word2Vec, GloVe) to convert text data into numerical form.
- Train-Test Split: Divide your dataset into a training set and a test set for model evaluation.
- Model Selection:
- Choose a sentiment analysis model like Naive Bayes, Support Vector Machines, Recurrent Neural Networks (RNNs), or Transformers (e.g., BERT).

- Model Training:
- Train the selected model using the preprocessed training data.
- Model Evaluation:
- Evaluate your model’s performance using metrics like accuracy, precision, recall, and F1-score on the test dataset.
- Hyperparameter Tuning:
- Fine-tune your model by adjusting hyperparameters for better performance.

- Deployment:
- Integrate the model into your application or service for real-time sentiment analysis.
- Remember to choose the appropriate tools and libraries for each step. Common choices include Python, libraries like NLTK, scikit-learn, TensorFlow, and PyTorch, and pre-trained language models like BERT for state-of-the-art performance