SANTMENT ANALYSIS FOR MARKETIG NIMMI HASSAN.F(411521104075)

**Dataset Selection:**

First, choose a dataset that contains text and corresponding sentiment labels (positive, negative, neutral, etc.). Popular datasets include IMDb movie reviews, Twitter sentiment data, or your own collected data.

**Data Preprocessing:**

Text Cleaning: Remove special characters, punctuation, and numbers from the text.

**Tokenization**:

Split the text into individual words or tokens.

Lowercasing: Convert all text to lowercase to ensure uniformity.

**Stopword Removal:**

Eliminate common words (e.g., “the,” “and”) that don’t carry sentiment information.

**Stemming or Lemmatization:**

Reduce words to their base form to handle variations.

**Data Labeling:**

Ensure that your dataset has accurate sentiment labels associated with each text entry.

Text Vectorization: Transform the text data into numerical vectors using techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings (e.g., Word2Vec or GloVe).

**Split Data:**

Divide your dataset into training and testing sets to evaluate your model’s performance.

**Select a Model:**

Choose a machine learning or deep learning model for sentiment analysis. Common choices include Logistic Regression, Naive Bayes, Support Vector Machines, or recurrent neural networks (RNNs) and transformers like BERT for more advanced tasks.

**Train the Model:**

Feed the training data into the chosen model and train it to learn the sentiment patterns.

Evaluate the Model: Use the testing data to assess the model’s performance. Common evaluation metrics include accuracy, precision, recall, F1-score, and confusion matrices.

Tune Hyperparameters: Optimize model hyperparameters to achieve better results.

**Deploy the** Model:

Once you’re satisfied with the model’s performance, deploy it for sentiment analysis tasks. You can create an API or integrate it into your application.

Continual Monitoring and **Maintenance:**

Regularly monitor the model’s performance and retrain it with new data to keep it up to date.

There are various libraries and frameworks available in Python, such as scikit-learn and TensorFlow/Keras, that can help you implement these steps. Additionally, pre-trained models like BERT can significantly simplify the process of sentiment analysis.

**LOADING DATA:**

**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.

**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.

**Evaluation of model:**

Use the test set to assess your model's performance by calculating metrics such as accuracy, precision, recall, and F1-score.

**Inference**:

Once your model is trained and evaluated, you can use it to analyze the sentiment of new text data.Here's a basic Python example using scikit-learn and a simple TF-IDF vectorizer along with a

**Logistic Regression classifier:** 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

# Load your dataset

# Preprocess and clean the data

# Encode labels

# Split data into train, validation, and test sets

# Vectorize text using TF-IDFtfidf\_vectorizer = TfidfVectorizer(max\_features=5000)X\_train\_tfidf = tfidf\_vectorizer.fit\_transform(X\_train)X\_test\_tfidf = tfidf\_vectorizer.transform(X\_test)

# Train a Logistic Regression modelmodel = LogisticRegression()model.fit(X\_train\_tfidf, y\_train)# Predict sentiment.y\_pred = model.predict(X\_test\_tfidf)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

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.

**program:**

Import nltk

From nltk.sentiment.vader import SentimentIntensityAnalyzer

# Sample text data

Text = “Your product is amazing! I love it.”

# Initialize the sentiment analyzer

Nltk.download(‘vader\_lexicon’)

Sid = SentimentIntensityAnalyzer()

# Get sentiment scores

Sentiment\_scores = sid.polarity\_scores(text)

# Analyze the sentiment

If sentiment\_scores[‘compound’] >= 0.05:

Sentiment = “positive”

Elif sentiment\_scores[‘compound’] <= -0.05:

Sentiment = “negative”

Else:

Sentiment = “neutral”

Print(“Sentiment:”, sentiment)

This is a basic example, and in practice, you’ll need to load a larger dataset and apply more advanced preprocessing and analysis techniques. You can also consider using machine learning models for more accurate sentiment analysis.