Aim: To develop a Bag of Words Model for Sentiment Analysis

Theory:

Sentiment Analysis (Opinion Mining):
Sentiment Analysis, also known as Opinion Mining, is a natural language processing (NLP) technique used to determine the sentiment or emotional tone in a piece of text, such as a review, comment, or social media post. Sentiment Analysis aims to classify text into categories like positive, negative, or neutral to understand the expressed sentiment of the author.

Bag of Words (BoW) Model:
The Bag of Words model is a fundamental and straightforward technique used in NLP for text analysis. It represents text data as a collection of individual words (or tokens) disregarding the order and structure of the text. The BoW model is constructed by creating a vocabulary of unique words in the dataset and then counting the frequency of each word in a document. It results in a matrix where each row corresponds to a document, and each column corresponds to a unique word in the vocabulary.

Steps to Develop a Bag of Words Model for Sentiment Analysis:
1. *Data Collection*:
   Gather a dataset containing text and corresponding sentiment labels (e.g., positive, negative, neutral). The dataset should be well-labeled to enable supervised learning.
2. *Data Preprocessing*:
   - Remove any irrelevant information or metadata (e.g., URLs, special characters, punctuation).
   - Tokenize the text: Split the text into individual words or tokens.
   - Convert text to lowercase to ensure uniformity.
   - Remove stopwords (common words like "the," "and," "is") that do not contribute much to sentiment analysis.
3. *Creating the Bag of Words*:
   - Build a vocabulary by collecting unique words from the entire dataset.
   - Create a matrix where each row represents a document, and each column represents a word from the vocabulary.
   - Populate the matrix by counting the frequency of each word in each document. This results in a term-document matrix.

4. *Feature Engineering*:
   - The term-document matrix serves as input features for a machine learning model.
   - Encode sentiment labels as numerical values (e.g., 0 for negative, 1 for neutral, and 2 for positive).
5. *Model Building*:
   - Choose a machine learning model suitable for text classification, such as Logistic Regression, Naive Bayes, or Support Vector Machines (SVM).
   - Split the dataset into training and testing sets to evaluate the model's performance.
6. *Model Training*:
   - Train the chosen model on the training data, using the term-document matrix as features and sentiment labels as targets.
7. *Model Evaluation*:
   - Evaluate the model's performance on the testing data using appropriate evaluation metrics (e.g., accuracy, precision, recall, F1-score).
   - Analyze the confusion matrix to understand the model's predictions.
8. *Predictions*:
   - Apply the trained model to predict sentiment labels for new or unseen text data.
9. *Interpretation and Improvements*:
   - Analyze the results and understand which words or phrases contribute to specific sentiment predictions.
   - Fine-tune the model by adjusting hyperparameters or trying different algorithms for better performance.
   - Iterate and improve the model as needed.

Code:

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer

data = pd.read_csv("/content/TweetData.csv", encoding='unicode_escape')
```

```
columns_to_drop = ['textID', 'Population -2020', 'Time of Tweet', 'Age of User',
'Country', 'Land Area (Km²)', 'Density (P/Km²)']
data = data.drop(columns=columns_to_drop, axis=1)
data = data.dropna()
data.isna().sum()
X = data['text']
y = data['sentiment']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=10)

print('Training Data: ', X_train.shape)
print('Testing Data: ', X_test.shape)

# Handle missing values in both training and test data
imputer = SimpleImputer(strategy='constant', fill_value='')
X_train = X_train.fillna('')
X_test = X_test.fillna('')

# Vectorize the text data
cv = CountVectorizer()
X_train_cv = cv.fit_transform(X_train)
X_test_cv = cv.transform(X_test)
lr = LogisticRegression()
lr.fit(X_train_cv, y_train)
predictions = lr.predict(X_test_cv)

y_test_cv = cv.transform(y_test)
print(classification_report(y_test, predictions))

sample_input = ["I love this product, I find it very useful"]
sample_input = cv.transform(sample_input)
y_pred = lr.predict(sample_input)
print("Sample Input 1 Prediction:", y_pred)
```

```
sample_input2 = ["I hate this product, I find it useless"]
sample_input2 = cv.transform(sample_input2)
y_pred2 = lr.predict(sample_input2)
print("Sample Input 2 Prediction:", y_pred2)
```

Output:

```
Training Data:  (2827,)
Testing Data:  (707,)
              precision    recall  f1-score   support

    negative       0.61      0.51      0.56       189
     neutral       0.58      0.70      0.64       300
    positive       0.74      0.63      0.68       218

    accuracy                           0.63       707
   macro avg       0.64      0.61      0.62       707
weighted avg       0.64      0.63      0.63       707

Sample Input 1 Prediction: ['positive']
Sample Input 2 Prediction: ['negative']
```

Conclusion:

The Bag of Words model is a useful and interpretable approach for sentiment analysis. By representing text as a matrix of word frequencies, it allows for the analysis of large volumes of textual data to understand the underlying sentiment. However, it is important to note that the BoW model doesn't capture the context and order of words in a document, which can limit its accuracy in certain cases. Nonetheless, it provides a solid foundation for text classification tasks and can be a starting point for more advanced NLP techniques.