Title: - Text Classification for Sentiment Analysis using K-Nearest Neighbors (KNN)

Aim/Objective: - To perform text classification for sentiment analysis using the K-Nearest Neighbors (KNN) algorithm and predict whether a given text (tweet) has a positive or negative sentiment.

Software Required:

- Python programming environment (Jupyter Notebook, Google Colab, or any Python IDE)
- Libraries: pandas, scikit-learn, nltk

Hardware Required:

- 4GB RAM
- Intel i3 or higher / AMD equivalent
- GPU for accelerated computations (if using deep learning frameworks)
- 120 GB SSD

Theory:

Sentiment Analysis is a type of natural language processing (NLP) where the aim is to determine the sentiment expressed in a piece of text — whether positive, negative, or neutral.

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm that classifies a data point based on how its neighbors are classified. It doesn't build a model explicitly; instead, it stores the entire training dataset and classifies new instances based on similarity measures (like Euclidean distance).

In **text classification**, we first need to:

- Preprocess text (cleaning, removing stopwords & punctuations)
- Convert text into numeric features (using techniques like Bag of Words or TF-IDF)
- Train KNN on these numeric features.

Procedure:

Procedure

1. **Import Libraries:** Import necessary Python libraries.
2. **Load Dataset:** Load a Twitter Sentiment dataset (or any text dataset).
3. **Preprocessing Text:**
    - Lowercasing, removing special characters, stopwords removal.
4. **Feature Extraction:**

   - Convert cleaned text into numerical features using TF-IDF Vectorizer.

5. **Model Training:**
   - Train KNN Classifier on the training data.

6. **Model Testing:**
   - Test the KNN Classifier on the unknown data given

Observations:

1. Text data was successfully preprocessed and transformed into numerical format.
2. KNN classified the tweets into positive or negative sentiments.
3. Model performance varied depending on the value of k.

Code:

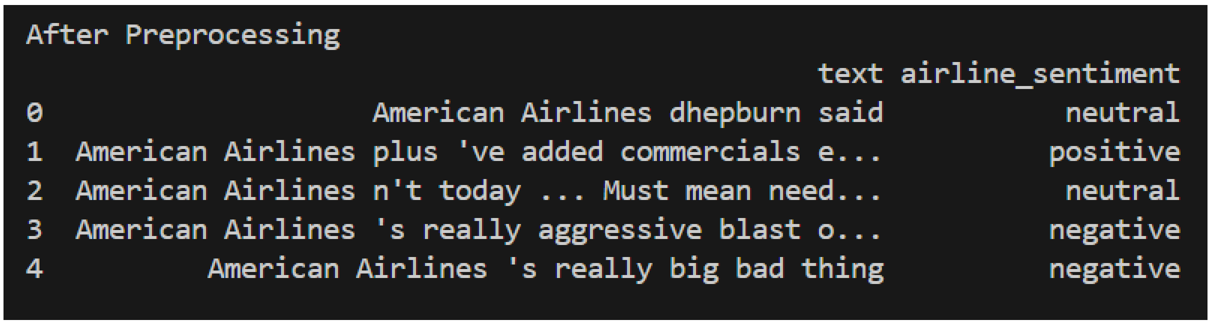
```python
# Import Required Libraries
import pandas as pd
import string
import nltk
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier

# Load Dataset
df = pd.read_csv("Tweets.csv")

# Feature Selection
df = df.iloc[:, [10, 1]]  # Select only relevant columns: 'text' and the
'label' column

#Print Before Preprocessing
print("Before Preprocessing")
print(df[['text','airline_sentiment']].head())
print("\n")

# Preprocessing Text: It involves 2 Step: 1. Removing (Stopwords &
Punctuations) & Cleaning, 2. Text to numerical form
# Step 1: Removing Stopwords & Punctuations
# Step 1.1: Pre-requisite - Download list of stopwords & punctuations from
nltk library
#Note: - One Time download afer then comment line no.19,21,22
#nltk.download('stopwords')  # Stopwords are commonly used words in a language
(like "is", "the", "and", "to")
                             # that don't contribute significant meaning in
text analysis.
#nltk.download('punkt')      # Punctuation list for tokenization
#nltk.download('punkt_tab')  # To reduce noise and standardize the text data.
```

```python
# Step 1.2: Remove Stopwords & Punctuation from the text and tokenize the words from text
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Get the list of stopwords
stop_words = set(stopwords.words('english'))

# Function to clean text by removing stopwords and punctuation
def clean_text(text):
    # Tokenize the text
    tokens = word_tokenize(text)
    # Remove stopwords and punctuation
    cleaned_text = [word for word in tokens if word.lower() not in stop_words and word not in string.punctuation]
    # Join the cleaned words back into a sentence
    return ' '.join(cleaned_text)

# Step 1.3: Apply the clean_text function to the 'text' column
df['text'] = df['text'].apply(clean_text)

#Print After Cleaning of Text
print("After Cleaning of Text Column")
print(df[['text','airline_sentiment']].head())
print("\n")

#Step 2: Feature Extraction: Using TF-IDF to convert text to numerical features
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(df['text'])
#After Pre-Processing now we need to do Model Training

#Print After Pre-Processing
print("After Preprocessing")
print(df[['text','airline_sentiment']].head())
print("\n")

#Note:- During we Processing, X is obtained therefore
# Target variable
y=df['airline_sentiment']

# Model Training: K-Nearest Neighbors Classifier
knn = KNeighborsClassifier(n_neighbors=5)  # You can experiment with different values of k
knn.fit(X, y)

#Print After Model Training
print("After Model Training")
print(df[['text','airline_sentiment']].head())
print("\n")

# Predict Sentiment for Unknown Reviews
def predict_sentiment(review):
    # Preprocess the review (clean and vectorize)
    cleaned_review = clean_text(review)
    review_vectorized = vectorizer.transform([cleaned_review])

    # Predict using the trained KNN model
    prediction = knn.predict(review_vectorized)
    return prediction[0]

# Example usage: Predict sentiment for an unknown review
unknown_review = "The product was amazing, I really loved it!"
predicted_sentiment = predict_sentiment(unknown_review)
#Note you can even give a feature to test via importing another dataset or use
Input function to manually write
print(unknown_review)
print(f"Predicted Sentiment: {predicted_sentiment}")
```

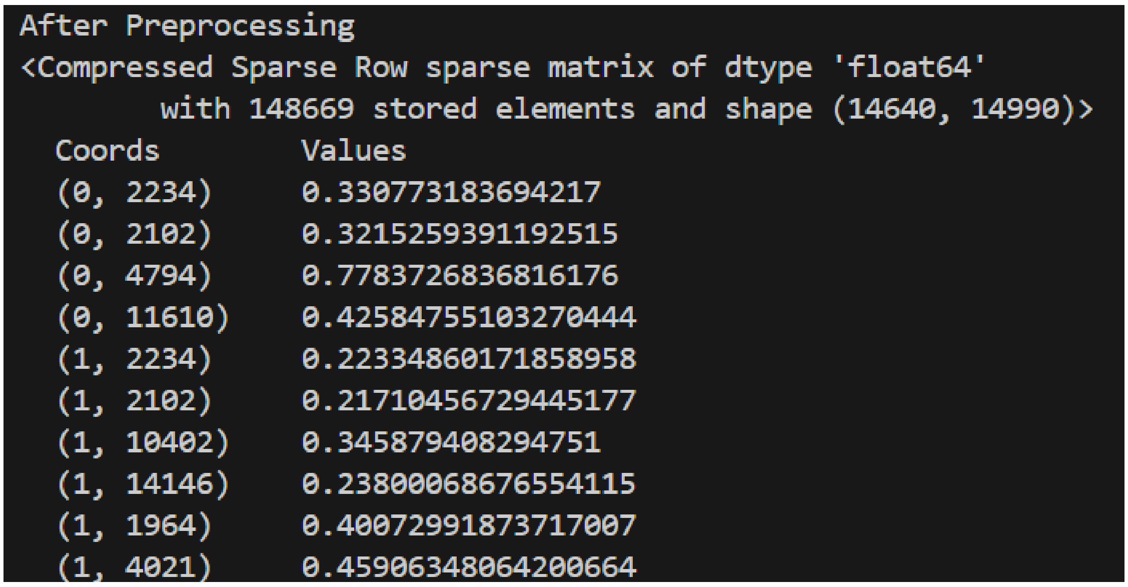
Output:

```
Before Preprocessing
                                                text airline_sentiment
0            @American Airlines What @dhepburn said.           neutral
1  @American Airlines plus you've added commercia...          positive
2  @American Airlines I didn't today... Must mean...           neutral
3  @American Airlines it's really aggressive to b...          negative
4  @American Airlines and it's a really big bad t...          negative
```

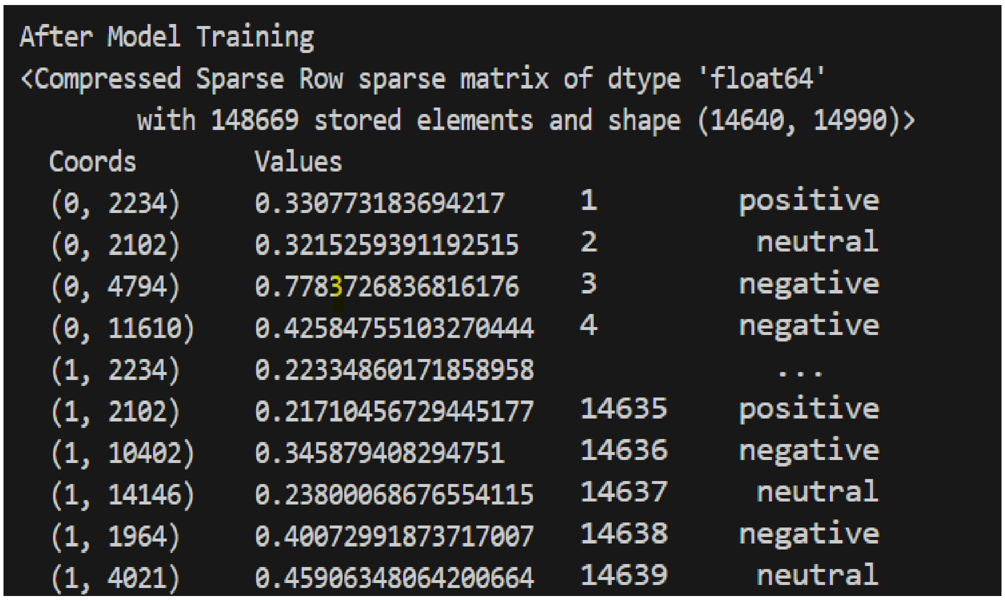
```
After Cleaning of Text Column
                                                text airline_sentiment
0                     American Airlines dhepburn said           neutral
1  American Airlines plus 've added commercials e...          positive
2  American Airlines n't today ... Must mean need...           neutral
3  American Airlines 's really aggressive blast o...          negative
4          American Airlines 's really big bad thing          negative
```



```
After Preprocessing
                                                  text airline_sentiment
0                    American Airlines dhepburn said           neutral
1  American Airlines plus 've added commercials e...          positive
2  American Airlines n't today ... Must mean need...           neutral
3  American Airlines 's really aggressive blast o...          negative
4           American Airlines 's really big bad thing         negative
```



```
After Preprocessing
<Compressed Sparse Row sparse matrix of dtype 'float64'
        with 148669 stored elements and shape (14640, 14990)>
  Coords        Values
  (0, 2234)     0.330773183694217
  (0, 2102)     0.3215259391192515
  (0, 4794)     0.7783726836816176
  (0, 11610)    0.42584755103270444
  (1, 2234)     0.22334860171858958
  (1, 2102)     0.21710456729445177
  (1, 10402)    0.345879408294751
  (1, 14146)    0.23800068676554115
  (1, 1964)     0.40072991873717007
  (1, 4021)     0.45906348064200664
```



```
After Model Training
<Compressed Sparse Row sparse matrix of dtype 'float64'
        with 148669 stored elements and shape (14640, 14990)>
  Coords        Values
  (0, 2234)     0.330773183694217     1        positive
  (0, 2102)     0.3215259391192515    2         neutral
  (0, 4794)     0.7783726836816176    3        negative
  (0, 11610)    0.42584755103270444   4        negative
  (1, 2234)     0.22334860171858958              ...
  (1, 2102)     0.21710456729445177   14635    positive
  (1, 10402)    0.345879408294751     14636    negative
  (1, 14146)    0.23800068676554115   14637     neutral
  (1, 1964)     0.40072991873717007   14638    negative
  (1, 4021)     0.45906348064200664   14639     neutral
```

Conclusion:

In this exercise, we have demonstrated how to preprocess text data, train a K-Nearest Neighbors (KNN) model for sentiment classification, and make predictions on tweets or reviews. Here's a summary of the key steps and the overall process:

Data Preprocessing:

- We started by cleaning the text data. This involved removing unnecessary **stopwords** (e.g., "the", "is", "and") and **punctuations** (e.g., ".", ",", "@"). This helped to focus on meaningful words in the text, reducing noise in the data and improving the model's performance.

Feature Extraction:

- The cleaned text data was then converted into numerical form using **TF-IDF (Term Frequency-Inverse Document Frequency)**. This step transformed the text into a format that can be fed into the machine learning model. TF-IDF helps to identify the most important words in the text by weighing terms based on their frequency and relevance.

Label Encoding:

- The sentiment labels ("Positive" and "Negative") were converted into **binary values**: 1 for Positive and 0 for Negative. This step is crucial for the model, as most machine learning algorithms work with numerical data.

Model Training:

- The K-Nearest Neighbors (KNN) algorithm was used to train the model. This algorithm is a simple yet effective classification method that makes predictions based on the majority label of the nearest neighbors in the feature space.

Model Prediction:

- After training, the model was used to predict the sentiment (Positive or Negative) for all rows in the dataset. The predicted sentiments were stored in a new column in the DataFrame.

Results:

By running this process, you can evaluate the model's performance by comparing the predicted sentiment with the actual sentiment. If the model's predictions match the actual labels (0 for Negative, 1 for Positive), it indicates that the KNN model has learned well from the data. If the model doesn't perform as expected, further steps such as hyperparameter tuning, using different models (e.g., Logistic Regression, SVM), or improving the feature extraction method could be considered.