Aim: - implement document classifier in multi category dataset

Requirement: -Python versions 3.7, 3.8, 3.9, 3.10 or 3.11, jupyter notebook

Theory: -

1. Data Preprocessing:
   Load Data: Load your dataset containing documents and their respective categories.
   Text Cleaning: Remove stopwords, punctuation, and perform stemming/lemmatization.
   Vectorization: Convert text data into numerical features using techniques like TF-IDF or word embeddings (Word2Vec, GloVe).
2. Split the Data:
   Train-Test Split: Divide the dataset into training and testing sets. For multi-category, you might consider stratified splitting to maintain class balance.
3. Model Selection and Training:
   Choose a Classifier: Select a suitable classification algorithm (e.g., Naive Bayes, SVM, Random Forest, Neural Networks).
   Fit the Model: Train the chosen classifier on the training data.
4. Evaluation:
   Predictions: Make predictions on the test set.
   Evaluation Metrics: Calculate evaluation metrics like accuracy, precision, recall, and F1-score to assess the model's performance.

Steps :-

1. Load Data: Fetch the dataset using fetch_20newsgroups() from sklearn.datasets.
2. Text Vectorization: Convert text data into numerical features using TfidfVectorizer.

3. Split Data: Split the dataset into training and testing sets.
4. Model Training: Initialize the Multinomial Naive Bayes classifier and fit it to the training data.
5. Predictions: Make predictions on the test data.
6. Evaluation: Print out a classification report to evaluate the performance of the classifier.
Replace newsgroups_train.data and newsgroups_test.data with your document data, and adjust categories accordingly. Additionally, you might want to preprocess your text data (cleaning, normalization) before vectorizing it, depending on your specific dataset's characteristics.

Code:-

```python
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Load the dataset (Example: 20 Newsgroups dataset)
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'sci.med']
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories)
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories)

# Convert text data to numerical features using TF-IDF
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(newsgroups_train.data)
X_test = vectorizer.transform(newsgroups_test.data)
y_train = newsgroups_train.target
y_test = newsgroups_test.target

# Split the 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)

# Initialize and train the classifier
clf = MultinomialNB()
clf.fit(X_train, y_train)

# Make predictions
predicted = clf.predict(X_test)

# Evaluate the classifier
print(classification_report(y_test, predicted))
```

```
              precision    recall  f1-score   support

           0       0.97      0.60      0.74       319
           1       0.96      0.89      0.92       389
           2       0.97      0.81      0.88       396
           3       0.65      0.99      0.78       398

    accuracy                           0.83      1502
   macro avg       0.89      0.82      0.83      1502
weighted avg       0.88      0.83      0.84      1502
```

```
# !pip install pytesseract pillow
# !apt install tesseract-ocr


from PIL import Image as image
import pytesseract

ipath = '/content/textVsImage.png'
imagetext = image.open(ipath)

extractedText = pytesseract.image_to_string(imagetext)

print(extractedText)
```

```
People following directions with text and illustrations do 323% better
than those following directions without illustrations.
```

Source: Levie, W. Howard, and Richard Lentz. “Effects of Text Illustrations: A Review of Research.”