Problem 2 : Data Engineering Challenge

This is an overview of the preprocessing steps and feature extraction methods used for text classification on the BBC articles dataset.

- `read_text_file(file_path)`: This function reads the content of a text file and returns it as a string.

- `create_csv_from_text_files(folder_path, csv_file_path)`: This function creates a CSV file from text files in a specified folder. It reads all the text files in the folder, extracts the article ID, category, and text content from each file, and writes this information into a CSV file.

  Args:
  - `folder_path (str)`: Path to the folder containing text files.
  - `csv_file_path (str)`: Path to the CSV file to be created.

Main block of code:
The main block of code does the following:

- Calls`create_csv_from_text_files(folder_path, csv_file_path)` to generate a structured CSV file from text files in the specified folder.
- Reads the structured CSV file into a Pandas DataFrame.

- Text Preprocessing:
  The `preprocess_text(text)` function is used to preprocess the text data. It performs the following steps:

  - Convert text to lowercase.
  - Remove punctuation and numbers.
  - Tokenize the text.
  - Remove stopwords.

```
    text = text.lower()

    # Remove punctuation and numbers

    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Tokenize the text

    tokens = word_tokenize(text)

    # Remove stopwords
    tokens = [word for word in tokens if word not in
stopwords.words('english')]

    return ' '.join(tokens)
```

- Apply the `preprocess_text(text)` function to preprocess the text data.

```
df['processed_text'] = df['text'].apply(preprocess_text)
```

- Perform TF-IDF vectorization on the preprocessed text data using `TfidfVectorizer`.

```
tfidf_vectorizer = TfidfVectorizer(max_features=1000)  # Limiting to
1000 features

tfidf_features = tfidf_vectorizer.fit_transform(df['processed_text'])
```

1. **TF-IDF Representation:**
   - `TfidfVectorizer` converts text data into numerical vectors using the TF-IDF (Term Frequency-Inverse Document Frequency) representation.
   - TF-IDF takes into account the frequency of a term in a document and the rarity of the term across all documents, capturing the importance of the term in the document relative to its importance in the entire corpus.
2. **Normalisation:**
   - TF-IDF normalises the vector representation of text data, making it robust to varying document lengths.
   - This ensures that the model focuses on the significance of terms rather than their raw frequencies.
3. **Dimensionality Reduction:**
   - By using `(max_features=1000)`, `TfidfVectorizer` limits the number of features to the top 1000 most important terms.
   - This reduces the dimensionality of the feature space, making the model more efficient and less prone to overfitting.

- Convert category labels to numerical values using `label_encoder.`

```
label_encoder = LabelEncoder()

df['category_label'] =
label_encoder.fit_transform(df['category'])
```

- Create a DataFrame (df_final) with TF-IDF features and category labels.

```
df_features = pd.DataFrame(tfidf_features.toarray(),
columns=tfidf_vectorizer.get_feature_names_out())

df_final = pd.concat([df_features, df['category_label']], axis=1)
```

- Save the new DataFrame `df_final` as a CSV file named 'vectorized_dataset.csv'.

  ```
  df_final.to_csv('vectorized_dataset.csv', index=False)
  ```

To run the provided code, make sure you have the following dependencies installed:

- Python (version 3.6 or higher)
  - You can download and install Python from https://www.python.org/
- Required Python packages:
  - pandas
  - Nltk
  - scikit-learn

These packages can be installed using pip in terminal:

```
pip install pandas nltk scikit-learn
```

- NLTK Resources:
  - Before running the code, you need to download NLTK resources for tokenization and stopwords.