## **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:

- Callscreate\_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.