IR - Information Retrieval

Assignment 01

- Daniel Attali - 328780879

Input

We are given 4 groups of 600 document taken from JP, BBC, NYT, AJ. The task it to perform a research on the data.

Part 1 - Data Cleaning and Tokenization

Data Cleaning

The first part is to clean the data we have to first load that data using `pandas` into 4 dataframes.

Then after loading we need to take the important columns from the dataframes (each source has different columns) then for each source combine the columns (title, sub-title, content) into a single column of `documents` and add an `id` column to each document (i.e. `jp_1` etc).

Before continuing we need to find and replace all the different kind of characters such as backtik, double quotes, single quotes, etc. with their respective ASCII characters.

```
def clean_text(text: str) -> str:
    # Normalize all types of single and double quotation marks to standard
forms
    text = re.sub(r"[‘’`]", "'", text)  # Convert all single quote
variations to '
    text = re.sub(r"[“”]", '"', text)  # Convert all double quote variations
to "

    return text
```

Tokenization - Part 1

The first part of is word tokenization. Where we are tasked to separate the words from the punctuation and other characters. We are also tasked to remove the stop words from the documents.

For example `I am a student. Are you?` will be tokenized into `I am a student . Are you ?`. Where we keep the punctuation for usage further down the line.

Example:

```
text = "How are you? I'm well thank you. doesn't."
clean_text(text)
```

Output:

```
How are you ? I'm well thank you . doesn't .
```

Then we apply this on all the 4 groups of ~600 documents.

Tokenization - Part 2

Lemmatization is the process of converting a word to its base form. For example, the word "running" is converted to "run". This is done using the `nltk` library.

We first need to download the needed resources

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

nltk.download("punkt")
nltk.download("stopwords")
nltk.download('punkt_tab')
```

The function:

```
def clean_text_lemma(text):
  # replace all the I'm to I am etc
  text = re.sub(r"I'm", "I am", text)
  text = re.sub(r"you're", "you are", text)
  text = re.sub(r"he's", "he is", text)
```

```
    text = re.sub(r"she's", "she is", text)
    text = re.sub(r"it's", "it is", text)
    text = re.sub(r"we're", "we are", text)
    text = re.sub(r"they're", "they are", text)
    text = re.sub(r"that's", "that is", text)
    text = re.sub(r"what's", "what is", text)
    text = re.sub(r"where's", "where is", text)
    text = re.sub(r"how's", "how is", text)
    text = re.sub(r"i'll", "I will", text)

    # remove from the text all the punctatution
    text = re.sub(r'[^\w\s]', '', text)

    # tokenize the text
    tokens = word_tokenize(text)

    # remove all the numbers and dates etc
    tokens = [word for word in tokens if not any(char.isdigit() for char in
word)]

    # remove the stopwords
    tokens = [word for word in tokens if not word.lower in stop_words]

    doc = nlp(' '.join(tokens))
    lemmatized_text = ' '.join(token.lemma_ for token in doc)
    return lemmatized_text
```

Explanation:

1. Replace all the contractions with their full form. (To not loose their meaning)
2. Remove all the punctuation from the text.
3. Tokenize the text.
4. Remove all the numbers and dates etc.
5. Remove the stopwords.
6. Lemmatize the text.

Tokenization - Part 3

Saving the data into `.csv` file:

```
df_aj_word.to_csv("A_J_word.csv", index=False)
```

Tokenization - Output

The output of this process is 2 groups of 4 dataframe and in each dataframe we have about 600 documents. Each document is tokenized and lemmatized or cleaned.

The columns of the dataframe are `id` and `document`.

Part 2 - TF-IDF

We are tasked to create a TF-IDF matrix for each of the 4 groups of documents.

TF-IDF - Part 1

The formula for TF-IDF we will use is TF-IDF with BM25 + Okapi.

$$f(q,d) = \sum_{w \in q \cap d} \frac{(k+1)c(w,d)}{c(w,d) + k\left(1 - b + b \cdot \frac{L_d}{L_{avg}}\right)} \cdot \log\left(\frac{M+1}{df(w)}\right)$$

$$b \in [0,1], k \in [0, +\infty)$$

Where:

- $f(q,d)$ is the score of the query q in the document d
- w is the word in the query q and the document d
- $c(w,d)$ is the frequency of the word w in the document d
- $df(w)$ is the document frequency of the word w
- M is the total number of documents
- L_d is the length of the document d
- L_{avg} is the average length of the documents

The process of creating the TF-IDF matrix is as follows:

1. Find the corpus of the documents.
2. Find the vocabulary of the corpus.
3. Create the TF matrix.
4. Create the IDF matrix.
5. Find the L_{avg}.

Step 1: Corpus

```
def create_corpus(corpus):
    processed_corpus = []

    for doc in corpus:
        # Convert to lowercase
        doc = doc.lower()
        # Remove punctuation
        doc = re.sub(r"[^\w\s]", "", doc)
        # Remove numbers
        doc = re.sub(r"\d+", "", doc)
        # Tokenize the document
        words = word_tokenize(doc)
        # Remove stop words
        words = [word for word in words if word not in STOP_WORDS]
        # Join the words back into a string
        processed_doc = " ".join(words)
        processed_corpus.append(processed_doc)
    return processed_corpus
```

Step 2: Vocabulary

```
def create_vocabulary(corpus):
    vocabulary = set()
    for doc in corpus:
        words = doc.split()
        for word in words:
            vocabulary.add(word)
    return vocabulary
```

The vocabulary size for the word docs are in the 4000 range and for the lemma docs are in the 3000 range.

Step 3: TF Matrix

```
def create_tf_matrix(corpus, vocab, threshold=5):
    n_docs = len(corpus)
    n_terms = len(vocab)

    tf_matrix = np.zeros((n_docs, n_terms)).astype(np.int32)

    for doc_idx, doc in enumerate(corpus):
        words = doc.split()
```

```
        for v_idx, v in enumerate(vocab):
            # if the count is lower than threshold don't put it in
            c = words.count(v)
            if c >= threshold:
                tf_matrix[doc_idx, v_idx] = words.count(v)

    return tf_matrix
```

Step 4: IDF Matrix

```
def calculate_df(tf_matrix, vocab):
    df = {}
    for term_idx in range(tf_matrix.shape[1]):
        df[vocab[term_idx]] = np.count_nonzero(tf_matrix[:, term_idx])
    return df
```

Step 5: L_{avg}

```
def calculate_avg_doc_len(corpus):
    total_len = sum(len(doc.split()) for doc in corpus)
    return total_len / len(corpus)
```

TF-IDF - Part 2

Calculate the TF-IDF matrix:

```
def tfidf_bm25_okapi(tf_matrix, df, processed_corpus, vocab, L_avg, k=1.2,
b=0.75):
    M = tf_matrix.shape[0]  # Total number of documents

    tfidf_matrix = np.zeros_like(tf_matrix, dtype=np.float64)

    for doc_idx in range(tf_matrix.shape[0]):
        doc_len = len(processed_corpus[doc_idx].split())
        for term_idx in range(tf_matrix.shape[1]):
            term = vocab[term_idx]
            c_wd = tf_matrix[doc_idx, term_idx]  # Term frequency in the
document

            if c_wd > 0:  # Only calculate if the term is present
```

```
                idf = np.log((M + 1) / df[term])

                numerator = (k + 1) * c_wd
                denominator = c_wd + k * (1 - b + b * (doc_len / L_avg))

                tfidf_matrix[doc_idx, term_idx] = (numerator / denominator)
* idf

    return tfidf_matrix
```

The default value for K and b are 1.2 and 0.75 respectively. Based on the literature these are the most common values. here in the wiki article we can see the the most common values for k are $k \in [1.2, 2.0]$ and for b are $b \in [0.75, 1.0]$.

TF-IDF - Part 3

The 3d part is to analize the TF-IDF matrix and find the top 10 words for each document.

Using two functions:

1. Information Gain
2. Gain Ratio

Information Gain

```
def information_gain(tfidf_matrix):
    # calculate the entropy of the tfidf matrix
    entropy = -tfidf_matrix * np.log2(tfidf_matrix)
    entropy = np.sum(entropy, axis=1)
    # calculate the entropy of the tfidf matrix
    entropy = -tfidf_matrix * np.log2(tfidf_matrix)
    entropy = np.sum(entropy, axis=1)
    # calculate the information gain
    ig = np.sum(entropy) - entropy
    return ig
```

The formula as follows:

$$IG = \sum_{i=1}^{n} \left(\sum_{j=1}^{m} -tfidf_{ij} \cdot \log_2(tfidf_{ij}) \right) - \sum_{i=1}^{n} \left(\sum_{j=1}^{m} -tfidf_{ij} \cdot \log_2(tfidf_{ij}) \right)$$

Where:

- IG is the information gain
- $tfidf_{ij}$ is the value of the i-th document and j-th term in the TF-IDF matrix
- n is the number of documents
- m is the number of terms

Gain Ratio

```
def gain_ration(tfidf_matrix):
    # calculate the entropy of the tfidf matrix
    entropy = -tfidf_matrix * np.log2(tfidf_matrix)
    entropy = np.sum(entropy, axis=1)
    # calculate the information gain
    ig = np.sum(entropy) - entropy
    # calculate the gain ratio
    gr = ig / entropy
    return gr
```

The formula as follows:

$$GR = \frac{IG}{H}$$

Where:

- GR is the gain ratio
- IG is the information gain

Output of 20 terms

```
Building IG and GR for file:A_J_word.csv
Top 20 terms by Information Gain:
 ['barricade', 'algerias', 'alfitr', 'alexius', 'alexandria', 'alert',
'aleaziz', 'ale', 'aldaqran', 'albaressaid', 'albany', 'albalah', 'albag',
'alawda', 'alassad', 'alarmed', 'alarm', 'algeria', 'ali', 'allow']
Top 20 terms by Gain Ratio:
 ['barricade', 'algerias', 'alfitr', 'alexius', 'alexandria', 'alert',
'aleaziz', 'ale', 'aldaqran', 'albaressaid', 'albany', 'albalah', 'albag',
'alawda', 'alassad', 'alarmed', 'alarm', 'algeria', 'ali', 'allow']
Building IG and GR for file:BBC_word.csv
```

```
Top 20 terms by Information Gain:
 ['badfaith', 'ajoint', 'alan', 'alam', 'alaliin', 'alali', 'ala', 'al',
'airstrikein', 'algerias', 'airstrike', 'airportstyle', 'airlift',
'airdrop', 'aircraft', 'airborne', 'alaqsa', 'alarge', 'alarm', 'alarmed']
Top 20 terms by Gain Ratio:
 ['badfaith', 'ajoint', 'alan', 'alam', 'alaliin', 'alali', 'ala', 'al',
'airstrikein', 'algerias', 'airstrike', 'airportstyle', 'airlift',
'airdrop', 'aircraft', 'airborne', 'alaqsa', 'alarge', 'alarm', 'alarmed']
Building IG and GR for file:J_P_word.csv
Top 20 terms by Information Gain:
 ['barricade', 'algerias', 'alfitr', 'alexius', 'alexandria', 'alert',
'aleaziz', 'ale', 'aldaqran', 'albaressaid', 'albany', 'albalah', 'albag',
'alawda', 'alassad', 'alarmed', 'alarm', 'algeria', 'ali', 'allow']
Top 20 terms by Gain Ratio:
 ['barricade', 'algerias', 'alfitr', 'alexius', 'alexandria', 'alert',
'aleaziz', 'ale', 'aldaqran', 'albaressaid', 'albany', 'albalah', 'albag',
'alawda', 'alassad', 'alarmed', 'alarm', 'algeria', 'ali', 'allow']
Building IG and GR for file:NYT_word.csv
Top 20 terms by Information Gain:
 ['barricade', 'algerias', 'alfitr', 'alexius', 'alexandria', 'alert',
'aleaziz', 'ale', 'aldaqran', 'albaressaid', 'albany', 'albalah', 'albag',
'alawda', 'alassad', 'alarmed', 'alarm', 'algeria', 'ali', 'allow']
Top 20 terms by Gain Ratio:
 ['barricade', 'algerias', 'alfitr', 'alexius', 'alexandria', 'alert',
'aleaziz', 'ale', 'aldaqran', 'albaressaid', 'albany', 'albalah', 'albag',
'alawda', 'alassad', 'alarmed', 'alarm', 'algeria', 'ali', 'allow']
Building IG and GR for file:A_J_lemma.csv
Top 20 terms by Information Gain:
 ['barricade', 'algerias', 'alfitr', 'alexius', 'alexandria', 'alert',
'aleaziz', 'ale', 'aldaqran', 'albaressaid', 'albany', 'albalah', 'albag',
'alawda', 'alassad', 'alarmed', 'alarm', 'algeria', 'ali', 'allow']
Top 20 terms by Gain Ratio:
 ['barricade', 'algerias', 'alfitr', 'alexius', 'alexandria', 'alert',
'aleaziz', 'ale', 'aldaqran', 'albaressaid', 'albany', 'albalah', 'albag',
'alawda', 'alassad', 'alarmed', 'alarm', 'algeria', 'ali', 'allow']
Building IG and GR for file:BBC_lemma.csv
Top 20 terms by Information Gain:
 ['badfaith', 'ajoint', 'alan', 'alam', 'alaliin', 'alali', 'ala', 'al',
'airstrikein', 'algerias', 'airstrike', 'airportstyle', 'airlift',
'airdrop', 'aircraft', 'airborne', 'alaqsa', 'alarge', 'alarm', 'alarmed']
```

```
Top 20 terms by Gain Ratio:
 ['badfaith', 'ajoint', 'alan', 'alam', 'alaliin', 'alali', 'ala', 'al',
'airstrikein', 'algerias', 'airstrike', 'airportstyle', 'airlift',
'airdrop', 'aircraft', 'airborne', 'alaqsa', 'alarge', 'alarm', 'alarmed']
Building IG and GR for file:J_P_lemma.csv
Top 20 terms by Information Gain:
 ['barricade', 'algerias', 'alfitr', 'alexius', 'alexandria', 'alert',
'aleaziz', 'ale', 'aldaqran', 'albaressaid', 'albany', 'albalah', 'albag',
'alawda', 'alassad', 'alarmed', 'alarm', 'algeria', 'ali', 'allow']
Top 20 terms by Gain Ratio:
 ['barricade', 'algerias', 'alfitr', 'alexius', 'alexandria', 'alert',
'aleaziz', 'ale', 'aldaqran', 'albaressaid', 'albany', 'albalah', 'albag',
'alawda', 'alassad', 'alarmed', 'alarm', 'algeria', 'ali', 'allow']
Building IG and GR for file:NYT_lemma.csv
Top 20 terms by Information Gain:
 ['barricade', 'algerias', 'alfitr', 'alexius', 'alexandria', 'alert',
'aleaziz', 'ale', 'aldaqran', 'albaressaid', 'albany', 'albalah', 'albag',
'alawda', 'alassad', 'alarmed', 'alarm', 'algeria', 'ali', 'allow']
Top 20 terms by Gain Ratio:
 ['barricade', 'algerias', 'alfitr', 'alexius', 'alexandria', 'alert',
'aleaziz', 'ale', 'aldaqran', 'albaressaid', 'albany', 'albalah', 'albag',
'alawda', 'alassad', 'alarmed', 'alarm', 'algeria', 'ali', 'allow']
```

So we can learn from the top 20 terms in each TF-IDF Matrix that terms are similar across all the documents which means that the documents are similar (we now that becasuse we took the articles about the same topic).

TF-IDF - Part 4

Saving the TF-IDF matrix. Since the TF-IDF is a sparse matrix we can save it as a `.npz` file.

```
def save_sparse_matrix(matrix, vocab, doc_ids, path):
    # convert the matrix to a sparse matrix
    sparse_matrix = sparse.csr_matrix(matrix)
    # save the sparse matrix
    sparse.save_npz(path + "_matrix.npz", sparse_matrix)
    # save the vocabulary
    pd.DataFrame(list(vocab), columns=["term"]).to_csv(path + "_vocab.csv",
index=False)
    # save the document id
```

```
    pd.DataFrame(doc_ids, columns=["doc_id"]).to_csv(path + "_doc_id.csv",
index=False)
```

We also have a load function to reload saved matrix.

```
def load_sparse_matrix(path):
    # load the sparse matrix
    matrix = sparse.load_npz(path + "_matrix.npz")
    # load the vocabulary
    vocab = pd.read_csv(path + "_vocab.csv")["term"].tolist()
    # load the document id
    doc_ids = pd.read_csv(path + "_doc_id.csv")["doc_id"].tolist()

    # transform the sparse matrix to a dense matrix and create a df with
vocab and doc_id
    matrix = matrix.todense()
    df = pd.DataFrame(matrix, columns=vocab, index=doc_ids)
    return df, matrix, vocab, doc_ids
```

TF-IDF - Output

The output of this process is 4 groups of TF-IDF matrix. Each matrix is saved as a `.npz` file and the vocabulary and document id are saved as `.csv` files.

Part 3 - Vectorization

In this part we are tasked to take the output of the first part and calculate the vector representation of the documents.

1. Calculate the vector of the word using `Word2Vec`.
2. Calculate the vector of the document using `Doc2Vec`.
3. Calculate the vector of the word using `BERT`.
4. Calculate the vector of the document using `BERT`.
5. Calculate the vector of sentence using `Sentence-BERT`.

Vectorization - Part 1

Word2Vec

```
from gensim.models import Word2Vec

def word2vec(corpus, size=100, window=5, min_count=1, workers=4):
    model = Word2Vec(corpus, size=size, window=window, min_count=min_count,
workers=workers)
    return model
```

The vector size for the word2vec is 100 and the window size is 5.

Vectorization - Part 2

Doc2Vec

```
from gensim.models import Doc2Vec
from gensim.models.doc2vec import TaggedDocument

def doc2vec(corpus, size=100, window=5, min_count=1, workers=4):
    tagged_corpus = [TaggedDocument(words=word_tokenize(doc), tags=[str(i)])
for i, doc in enumerate(corpus)]
    model = Doc2Vec(tagged_corpus, vector_size=size, window=window,
min_count=min_count, workers=workers)
    return model
```

The vector size for the doc2vec is 100 and the window size is 5.

Vectorization - Part 3

BERT

```
from transformers import BertTokenizer, BertModel

def bert(corpus, model_name="bert-base-uncased"):
    tokenizer = BertTokenizer.from_pretrained(model_name)
    model = BertModel.from_pretrained(model_name)

    vectors = []
    for doc in corpus:
        inputs = tokenizer(doc, return_tensors="pt")
```

```
        outputs = model(**inputs)

vectors.append(outputs.last_hidden_state.mean(dim=1).detach().numpy())

    return np.array(vectors)
```

The default model name is `bert-base-uncased`. The dimension of the vector is 768.

Vectorization - Part 4

Sentence-BERT

```
from sentence_transformers import SentenceTransformer

def sentence_bert(corpus, model_name="stsb-roberta-base"):
    model = SentenceTransformer(model_name)
    vectors = model.encode(corpus)
    return vectors
```

The default model name is `stsb-roberta-base`. The dimension of the vector is 768.

Vectorization - Part 5

Saving the vectors:

```
def save_vectors(vectors, doc_ids, path):
    np.save(path + "_vectors.npy", vectors)
    pd.DataFrame(doc_ids, columns=["doc_id"]).to_csv(path + "_doc_id.csv",
index=False)
```

Vectorization - Part 6

Loading the vectors:

```
def load_vectors(path):
    vectors = np.load(path + "_vectors.npy")
    doc_ids = pd.read_csv(path + "_doc_id.csv")["doc_id"].tolist()
    return vectors, doc_ids
```

Vectorization - Output

The output of this process is 4 groups of vectors. Each vector is saved as a `.npy` file and the document id is saved as a `.csv` file.

Summary

In this assignment we have tokenized and lemmatized the documents. Then we have created the TF-IDF matrix for each group of documents. Finally we have created the vector representation of the documents.

The output of this process is 4 groups of vectors and 4 groups of TF-IDF matrix.
