# Information Retrieval

## Assignment 03

**context**: in the first assignment we were given a dataset of of articles from 4 sources (AJ, BBC, NYT, J-P) with articles per source.

In this assignment we are asked to perform sentiment analysis on the article.

We have divided the assignment into stages:

1. Data Preprocessing - Tokenization, Stopwords Removal, Stemming (Done in Assignment 02)
2. Sentence Extraction - we take a database of words with positive and negative connotation and we extract the sentences that contain these words. This will be the base of our sentiment analysis.
3. Sentiment Analysis - we will use the extracted sentences to perform sentiment analysis on the articles. We will use many models from huggingface transformers to perform this task. And for each sentence we make the models vote on the sentence wether it is positive or negative. We will then use the majority vote to determine the sentiment of the article.

## Stage 2: Sentence Extraction

In this section we will perform the sentence extraction.

We have two databases of words with positive and negative connotation. We will use these words to extract the sentences that contain these words.

Those are the israel.txt and palestine.txt files. With each about 100 words.

def extract\_relevant\_sentences(df, pro\_israeli\_words, pro\_palestinian\_words):  
 extracted = []  
  
 for idx, row in df.iterrows():  
 doc\_id = row['id']  
 document = row['document']  
  
 # Split into sentences  
 sentences = re.split(r'[.!?]', document) # Basic sentence splitting  
  
 for sentence in sentences:  
 sentence = sentence.strip().lower()  
 is\_pro\_israeli = any(word in sentence for word in pro\_israeli\_words)  
 is\_pro\_palestinian = any(word in sentence for word in pro\_palestinian\_words)  
  
 if is\_pro\_israeli and not is\_pro\_palestinian:  
 extracted.append((doc\_id, sentence, 'pro-israeli'))  
 elif is\_pro\_palestinian and not is\_pro\_israeli:  
 extracted.append((doc\_id, sentence, 'pro-palestinian'))  
  
 return pd.DataFrame(extracted, columns=['id', 'sentence', 'type'])

As you can see from the code snippet about, the function extract\_related\_sentences will extract the sentences that contain the words from the database. The idea behind is very simple if the word then the sentence is positive and vice versa.

The rest of the code is just loading the files from the GitHub repo and performing the same cleaning steps as in the previous assignment.

# remove special characters from the sentences  
df\_results[0]["sentence"] = df\_results[0]["sentence"].apply(clean\_text)  
df\_results[1]["sentence"] = df\_results[1]["sentence"].apply(clean\_text)  
df\_results[2]["sentence"] = df\_results[2]["sentence"].apply(clean\_text)  
df\_results[3]["sentence"] = df\_results[3]["sentence"].apply(clean\_text)  
  
# Combine results  
print("Combining results")  
df\_extracted = pd.concat(df\_results)  
df\_extracted.to\_csv("extracted\_sentences.csv", index=False)

The last step in this stage is concatenating the sentences and saving them to a file extracted\_sentences.csv for the next stage.

## Stage 3: Sentiment Analysis

In this stage we will perform the sentiment analysis on the extracted sentences.

link = "https://github.com/dattali18/IR\_Assignments/blob/main/Assignment.03/extracted\_sentences.csv?raw=true"

import pandas as pd  
from transformers import pipeline  
from tqdm import tqdm  
import torch  
  
# Download your CSV  
df = pd.read\_csv(link)  
print("Dataset shape:", df.shape)  
print("\nFirst few rows:")  
print(df.head())  
print("\nColumn names:", df.columns.tolist())

def load\_sentiment\_models():  
 model\_paths = {  
 'model1': "cardiffnlp/twitter-roberta-base-sentiment-latest",  
 'model2': "nlptown/bert-base-multilingual-uncased-sentiment",  
 'model3': "cardiffnlp/twitter-xlm-roberta-base-sentiment",  
 'model4': "siebert/sentiment-roberta-large-english",  
 'model5': "lxyuan/distilbert-base-multilingual-cased-sentiments-student",  
 'model6': "finiteautomata/bertweet-base-sentiment-analysis",  
 'model7': "j-hartmann/sentiment-roberta-large-english-3-classes"  
 }  
  
 loaded\_models = {}  
 for name, path in model\_paths.items():  
 try:  
 loaded\_models[name] = pipeline("sentiment-analysis", model=path)  
 print(f"Loaded {name}")  
 except Exception as e:  
 print(f"Failed to load {name}: {e}")  
  
 return loaded\_models  
  
# Load models  
models = load\_sentiment\_models()

As you can see from the code above, we are loading a total of 7 models from huggingface transformers. We will use these models to perform the sentiment analysis on the extracted sentences.

def interpret\_sentiment(sentiment\_label, score, sentence\_type):  
 """Convert model sentiment to pro-israeli/pro-palestinian context"""  
 if 'neutral' in sentiment\_label.lower():  
 return 'NEUTRAL'  
  
 # If sentence is about Israel  
 if sentence\_type == 'pro-israeli':  
 return 'POS' if sentiment\_label.lower() in ['positive', 'pos'] else 'NEG'  
 # If sentence is about Palestine  
 elif sentence\_type == 'pro-palestinian':  
 return 'POS' if sentiment\_label.lower() in ['positive', 'pos'] else 'NEG'  
  
 return 'NEUTRAL'  
  
def analyze\_sentence(sentence, sentence\_type, models):  
 results = {}  
 for name, model in models.items():  
 try:  
 prediction = model(sentence)[0]  
 results[f"{name}\_score"] = prediction['score']  
 results[f"{name}\_label"] = interpret\_sentiment(  
 prediction['label'],  
 prediction['score'],  
 sentence\_type  
 )  
 except Exception as e:  
 print(f"Error with {name} on sentence: {sentence[:50]}... Error: {e}")  
 results[f"{name}\_score"] = None  
 results[f"{name}\_label"] = None  
 return results  
  
def get\_majority\_decision(row):  
 """Get the majority decision across all models"""  
 labels = [v for k, v in row.items() if '\_label' in k and v is not None]  
 if not labels:  
 return 'UNKNOWN'  
  
 from collections import Counter  
 count = Counter(labels)  
 return count.most\_common(1)[0][0]

As you can see, the idea behind is for every sentence we will use all the models to vote on the sentiment of the sentence. We will then use the majority vote to determine the sentiment of the sentence. We will also extract the score + label from each model as instructed.

# Process each sentence  
results = []  
for idx, row in tqdm(df.iterrows(), total=len(df)):  
 result = {  
 'newspaper': row['id'].split('\_')[0],  
 'article\_id': row['id'],  
 'sentence': row['sentence'],  
 'type': row['type']  
 }  
  
 # Add model predictions  
 result.update(analyze\_sentence(row['sentence'], row['type'], models))  
  
 # Add to results  
 results.append(result)  
  
# Create final DataFrame  
output\_df = pd.DataFrame(results)  
  
# Add majority decision  
output\_df['majority\_decision'] = output\_df.apply(get\_majority\_decision, axis=1)  
  
# Calculate average score for majority decision  
score\_columns = [col for col in output\_df.columns if '\_score' in col]  
output\_df['avg\_majority\_score'] = output\_df[score\_columns].mean(axis=1)  
  
print("Analysis complete!")

# Save to Excel  
output\_df.to\_excel('sentiment\_analysis\_results.xlsx', index=False)  
  
# Download the file (in Colab)  
from google.colab import files  
files.download('sentiment\_analysis\_results.xlsx')  
  
# Display some summary statistics  
print("\nSummary of results:")  
print("\nMajority decisions distribution:")  
print(output\_df['majority\_decision'].value\_counts())  
print("\nAverage scores by newspaper:")  
print(output\_df.groupby('newspaper')['avg\_majority\_score'].mean())

The output is:

Summary of results:  
  
Majority decisions distribution:  
majority\_decision  
NEG 652  
NEUTRAL 258  
POS 90  
Name: count, dtype: int64  
  
Average scores by newspaper:  
newspaper  
aj 0.754490  
bbc 0.744012  
jp 0.740172  
nyt 0.739272  
Name: avg\_majority\_score, dtype: float64

**NOTE** in this context positive mean pro-palestinian and negative means pro-israel meaning there is a clear bias in those articles.