**Report for Experiment 1**

In the first experiment, the main idea was to create a model that could take two headlines and check if they were similar semantically or not. In order to this, we first tried a couple of different approaches.

**Find same titles:**

In this first sub experiment a json document containing 10000 json objects was parsed and fed into a model to find if there were json objects that were copies of each other. For a specific json object , we would gather all its copies objectids and headline field values and store them as an array for the json objects field’s names as similar\_objectids and similar\_headlines. The below code snippet provides a general idea of the code and how it works:

#read json file

import io

import pandas as pd

from google.colab import files

uploaded = files.upload()

df = pd.read\_json(io.StringIO(uploaded.get('article\_profile\_business\_10000.json').decode('utf-8')))

df.head()

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

import nltk

import json

import re

nltk.download('stopwords')

nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()

stop\_words = set(stopwords.words('english'))

def preprocess(text):

    text = text.lower()

    text = re.sub(r'\d+', '', text)  # Remove numbers

    text = re.sub(r'\b\d+\b', '', text)  # Remove standalone numbers

    text = re.sub(r'\s+', ' ', text)  # Replace multiple spaces with single space

    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation

    words = text.split()

    words = [lemmatizer.lemmatize(word) for word in words if word not in stop\_words]

    return ' '.join(words)

# Extract and preprocess titles

titles = df['title']

processed\_titles = titles.apply(preprocess)  # Apply preprocessing to each title

# Vectorize the processed titles

vectorizer = TfidfVectorizer().fit\_transform(processed\_titles)

vectors = vectorizer.toarray()

# Calculate cosine similarity matrix

cosine\_sim\_matrix = cosine\_similarity(vectors)

# Remove similar headlines

threshold = 0.8  # Similarity threshold

to\_remove = set()

duplicate\_info = {}

for i in range(len(df)):

    for j in range(len(df)):

        if j != i:

          if cosine\_sim\_matrix[i, j] > threshold:

            if i not in duplicate\_info:

              duplicate\_info[i] = {'titles': [], 'json\_objectids': []}

            duplicate\_info[i]['titles'].append(df.iloc[j]['title'])

            duplicate\_info[i]['json\_objectids'].append(df.iloc[j]['\_id'])

df['duplicate\_text\_json'] = None

df['duplicate\_text\_json\_objectid'] = None

for index, info in duplicate\_info.items():

    df.at[index, 'duplicate\_text\_json'] = json.dumps(info['titles'])

    df.at[index, 'duplicate\_text\_json\_objectid'] = json.dumps(info['json\_objectids'])

df

# Filter the dataset

#cleaned\_data = df.drop(df.index[list(to\_remove)]).to\_dict('records')

import json

# Convert cleaned\_data to JSON

json\_data = df.to\_json('experiment1.json', orient='records', lines=True)

print("JSON file 'experiment1.json' has been created and is ready for download.")

**Entity extraction:**

The second approach was to extract any entities from the headlines label them accordingly. The idea was that this may give us a better way of determining if two headlines were similar or copies of each other.

#read json file

import io

import pandas as pd

from google.colab import files

uploaded = files.upload()

df = pd.read\_json(io.StringIO(uploaded.get('article\_profile\_business\_10000.json').decode('utf-8')))

df.head()

import spacy.cli

spacy.cli.download("en\_core\_web\_trf")

import spacy

nlp = spacy.load("en\_core\_web\_trf")

def extract\_entities(text):

    # Process the text using the NER model

    doc = nlp(text)

    # Extract the entities and their types

    entities = []

    for ent in doc.ents:

        entities.append({

                "text": ent.text,

                "type": ent.label\_

        })

    return entities

# Apply the extract\_entities function to the 'title' column and store the results in a new column

df['entities\_and\_their\_type'] = df['title'].apply(extract\_entities)

# Display the updated DataFrame

df.head()

import json

# Convert cleaned\_data to JSON

json\_data = df.to\_json('experiment2.json', orient='records', lines=True)

print("JSON file 'experiment2.json' has been created and is ready for download.")

**Result:**

What we observed was that entity extraction would require a model to be trained on a greater dataset, require more time and internal resources and would not be an affordable way to determine if two headlines are similar or not. Thus we decided to take the approach of sentence embeddings by choosing a better model and doing pre and post text processing methods as well as threshold value tuning.

**Final Model:**

This final model used a pre trained model from HuggingFace as well as various pre and post text processing methods to correctly identify if two headlines were similar or not. The final accuracy of this model was 80 %.

# Read JSON file

import pandas as pd

from google.colab import files

uploaded = files.upload()

# Save the uploaded file to a local file

with open('output\_file\_prompt3.csv', 'wb') as f:

    f.write(uploaded['output\_file\_prompt31.csv'])

# Read the local file into a pandas DataFrame

df = pd.read\_csv('output\_file\_prompt31.csv')

df = df.drop(df.index[0])

df = df.drop(df.columns[[0, -1]], axis=1)

df3 = df[:929]

import nltk

nltk.download('punkt')

nltk.download('stopwords')

pip install sentence-transformers

from sentence\_transformers import SentenceTransformer

from tqdm import tqdm

import pandas as pd

from scipy import spatial

import re

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

# Load the most accurate Sentence Transformer model

model = SentenceTransformer('multi-qa-MiniLM-L6-cos-v1')

# Function to preprocess text

def preprocess\_text(text):

    # Remove non-alphanumeric characters and convert to lowercase

    text = re.sub(r'[^a-zA-Z0-9\s]', '', text.lower())

    # Tokenize the text

    tokens = word\_tokenize(text)

    # Remove stopwords

    stop\_words = set(stopwords.words('english'))

    tokens = [word for word in tokens if word not in stop\_words]

    # Join the tokens back into a single string

    preprocessed\_text = ' '.join(tokens)

    return preprocessed\_text

# Function to calculate similarity using Sentence Transformer embeddings

def sentence\_transformer\_similarity\_model(headline1, headline2):

    # Preprocess the headlines

    headline1 = preprocess\_text(headline1)

    headline2 = preprocess\_text(headline2)

    # Encode the preprocessed headlines

    embeddings = model.encode([headline1, headline2])

    # Calculate the cosine similarity between the embeddings

    similarity\_score = 1 - spatial.distance.cosine(embeddings[0], embeddings[1])

    return similarity\_score

# Update the dofunc function to use sentence\_transformer\_similarity\_model

def dofunc(df):

    # Iterate through the DataFrame and set the last column based on similarity

    for idx, row in tqdm(df.iterrows()):

        headline1 = row['Unnamed: 1']

        headline2 = row['Unnamed: 2']

        similarity = sentence\_transformer\_similarity\_model(headline1, headline2)

        df.at[idx, 'Unnamed: 4'] = similarity >= 0.73

dofunc(df3)

def extract\_month(text):

    # Regular expression to extract month names (full and abbreviated)

    months = {

        'january': 1, 'february': 2, 'march': 3, 'april': 4,

        'may': 5, 'june': 6, 'july': 7, 'august': 8,

        'september': 9, 'october': 10, 'november': 11, 'december': 12,

        'jan': 1, 'feb': 2, 'mar': 3, 'apr': 4,

        'may': 5, 'jun': 6, 'jul': 7, 'aug': 8,

        'sep': 9, 'oct': 10, 'nov': 11, 'dec': 12

    }

    pattern = r'\b(?:' + '|'.join(months.keys()) + r')\b'

    match = re.search(pattern, text.lower())

    if match:

        return months[match.group(0)]

    return None

def extract\_date(text):

    # Regular expression to extract dates in the format "Month Day"

    pattern = r'\b(?:january|february|march|april|may|june|july|august|september|october|november|december)\b\s+\d+'

    match = re.search(pattern, text.lower())

    if match:

        return match.group(0)

    return None

def dofunc1(df):

    for idx, row in tqdm(df.iterrows(), total=len(df)):

        headline1 = row['Unnamed: 1']

        headline2 = row['Unnamed: 2']

        # Extract months from headlines

        month1 = extract\_month(headline1)

        month2 = extract\_month(headline2)

        date1 = extract\_date(headline1)

        date2 = extract\_date(headline2)

        # Check if both headlines contain months and if they are the same

        if month1 is not None and month2 is not None and month1 != month2:

            df.at[idx, 'Unnamed: 4'] = False

        if date1 is not None and date2 is not None and date1 != date2:

            df.at[idx, 'Unnamed: 4'] = False

dofunc1(df3)

true\_labels = df3['Unnamed: 3']

true\_labels = true\_labels.str.strip().replace({'True': True, 'False': False})

predicted\_labels = df3['Unnamed: 4']

predicted\_labels =  predicted\_labels.astype(bool)

from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score, f1\_score

# Calculate the confusion matrix

conf\_matrix = confusion\_matrix(true\_labels, predicted\_labels)

# Display the confusion matrix

confusion\_matrix\_df = pd.DataFrame(conf\_matrix, index=['Actual Negative', 'Actual Positive'], columns=['Predicted Negative', 'Predicted Positive'])

print(confusion\_matrix\_df)

# Compute accuracy

accuracy = accuracy\_score(true\_labels, predicted\_labels)

# Compute precision

precision = precision\_score(true\_labels, predicted\_labels)

# Compute recall

recall = recall\_score(true\_labels, predicted\_labels)

# Compute F1 score

f1 = f1\_score(true\_labels, predicted\_labels)

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1 Score:", f1)

Predicted Negative Predicted Positive

Actual Negative 410 64

Actual Positive 119 336

Accuracy: 0.8030139935414424

Precision: 0.84

Recall: 0.7384615384615385

F1 Score: 0.7859649122807019

This model will be used in addition to other models and functions to label and sort out similar news headlines and json objects and will be deployed in real time environment.