Tal Melnikov, 206961377

Omri Chen, 31148720

**Methodology and Results**

**Structure of the Datasets**

BBC Dataset (bbc.csv):

Entries: 835 rows

Columns:

link: URL of the tweet

text: Content of the tweet

date: Timestamp of the tweet

No\_of\_Likes: Number of likes

No\_of\_Retweets: Number of retweets

CNN Dataset (cnn.csv):

Entries: 850 rows

Columns: Same as BBC

This part is used to process and analyze a text corpus using the spaCy library, which is a powerful tool for Natural Language Processing (NLP) in Python. The code includes functions for processing the corpus, tokenizing sentences, and lemmatizing tokens while filtering out stop words.

**'process corpus'**:  
This function processes a list of sentences (corpus) using the spaCy language model.

Parameters:

nlp:  
This is the spaCy language model that you've loaded (en\_core\_web\_sm in this case).  
It is used to process the text.

corpus:  
This is a list of sentences (strings) that you want to analyze.

The function returns a list of Doc objects, where each Doc object corresponds to a processed sentence in the corpus. These Doc objects contain tokens and other linguistic features like parts of speech, named entities, etc.

**'tokenize\_sentences':**  
This function tokenizes the sentences in the processed corpus and filters out punctuation, stop words, and spaces.

Parameters:

processed\_corpus:  
A list of Doc objects, which are the processed sentences returned by the process\_corpus function.

The function returns A list of lists, where each inner list contains Token objects for a sentence. These tokens exclude punctuation marks, stop words (common words like "and," "the," etc.), and whitespace characters. This results in a more meaningful set of tokens for analysis.

**'lemmatize\_tokens':**  
This function lemmatizes the tokens, converting them to their base or dictionary form (e.g., "running" to "run"), and filters out any tokens that are stop words.

Parameters:

tokens:  
A list of lists, where each inner list contains Token objects from the  
tokenize\_sentences function.

stop\_words:  
(Optional) A set of stop words to be filtered out. If not provided, the function uses spaCy's built-in stop words.

tf\_idf:  
Calculates the Term Frequency-Inverse Document Frequency (TF-IDF) scores for words in a corpus and returns a sorted DataFrame of the most important words.

Process:

Vectorizer Initialization:  
Initializes a TfidfVectorizer with English stop words removed.

Fit and Transform:  
Converts the corpus into a TF-IDF matrix.

Feature Names:  
Retrieves the words (features) used in the matrix.

TF Score Calculation:  
Sums the term frequency (TF) scores across all documents.

Create DataFrame:  
Puts the words and their TF scores into a DataFrame, sorted by score in descending order.

The function returns A sorted DataFrame of words and their corresponding TF scores, highlighting the most important terms in the corpus.

**'plot\_tfidf\_bar\_chart'**  
Creates a bar chart to visualize and compare the top TF-IDF scores of words from  
two different DataFrames.

Parameters:

tf\_df1:  
The first DataFrame containing words and their TF scores.

tf\_df2:  
The second DataFrame containing words and their TF scores.

top\_n:  
(Optional) The number of top words to display from each DataFrame. Defaults to 10.

Process:

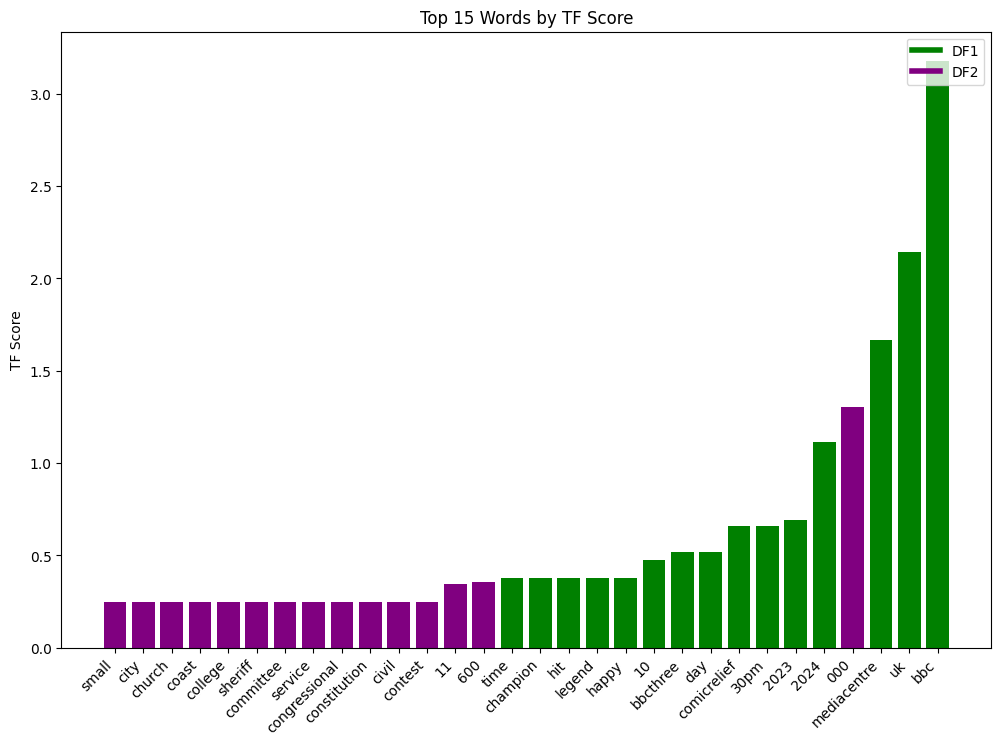
Extract Top Words:  
Retrieves the top top\_n words from each DataFrame.

Combine Data:  
Concatenates the top words from both DataFrames into a single DataFrame, with a new column indicating the source (DF1 or DF2).

Sort for Visualization:  
Sorts the combined DataFrame by TF score for better plotting.

Create Bar Chart:  
Plots a bar chart using different colors (green for DF1, purple for DF2) to represent the top words and their TF scores.

Add Legend:  
Adds a legend to distinguish between the two sources.

Display Chart:  
Shows the bar chart with word labels rotated for readability.

**'plot\_word\_clouds'**  
Generates and displays word clouds for the top words from two different DataFrames based on their TF-IDF scores.

Parameters:

tf\_df1:  
The first DataFrame containing words and their TF scores.

tf\_df2:  
The second DataFrame containing words and their TF scores

top\_n:  
(Optional) The number of top words to display in each word cloud. Defaults to 5.

Process:

Extract Top Words:  
Retrieves the top top\_n words from each DataFrame.

Create Word Frequency Dictionaries:  
Converts the top words and their TF scores into dictionaries for word cloud generation.

Generate Word Clouds:

Creates two word clouds:

wordcloud1 for tf\_df1 with a green color scheme.

wordcloud2 for tf\_df2 with a purple color scheme.

Both word clouds have a white background, black contour borders, and a contour width of 3.

Plot Word Clouds:

Displays the word clouds side by side in a single figure:

The first subplot shows the word cloud for tf\_df1.

The second subplot shows the word cloud for tf\_df2.

A close up of a logo

Description automatically generated

**'Get\_top\_frequent\_word':**  
Trains a Word2Vec model on a list of tokenized text and retrieves the top N  
most frequent words.

Parameters:

token\_list:  
A list of tokenized documents, where each document is a list of tokens.

n:  
(Optional) The number of top words to retrieve. Defaults to 10.

vector\_size:  
(Optional) The dimensionality of the word vectors. Defaults to 100.

window:  
(Optional) The maximum distance between the current and predicted word within a sentence. Defaults to 5.

min\_count:  
(Optional) Ignores words with a total frequency lower than this. Defaults to 1.

workers:  
(Optional) The number of worker threads to train the model. Defaults to 4.

Process:

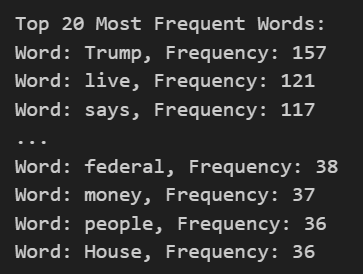
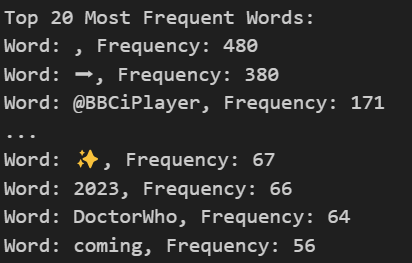
Train Word2Vec Model:  
Trains a Word2Vec model using the tokenized text from token\_list.

Extract Word Frequencies:  
Retrieves the frequency counts of words from the model's vocabulary.

Get Top N Words:  
Sorts the words by their frequency in descending order and selects the top N most frequent words.

The function returns a list of tuples, where each tuple contains a word and its frequency.

The model returns The trained Word2Vec model.

CNN results BBC results

**'autoencoder\_significance\_analysis'**  
Analyzes the significance of words in a corpus by calculating reconstruction errors using an autoencoder trained on word embeddings. Words with higher reconstruction errors are considered potentially more significant.

Parameters:

w2v\_model:  
A trained Word2Vec model used to obtain word embeddings.

tokenized\_corpus:  
A list of tokenized documents, where each document is a list of tokens.

top\_n:  
(Optional) The number of top significant words to display. Defaults to 10.

Process:

Create Word Embeddings:

Extracts and stores word embeddings from the Word2Vec model for each unique word in the corpus.

Prepare Embedding Matrix:

Converts the collected word embeddings into a NumPy array (embedding\_matrix) for processing.

Define and Train Autoencoder:

Constructs an autoencoder with an input layer, an encoding layer (32 dimensions), and a decoding layer.

Trains the autoencoder on the embedding\_matrix to reconstruct the word embeddings.

Calculate Reconstruction Errors:

Predicts the reconstructed embeddings using the trained autoencoder.

Calculates the mean squared reconstruction error for each word.

Identify Significant Words:

Sorts words by their reconstruction error in descending order. Higher errors suggest that the word may be more significant.

Displays the top top\_n words with the highest reconstruction errors.

**'compare\_results':**compares the top words from TF-IDF and autoencoder analysis for BBC and CNN datasets, and visualizes the results with word clouds

Parameters:

bbc\_significance:  
Top significant words from BBC (autoencoder).

cnn\_significance:  
Top significant words from CNN (autoencoder).

bbc\_tf\_df:  
BBC words and TF scores DataFrame.

cnn\_tf\_df:  
CNN words and TF scores DataFrame.

top\_n:  
(Optional) Number of top words to compare. Defaults to 10.

Process:

Extract Top Words:  
Retrieves the top top\_n words from TF-IDF and autoencoder for both datasets.

Compare Words:  
Identifies common and unique words between TF-IDF and autoencoder for BBC and CNN.

Print Results:  
Outputs common and unique words for both datasets.

Visualize with Word Clouds:  
Generates and displays word clouds for the top words from both methods (TF-IDF and autoencoder) for BBC and CNN.



**'print\_NER':**  
Processes a text corpus using a spaCy model and prints named entities (NER) along with their labels and explanations.

Parameters:

corpus:  
A list of text documents (strings) to be analyzed.

nlp\_model:  
A spaCy language model for processing the text and identifying entities.

Process:

Process Corpus:  
Applies the nlp\_model to each document in the corpus to extract named entities.

Print Named Entities:  
For each document:

Displays the document number.

Prints each entity's text, its NER label, and an explanation of the label.

**'plot\_hourly\_metrics':**  
Visualizes the average number of tweets, likes and retweets per hour  
from a given DataFrame.

Parameters:

df:  
The DataFrame containing the tweet data, including columns for the date, likes, and retweets.

df\_name:  
A string representing the name of the DataFrame, used in the plot title.

Process:

Date Conversion:  
Converts the 'date' column to a datetime format. Drops any rows where date conversion fails.

Column Check:  
Ensures that the DataFrame contains the necessary columns ('No\_of\_Likes' and 'No\_of\_Retweets').

Extract Hour:  
Extracts the hour from the 'date' column to analyze activity by time of day.

Calculate Averages:  
Computes the average number of likes, retweets, and posts per hour.

Reorder Hours:  
Adjusts the order of hours for better visual representation (starting from 1 AM and ending with midnight).

Plot Data:  
Plots a bar chart showing the average number of likes, retweets, and posts per hour, using different colors for each metric.

A graph with green and blue lines

Description automatically generated

A graph with green and blue lines

Description automatically generated

**'sent\_analysis':**  
Analyzes and compares the sentiment (positive, negative, neutral) in two text corpora and visualizes the results with a bar chart.

Parameters:

corpus1:  
A list of text documents (strings) representing the first user's corpus.

user1:  
A string representing the name of the first user (used in the plot).

corpus2:  
A list of text documents (strings) representing the second user's corpus.

user2:  
A string representing the name of the second user (used in the plot).

Process:

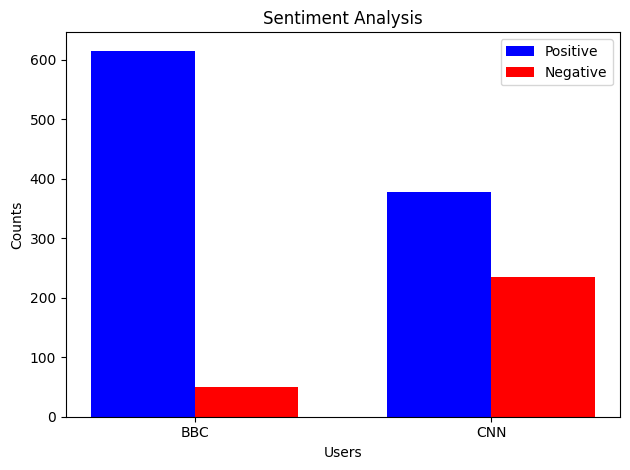
Sentiment Analysis:  
Uses the analyze\_sentiment function to determine the sentiment of each text in both corpora.

Count Sentiments:  
Counts the number of positive and negative sentiments in each corpus.

Plot Results:

Creates a bar chart comparing the positive and negative sentiment counts between the two users.

Each user's sentiments are represented with bars of different colors (blue for positive, red for negative).



**'summarize\_text':**Generates a summary of a given text using a specified pre-trained model.

Parameters:

text:  
The input text to be summarized.

model\_name:  
(Optional) The name of the pre-trained model to use. Defaults to 'facebook/bart-large-cnn'.

max\_length:  
(Optional) The maximum length of the summary. Defaults to 150 words.

min\_length:  
(Optional) The minimum length of the summary. Defaults to 30 words.

device:  
(Optional) The device to run the model on (-1 for CPU, 0 for GPU). Defaults to -1.

Process:

Summarizer Pipeline:  
Initializes the summarization pipeline using the specified model.

Adjust Lengths:  
Adjusts max\_length and min\_length based on the input text length to ensure the summary is appropriate.

Generate Summary:  
Generates and returns a summary of the text using the pipeline.

Returns:  
The summarized text.

**'summarize\_dataframe':**  
Applies text summarization to a specified column in a DataFrame and adds the summaries as a new column.

Parameters:

df:  
The DataFrame containing the text data.

text\_column:  
The name of the column in the DataFrame containing the text to be summarized.

model\_name:  
(Optional) The name of the pre-trained model to use.  
Defaults to 'facebook/bart-large-cnn'.

max\_length:  
(Optional) The maximum length of the summary. Defaults to 150 words.

min\_length:  
(Optional) The minimum length of the summary. Defaults to 30 words.

device:  
(Optional) The device to run the model on (-1 for CPU, 0 for GPU). Defaults to -1.

Process:

Apply Summarization:  
Applies the summarize\_text function to each entry in the specified text column and stores the results in a new column called 'summary'

This Prepares text data for training a character-level Recurrent Neural Network (RNN) and builds an RNN model using LSTM layers to predict the next character in a sequence.

Text Preprocessing and Tokenization:

Text Cleaning: Converts text to lowercase, removes extra spaces and punctuation.

Tokenization: Tokenizes text at the character level, converting characters into integer sequences.

Sequence Creation: Generates input (X) and target (y) sequences for training, where each X[i] is a sequence of 40 characters, and y[i] is a one-hot encoded vector representing the next character.

Model Building:

Embedding Layer: Maps input characters (encoded as integers) into dense 128-dimensional vectors.

LSTM Layers: Two LSTM layers with 128 units each:

The first LSTM layer returns sequences, feeding into the second LSTM layer.

The second LSTM layer outputs the final hidden state, used for prediction.

Dense Layer: Outputs a probability distribution over all possible characters using the softmax activation function.

Compilation and Training:

Compile: Uses the Adam optimizer and categorical crossentropy loss, suitable for multi-class classification.

Train: Trains the model for 100 epochs with a batch size of 64.

**'generate\_text':**Generates text character-by-character based on a seed input using a trained RNN model.

Parameters:

seed\_text:  
Initial string to start generating text.

num\_chars:  
Number of characters to generate.

Process:

Initialize:  
Begins with the seed\_text as the starting point for generation.

Generate Characters:

Iteratively generates the next character based on the current seed\_text.

Converts the seed\_text into a sequence, pads it to the required seq\_length, and reshapes it for the model.

Predicts the next character using the trained model, adds it to the generated\_text, and updates the seed\_text for the next prediction.

Return Result:

Returns the full generated\_text after generating the specified number of characters.

**GPT MODEL:**

Process:

Load GPT-2 Model and Tokenizer:

GPT2Tokenizer and GPT2LMHeadModel: Load the tokenizer and model from the pre-trained GPT-2.

Set Pad Token: Sets the padding token to the end-of-sequence (EOS) token, which is used to ensure that all sequences have the same length.

**'encode\_texts':**

Encodes a list of texts into token sequences suitable for GPT-2 input.

Parameters:

texts:  
List of text strings to encode.

tokenizer:  
The GPT-2 tokenizer.

max\_length:  
(Optional) Maximum length of the encoded sequences. Defaults to 512 tokens.

Process:

Encodes each text in texts, truncating or padding it to max\_length tokens.

Returns a list of encoded sequences.

Encode Corpus:

Encodes the combain\_corpus using the encode\_texts function with the GPT-2 tokenizer.

**'generate\_text\_1'**:  
Generates text using GPT-2 based on a seed text, with control over the length and number of output sequences.

Parameters:

seed\_text:  
The initial text to start the generation.

max\_length:  
(Optional) Maximum length of the generated text, including the seed text. Defaults to 100 characters.

num\_return\_sequences:  
(Optional) Number of different text sequences to generate. Defaults to 1.

Process:

Encode Seed Text:

Converts the seed\_text into input IDs using the GPT-2 tokenizer.

Generate Text:

Model Generation:  
Uses the GPT-2 model to generate text, considering the following parameters:

max\_length:  
The total length of the generated text, including the seed text.

num\_return\_sequences:  
Specifies how many different text sequences to generate.

Diversity Controls:

no\_repeat\_ngram\_size: Ensures that no n-grams are repeated within the generated text.

top\_k and top\_p: Parameters to control randomness and diversity in the generation process.

temperature: Controls the randomness of predictions by scaling the logits before applying softmax.

pad\_token\_id: Uses the end-of-sequence (EOS) token as the padding token.