Copy of Refactored Content Analysis

February 4, 2024

1 A computational Literature Review of Health psychology Intervention Development

1.1 Storm Hiskens-Ravest - 218685427

1.1.1 SIT723 Thesis

1.2 Mount Google Drive

[1]:
```
# Google Colab
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

1.3 Install Libraries

[2]:
```
%%capture
!git clone https://github.com/jupyter/nbconvert.git
!cd nbconvert

!pip install -e .

!apt-get install pandoc

!apt-get update
!apt-get install inkscape
!add-apt-repository --yes universe
!add-apt-repository --yes ppa:inkscape.dev/stable
!apt-get update
!apt-get install -y inkscape

!apt-get update
!apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-generic␣
↪texlive-latex-extra -y

!pip install optimization
!pip install octis
!pip install openai
```

```
!pip install PyPDF2
!pip install keybert
!pip install bertopic
!pip install tiktoken
!pip install transformers
!pip install pyspellchecker
!pip install sentence-transformers
!pip install --upgrade typing_extensions
```

[238]:
```
# Data Manipulation
import numpy as np
import pandas as pd

# PDF Manipulation
import PyPDF2

# Text Processing and Regular Expressions
import re
import string

# Machine Learning
import umap
from umap import UMAP
import hdbscan
from hdbscan import HDBSCAN

# Data Visualisation
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
from typing import List, Union
import plotly.graph_objects as go
from sklearn.preprocessing import normalize

# Natural Language Processing (NLP)
import nltk
from keybert import KeyBERT
from nltk.corpus import stopwords
from spellchecker import SpellChecker
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
nltk.download('punkt')
nltk.download('stopwords')
from transformers.pipelines import pipeline
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
```

```
from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTransformer,␣
 ↪CountVectorizer

# Topic Modeling
from bertopic import BERTopic
from bertopic.representation import KeyBERTInspired
from bertopic.vectorizers import ClassTfidfTransformer

# APIs and External Tools Integration
import openai
from bertopic.representation import OpenAI

# Topic Modeling Evaluation
from octis.dataset.dataset import Dataset
from octis.evaluation_metrics.coherence_metrics import Coherence
from octis.evaluation_metrics.diversity_metrics import TopicDiversity

# Miscellaneous
import os
import csv
import torch
import tiktoken
from tqdm import tqdm
from typing import Tuple
from IPython.display import display
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

2 Define Classes

[254]:
```
class PDFProcessor:
    def __init__(self, pdf_directory, txt_directory):
        self.pdf_directory = pdf_directory
        self.txt_directory = txt_directory
        os.makedirs(self.txt_directory, exist_ok=True)

    def extract_text_from_pdf(self, pdf_file_path):
        with open(pdf_file_path, 'rb') as file:
            pdf_reader = PyPDF2.PdfReader(file)
            text = ''.join([page.extract_text() + '\n' for page in pdf_reader.
 ↪pages if page.extract_text()])
        return text
```

```
    def save_text_to_file(self, text, filename):
        output_path = os.path.join(self.txt_directory, filename.replace('.pdf',␣
↪'.txt'))
        with open(output_path, 'w', encoding='utf-8') as file:
            file.write(text)

    def process_pdfs(self):
        pdf_files = [f for f in os.listdir(self.pdf_directory) if f.endswith('.
↪pdf')]
        for pdf_file in tqdm(pdf_files, desc="Processing PDFs"):
            pdf_file_path = os.path.join(self.pdf_directory, pdf_file)
            try:
                text = self.extract_text_from_pdf(pdf_file_path)
                self.save_text_to_file(text, pdf_file)
            except Exception as e:
                print(f"Error processing {pdf_file}: {e}")

class TextPreprocessor:
    def __init__(self, remove_punctuation=True, punctuation=string.punctuation,
                 stopword_list=None, min_chars=2, min_words_docs=1, min_df=0.1,
                 max_df=0.8, spellchecker=None):
        self.remove_punctuation = remove_punctuation
        self.punctuation = punctuation
        self.stopword_list = set(stopwords.words('english')) if stopword_list␣
↪is None else stopword_list
        self.min_chars = min_chars
        self.min_words_docs = min_words_docs
        self.min_df = min_df
        self.max_df = max_df
        self.spell = spellchecker or SpellChecker()

    def extract_relevant_section(self, text):
        """
        Extracts text between 'abstract' and 'references'.
        """
        pattern = r"\babstract\b(.*?)\breferences\b"
        match = re.search(pattern, text, flags=re.DOTALL | re.IGNORECASE)
        if match:
            return match.group(1).strip()  # Extract only the matching group
        else:
            return text  # Return original text if no match found

    def split_merged_words(self, text):
        # Split words that are incorrectly merged
        words = text.split()
        new_text = []
        for word in words:
```

```
            if self.spell.unknown([word]):
                for i in range(1, len(word)):
                    part1, part2 = word[:i], word[i:]
                    if not self.spell.unknown([part1]) and not self.spell.
↪unknown([part2]):
                        new_text.extend([part1, part2])
                        break
                else:
                    new_text.append(word)
            else:
                new_text.append(word)
        return ' '.join(new_text)

    def recombine_split_words(self, text):
        # Logic to recombine incorrectly split words
        words = text.split()
        new_words = []
        temp_word = ''
        for word in words:
            if len(word) == 1 and word.isalpha():
                temp_word += word
            else:
                if temp_word:
                    new_words.append(temp_word)
                    temp_word = ''
                new_words.append(word)
        if temp_word:
            new_words.append(temp_word)
        return ' '.join(new_words)

    def preprocess_text(self, text):
        # Initial encoding to ASCII and decoding
        text = text.encode('ascii', 'ignore').decode('ascii')

        # Remove extra spaces introduced by encoding-decoding
        text = re.sub(r'(?<=\b\w) (?=\w\b)', '', text)
        extracted_text = self.extract_relevant_section(text)

        # Normalize text, remove punctuation, etc.
        # Lowercase conversion moved up before extracting relevant section
        text = text.lower()
        text = text.translate(str.maketrans('', '', self.punctuation))

        # Split into words and remove stopwords
        words = text.split()
        words = [word for word in words if word not in self.stopword_list and␣
↪len(word) >= self.min_chars]
```

```
        text = ' '.join(words)

        # Apply the same pre-processing steps to the extracted text
        extracted_text = extracted_text.lower()
        extracted_text = extracted_text.translate(str.maketrans('', '', self.
↪punctuation))
        extracted_words = extracted_text.split()
        extracted_words = [word for word in extracted_words if word not in self.
↪stopword_list and len(word) >= self.min_chars]
        extracted_text = ' '.join(extracted_words)

        # Additional normalization and cleaning for extracted text
        extracted_text = re.sub(r'-\n+', '', extracted_text)
        extracted_text = extracted_text.encode('ascii', 'ignore').
↪decode('ascii')
        extracted_text = re.sub(r'\s+', ' ', extracted_text)
        extracted_text = re.sub(r'http\S+|www\.\S+', '', extracted_text)
        extracted_text = re.sub(r'\[.*?\]|\(.*?\)|\{.*?\}', '', extracted_text)

        # Replace specific characters
        replace_dict = {'&': 'and', ' ': 'ffi', ' ': 'ff', ' ': 'fi', ' ': 'fl'}
        for key, val in replace_dict.items():
            extracted_text = extracted_text.replace(key, val)

        # Remove all punctuation, including periods
        punctuation_to_remove = string.punctuation
        extracted_text = extracted_text.translate(str.maketrans('', '',␣
↪punctuation_to_remove))
        extracted_text = extracted_text.lower().translate(str.maketrans('', '',␣
↪string.digits))

        # Remove specific words and phrases
        phrases_to_remove = [#r'\bimplementation\b',
                             #r'\bstudy\b',
                             #r'\bresearch\b'
                             #r'\bhealth\b',
                             r'\bcopyright\b',
                             r'\bamerican\b',
                             r'\bpsychological\b',
                             r'\bassociation\b',
                             r'\bauthor\b',
                             #r'\bintervention\b',
                             #r'\binterventions\b',
                             #r'\bpsychology\b',
                             r'\bissn\b',
                             r'\bprint\b',
                             r'\bbackground\b',
```

```
                                r'\bfull\b',
                                r'\bterms\b',
                                r'\bconditions\b',
                                r'\baccess\b',
                                r'\buse\b',
                                r'\bfound\b',
                                r'\breview\b',
                                r'\bjournal\b']

        for phrase in phrases_to_remove:
            extracted_text = re.sub(phrase, '', extracted_text, flags=re.
↪IGNORECASE)

        return extracted_text

    def process_documents(self, docs, filenames, preprocessed_txt_path):
        for i in tqdm(range(len(docs)), desc="Processing Documents"):
            preprocessed_text = self.preprocess_text(docs[i])
            filename = filenames[i]
            output_path = os.path.join(preprocessed_txt_path, filename)
            with open(output_path, 'w', encoding='utf-8') as file:
                file.write(preprocessed_text)

class DataLoader:
    """
    A class for loading text data from a directory and extracting metadata.

    Attributes:
        directory (str): The directory containing text files.
    """

    def __init__(self, directory):
        """
        Initialises a DataLoader object with the specified directory.

        Args:
            directory (str): The directory containing text files.
        """
        self.directory = directory

    def extract_year_from_filename(self, filename):
        """
        Extracts the year from a filename with a specific format.

        Args:
            filename (str): The filename from which to extract the year.
```

```
        Returns:
            str or None: The extracted year or None if not found.
        """
        match = re.search(r'_\d{4}_', filename)
        return match.group(0)[1:-1] if match else None

    def load_texts(self):
        """
        Loads text data from files in the specified directory.

        Returns:
            list of str: A list of text content from the loaded files.
        """
        texts = []
        file_list = [f for f in os.listdir(self.directory) if f.endswith('.
↪txt')]
        for filename in tqdm(file_list, desc='Loading files'):
            with open(os.path.join(self.directory, filename), 'r',␣
↪encoding='utf-8') as file:
                texts.append(file.read())
        return texts

    def create_corpus_file(self, output_dir, filename="corpus.tsv"):
        """
        Creates a corpus file in a format suitable for OCTIS.

        Args:
            output_dir (str): The directory to save the corpus file.
            filename (str): The name of the corpus file.
        """
        file_path = os.path.join(output_dir, filename)
        with open(file_path, 'w', encoding='utf-8') as file:
            file_list = [f for f in os.listdir(self.directory) if f.endswith('.
↪txt')]
            for filename in tqdm(file_list, desc='Creating corpus file'):
                year = self.extract_year_from_filename(filename)
                with open(os.path.join(self.directory, filename), 'r',␣
↪encoding='utf-8') as text_file:
                    content = text_file.read().strip()
                    file.write(f"{content}\t{year}\n")

    def create_vocabulary_file(self, output_dir, filename="vocabulary.txt"):
        """
        Creates a vocabulary file from the texts in the specified directory.

        Args:
            output_dir (str): The directory to save the vocabulary file.
```

```
            filename (str): The name of the vocabulary file.
        """
        vocab = collections.Counter()
        file_list = [f for f in os.listdir(self.directory) if f.endswith('.
↪txt')]
        for filename in tqdm(file_list, desc='Building vocabulary'):
            with open(os.path.join(self.directory, filename), 'r',␣
↪encoding='utf-8') as text_file:
                words = text_file.read().split()
                vocab.update(words)

        vocab_path = os.path.join(output_dir, filename)
        with open(vocab_path, 'w', encoding='utf-8') as vocab_file:
            for word in sorted(vocab):
                vocab_file.write(word + '\n')
    def write_documents_to_tsv(self, tsv_path, documents):
        """
        Writes the documents to a TSV file.

        Args:
            tsv_path (str): The path to the TSV file where documents will be␣
↪written.
            documents (list of str): A list of documents to be written to the␣
↪TSV file.
        """
        with open(tsv_path, 'w', encoding='utf-8') as tsv_file:
            for idx, document in enumerate(documents):
                cleaned_document = document.replace('\t', ' ').replace('\n', '␣
↪')
                tsv_file.write(cleaned_document + '\n')
                if '\t' in document or '\n' in document:
                    print(f"Warning: Document at index {idx} contained tabs or␣
↪newlines and was cleaned.")

class EmbeddingModel:
    """
    A class for generating embeddings from text data using a pre-trained model.

    Attributes:
        model_name (str): The name of the pre-trained embedding model.
    """

    def __init__(self, model_name):
        """
        Initializes an EmbeddingModel object with the specified pre-trained␣
↪model name.
```

```
        Args:
            model_name (str): The name of the pre-trained embedding model.
        """
        self.model = SentenceTransformer(model_name)

    def generate_embeddings(self, documents, is_sentences=False):
        """
        Generates embeddings for a list of documents or sentences.

        Args:
            documents (list of str): The list of documents or sentences to␣
↪generate embeddings for.
            is_sentences (bool): True if the input is a list of sentences,␣
↪False if it's a list of documents.

        Returns:
            list of numpy.ndarray: A list of embeddings for the input documents␣
↪or sentences.
        """
        if is_sentences:
            sentences = [sentence for doc in documents for sentence in␣
↪sent_tokenize(doc)]
            return self.model.encode(sentences, show_progress_bar=True)
        return self.model.encode(documents, show_progress_bar=True)

class UMAPDimensionalityReduction:
    def __init__(self, n_neighbors, n_components, min_dist, metric,␣
↪random_state=42):
        self.umap_model = UMAP(n_neighbors=n_neighbors,
                               n_components=n_components,
                               min_dist=min_dist,
                               metric=metric,
                               random_state=random_state)

    def fit_transform(self, embeddings):
        return self.umap_model.fit_transform(embeddings)


    def fit_hdbscan_model(self, embeddings):
        """
        Fits the HDBSCAN clustering model to embeddings.

        Args:
            embeddings (list of numpy.ndarray): The embeddings on which HDBSCAN␣
↪will perform clustering.
```

```
        Returns:
            HDBSCAN: The fitted HDBSCAN clustering model.
        """
        return self.hdbscan_model.fit(embeddings)

class HDBSCANClustering:
    def __init__(self, min_cluster_size, metric, cluster_selection_method,␣
 ↪prediction_data=True):
        self.hdbscan_model = HDBSCAN(min_cluster_size=min_cluster_size,
                                     metric=metric,
                                    ␣
 ↪cluster_selection_method=cluster_selection_method,
                                     prediction_data=prediction_data)

    def fit(self, embeddings):
        return self.hdbscan_model.fit(embeddings)

class TopicModeling:
    """
    A class for performing topic modeling using BERTopic and related components.

    Attributes:
        embedding_model: The embedding model for text data.
        umap_model: The UMAP dimensionality reduction model.
        hdbscan_model: The HDBSCAN clustering model.
    """
    def __init__(self, embedding_model, umap_model, hdbscan_model, top_n_words,␣
 ↪nr_topics):
        """
        Initializes a TopicModeling object with the specified components.

        Args:
            embedding_model: The embedding model for text data.
            umap_model: The UMAP dimensionality reduction model.
            hdbscan_model: The HDBSCAN clustering model.
            top_n_words (int): The number of top words for each topic in␣
 ↪BERTopic.
            nr_topics (int): The number of topics to extract in BERTopic.
        """
        self.vectorizer_model = CountVectorizer(stop_words="english",␣
 ↪ngram_range=(1, 3))
        self.representation_model = KeyBERT()
        self.topic_model = BERTopic(
            calculate_probabilities=True,
            embedding_model=embedding_model,
            umap_model=umap_model,
```

```
            hdbscan_model=hdbscan_model,
            vectorizer_model=self.vectorizer_model,
            representation_model=self.representation_model,
            top_n_words=top_n_words,
            nr_topics=nr_topics,
            verbose=True
        )

    def fit_transform(self, documents, embeddings=None):
        """
        Fits the BERTopic model to documents, optionally using precomputed␣
↪embeddings.

        Args:
            documents (list of str): The input documents.
            embeddings (list of numpy.ndarray, optional): Precomputed␣
↪embeddings for the input documents.

        Returns:
            tuple: A tuple containing topics and probabilities. If embeddings␣
↪are provided, uses those; otherwise, generates embeddings.
        """
        if embeddings is None:
            embeddings = self.model.encode(documents, show_progress_bar=True)
        topics, probabilities = self.topic_model.fit_transform(documents,␣
↪embeddings)
        return topics, probabilities

    def get_topic_info(self):
        """
        Gets information about topics including words and probabilities.

        Returns:
            pandas.DataFrame: A DataFrame containing topic information.
        """
        return self.topic_model.get_topic_info()

    def fit_hierarchical_topics(self, documents):
        """
        Fits hierarchical topics to the input documents.

        Args:
            documents (list of str): The input documents.

        Returns:
            str: A string representation of the hierarchical topics.
        """
```

```
        return self.topic_model.hierarchical_topics(documents)

    def topics_over_time(self, documents, timestamps, nr_bins):
        """
        Performs dynamic topic modeling.

        Args:
            documents (list of str): The input documents.
            timestamps (list of str): Timestamps corresponding to each document.
            topics (list of int): The topics assigned to each document.
            nr_bins (int): The number of bins to divide the timestamps.

        Returns:
            DataFrame: A DataFrame containing dynamic topics over time.
        """
        return self.topic_model.topics_over_time(documents, timestamps,␣
↪nr_bins=nr_bins)

    def reduce_outliers(self, documents, topics, probabilities=None,␣
↪strategy="probabilities"):

        """
        Reduces outliers using the specified strategy.

        Args:
            documents (list of str): The documents to process.
            topics (list of int): The initial topics assigned to each document.
            probabilities (list of float, optional): The probabilities of each␣
↪document belonging to its topic.
            strategy (str): The strategy to use for reducing outliers. Options␣
↪are 'probabilities', 'distributions',
                            'c-tf-idf', 'embeddings'.

        Returns:
            list of int: The updated topics with reduced outliers.
        """
        new_topics = []

        if strategy == "probabilities":
            if probabilities is None:
                raise ValueError("Probabilities must be provided for the␣
↪'probabilities' strategy.")
            new_topics = self.topic_model.reduce_outliers(documents, topics,␣
↪probabilities=probabilities, strategy=strategy)

        return new_topics
```

```
    def merge_topics(self, documents, topics_to_merge):
        """
        Merges topics in the BERTopic model.

        Args:
            documents (list of str): The documents used in topic modeling.
            topics_to_merge (list of list of int): A list where each element is␣
↪a list containing two topic numbers to be merged.
        """
        # Check if the BERTopic instance has the merge_topics method
        if hasattr(self.topic_model, 'merge_topics'):
            for topics_list in topics_to_merge:
                # Ensure there's more than one topic in the list to merge
                if len(topics_list) > 1:
                    # The first topic in the list is the target topic we merge␣
↪other topics into
                    target_topic = topics_list[0]
                    # Topics to be merged into the target topic, excluding the␣
↪target topic itself
                    topics_to_be_merged = topics_list[1:]

                    # Merge each topic into the target topic
                    for topic in topics_to_be_merged:
                        # BERTopic's merge_topics method might be expecting the␣
↪target topic and a single topic to merge at a time
                        # You may need to adjust this call depending on the␣
↪exact requirements of BERTopic's merge_topics method
                        self.topic_model.merge_topics(documents, [target_topic,␣
↪topic])
                else:
                    print(f"Invalid topics list for merging: {topics_list}")
        else:
            print("The BERTopic instance does not support topic merging.")

    def update_topics(self, documents, new_topics):
        """
        Updates the BERTopic model with new topic assignments.

        Args:
            documents (list of str): The documents used in topic modeling.
            new_topics (list of int): The new topic assignments for each␣
↪document.
        """
        self.topic_model.update_topics(documents, topics=new_topics,␣
↪n_gram_range=(1, 3))
```

```python
class TopicVisualisation:
    """
    A class dedicated to visualising topics and their relationships.
    """
    def __init__(self, topic_model):
        self.topic_model = topic_model

    def visualise_topics(self):
        """
        Visualises the topics generated by the topic model.

        Returns:
            plotly.Figure: A figure displaying the visualisation of topics.
        """
        topic_info = self.topic_model.get_topic_info()
        if topic_info.shape[0] > 1:  # Check if there are topics
            return self.topic_model.visualize_topics()
        else:
            print("No topics to visualize.")
            return None

    def visualise_barchart(self, top_n_topics):
        """
        Visualises a bar chart of the top topics generated by the topic model.

        Args:
            top_n_topics (int, optional): The number of top topics to visualise.

        Returns:
            plotly.Figure: A figure displaying the bar chart.
        """
        return self.topic_model.visualize_barchart(top_n_topics=top_n_topics)

    def visualise_hierarchy(self, hierarchical_topics):
        """
        Visualises hierarchical topics.

        Args:
            hierarchical_topics: Hierarchical topics data.

        Returns:
            plotly.Figure: A figure displaying the hierarchical topics␣
 ↪visualization.
        """
```

```
        return self.topic_model.
↪visualize_hierarchy(hierarchical_topics=hierarchical_topics)

    def visualise_heatmap(self):
        """
        Visualises a heatmap of the topic model.

        Returns:
            plotly.Figure: A figure displaying the heatmap.
        """
        return self.topic_model.visualize_heatmap()

    def visualise_topics_over_time(self, topics_over_time: pd.DataFrame,␣
↪top_n_topics: int = None,
                                   topics: List[int] = None,␣
↪normalize_frequency: bool = False,
                                   custom_labels: Union[bool, str] = False,␣
↪title: str = "<b>Topics over Time</b>",
                                   width: int = 1250, height: int = 450,␣
↪colormap: str = 'Viridis') -> go.Figure:
        """
        Visualize topics over time using labels from a dictionary, excluding␣
↪topic -1,
        and allowing any Plotly colormap.
        """
        # Handle Plotly's built-in color scales if a string is provided
        if isinstance(colormap, str):
            try:
                # Attempt to fetch the colormap by name from Plotly Express
                colors = getattr(px.colors.sequential, colormap)
            except AttributeError:
                # If colormap is not found, default to Viridis
                colors = px.colors.sequential.Viridis
        else:
            # If a list is provided, use it directly as the colormap
            colors = colormap

        # Exclude topic -1 from visualization
        data = topics_over_time[topics_over_time['Topic'] != -1].copy()

        # If specific topics are requested, filter by those; otherwise, select␣
↪top N topics if specified
        if topics is not None:
            data = data[data['Topic'].isin(topics)]
        elif top_n_topics is not None:
            top_topics = data['Topic'].value_counts().head(top_n_topics).index
```

```
            data = data[data['Topic'].isin(top_topics)]

        # Initialize Plotly figure
        fig = go.Figure()

        # Iterate over topics to create traces
        for index, topic in enumerate(sorted(data['Topic'].unique(), key=lambda␣
↪x: (x != -1, x))):
            topic_data = data[data['Topic'] == topic]
            label = topic_labels.get(topic, f"Topic {topic}")  # Get label from␣
↪dictionary

            # Normalize frequency if requested
            if normalize_frequency:
                y = normalize(topic_data['Frequency'].values.reshape(1, -1))[0].
↪tolist()
            else:
                y = topic_data['Frequency'].values

            # Determine the color for the trace
            color = colors[index % len(colors)]

            # Add trace to the figure for this topic
            fig.add_trace(go.Scatter(x=topic_data['Timestamp'], y=y,␣
↪mode='lines+markers',
                                     marker_color=color, hoverinfo="text",
                                     name=label,  # Use label for legend
                                     hovertext=[f'<b>{label}</b><br>Words:␣
↪{words}' for words in topic_data['Words']]))

        # Update layout with legend on the right
        fig.update_layout(
            yaxis_title="Frequency",
            title={'text': title, 'y': 0.9, 'x': 0.5, 'xanchor': 'center',␣
↪'yanchor': 'top'},
            template="plotly_white", width=width, height=height,
            hoverlabel=dict(bgcolor="white", font_size=12,␣
↪font_family="Rockwell"),
            legend=dict(
                title="<b>Legend</b>",
                orientation="v",
                yanchor="middle",
                y=0.5,
                xanchor="left",
                x=1.05
            )
```

```
        )
        return fig

class TopicModelEvaluation:
    """
    A class for evaluating topic models focusing on NPMI coherence, topic␣
 ↪diversity,
    and KL background metrics.
    """
    def __init__(self, topic_model, dataset_path):
        """
        Initializes the TopicModelEvaluation object with a topic model and␣
 ↪dataset path.

        Args:
            topic_model: The topic model to be evaluated.
            dataset_path (str): The path to the OCTIS dataset for evaluation.
        """
        self.topic_model = topic_model
        self.dataset_path = dataset_path
        self.octis_dataset = self._load_dataset()

    def _load_dataset(self):
        """
        Loads the dataset from the provided dataset path.

        Returns:
            octis_dataset: The OCTIS dataset object loaded from the dataset␣
 ↪path.
        """
        octis_dataset = Dataset()
        octis_dataset.load_custom_dataset_from_folder(self.dataset_path)
        return octis_dataset

    def evaluate_npmi_coherence(self, topk):
        """
        Evaluates the topic model using the NPMI coherence metric.

        Args:
            topk (int, optional): The number of top words to consider for␣
 ↪coherence calculation.

        Returns:
            float: The NPMI coherence score.
        """
```

```
        topics_for_octis = [[word for word, _ in self.topic_model.
↪get_topic(topic_id)[:topk]] for topic_id in self.topic_model.get_topics().
↪keys()]
        model_output = {"topics": topics_for_octis}
        coherence_metric = Coherence(texts=self.octis_dataset.get_corpus(),␣
↪topk=topk, measure='c_npmi')
        npmi_score = coherence_metric.score(model_output)
        return npmi_score

    def evaluate_topic_diversity(self, topk=5):
        """
        Evaluates the topic diversity of the model.

        Args:
            topk (int, optional): The number of top words to consider for␣
↪diversity calculation.

        Returns:
            float: The topic diversity score.
        """
        topics = [[word for word, _ in self.topic_model.get_topic(topic_id)[:
↪topk]] for topic_id in self.topic_model.get_topics().keys()]
        model_output = {"topics": topics}
        topic_diversity = TopicDiversity(topk=topk)
        diversity_score = topic_diversity.score(model_output)
        return diversity_score

    def evaluate_model(self, topk):
        """
        Performs a comprehensive evaluation using NPMI coherence, topic␣
↪diversity.

        Args:
            topk (int, optional): The number of top words to consider for␣
↪evaluation.

        Returns:
            dict: A dictionary containing scores for each of the evaluated␣
↪metrics.
        """
        npmi_score = self.evaluate_npmi_coherence(topk)
        diversity_score = self.evaluate_topic_diversity(topk)

        return {
            "NPMI Coherence": npmi_score,
            "Topic Diversity": diversity_score,
```

```
        }

    def evaluate_individual_topics(self, topk):
        """
        Evaluates individual topics generated by the topic model using OCTIS␣
↪metrics.

        Args:
            topk (int): The number of top words to consider for coherence␣
↪calculation.

        Returns:
            dict: A dictionary containing OCTIS coherence scores for each topic.
        """
        extracted_topics = self.topic_model.get_topics()
        topics_for_octis = {topic_id: [word for word, _ in self.topic_model.
↪get_topic(topic_id)[:topk]]
                            for topic_id in extracted_topics.keys()}

        individual_coherence_scores = {}
        for topic_id, words in topics_for_octis.items():
            model_output = {"topics": [words[:topk]]}  # Ensure words list has␣
↪at least 'topk' elements
            npmi_octis = Coherence(texts=self.octis_dataset.get_corpus(),␣
↪topk=len(words[:topk]), measure="c_npmi")
            individual_coherence_scores[topic_id] = npmi_octis.
↪score(model_output)

        return individual_coherence_scores
```

2.1 Set Paths and Load Data

[5]:
```
# Set Paths
pdf_path = '/content/drive/My Drive/Colab Notebooks/SIT723/Data/Articles/PDF'
txt_path = '/content/drive/My Drive/Colab Notebooks/SIT723/Data/Articles/txt'
raw_dataset_path = '/content/drive/My Drive/Colab Notebooks/SIT723/Data/
 ↪Articles/txt'
dataset_path = '/content/drive/My Drive/Colab Notebooks/SIT723/Data/Articles/
 ↪txt/preprocessed'
tsv_directory = '/content/drive/My Drive/Colab Notebooks/SIT723/Data/Articles/
 ↪tsv'
tsv_path = '/content/drive/My Drive/Colab Notebooks/SIT723/Data/Articles/tsv/
 ↪corpus.tsv'
```

2.2 Extract Text from PDFs

[ ]:
```
# Extract Text from PDF
pdf_processor = PDFProcessor(pdf_path, txt_path)
pdf_processor.process_pdfs()
```

[208]:
```
# Initialize empty lists to store text content and filenames
docs, filenames = [], []

# Iterate through files in the specified directory
for filename in os.listdir(txt_path):
    # Check if the file has a ".txt" extension
    if filename.endswith(".txt"):
        # Construct the full file path
        filepath = os.path.join(txt_path, filename)
        # Open the file for reading
        with open(filepath, 'r') as file:
            # Append the file's content to the docs list
            docs.append(file.read())
            # Append the filename to the filenames list
            filenames.append(filename)
```

2.3 Load Raw .txt Files

[209]:
```
# Loading raw txt files
data_loader = DataLoader(directory=raw_dataset_path)
raw_documents = data_loader.load_texts()
```

```
Loading files: 100%|      | 414/414 [00:01<00:00, 398.31it/s]
```

2.4 Preprocess Raw .txt Files

[210]:
```
preprocessor = TextPreprocessor(
    remove_punctuation=True,
    punctuation=string.punctuation,
    stopword_list=set(stopwords.words('english')),
    min_chars=4,
    min_words_docs=1,
    #min_df=0.1,
    #max_df=0.8,
    spellchecker=SpellChecker()
)

preprocessor.process_documents(docs, filenames, dataset_path)
```

```
Processing Documents: 100%|      | 414/414 [00:13<00:00, 31.49it/s]
```

2.5 Load Preprocessed Documents

[211]:
```
# Loading preprocessed txt files
# Initialize DataLoader
data_loader = DataLoader(directory=dataset_path)

# Load preprocessed documents
preprocessed_documents = data_loader.load_texts()

documents = preprocessed_documents

# Write preprocessed documents to TSV
data_loader.write_documents_to_tsv(tsv_path, preprocessed_documents)

# Loading tsv file
dataset = Dataset()
dataset.load_custom_dataset_from_folder(tsv_directory)
```

```
Loading files: 100%|      | 414/414 [00:02<00:00, 196.04it/s]
```

2.6 Generate Embeddings

[212]:
```
# Generate Embeddings
embedding_model_instance = EmbeddingModel(model_name='all-mpnet-base-v2')
embeddings = embedding_model_instance.generate_embeddings(documents,␣
 ↪is_sentences=True)
```

```
Batches:   0%|          | 0/13 [00:00<?, ?it/s]
```

2.7 Initialise Dimension Reduction and Clustering Models

[213]:
```
# Apply UMAP Dimensionality Reduction on Embeddings
umap_instance = UMAPDimensionalityReduction(n_neighbors=5,
                                            n_components=2,
                                            min_dist=1,
                                            metric='euclidean')
umap_embeddings = umap_instance.fit_transform(embeddings)

# Apply HDBSCAN Clustering on Reduced Embeddings
hdbscan_instance = HDBSCANClustering(min_cluster_size=5,
                                     metric='euclidean',
                                     cluster_selection_method='leaf')
clusters = hdbscan_instance.fit(umap_embeddings)
```

2.8 Fit BERTopic Topic Model

[214]:
```
# Initialize TopicModeling with custom parameters
topic_modeling = TopicModeling(
    embedding_model_instance.model,
    umap_instance.umap_model,
    hdbscan_instance.hdbscan_model,
    top_n_words=5,
    nr_topics=22  # Dynamic number of topics
)
topics, probabilities = topic_modeling.fit_transform(documents, umap_embeddings)
```

```
2024-02-04 06:16:06,777 - BERTopic - Dimensionality - Fitting the dimensionality
reduction algorithm
2024-02-04 06:16:07,985 - BERTopic - Dimensionality - Completed
2024-02-04 06:16:07,987 - BERTopic - Cluster - Start clustering the reduced
embeddings
2024-02-04 06:16:08,025 - BERTopic - Cluster - Completed
2024-02-04 06:16:08,026 - BERTopic - Representation - Extracting topics from
clusters using representation models.
2024-02-04 06:16:33,133 - BERTopic - Representation - Completed
2024-02-04 06:16:33,216 - BERTopic - Topic reduction - Reducing number of topics
2024-02-04 06:16:58,394 - BERTopic - Topic reduction - Reduced number of topics
from 26 to 22
```

2.9 View Topics

[215]:
```
# Retrieve and Print Topic Info
topic_info = topic_modeling.get_topic_info()
print(topic_info)
```

```
    Topic  Count                                              Name  \
0      -1    176        -1_intervention_health_study_implementation
1       0     26             0_intervention_weight_change_behavior
2       1     24                  1_behaviour_health_change_theory
3       2     20      2_intervention_implementation_health_practice
4       3     17             3_intervention_activity_physical_study
5       4     17                  4_intervention_health_women_study
6       5     14             5_children_language_trial_intervention
7       6     12      6_implementation_safecare_strategies_training
8       7     12          7_implementation_research_outcomes_health
9       8      9  8_alcohol_consumption_drinking_alcohol consump…
10      9      9  9_stroke_intervention_rehabilitation_stroke su…
11     10      9                  10_physical_children_school_data
12     11      9    11_activity_physical_physical activity_messages
13     12      8             12_anxiety_patients_intervention_study
14     13      8            13_cancer_intervention_distress_studies
15     14      7         14_veterans_implementation_missionvet_care
```

```
16     15      7                  15_smoking_smokers_quit_cessation
17     16      7    16_intervention_activity_physical_interventions
18     17      7                17_food_sample_control_consumption
19     18      6                18_singing_cognitive_life_dementia
20     19      5         19_behaviour_behavior_change_interventions
21     20      5                  20_care_depression_mental_health

                                      Representation  \
0   [intervention, health, study, implementation, …
1   [intervention, weight, change, behavior, resea…
2     [behaviour, health, change, theory, psychology]
3   [intervention, implementation, health, practic…
4   [intervention, activity, physical, study, chil…
5        [intervention, health, women, study, sexual]
6   [children, language, trial, intervention, pare…
7   [implementation, safecare, strategies, trainin…
8   [implementation, research, outcomes, health, f…
9   [alcohol, consumption, drinking, alcohol consu…
10  [stroke, intervention, rehabilitation, stroke …
11       [physical, children, school, data, activity]
12  [activity, physical, physical activity, messag…
13     [anxiety, patients, intervention, study, pain]
14   [cancer, intervention, distress, studies, study]
15  [veterans, implementation, missionvet, care, s…
16  [smoking, smokers, quit, cessation, smoking ce…
17  [intervention, activity, physical, interventio…
18      [food, sample, control, consumption, craving]
19  [singing, cognitive, life, dementia, intervent…
20  [behaviour, behavior, change, interventions, i…
21  [care, depression, mental, health, mental health]

                                Representative_Docs
0   [ dating relationship violence intimate partne…
1   [weight loss individuals regain lost weight in…
2   [      health psychology      online  homepage…
3   [ researchers publish processes develop interv…
4   [ premature birth gestational weeks associated…
5   [ perinatal period common mental disorders cmd…
6   [ number children experience difficulties soci…
7   [ increasing number schools rural settings imp…
8   [ facilitation entified literature tentially c…
9   [      health psychology      online  homepage…
10  [ poststroke physical activity reduces disabil…
11  [ inactive lifestyles becoming norm creative a…
12  [heartphone mobile evaluative conditioning enh…
13  [ anxiety depressive disorders highly prevalen…
14  [around individuals cancer experience distress…
15  [ homeless veterans often multiple health care…
```

```
16  [using internet assist smoking prevention cess…
17  [introduction method section prior sedentary b…
18  [health impulse selfcontrol promotes healthy f…
19  [ playlist life brief inexpensive music listen…
20  [rates chronic diseases high among black south…
21  [ integrating mental health providers primary …
```

2.10 Evaluate Topic Model Performance

[216]:
```
# Evaluation
# Create an instance of TopicModelEvaluation
evaluation = TopicModelEvaluation(topic_model=topic_modeling.topic_model,␣
 ↪dataset_path=tsv_directory)

# Call evaluate_model on the instance
scores = evaluation.evaluate_model(topk=5)

# Print the evaluation scores
print("\nTopic Model Evaluation:")
print(scores)
```

```
Topic Model Evaluation:
{'NPMI Coherence': 0.10871274824130438, 'Topic Diversity': 0.5909090909090909}
```

[217]:
```
# Evaluating individual topics
evaluation = TopicModelEvaluation(topic_model=topic_modeling.topic_model,␣
 ↪dataset_path=tsv_directory)
individual_coherence_scores = evaluation.evaluate_individual_topics(topk=5)

# Print individual coherence results
print("\nIndividual Topic Coherence Scores (NPMI):")
for topic_id, score in individual_coherence_scores.items():
    print(f"Topic {topic_id}: {score}")
```

```
Individual Topic Coherence Scores (NPMI):
Topic -1: -0.031245511489278517
Topic 0: 0.015928100651445617
Topic 1: 0.16670796516766523
Topic 2: 0.0011087846247241368
Topic 3: 0.05865734264336738
Topic 4: -0.001968814570600449
Topic 5: 0.06688267411305135
Topic 6: 0.021514191073156344
Topic 7: 0.03225444117952737
Topic 8: 0.16025791824995625
Topic 9: 0.26105556928620893
```

```
Topic 10: 0.06851660186865188
Topic 11: 0.23682781792433874
Topic 12: 0.0031982432224103574
Topic 13: 0.010861886338152292
Topic 14: 0.1171489176115433
Topic 15: 0.47451096785742514
Topic 16: 0.1362825643096143
Topic 17: 0.1774048996447632
Topic 18: 0.08270599205687167
Topic 19: 0.11319308689675851
Topic 20: 0.21987682264894307
```

2.11 Outlier Reduction

[218]:
```
# Reduce outliers
# Initialize TopicModeling with custom parameters
topic_modeling_outlier_reduction = TopicModeling(
    embedding_model_instance.model,
    umap_instance.umap_model,
    hdbscan_instance.hdbscan_model,
    top_n_words=5,
    nr_topics=22  # Dynamic number of topics
)
# Fit Model
topics, probabilities = topic_modeling_outlier_reduction.
 ↪fit_transform(documents, umap_embeddings)

# Reduce Outliers
new_topics = topic_modeling_outlier_reduction.reduce_outliers(documents,␣
 ↪topics, probabilities=probabilities)

# Update Model
topic_modeling_outlier_reduction.update_topics(documents, new_topics)

# view Topics
topic_info = topic_modeling_outlier_reduction.topic_model.get_topic_info()
print(topic_info)

# Evaluation
# Create an instance of TopicModelEvaluation
evaluation = TopicModelEvaluation(topic_model=topic_modeling_outlier_reduction.
 ↪topic_model, dataset_path=tsv_directory)

# Call evaluate_model on the instance
scores = evaluation.evaluate_model(topk=5)

# Print the evaluation scores
```

```
print("\nTopic Model Evaluation:")
print(scores)
```

```
2024-02-04 06:20:51,013 - BERTopic - Dimensionality - Fitting the dimensionality
reduction algorithm
2024-02-04 06:20:53,855 - BERTopic - Dimensionality - Completed
2024-02-04 06:20:53,858 - BERTopic - Cluster - Start clustering the reduced
embeddings
2024-02-04 06:20:53,897 - BERTopic - Cluster - Completed
2024-02-04 06:20:53,898 - BERTopic - Representation - Extracting topics from
clusters using representation models.
2024-02-04 06:21:18,399 - BERTopic - Representation - Completed
2024-02-04 06:21:18,480 - BERTopic - Topic reduction - Reducing number of topics
2024-02-04 06:21:42,973 - BERTopic - Topic reduction - Reduced number of topics
from 26 to 22
2024-02-04 06:21:49,644 - BERTopic - WARNING: Using a custom list of topic
assignments may lead to errors if topic reduction techniques are used
afterwards. Make sure that manually assigning topics is the last step in the
pipeline.Note that topic embeddings will also be created through weightedc-TF-
IDF embeddings instead of centroid embeddings.
```

	Topic	Count	Name	\
0	0	28	0_intervention_weight_change_research	
1	1	44	1_behaviour_change_health_intervention	
2	2	51	2_intervention_implementation_health_intervent…	
3	3	35	3_social_health_study_intervention	
4	4	21	4_intervention_women_health_study	
5	5	17	5_children_language_intervention_trial	
6	6	22	6_implementation_health_training_study	
7	7	37	7_implementation_health_research_study	
8	8	11	8_alcohol_community_consumption_drinking	
9	9	11	9_stroke_intervention_stroke survivors_survivors	
10	10	14	10_intervention_physical_study_data	
11	11	11	11_activity_physical_physical activity_interve…	
12	12	12	12_anxiety_treatment_intervention_patients	
13	13	13	13_cancer_intervention_interventions_studies	
14	14	14	14_implementation_veterans_missionvet_sites	
15	15	8	15_smoking_smokers_quit_cessation	
16	16	17	16_intervention_physical_activity_participants	
17	17	11	17_food_control_sample_consumption	
18	18	12	18_music_dementia_cognitive_study	
19	19	9	19_behaviour_change_behavior_interventions	
20	20	16	20_care_mental_health_mental health	

	Representation	\
0	[intervention, weight, change, research, behav…	
1	[behaviour, change, health, intervention, theo…	
2	[intervention, implementation, health, interve…	

```
3   [social, health, study, intervention, physical…
4   [intervention, women, health, study, participa…
5   [children, language, intervention, trial, viol…
6   [implementation, health, training, study, scho…
7   [implementation, health, research, study, inte…
8   [alcohol, community, consumption, drinking, in…
9   [stroke, intervention, stroke survivors, survi…
10  [intervention, physical, study, data, particip…
11  [activity, physical, physical activity, interv…
12  [anxiety, treatment, intervention, patients, s…
13  [cancer, intervention, interventions, studies,…
14  [implementation, veterans, missionvet, sites, …
15  [smoking, smokers, quit, cessation, smoking ce…
16  [intervention, physical, activity, participant…
17  [food, control, sample, consumption, study, pa…
18  [music, dementia, cognitive, study, participan…
19  [behaviour, change, behavior, interventions, h…
20  [care, mental, health, mental health, depressi…

                                Representative_Docs
0   [weight loss individuals regain lost weight in…
1   [      health psychology      online  homepage…
2   [ researchers publish processes develop interv…
3   [ premature birth gestational weeks associated…
4   [ perinatal period common mental disorders cmd…
5   [ number children experience difficulties soci…
6   [ increasing number schools rural settings imp…
7   [ facilitation entified literature tentially c…
8   [      health psychology      online  homepage…
9   [ poststroke physical activity reduces disabil…
10  [ inactive lifestyles becoming norm creative a…
11  [heartphone mobile evaluative conditioning enh…
12  [ anxiety depressive disorders highly prevalen…
13  [around individuals cancer experience distress…
14  [ homeless veterans often multiple health care…
15  [using internet assist smoking prevention cess…
16  [introduction method section prior sedentary b…
17  [health impulse selfcontrol promotes healthy f…
18  [ playlist life brief inexpensive music listen…
19  [rates chronic diseases high among black south…
20  [ integrating mental health providers primary …

Topic Model Evaluation:
{'NPMI Coherence': 0.08528462888273036, 'Topic Diversity': 0.5333333333333333}
```

[219]:
```
# Evaluation
# Create an instance of TopicModelEvaluation
```

```
evaluation = TopicModelEvaluation(topic_model=topic_modeling_outlier_reduction.
 ↪topic_model, dataset_path=tsv_directory)

# Call evaluate_model on the instance
scores = evaluation.evaluate_model(topk=5)

# Print the evaluation scores
print("\nTopic Model Evaluation:")
print(scores)
```

```
Topic Model Evaluation:
{'NPMI Coherence': 0.08528462888273036, 'Topic Diversity': 0.5333333333333333}
```

[220]:
```
# Evaluating individual topics
evaluation = TopicModelEvaluation(topic_model=topic_modeling_outlier_reduction.
 ↪topic_model, dataset_path=tsv_directory)
individual_coherence_scores = evaluation.evaluate_individual_topics(topk=5)

# Print individual coherence results
print("\nIndividual Topic Coherence Scores (NPMI):")
for topic_id, score in individual_coherence_scores.items():
    print(f"Topic {topic_id}: {score}")
```

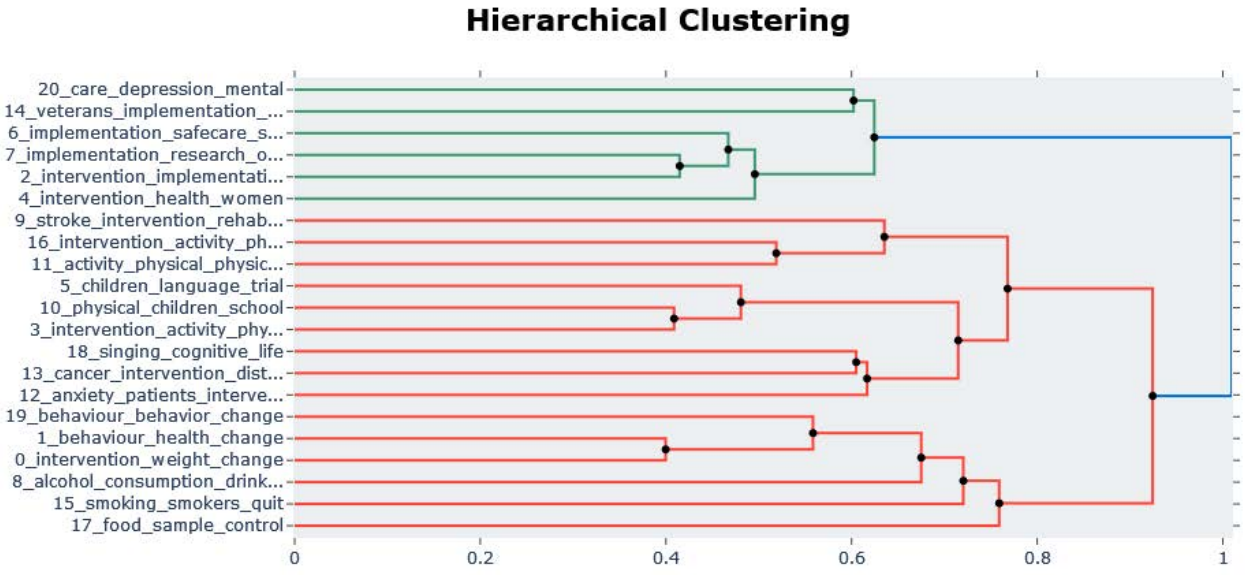
```
Individual Topic Coherence Scores (NPMI):
Topic 0: 0.015928100651445624
Topic 1: 0.10633362377923024
Topic 2: 0.0056720488033015545
Topic 3: -0.02658997850314469
Topic 4: -0.017857950912378327
Topic 5: -0.006516021381712466
Topic 6: -0.02090668996365625
Topic 7: -0.005060034971591106
Topic 8: 0.13046591700976765
Topic 9: 0.26105556928620893
Topic 10: -0.006431204419471303
Topic 11: 0.12929392581618424
Topic 12: 0.015018625561014676
Topic 13: 0.03813356767864303
Topic 14: 0.11714891761154331
Topic 15: 0.47451096785742514
Topic 16: 0.12929392581618424
Topic 17: 0.06525767401730395
Topic 18: 0.02917048110919405
Topic 19: 0.1371789190429022
Topic 20: 0.21987682264894312
```

Outlier reduction reduced topic coherence and diversity. We will continue with the original solution.

2.12 Fit and Visualise Hierarchichal Model

[221]:
```
# Create an instance of VisualisationAndEvaluation
visualisation = TopicVisualisation(topic_modeling.topic_model)

# Hierarchical Topic Modeling
hierarchical_topics = topic_modeling.fit_hierarchical_topics(documents)

# Visualize the hierarchical topics
hierarchy_vis = visualisation.visualise_hierarchy(hierarchical_topics)
display(hierarchy_vis)
```

```
100%|      | 20/20 [00:01<00:00, 14.50it/s]
```



2.13 Merging Topics

[222]:
```
# # List of topics to merge
# topics_to_merge = [2, 7],

# # Merging Topics
# topic_modeling.merge_topics(documents, topics_to_merge)
# # view Topics
# topic_info = topic_modeling.topic_model.get_topic_info()
# print(topic_info)
# # Create an instance of VisualisationAndEvaluation
# visualisation = TopicVisualisation(topic_modeling.topic_model)

# # Hierarchical Topic Modeling
# hierarchical_topics = topic_modeling.fit_hierarchical_topics(documents)

# # Visualize the hierarchical topics
# hierarchy_vis = visualisation.visualise_hierarchy(hierarchical_topics)
# display(hierarchy_vis)
# # Evaluation
# # Create an instance of TopicModelEvaluation
# evaluation =␣
 ↪TopicModelEvaluation(topic_model=topic_modeling_outlier_reduction.
 ↪topic_model, dataset_path=tsv_directory)

# # Call evaluate_model on the instance
# scores = evaluation.evaluate_model(topk=5)

# # Print the evaluation scores
# print("\nTopic Model Evaluation:")
# print(scores)
```

```
# # Evaluating individual topics
# evaluation = TopicModelEvaluation(topic_model=topic_modeling_outlier_reduction.topic_model, dataset_path=tsv_directory)
# individual_coherence_scores = evaluation.evaluate_individual_topics(topk=5)

# # Print individual coherence results
# print("\nIndividual Topic Coherence Scores (NPMI):")
# for topic_id, score in individual_coherence_scores.items():
#     print(f"Topic {topic_id}: {score}")
```

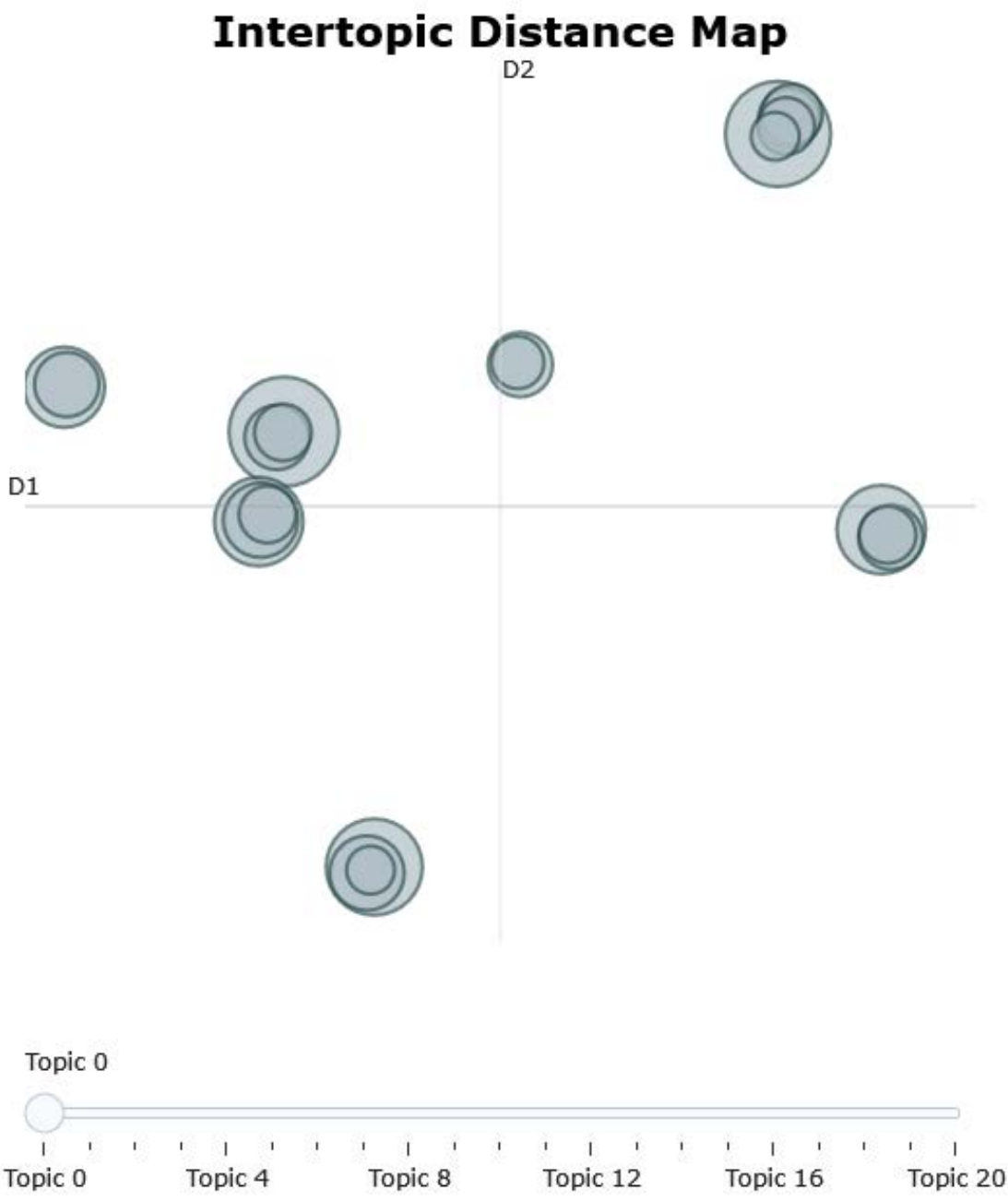
[ ]:
```
# # Evaluation
# # Create an instance of TopicModelEvaluation
# evaluation = TopicModelEvaluation(topic_model=topic_modeling_outlier_reduction.topic_model, dataset_path=tsv_directory)

# # Call evaluate_model on the instance
# scores = evaluation.evaluate_model(topk=5)

# # Print the evaluation scores
# print("\nTopic Model Evaluation:")
# print(scores)
```

[ ]:
```
# # Evaluating individual topics
# evaluation = TopicModelEvaluation(topic_model=topic_modeling_outlier_reduction.topic_model, dataset_path=tsv_directory)
# individual_coherence_scores = evaluation.evaluate_individual_topics(topk=5)

# # Print individual coherence results
# print("\nIndividual Topic Coherence Scores (NPMI):")
# for topic_id, score in individual_coherence_scores.items():
#     print(f"Topic {topic_id}: {score}")
```

2.14 Visualise Topics

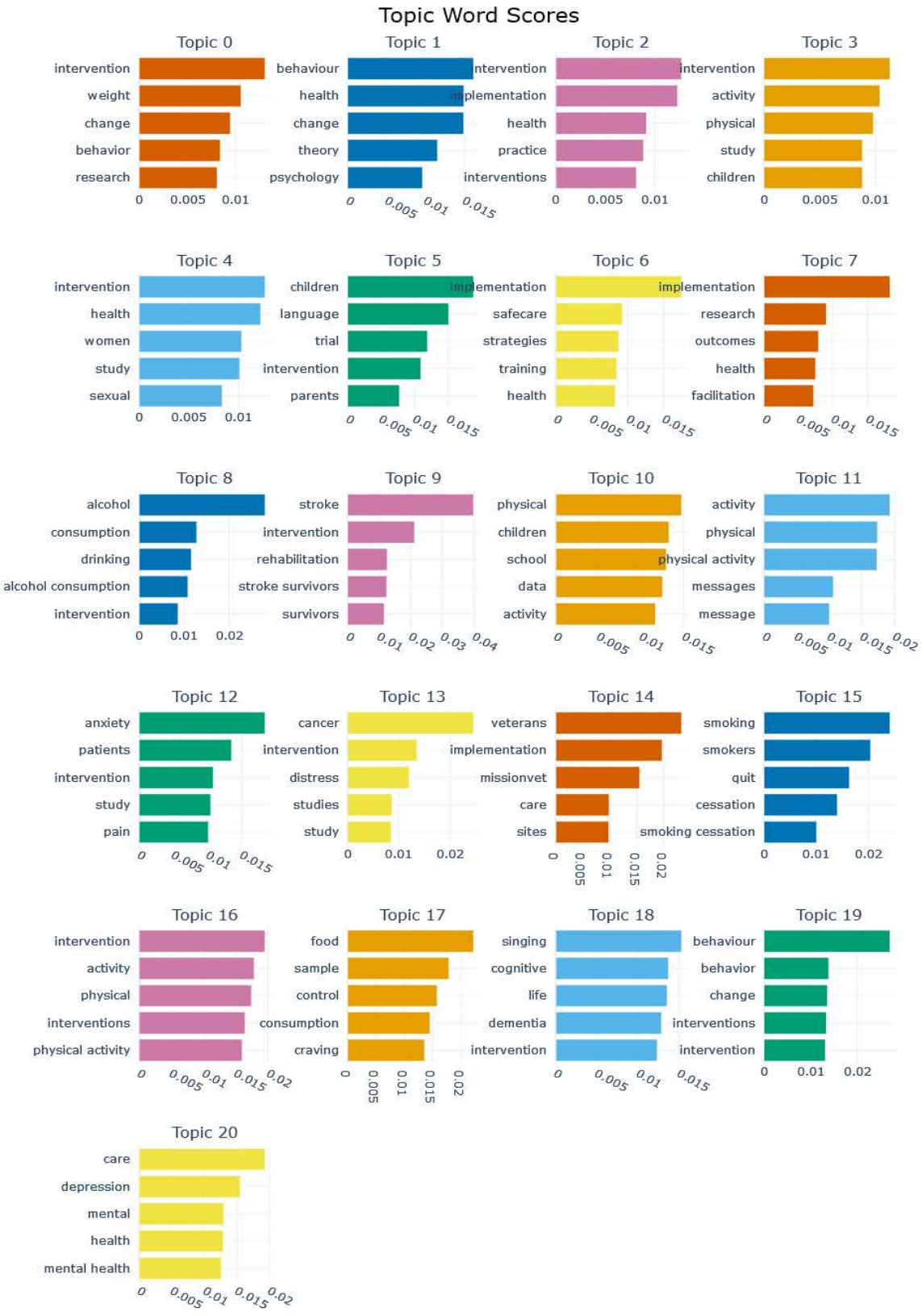
2.14.1 Intertopic Distance Map

[223]:
```
# Visualiser Initialisation
visualisation = TopicVisualisation(topic_modeling.topic_model)

# Visualise intertopic distance map
topics_vis = visualisation.visualise_topics()
display(topics_vis)
```


Intertopic Distance Map
D2
D1
Topic 0
Topic 0
Topic 4
Topic 8
Topic 12
Topic 16
Topic 20

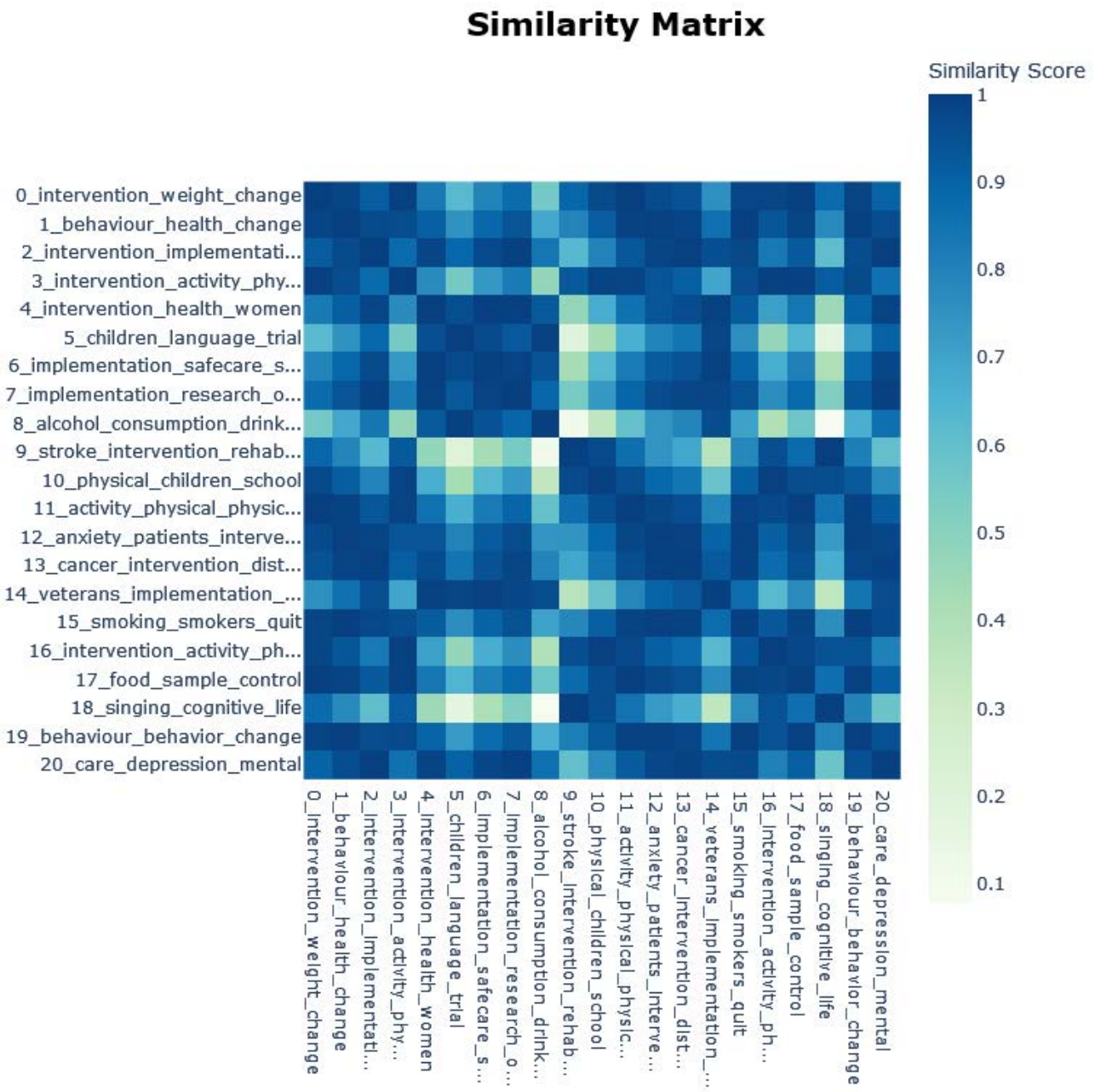
2.14.2 Barchart

[224]:
```
# Visualise topic barcharts
barchart_vis = visualisation.visualise_barchart(top_n_topics=23)
display(barchart_vis)
```



2.14.3 Topic Heatmap

[225]:
```
# Create an instance of VisualisationAndEvaluation
visualisation = TopicVisualisation(topic_modeling.topic_model)

# Visualize heatmap
heatmap_vis = visualisation.visualise_heatmap()
display(heatmap_vis)
```

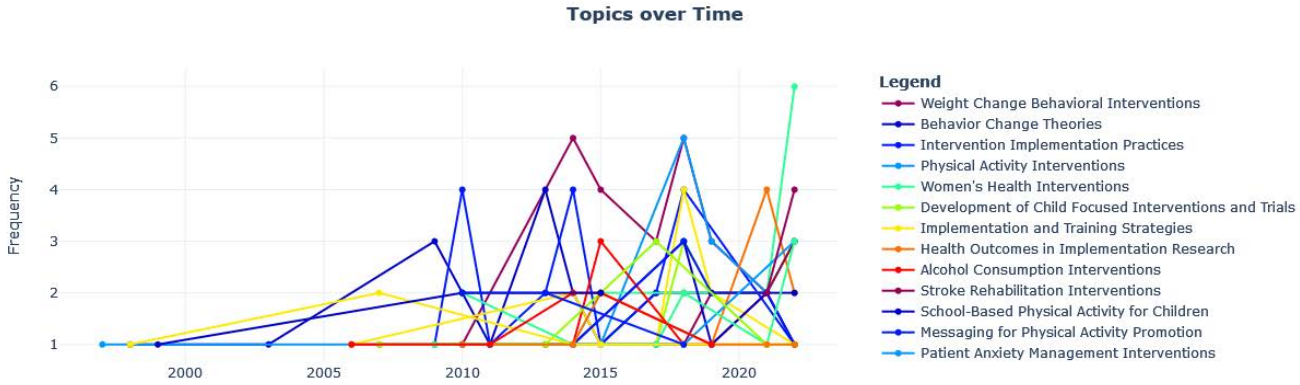


2.15 Fit and Visualise Dynamic Topic Model

[226]:
```
# Create an instance of VisualisationAndEvaluation
visualisation = TopicVisualisation(topic_modeling.topic_model)

# Dynamic Topic Modeling
# Ensure each document has a valid timestamp
timestamps = [data_loader.extract_year_from_filename(f) for f in os.
 ↪listdir(data_loader.directory)]
timestamps = [t for t in timestamps if t is not None]

# Perform dynamic topic modeling
topics_over_time = topic_modeling.topics_over_time(documents=documents,␣
 ↪timestamps=timestamps, nr_bins=29)
```

```
20it [11:26, 34.35s/it]
```

[255]:
```
# Updated dictionary mapping from the provided image
topic_labels = {
    -1: "Intervention Implementation Research",
    0: "Weight Change Behavioral Interventions",
    1: "Behavior Change Theories",
    2: "Intervention Implementation Practices",
    3: "Physical Activity Interventions",
    4: "Women's Health Interventions",
    5: "Development of Child Focused Interventions and Trials",
    6: "Implementation and Training Strategies",
    7: "Health Outcomes in Implementation Research",
    8: "Alcohol Consumption Interventions",
    9: "Stroke Rehabilitation Interventions",
    10: "School-Based Physical Activity for Children",
    11: "Messaging for Physical Activity Promotion",
    12: "Patient Anxiety Management Interventions",
    13: "Cancer-Related Interventions",
    14: "Implementation in Veteran Care",
    15: "Smoking Cessation Interventions",
    16: "Physical Activity Interventions",
    17: "Weight Loss and Nutrition Interventions",
    18: "Music Interventions for Dementia",
    19: "Behavioral Change Interventions",
    20: "Mental Health Care for Depression"
}

# Make sure this instance is of the updated class that includes the 'colormap'␣
 ↪parameter
visualisation = TopicVisualisation(topic_modeling.topic_model)

# Now call the method with the 'colormap' parameter
topics_over_time_vis = visualisation.
 ↪visualise_topics_over_time(topics_over_time, top_n_topics=21,␣
 ↪colormap='Rainbow')
topics_over_time_vis.show()
```


Topics over Time
Frequency
6
5
4
3
2
1
2000
2005
2010
2015
2020
Legend
Weight Change Behavioral Interventions
Behavior Change Theories
Intervention Implementation Practices
Physical Activity Interventions
Women's Health Interventions
Development of Child Focused Interventions and Trials
Implementation and Training Strategies
Health Outcomes in Implementation Research
Alcohol Consumption Interventions
Stroke Rehabilitation Interventions
School-Based Physical Activity for Children
Messaging for Physical Activity Promotion
Patient Anxiety Management Interventions

[229]:

```
!jupyter nbconvert --to pdf "/content/drive/MyDrive/Colab Notebooks/SIT723/Copy␣
↪of Refactored Content Analysis.ipynb" --output-dir "/content/drive/MyDrive/
↪Colab Notebooks/SIT723/"
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab
Notebooks/SIT723/Copy of Refactored Content Analysis.ipynb to pdf
/usr/local/lib/python3.10/dist-packages/nbconvert/filters/datatypefilter.py:41:
UserWarning: Your element with mimetype(s) dict_keys(['text/html']) is not able
to be represented.
  warn(
/usr/local/lib/python3.10/dist-packages/nbconvert/filters/datatypefilter.py:41:
UserWarning: Your element with mimetype(s) dict_keys(['text/html']) is not able
to be represented.
  warn(
/usr/local/lib/python3.10/dist-packages/nbconvert/filters/datatypefilter.py:41:
UserWarning: Your element with mimetype(s) dict_keys(['text/html']) is not able
to be represented.
  warn(
/usr/local/lib/python3.10/dist-packages/nbconvert/filters/datatypefilter.py:41:
UserWarning: Your element with mimetype(s) dict_keys(['text/html']) is not able
to be represented.
  warn(
/usr/local/lib/python3.10/dist-packages/nbconvert/filters/datatypefilter.py:41:
UserWarning: Your element with mimetype(s) dict_keys(['text/html']) is not able
to be represented.
  warn(
[NbConvertApp] Writing 141839 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 129846 bytes to /content/drive/MyDrive/Colab
Notebooks/SIT723/Copy of Refactored Content Analysis.pdf
```