TECHNICAL INFORMATION

README: THE CODE TO CREATE THE COVER & TOC GRAPHS

David J. Cox[1,2]

[1] ENDICOTT COLLEGE, [2] RETHINKFIRST

Visualizing behavioral data in unique ways may lead to novel methods of analyses and, perhaps, new ways of thinking about environment-behavior relations. In the spirit of transparency and to help others discover interesting things in their own textual data, below is the code to create the plots shown on the cover and table of contents in this volume. Some familiarity with Python is needed (i.e., how to open a script, read in files, and execute the program). A downloadable file of this Python script is also available here: https://osf.io/p8f7j. Otherwise, happy coding and playing.

Keywords: Python, visual analysis, natural language processing

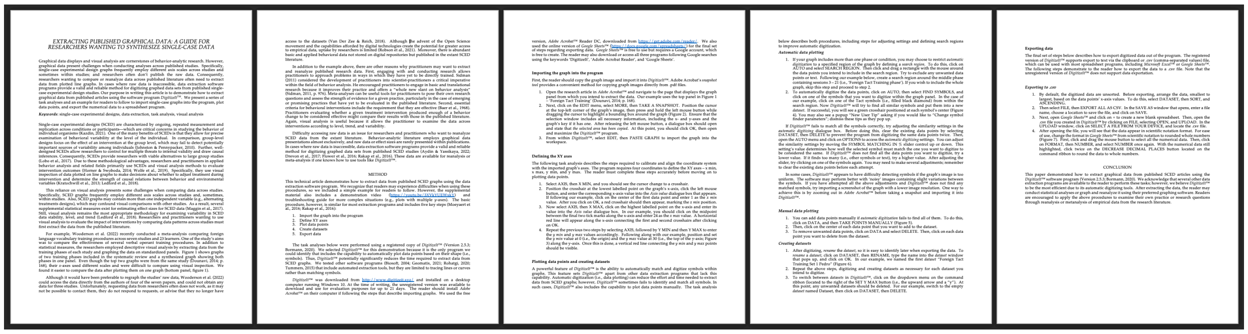

Figure 1. How the text file was saved before running the code. Note the following has been removed: headers and page numbers, figures, references, columnar formatting. It is a simple .docx format.

```
# -*- coding: utf-8 -*-
'''
Automatically generated by Colaboratory.
Original file is located at: https://colab.research.google.com/drive/195B5MPyzA9RkOYr1r6mT_q75tYef7VwZ
'''

# Packages and Modules
# System
from IPython.display import clear_output
from itertools import tee
from collections import Counter

# Data manipulation
import docx
from docx import Document
import networkx as nx
import numpy as np
import pandas as pd

# NLP
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.util import ngrams
from nltk.tokenize import word_tokenize
from transformers import DistilBertModel, DistilBertTokenizer
from sklearn.decomposition import PCA

# Data Visualization
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import plotly.graph_objects as go
```

```
"""# Functions We'll Use"""
def extract_text_from_docx(docx_path):
    doc = docx.Document(docx_path)
    return ' '.join([para.text for para in doc.paragraphs])

def tokenize_and_filter(text):
    stop_words = set(stopwords.words('english'))
    words = word_tokenize(text.lower())
    return [word for word in words if word.isalpha() and word not in stop_words]

def create_corpus(documents):
    ordered_words = []
    seen_words = set()
    for doc in documents:
        filtered_words = tokenize_and_filter(doc)
        for word in filtered_words:
            if word not in seen_words:
                seen_words.add(word)
                ordered_words.append(word)
    return ordered_words

def get_word_embeddings(words):
    tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
    model = DistilBertModel.from_pretrained('distilbert-base-uncased')
    embeddings = []
    for word in words:
        inputs = tokenizer(word, return_tensors="pt", add_special_tokens=False)
        outputs = model(**inputs)
        word_embedding = outputs.last_hidden_state.mean(dim=1).detach().numpy().flatten()
        embeddings.append(word_embedding)
    return np.array(embeddings)

def create_bipartite_graph(corpus, text, normalized_embeddings):
    # Tokenize current document
    filtered_words = tokenize_and_filter(text)

    # Count pair occurrences
    pair_counts = Counter(zip(filtered_words, filtered_words[1:]))

    # Normalize counts to get alpha values
    max_count = max(pair_counts.values())
    alpha_values = {pair: 0.01 + 0.99 * (count / max_count) for pair, count in pair_counts.items()}

    # Create a bipartite graph
    B = nx.Graph()
    top_nodes = {f"top_{word}" for word in corpus}
    bottom_nodes = {f"bottom_{word}" for word in corpus}
    B.add_nodes_from(top_nodes, bipartite=0)
    B.add_nodes_from(bottom_nodes, bipartite=1)

    # Add edges with alpha values
    for word1, word2 in pair_counts:
        B.add_edge(f"top_{word1}", f"bottom_{word2}", alpha=alpha_values[(word1, word2)])

    # Plotting
    plt.figure(figsize=(10, 20))
    pos = {node: (0, i) for i, node in enumerate(top_nodes)}
    pos.update({node: (1, i) for i, node in enumerate(bottom_nodes)})

    # Create a color map based on embeddings
    assert len(normalized_embeddings) == len(corpus), "Length of embeddings and corpus do not match."
    color_map = {f"top_{word}": normalized_embeddings[i] for i, word in enumerate(corpus)}
    color_map.update({f"bottom_{word}": normalized_embeddings[i] for i, word in enumerate(corpus)})
    node_colors = [color_map[node] for node in B.nodes()]

    # Apply colormap
    colormap = cm.get_cmap('Spectral')
    node_colors = [colormap(color_map[node]) for node in B.nodes()]

    # Draw nodes
    nx.draw_networkx_nodes(B, pos, node_size=2, node_color=node_colors)
    # Draw edges with varying alpha
    for edge in B.edges(data=True):
        nx.draw_networkx_edges(B, pos, edgelist=[edge], alpha=edge[2]['alpha'])

    # Draw labels
    clean_labels = {node: node.split('_')[1] for node in B.nodes()}
    label_pos = {node: (pos[node][0] - 0.075 if node.startswith("top") else pos[node][0] + 0.075, pos[node][1]) for node in B.nodes()}
    nx.draw_networkx_labels(B, label_pos, labels=clean_labels, font_size=5)

    plt.subplots_adjust(left=0.15, right=0.85, top=0.95, bottom=0.05)
    plt.show()

# Normalize to range [0, 1] for coloring
def normalize_embeddings(embeddings):
    embeddings_range = max(embeddings) - min(embeddings)
    if embeddings_range == 0:
        return np.zeros_like(embeddings)
    else:
        return (embeddings - min(embeddings)) / embeddings_range


"""# Read in .docx documents and prep them for plotting"""
# Read in the docx files and convert to data we can use
gamified_operant = extract_text_from_docx('/content/gamified_human_operant_raw_text.docx')
extract_data = extract_text_from_docx('/content/extracting_published.docx')
scmo = extract_text_from_docx('/content/SCMO_plot.docx')

# Join the docs together for a single, combined text
combined_document = " ".join([gamified_operant, extract_data, scmo])

# List of documents to iterate over with each plot type
documents = [gamified_operant, extract_data, scmo, combined_document]



"""# Plot parallel bipartite plots of bigrams"""
corpus = create_corpus([gamified_operant, extract_data, scmo])
embeddings = get_word_embeddings(corpus)

pca = PCA(n_components=1)
reduced_embeddings = pca.fit_transform(embeddings).flatten()
normalized_embeddings = normalize_embeddings(reduced_embeddings)

for doc in documents:
    create_bipartite_graph(corpus, doc, normalized_embeddings)




"""# Function for double-helix and (cos(theta), 1-cos(theta)) rotating bipartite plots of bigrams"""
def create_bipartite_graph(corpus, text, normalized_embeddings, width_factor=0.5, double_helix=True):
    # Tokenize current document and count pair occurrences
    filtered_words = tokenize_and_filter(text)
    pair_counts = Counter(zip(filtered_words, filtered_words[1:]))

    # Normalize counts to get alpha values
    max_count = max(pair_counts.values())
    alpha_values = {pair: 0.01 + 0.99 * (count / max_count) for pair, count in pair_counts.items()}

    # Create a bipartite graph
    B = nx.Graph()
    top_nodes = {f"top_{word}" for word in corpus}
    bottom_nodes = {f"bottom_{word}" for word in corpus}
    B.add_nodes_from(top_nodes, bipartite=0)
    B.add_nodes_from(bottom_nodes, bipartite=1)

    # Add edges with alpha values
    for word1, word2 in pair_counts:
        B.add_edge(f"top_{word1}", f"bottom_{word2}", alpha=alpha_values[(word1, word2)])


# Plotting
    plt.figure(figsize=(5, 40))
    if double_helix==True:
        # Calculate positions for a double helix
        x_offset = width_factor  # Adjusts the horizontal spread
        y_positions = np.linspace(0, 1, len(top_nodes))
        pos = {}
        frequency_multiplier = 4  # Multiplier for frequency of helix turns
        helix_offset = np.pi / len(top_nodes)  # Offset between the two helices

        for i, node in enumerate(sorted(top_nodes)):
            angle = np.pi * y_positions[i] * frequency_multiplier
            pos[node] = np.array([np.cos(angle) * x_offset+0.08, y_positions[i]])

        for i, node in enumerate(sorted(bottom_nodes)):
            angle = np.pi * y_positions[i] * frequency_multiplier + helix_offset
            pos[node] = np.array([np.cos(angle) * x_offset, y_positions[i]])

    else:
        # Calculate positions using (cos(theta), 1-cos(theta)) offset
        x_offset = width_factor  # Use the width_factor to adjust the horizontal spread of the plot
        y_positions = np.linspace(0, 1, len(top_nodes))
        pos = {}
        frequency_multiplier = 4  # Multiplier to increase the frequency the coils

        for i, node in enumerate(sorted(top_nodes)):
            angle = np.pi * y_positions[i] * frequency_multiplier
            pos[node] = np.array([(np.cos(angle) + 1) * x_offset, y_positions[i]])

        for i, node in enumerate(sorted(bottom_nodes)):
            angle = np.pi * y_positions[i] * frequency_multiplier
            pos[node] = np.array([1 - (np.cos(angle) + 1) * x_offset, y_positions[i]])

    # Create a color map based on embeddings
    assert len(normalized_embeddings) == len(corpus), "Length of embeddings and corpus do not match."

    # Ensuring we have color for each node in top_nodes and bottom_nodes:
    color_map = {f"top_{word}": normalized_embeddings[i] for i, word in enumerate(corpus)}
    color_map.update({f"bottom_{word}": normalized_embeddings[i] for i, word in enumerate(corpus)})

    # Apply colormap
    colormap = cm.get_cmap('Spectral')
    node_colors = [colormap(color_map[node]) for node in B.nodes()]

    # The node_color list should now have an entry for every node in the graph:
    node_colors = [color_map[node] for node in top_nodes] + [color_map[node] for node in bottom_nodes]

    # Draw nodes
    nx.draw_networkx_nodes(B, pos, node_size=2, node_color=node_colors)

    # Draw edges with varying alpha
    for edge in B.edges(data=True):
        nx.draw_networkx_edges(B, pos, edgelist=[edge], alpha=edge[2]['alpha'], edge_color='grey')

    # Draw labels
    clean_labels = {node: node.split('_')[1] for node in B.nodes()}
    label_pos = {k: (v[0], v[1] - 0.02) for k, v in pos.items()}  # Adjust label positions
    nx.draw_networkx_labels(B, label_pos, labels=clean_labels, font_size=5)

    plt.axis('off')
    plt.show()



# Plot coiled visual for each article and overall corpus
gamified_operant = extract_text_from_docx('/content/gamified_human_operant_raw_text.docx')
extract_data = extract_text_from_docx('/content/extracting_published.docx')
scmo = extract_text_from_docx('/content/SCMO_plot.docx')
combined_document = " ".join([gamified_operant, extract_data, scmo])

corpus = create_corpus([gamified_operant, extract_data, scmo])
documents = [gamified_operant, extract_data, scmo, combined_document]
```

```
embeddings = get_word_embeddings(corpus)

pca = PCA(n_components=1)
reduced_embeddings = pca.fit_transform(embeddings).flatten()
normalized_embeddings = normalize_embeddings(reduced_embeddings)

for doc in documents:
    create_bipartite_graph(corpus, doc, normalized_embeddings, double_helix=False)
```