TECHNICAL INFORMATION

THE LOGIC AND CODE BEHIND THE COVER & TOC GRAPHS

David J. Cox[1,2]

[1] INSTITUTE FOR APPLIED BEHAVIORAL SCIENCE, ENDICOTT COLLEGE, [2] RETHINKFIRST

Visualizing behavioral data in unique ways may lead to novel methods of analysis 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 the referenced Python script is available here: https://osf.io/b6xuh/. If the code is all you're after, you can stop reading here. Otherwise, the text below describes a bit more about the mathematical and computational logic behind the individual article figures, how they relate to the data shown on the cover page, and why certain decisions were made. Happy coding and playing!

Keywords: natural language processing, embeddings, Euclidean distance, Convex Hull volume

The theme of this special issue is "emerging topics from emerging voices". To capture this visually, I thought it might be interesting to visualize how the topics within each article emerged while a reader reads an article from the first sentence to the last. To do this, two things are required: 1) get the topics for each sentence, and 2) show how the topics emerge sequentially.

There are many ways someone might define the topic(s) of a sentence. Most sentences are about things (nouns) and what those things do (verbs). Thus, once basic pre-processing of each article was completed (e.g., Bengfort et al., 2018; Silge & Robinson, 2017), each sentence was trimmed to only the nouns and verbs in that sentence (Figure 1).

To do mathematical things with textual stimuli, we need to turn the text into numbers. One way to do this is through embeddings. In natural language processing, embeddings are numeric representations of textual stimuli in the form of a vector of real numbers. There are many available techniques to turn textual stimuli into a vector of real numbers and readers interested in learning more about these techniques are encouraged to review Birunda & Devi (2021) and Johnson et al. (2024). Creating a novel set of embeddings, however, typically requires a large amount of data. As such, every sentence for every manuscript in this issue was embedded using the pre-trained embedding model paraphrase-MiniLM-L6-v2 (Reimers & Gurevych, 2019). This model is a modified version of the pretrained BERT network (Devlin et al., 2018) designed specifically to be lightweight and faster while maintaining accuracy as compared to the full BERT model.

As vectors, embeddings give a numeric point in a high-dimensional space that represents the content of the sentence. Thus, two points in the vector space close to one another are more similar in content than two points in the vector space that are further apart. A similar idea in a two-dimensional vector space is that of latitude and longitude to represent the global position of anything on the planet Earth. Using a vector of two numbers, for example, we know that the greatest baseball park of all time is located at [40.4475^{O} N, 80.0072^{O} W]. By comparing other latitude and longitude vectors, we can determine how close other geographical points are to the greatest baseball park of all time. The paraphrase-MiniLM-L6-v2 model accomplishes this same idea but with sentences and in a 384-dimensional vector space (Figure 1).

Humans have a difficult time "seeing" things in 384 dimensions. We live in a three-dimensional world, which can be tricky enough at times (ever try to hit a curveball?), and actively adding in the fourth dimension—time—can sometimes wreak havoc on our behavior (e.g., Amlung et al., 2019; Green et al., 1994). Thus, to aid in the visualization of each article for us mere humans, the total dimensions of each vector were reduced from 384 to three using principal component analysis (PCA; e.g., Gewers et al., 2021) via the scikit-learn package in Python (Pedregosa et al., 2011). The result is that each sentence now has a three-dimensional vector description of the nouns and verbs from that sentence (far right, Figure 1).

As a three-dimensional vector, we can graph each sentence as a point in a three-dimensional

https://doi.org/10.17605/OSF.IO/B6XUH

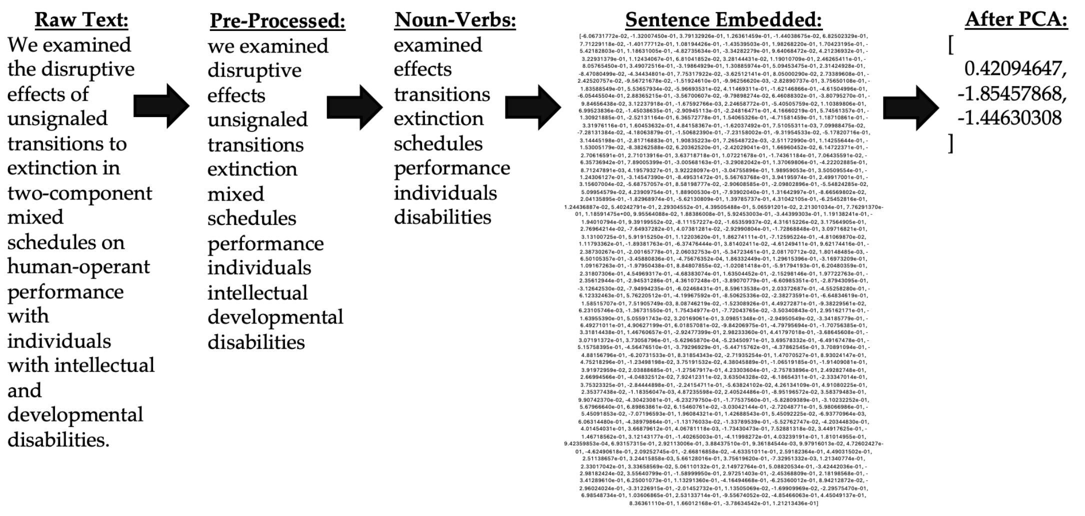
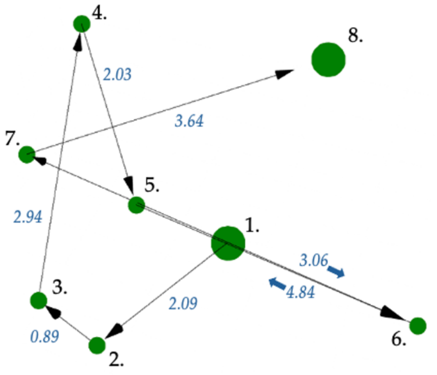

Raw Text:
We examined the disruptive effects of unsignaled transitions to extinction in two-component mixed schedules on human-operant performance with individuals with intellectual and developmental disabilities.
Pre-Processed:
we examined disruptive effects unsignaled transitions extinction mixed schedules performance individuals intellectual developmental disabilities
Noun-Verbs:
examined effects transitions extinction schedules performance individuals disabilities
Sentence Embedded:
After PCA:
[0.42094647, -1.85457868, -1.44630308]

Figure 1. Example of the sequence of textual transformations from a raw, published sentence through simple pre-processing, conversion to a noun-verb list, and sentence embedding.

space. These are the nodes (i.e., circular markers) in the visualizations created for the table of contents. That is, each node is the location of a sentence in the article as denoted by the three-dimensional vector space derived from the process described above.

Continuing this logic, as you read from sentence to sentence in an article, you move from node to node within a three-dimensional vector space (Figure 2). The arrows in each data visualization, therefore, show you the sequence of moving from sentence to sentence in a three-dimensional vector space. To help you find the start, the very first sentence is shown as the largest green node. And, to help you find where it ends, the very last sentence is shown as the largest red node. Lastly, because these things are still visually quite complex, the coloring of all the nodes (i.e., sentences) from first to last are colored from green to red along a systematically changing gradient. The greener the node means closer to the beginning of the article; yellow means you are in the middle; and the redder the node means closer to the end of the article. In total, the result of combining the above is that you can visually see how the topics of each article emerge relative to all other sentences as you read through the article from front to back.

But what does each visualization tell you about the article? One answer might be how much "movement" from topic-to-topic occurs within a manuscript. Because each sentence is a point in space, we can quantify the distance between two sentences via the length of a line connecting those two points (i.e., the Euclidean distance; e.g., Figure 2). A longer distance between two sentences would, thus, represent larger changes in sentence topics. And shorter distances between two sentences would represent shorter changes in sentence topics. Over the course of an article, we can get the cumulative sum of the total Euclidean distance traveled as we read from the first sentence to the last sentence in the article. This is the *y*-axis on the cover art. The greater the



Node	Dim 1 (x)	Dim 2 (y)	Dim 3 (z)	Distance = $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}$
1.	0.42	-1.85	-1.45	
				$2.09 \approx \sqrt{(1.05-0.42)^2+(0.06-(-1.85))^2+(-0.89-(-1.45))^2}$
2.	1.05	0.06	-0.89	
				$0.89 \approx \sqrt{(1.85-1.05)^2+(-0.27-0.06)^2+(-1.07-(-0.89))^2}$
3.	1.85	-0.27	-1.07	
				$2.94 \approx \sqrt{(2.43-1.85)^2+(-2.78-(-0.27))^2+(0.35-(-1.07))^2}$
4.	2.43	-2.78	0.35	
				$2.03 \approx \sqrt{(1.34-2.43)^2+(-1.53-(-2.78))^2+(-0.82-0.35)^2}$
5.	1.34	-1.53	-0.82	
				$3.06 \approx \sqrt{(-1.52-1.34)^2+(-0.65-(-1.53))^2+(-0.19-(-0.82))}$
6.	-1.52	-0.65	-0.19	
				$4.84 \approx \sqrt{(3.00-(-1.52)^2+(-2.07-(-0.65))^2+(-1.16-(-0.19)}$
7.	3.00	-2.07	-1.16	
				$3.64 \approx \sqrt{(-0.15-3.00)^2+(-3.20-(-2.07))^2+(0.26-(-1.16))^2}$
8.	-0.15	-3.20	0.26	

Figure 2. Example of how distance traveled is calculated from node to node (i.e., sentence-to-sentence) within each article.

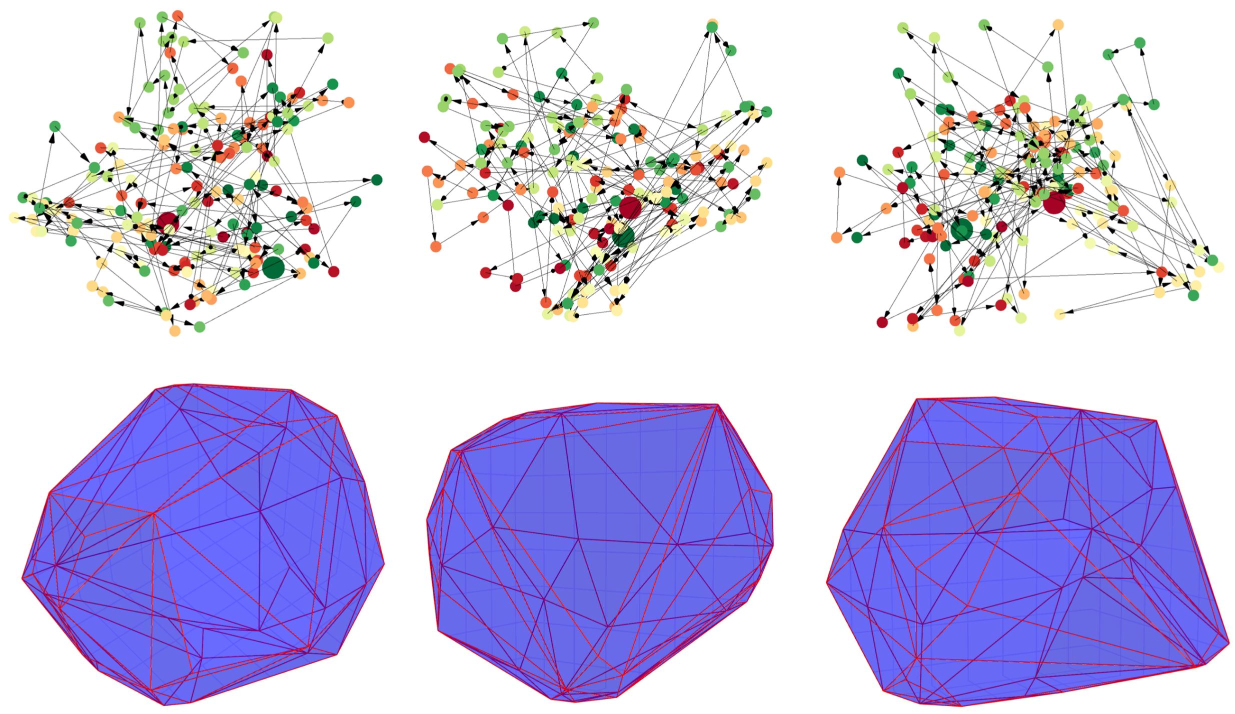


Figure 3. Visual demonstration of converting the topic space covered by an article into its convex hull.

Euclidean distance to cover, the greater the total movement across topics throughout the article.

Understanding the total distance traveled doesn't tell you the total scope or range of topics covered in the article. For example, one could bounce back and forth between two sentences, each being completely different in topic, and the total distance traveled would be quite large but would look like a straight line between two points (e.g., points 5, 6, and 7 that form a straight line in Figure 2).

One method to capture the total topic scope of an article might be through an idea from geometry. Each article leads to a unique structure in three-dimensional space created from the movement through the topic vector space from the first sentence to the last sentence. Now, imagine gift wrapping each of those visualizations using your favorite holiday gift wrapping. The result is a complex, geometrical shape unique to the topic space covered by that article (Figure 3 shows three angles for one of the articles). In geometry, this shape is termed a convex hull and our friends in computational geometry have given us the means to compute it easily using computers (e.g., Avis et al., 1997; Preparata & Shamos, 1985)—thanks, friends!

As an object with a known scale in three dimensions, we can use computers to compute the volume of each convex hull. This is the x-axis on the cover art. The greater the convex hull of an article, the greater the total topic space covered throughout the article. Of note is the absence of a perfect correlation between the total Euclidean distance and the convex hull volume suggesting they provide unique quantitative descriptions of each article.

But are these useful ways to quantitatively model scientific writing? Who knows. Only time will tell what methods for visualizing and quantitatively analyzing verbal behavior are useful. I can say they sure were fun to make. However, there are some obvious potential use cases for each measure that might prove fruitful for more serious academic work on the quantitative analyses of verbal behavior.

One use might be to quantitatively and more precisely define elements of writing style (e.g., Strunk & White, 1959; Williams, 1990). Using the total distance traveled metric, large jumps in topics from sentence to sentence and across the total landscape comprised by an article might suggest the article is more difficult to read when compared to sentence transitions that make the reader "work less". But, too small of jumps might also indicate too much handholding in topic transitions, be overly verbose, and state too many things obvious to the reader. Fun to think about is if "ideal" distances exist based on the influence the writer hopes to have on specific types of readers.

Convex hull volume also seems relevant to manuscript readability. Imagine two manuscripts with identical and overlapping convex hull volumes in three-dimensional space. The density of each might be calculated by dividing the total volume by the number of words or sentences in the article. This could give a metric for measuring relative claims about "omitting needless words" and sufficient brevity in one's writing. That is, if you can cover the same ground in fewer words, the less dense article might be easier to digest.

If you got this far, thanks for reading. I hope you enjoyed these data visualizations of this special issue as much as I enjoyed making them. Happy quantitative and computational analyzing!

REFERENCES

Amlung, M., Marsden, E., Holshausen, K., Morris, V., Patel, H., Vedalgo, L., Naish, K. R., Reed, D. D., & McCabe, R. E. (2019). Delay discounting as a transdiagnostic process in psychiatric disorders: A meta-analysis. *JAMA Psychiatry, 76*(11), 1176-1186. https://doi.org/10.1001/jamapsychiatry.2019.2102

Avis, D., Bremner, D., & Seidel, R. (1997). Hw good are convex hull algorithms? *Computational Geometry, 7*(5-6), 265-301. https://doi.org/10.1016/S0925-7721(96)00023-5

Bengfort, B., Bilbro, R., & Ojdea, T. (2018). *Applied Text Analysis with Python: Enabling Language-Aware Data Products with Machine Learning*. O'Reilly Media. ISBN: 1491963042

Birunda, S. S., & Devi, R. K. (2021). A review on word embedding techniques for text classification. In: J. S. Raj, A. M. Iliyasu, R. Bestak, & Z. A. Baig (Eds.) *Innovative Data Communication Technologies and Application. Lecture Notes on Data Engineering and Communications Technologies, vol 59* (pp. 267-282). Springer. https://doi.org/10.1007/978-981-15-9651-3_23

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. https://arxiv.org/abs/1810.04805

Gewers, F. L., Ferreira, G. R., De Arruda, H. F., Silva, F. N., Comin, C. H., Amancio, D. R., & Costa, L, D. F. (2021). Principal component analysis: A natiral approach to data exploration. *ACM Computing Surveys (CSUR), 54*(4), No. 70, 1-34. https://doi.org/10.1145/3447755

Green, L., Fristoe, N. & Myerson, J. (1994). Temporal discounting and preference reversals in choice between delayed outcomes. *Psychonomic Bulletin & Review 1*, 383–389. https://doi.org/10.3758/BF03213979

Johnson, S. J., Murty, M. R. & Navakanth, I. (2024). A detailed review on word embedding techniques with emphasis on word2vec. *Multimedia Tools and Applications, 83*, 37979–38007. https://doi.org/10.1007/s11042-023-17007-z

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., …, & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825-2830.

Preparata, F.P., Shamos, M.I. (1985). *Computational Geometry. Texts and Monographs in Computer Science*. Springer. https://doi.org/10.1007/978-1-4612-1098-6_3

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. http://arxiv.org/abs/1908.10084

Silge, J., & Robinson, D. (2017). *Text Mining with R*. O'Reilly Media. ISBN: 9781491981658

Strunk, W., & White, E. B. (1959). *The Elements of Style*. Harcourt.

Williams, J. M. (1990). *Style: Toward Clarity and Grace*. The University of Chicago Press.

< Code begins on next page. >

```
# -*- coding: utf-8 -*-

## Packages Needed
# Data manipulation
from IPython.display import clear_output
import pandas as pd
import numpy as np

try:
  from docx import Document
except:
  !pip install python-docx
  clear_output()
  from docx import Document

# NLP
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize
import string
nltk.download('punkt')
nltk.download('stopwords')
try:
  from sentence_transformers import SentenceTransformer
except:
  !pip install sentence-transformers
  clear_output()
  from sentence_transformers import SentenceTransformer

try:
  import spacy
  nlp = spacy.load('en_core_web_sm')
except:
  !pip install spacy
  !python -m spacy download en_core_web_sm
  clear_output()
  import spacy
  nlp = spacy.load('en_core_web_sm')

# Dimensionality reduction
from sklearn.decomposition import PCA

# Topology calculations
from scipy.spatial import ConvexHull
from scipy.spatial.distance import euclidean

# Data visualization
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.graph_objs as go
from matplotlib import cm

"""## Functions"""
def extract_text_from_docx(file_path):
    doc = Document(file_path)
    return "\n".join([para.text for para in doc.paragraphs])


def preprocess_text(text):
    stop_words = set(stopwords.words('english'))
    sentences = sent_tokenize(text)
    cleaned_sentences = []
    for sentence in sentences:
        words = word_tokenize(sentence)
        cleaned_words = [word.lower() for word in words if word.isalpha() and word not in stop_words]
        cleaned_sentences.append(" ".join(cleaned_words))
    return cleaned_sentences


def extract_nouns_verbs(sentence):
    # Process the sentence using spaCy
    doc = nlp(sentence)

    # Extract nouns and verbs
    nouns_verbs = [token.text for token in doc if token.pos_ in ('NOUN', 'VERB')]

    # Join the nouns and verbs back into a sentence
    return ' '.join(nouns_verbs)


def get_embeddings(sentences):
    model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
    return model.encode(sentences, show_progress_bar=False)


def calculate_area_and_distance(embeddings):
    # Calculate the convex hull
    hull = ConvexHull(embeddings)

    # Calculate the surface area of the convex hull
    area = hull.area

    # Calculate the total distance traveled by summing the distances between consecutive points
    total_distance = sum(euclidean(embeddings[i], embeddings[i+1]) for i in range(len(embeddings) - 1))

    return area, total_distance


def reduce_dimensions(embeddings, n_components=3):
    # Reduce the feature space
    pca = PCA(n_components=n_components)
    reduced_embeddings = pca.fit_transform(embeddings)

    # Calculate variance explained
    explained_variance = pca.explained_variance_ratio_
    total_explained_variance = np.sum(explained_variance)

    return reduced_embeddings, total_explained_variance


def plot_embeddings_with_arrows_plotly(embeddings, node_size=10, plot_height=800):
    # Create a gradient of colors from green to red
    cmap = cm.get_cmap('RdYlGn_r', len(embeddings))     # Reversed 'RdYlGn' to go from green to red
    colors = [cmap(i) for i in range(len(embeddings))]  # Get color for each point
    colors = ['rgba({}, {}, {}, 1)'.format(int(c[0]*255), int(c[1]*255), int(c[2]*255)) for c in colors]

    # Initialize figure
    fig = go.Figure()
```

```
# Add scatter points for all embeddings with gradient colors
scatter_sizes = [node_size*2 if i == 0 or i == len(embeddings) - 1 else node_size for i in
range(len(embeddings))]

for i in range(len(embeddings)):
    fig.add_trace(go.Scatter3d(
        x=[embeddings[i, 0]],
        y=[embeddings[i, 1]],
        z=[embeddings[i, 2]],
        mode='markers',
        marker=dict(size=scatter_sizes[i], color=colors[i]),
        name=f'Sentence {i+1}'
    ))

# Add arrows as lines between points with cones at the end
for i in range(len(embeddings) - 1):

    # Calculate the direction vector
    direction = embeddings[i + 1] - embeddings[i]

    # Normalize the direction vector and multiply by the node size to offset the arrow head
    norm_direction = direction / np.linalg.norm(direction)

    # Adjust position so the tip stops before the next node
    cone_start = embeddings[i + 1] - norm_direction * (scatter_sizes[i + 1] / 40)

    # Plot the arrow line
    fig.add_trace(go.Scatter3d(
        x=[embeddings[i, 0], cone_start[0]],
        y=[embeddings[i, 1], cone_start[1]],
        z=[embeddings[i, 2], cone_start[2]],
        mode='lines',
        line=dict(color='black', width=2, dash='solid'),
        opacity=0.6
    ))

    # Plot the cone
    fig.add_trace(go.Cone(
        x=[cone_start[0]],
        y=[cone_start[1]],
        z=[cone_start[2]],
        u=[norm_direction[0]],
        v=[norm_direction[1]],
        w=[norm_direction[2]],
        sizemode="absolute",
        sizeref=0.2,
        showscale=False,
        colorscale=[[0, 'black'], [1, 'black']]
    ))

# Update layout to remove gray background, axis labels, and ticks
fig.update_layout(
    title='',
    scene=dict(
        xaxis=dict(
            showbackground=False,  # Remove gray background from plot
            showticklabels=False,  # Remove axis ticks
            title='',              # Remove axis labels
            ),
            yaxis=dict(
                showbackground=False,
                showticklabels=False,
                title='',
            ),
            zaxis=dict(
                showbackground=False,
                showticklabels=False,
                title='',
            )
        ),
        showlegend=False,

        # Adjust margins to make better use of space
        margin=dict(l=0, r=0, b=0, t=40),

        # Set the height of the plot for easier interaction and recording
        height=plot_height
    )

    fig.show()


def plot_convex_hull_3d_plotly(embeddings, plot_height=1000):
    # Ensure embeddings are in 3D
    if embeddings.shape[1] > 3:
        embeddings = reduce_dimensions(embeddings, n_components=3)

    # Compute the convex hull
    hull = ConvexHull(embeddings)

    # Create the plotly figure
    fig = go.Figure()

    # Plot the convex hull by adding the triangles that make up the hull
    for simplex in hull.simplices:
        # Each simplex is a triangle in 3D space
        fig.add_trace(go.Mesh3d(
            x=embeddings[simplex, 0],
            y=embeddings[simplex, 1],
            z=embeddings[simplex, 2],
            color='rgba(0, 0, 255, 0.8)',  # Set a translucent color for the triangles
            opacity=0.5,
            showscale=False
        ))

    # Plot the red lines along the edges of the convex hull
    for i in range(len(hull.simplices)):
        for j in range(3):
            fig.add_trace(go.Scatter3d(
                x=[embeddings[hull.simplices[i, j], 0], embeddings[hull.simplices[i, (j+1) % 3], 0]],
                y=[embeddings[hull.simplices[i, j], 1], embeddings[hull.simplices[i, (j+1) % 3], 1]],
                z=[embeddings[hull.simplices[i, j], 2], embeddings[hull.simplices[i, (j+1) % 3], 2]],
                mode='lines',
                line=dict(color='red', width=3),
                showlegend=False
            ))
```

```
  # Update layout to remove gray background, axis labels, and ticks
  fig.update_layout(
    title='',
    scene=dict(
      xaxis=dict(
        showbackground=False,  # Remove gray background from plot
        showticklabels=False,  # Remove axis ticks
        title='',              # Remove axis labels
      ),
      yaxis=dict(
        showbackground=False,
        showticklabels=False,
        title='',
      ),
      zaxis=dict(
        showbackground=False,
        showticklabels=False,
        title='',
      )
    ),
    showlegend=False,

    # Adjust margins to make better use of space
    margin=dict(l=0, r=0, b=0, t=40),

    # Set the height of the plot for easier interaction and recording
    height=plot_height
  )

  # Show the plot
  fig.show()


"""## Read in the different articles"""
files = [
  'Falligant et al..docx',
  'Mohamed et al..docx',
  'Randall et al..docx',
  'Regaço et al..docx',
  'Craig et al..docx',
  'Simon.docx',
  'Williams et al..docx']
texts = [extract_text_from_docx(file) for file in files]


"""## Preprocess the Text"""
processed_texts = [preprocess_text(text) for text in texts]

noun_verb = []
for text in processed_texts:
 nouns_verbs_text = [extract_nouns_verbs(sentence) for sentence in text]
 nouns_verbs_text = [sent for sent in nouns_verbs_text if len(sent)>1]
 noun_verb.append(nouns_verbs_text)


"""## Generate Vector Embeddings"""
embeds = []
for text in noun_verb:
 embeds.append([get_embeddings(sentences) for sentences in text])
```

```
"""## Emergence of topics through each manuscript"""
vac = []
convex_hull = []
tot_dist = []

for article in embeds:
    reduced_embeddings, vac_embeds = reduce_dimensions(embeds[article])
    vac.append(vac_embeds)
    area, total_distance = calculate_area_and_distance(reduced_embeddings)
    convex_hull.append(area)
    tot_dist.append(total_distance)
    plot_embeddings_with_arrows_plotly(reduced_embeddings, plot_height=1000)
    plot_convex_hull_3d_plotly(reduced_embeddings)


# All texts
all_embeds = [item for sublist in embeds for item in sublist]
reduced_embeddings, vac_embeds = reduce_dimensions(all_embeds)
vac.append(vac_embeds)
area, total_distance = calculate_area_and_distance(reduced_embeddings)
convex_hull.append(area)
tot_dist.append(total_distance)
plot_embeddings_with_arrows_plotly(reduced_embeddings, plot_height=1000)
plot_convex_hull_3d_plotly(reduced_embeddings)


"""## Metrics from each article"""
docs = [
    'Falligant et al.',
    'Mohamed et al.',
    'Randall et al.',
    'Regaço et al.',
    'Simon',
    'Craig et al.',
    'Williams et al.',
    'All Articles'
]

metrics = pd.DataFrame({
    'document': docs,
    'vac':vac,
    'convex_hull':convex_hull,
    "total_distance":tot_dist
})


# Invert colors
plt.figure(figsize=(7, 7.5))
plt.scatter(metrics['convex_hull'], metrics['total_distance'], s=500, alpha=0.5, c='red')

# Axis labels with inverted color
plt.xlabel('Convex Hull Volume', fontsize=28, color='white', labelpad=12)
plt.ylabel('Euclidean Distance to Travel', fontsize=28, color='white', labelpad=12)

# Annotate bubbles with text in white
for i, txt in enumerate(metrics['document']):
    plt.annotate(txt, (metrics['convex_hull'][i]+5.5, metrics['total_distance'][i]),
            fontsize=12, ha='left', va='center', color='white')

# Set limits
# plt.ylim(250, 1100)
plt.xlim(120, 260)
plt.yscale("log")
ticks = [300, 600, 1200, 2400]
plt.yticks(ticks=ticks, labels=ticks)

# Invert colors for the plot background and spines
plt.gca().set_facecolor('#101010')
plt.gca().spines['top'].set_color('white')
plt.gca().spines['bottom'].set_color('white')
plt.gca().spines['left'].set_color('white')
plt.gca().spines['right'].set_color('white')

# Invert tick colors
plt.gca().tick_params(axis='x', colors='white')
plt.gca().tick_params(axis='y', colors='white')

# Remove the top and right spines
sns.despine(top=True, right=True, left=False, bottom=False)

# Set the figure background to black
plt.gcf().set_facecolor('#101010')

# Show the plot
plt.show()
```