Analyzing Pre-Indo-European Etruscan Theory Using Topological Data Analysis

Helena Welch

Background

High-dimensional data is often difficult to analyze because of the exponential growth of the size of the space in which the data lives as the dimension increases. [1, 2] One example of high-dimensional data comes from language, which contains many different characteristics (dimensions) with which it can be quantified, but not always sufficient data to detect patterns in it. This is especially true for ancient languages where there is a sparsity of texts. [3]

One such ancient language is Etruscan, which was spoken in the Etrurian region of Italy around 800 BC. With more than 13,000 examples of Etruscan text, we have enough data to understand a fair amount of vocabulary but not to concretely determine the origins of the language. [4] Possible families from which Etruscan originated range from the Tyrsenian group to the Pre-Indo-European family. [5, 6, 7]

For my project, I will compare the phonology of ancient and contemporary Indo-European languages to that of Etruscan using topological data analysis (TDA), with the aim of discovering which Indo-European languages are most closely related to Etruscan. Topological data analysis is a method of analyzing patterns in data using topological structures such as loops and holes. [8] While statistics aims to fit data points to lines or other geometries, TDA quantifies the distance between each data point in a point cloud. Previous studies have used TDA in detecting similarities between languages based on their phonological and syntactical structure. [9, 10] It is known to perform well in comparison to other techniques because of its insensitivity to sparsity of data and noise. [9, 10, 11]

Methodology

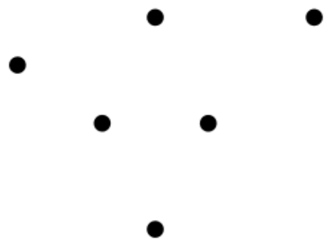
Following the procedure designed by Wolfram [12], I encode a list of words translated into Etruscan and each language to which I will compare Etruscan using their associated International Phonetic Alphabet (IPA) classification. [13, 14] Each sound in a word is encoded by an IPA character, and the list of sounds is universal across all languages. For each instance of an IPA character in a word list, I add the pair of characters that come before and after it, known as the context, to a list. Thus, each IPA character has a list of contexts. I quantify the relationship between two IPA characters in the word list by the cosine similarity C, which is given in Equation 1.

$$C = \frac{\text{number of contexts two characters share}}{\sqrt{\text{length of first context list}} * \sqrt{\text{length of second context list}}} \qquad (1)$$

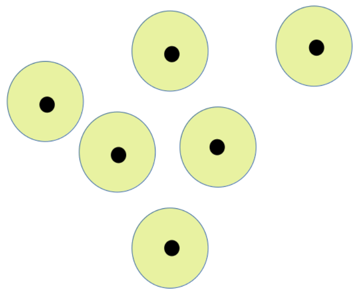
Next I define a coordinate system, where each axis corresponds to a pair of IPA characters. The position along that axis for a given language is the cosine similarity between those two IPA characters. Each language is described by a point in this coordinate system; the closer two points are, the more closely related their associated languages may be.

Thus far, I have calculated the cosine similarities for a given language; the code for this can be found in the Appendix. Those values are stored in a 2-dimensional matrix, where each axis is the list of IPA characters, and each element in the matrix is the cosine similarity for two characters.

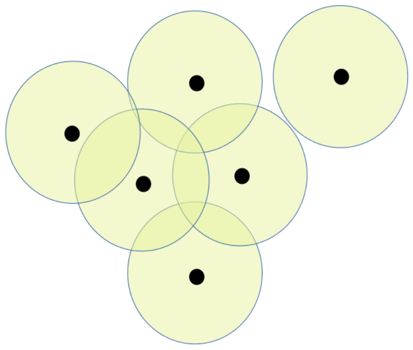
Next, I will perform TDA on the cloud of points within the coordinate system. A simplicial complex, or collection of simplices, is laid over the data and the persistence of each data point, or the amount of time before a given data point overlaps with another point as the simplex around it grows, is calculated (see Figure 1). From there, I can represent the persistence of each data point using a barcode graph. Similarities in the length of time each data point persists between different languages imply that those languages share phonological features and, thus, may be related.



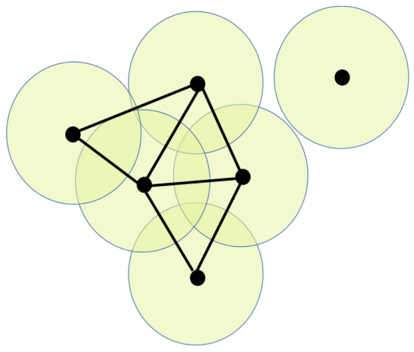
(a) Step 1: Point cloud



(b) Step 2: Generate circles around point cloud



(c) Step 3: Grow circles over a period of time



(d) Step 4: Lay simplicial complex over point cloud, connecting circles that overlap

Figure 1: Process of performing TDA

Next Steps

After calculating the persistence of the point cloud using traditional simplicial complexes, I will attempt to lay a recently-derived geometry over the point cloud. This geometry was discovered when researchers generalized the algorithm for generating Reuleaux triangles to higher dimensions. [15, 16] The Reuleaux triangle has the smallest volume of a shape of constant diameter in two dimensions, meaning that the distance from its center to any point on the boundary is always constant (see Figure 2). The new algorithm allows one to find constant-diameter shapes in any dimension, and it also demonstrates that volumes are a factor of 0.9^n times the volume of an equivalent nth dimensional ball. Thus, the Reuleaux construction's volume decreases exponentially as dimension increases. This makes it a potentially useful simplicial complex in finding relationships between Etruscan and other languages, where there is a sparsity of data in a high-dimensional data set.

Finally, I will compare results from using traditional simplicial complices and the higher-dimensional Reuleaux constructions. The ultimate goal is to discover 1) the significance of similarities between barcode graphs of different Indo-European languages and Etruscan and 2) the effectiveness of utilizing Reuleaux constructions in comparison to other simplicial complexes.

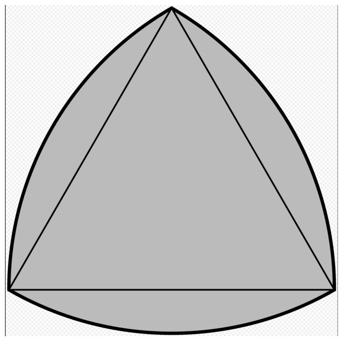


Figure 2: An example of a Reuleaux construction in two dimensions [17]

References

[1] Keogh, Eamonn; Mueen, Abdullah. "Curse of Dimensionality," *Enclyclopedia of Machine Learning and Data Mining*, **2017**.

[2] Altman, Naomi; Krzywinsk, Martin. "The curse(s) of dimensionality," *Nature Methods*, `https://www.nature.com/articles/s41592-018-0019-x`.

[3] Drikvandi, R.; Lawal, O. "Sparse Principal Component Analysis for Natural Language Processing," *Ann. Data. Sci. 10*, **2023**, `https://doi.org/10.1007/s40745-020-00277-x`, 25-41.

[4] Cartwright, Mark. "Etruscan Language," *World History Encyclopedia* **21 February 2017**, `https://www.worldhistory.org/Etruscan_Language/`.

[5] Horvath, Csaba. "Redefining Pre-Indo-European Language Families of Bronze Age Western Europe: A Study Based on the Synthesis of Scientific Evidence from Archaeology, Historical Linguistics and Genetics," *European Scientific Journal ESJ*, `15.10.19044/esj.2019.v15n26p1`.

[6] Rogers, Adelle. "Theories on the Origin of the Etruscan Language," *Open Access Theses*, **2018**, *1587*, `https://docs.lib.purdue.edu/open_access_theses/1587`.

[7] Van der Meer, Bouke. "Etruscan Origins," *Language and Archaeology, BABESCH*, 79, **2004**, 51-57.

[8] Saul, Nathaniel; Tralie, Chris. "Scikit-TDA: Topological Data Analysis for Python," Zenodo. `http://doi.org/10.5281/zenodo.2533369`.

[9] Port, A.; Gheorgita, I.; Guth, D.; et. al.. "Persistent Topology of Syntax," *Mat.Comput.Sci*, **2018** *12*, `https://doi.org/10.1007/s11786-017-0329-x`, 33-50.

[10] Singh, R.P.; Malott, N.O.; Sauerwein, B.; Mcgrogan, N.; Wilsey, P.A. "Generating High Dimensional Test Data for Topological Data Analysis," In: Hunold, S., Xie, B., Shu, K. (eds), *Benchmarking, Measuring, and Optimizing. Bench 2023.* **2024**. Spring, Singapore. `https://doi.org/10.1007/978-981-97-0316-6_2`.

[11] Taiwo, Funmilola Mary et al. "Explaining the Power of Topological Data Analysis in Graph Machine Learning," **2024**, ArXiv abs/2401.04250.

[12] Wolfram, Catherine. "Persistent homology on phonological data: a preliminary study," `https://math.uchicago.edu/~may/REU2017/REUPapers/Wolfram.pdf`.

[13] Salvucci, Claudio R.. "A Vocabulary of Etruscan: including the Etruscan gosees," Southampton, PA, *Evolution Publishing.*

[14] Bonfante, Giuliano; Bonfante, Larissa. "The Etruscan Language: an Introduction," *University of Manchester Press*, **2002**.

[15] Arman, Andrii; et. al. "Small bodies of constant width," arXiv:2405.18501.

[16] Arman, Andrii; et. al. “Convex bodies of constant width with exponential illumination number,” *Discrete and Computational Geometry*, **2024**.

[17] By Geoff Richards (Qef) - Own work, Public Domain, `https://commons.wikimedia.org/w/index.php?curid=4664686`

Appendix

The following code produces a matrix for a sample word list in American English.

InterimReportCode

December 20, 2024

[1]:
```
import numpy as np
```

[2]:
```
IPAamericanenglish = " aI, ju, hi, wi, ju, there, this, that, hir, ju, wut,␣
 ↪wer, wen, hau, nat, el, meni, sam, fju, raIt, left "
```

[3]:
```
def MatrixMaker(language_list):
    #turns string of words into a list
    language_list.replace("''","")
    language_list.replace(","," ")
    good_list = list(language_list)

    #creates list of letters that appear in word list
    LetterList = []
    for letter in good_list:
        if letter != ',':
            if letter != ' ':
                if letter not in LetterList:
                    LetterList.append(letter)

    counter0 = 0

    #creates numpy array of zeros
    arr0 = np.zeros((len(LetterList), len(LetterList)), dtype=object)


    for letter1 in LetterList:
        CosineSimilarities = []
        counter1 = 0
        for letter2 in LetterList:
            #creates list of letters that come before and after a given letter
            #in the word list (called the context of the letter)
            Contexts1 = []
            Contexts2 = []
            for i in range(0, len(good_list)):
                if good_list[i] == letter1:
                    context = [good_list[i-1], good_list[i+1]]
                    Contexts1.append(context)
                if good_list[i] == letter2:
                    context = [good_list[i-1], good_list[i+1]]
                    Contexts2.append(context)

            #counts how many pairs of contexts two letters have in common
            counter = 0
            shared_contexts = []
            j_contexts = []
            for i in range(0, len(Contexts1)):
                for j in range(0, len(Contexts2)):
                    if Contexts1[i] == Contexts2[j]:
                        if j_contexts.count(Contexts2[j]) < Contexts2.
↪count(Contexts2[j]):
                            j_contexts.append(Contexts2[j])
                            counter += 1

            #appends the cosine similarity of two letters
            #(how many contexts the letters had in common divided by
            #the square of the number of contexts each letter has)
            CosineSimilarities.append(counter/((len(Contexts1)**0.
↪5)*(len(Contexts2)**0.5)))

        #adds each element of the list of cosine similarities to a matrix
        for element in CosineSimilarities:
            arr0[counter0][counter1] = element
            counter1 += 1
        counter0 += 1

    return arr0
```

[ ]:
```
MatrixMaker(IPAamericanenglish)
```