Istanbul Technical University - Science and Letters Faculty

Mathematics Engineering Program

Word Embeddings and Phylogenetic Trees

Student Name: Arif Çakır

Student Number: 090190355

Code: MAT4092E

Advisor: Prof. Dr. Atabey Kaygun

Submission Date: February 11, 2024

Contents

1 Introduction

Natural Language Processing (NLP) is the subfield of machine learning that studies the processing of human language by computers. Due to its interesting and interdisciplinary subject, NLP is one of the most popular fields in the data science scene. Its popularity caused NLP to branch out into different subfields such as word embedding. Word embedding is one of the subfields of NLP that studies the semantic analysis of sets of words. This semantic analysis is done by word embedding generating vectors of words to mathematically analyze them. It can be used on various subjects such as translation, text classification, and sentiment analysis. There are several word embedding methods, such as Term Frequency - Inverse Document Frequency (TF-IDF) [10], Global Vectors of Word Representation (GloVe) [14], Bidirectional Encoder Representations from Transformers (BERT) [21], Skip-Gram, and Continous Bag Of Words (CBOW) [13].

1.1 Word Embedding Methods

Word embedding methods have different advantages and disadvantages. TF-IDF method is formed of two techniques: Term Frequency (TF) technique counts the occurrence of the words in a document by using log probability, while inverse document frequency (IDF) is practically the inverse of the TF method. IDF algorithm relies on the information gained from the words which are rarely used. As a disadvantage, the TF-IDF algorithm does not capture the semantic relationships between words and takes them as independent values [24]. Due to that disadvantage, TF-IDF is mostly used in stop word detection. GloVe is an unsupervised learning algorithm that captures global contextual information of words by calculating the global word-word co-occurrence matrix. GloVe performs well in word analogy and named entity recognition tasks due to it handling vocabulary words better. Bidirectional Encoder Representations from Transformers (BERT) is a family of masked-language models. It is a bidirectional and task-agnostic model which makes BERT to stand out. Skip-Gram and CBOW models are used in the Word2Vec technique, which will be discussed in the following chapters.

1.2 Language Clustering Using Word2Vec

In the first part of the graduation project, the clustering of languages was made by using Word2Vec. The main hypothesis of the project was showing the similarity of languages can be presented by the similarity of the common words in that language. To show this, a set of words in different languages was taken as a dataset and then the similarity of the top words in that text was calculated. The distances in the similarity matrix of top words should be similar to the linguistic distance of the languages.

The application was done by using The Divine Comedy in six different languages as the dataset, the data was taken from Project Gutenberg (https://www.gutenberg.org). The languages of the versions of The Divine Comedy used in the application were Dutch, English, Finnish, German, and Italian.

The application was made using Python. Firstly, the words were tokenized, sorted, and aligned for each language. After that, vectors of the top 50 words of each language was calculated. Finally, the cosine similarity of those vectors was calculated and dendrograms were constructed showing languages. One of the dendrograms can be seen in Figure 1.

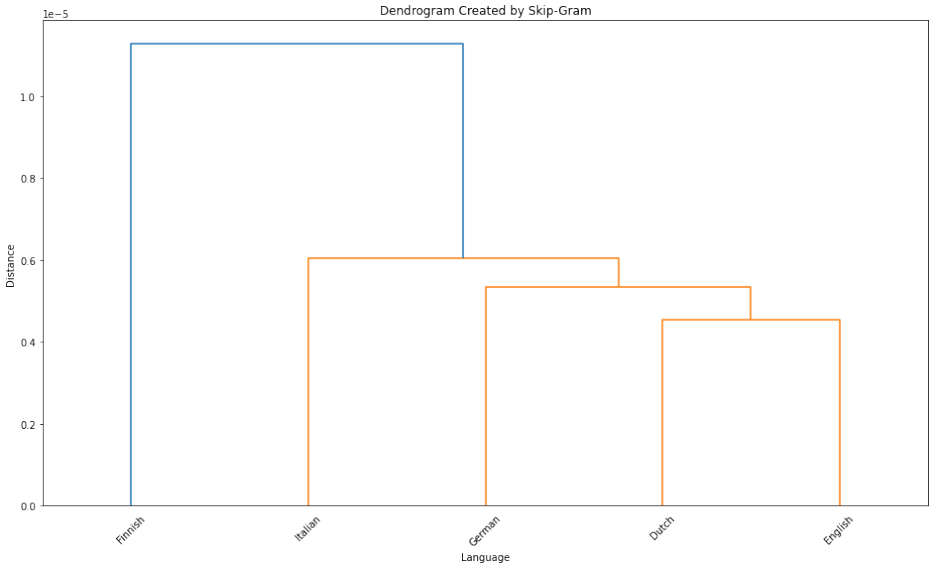


Figure 1

Language dendrogram generated by using Skip-Gram from the first graduation project.

1.3 Usage of Word Embedding Methods on Biological Sequences

As can be seen in the previous part, word embedding can be used to represent the semantic similarity between languages. This practice can be expanded to different fields, such as genome embedding. Genome embedding is the method of transforming genomes as " the language of nature" into semantic space. This way, as was the case in word embedding, it can be said that the similarity of genomes can be calculated by the similarity of the vectors they represent. Due to that, the similarities of the organisms can be calculated. By this organism similarity, a taxonomy can be formed. The main hypothesis of this project is that a taxonomy can be formed using genome vectors generated using genome embedding methods. To test this hypothesis, acidobacteriota genome data from the National Center of Biotechnology Information (NCBI) was used. ProtVec and Word2Vec were used to generate vectors.

The project consists of 5 different sections. In section 2, related works of the scientific literature to the subjects are given and summarized. This was followed by the discussion of the methodology in section 3. After that, details of the experiment were given in section 4. Finally, the results were discussed in section 5.

2 Related Work

In this section, a literature review of relevant topics was presented and discussed.

2.1 Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics

Asgari and Mofrad [15] proposed continuous embedding for biological sequences in 2015, for both gene vectors (GeneVec) and protein vectors (ProtVec). The method uses extracted k-mers in the sequence to learn the embedding word vector and uses the Skip-Gram neural network to train the embeddings. Authors use Swiss-Prot database (https://www.uniprot.org/uniprotkb?query=*&facets=reviewed%3Atrue) to train the model. The method tested for two different tasks: protein family classification and disordered protein detection. For the protein family classification task, the Protein Family

Database (Pfam) [26] is used to extract protein family information of protein sequences. With Support Vector Machines (SVM) [4], ProtVec resulted in a 93% accuracy score in this task. For the disordered protein classification tasks, the DisProt database and FG-Nups database to obtain disordered proteins. Two random sets of the same sizes from Protein Data Bank (PDB) were compared with the data from DisProt or FG-Nups databases by using SVM. ProtVec resulted in an accuracy score of 99.81% in this task.

2.2 dna2vec: Consistent vector representations of variable-length k-mers

Another method used for biological sequence embedding is Dna2Vec [19], which converts the sequence into overlapping fixed-length k-mer by sliding windows. Authors train Dna2vec is also based on Word2Vec, while training the model. The authors trained dna2vec in four stages. Firstly, they separated the genome into long, non-overlapping DNA fragments. Then they converted long DNA fragments into overlapping variable-length k-mers. After that, they applied unsupervised training of an aggregate embedding model, in which they used a two-layer neural network. Finally, they decomposed the aggregated model by k-mer lengths.

In the first stage, the genome sequence fragmented based on gap characteristics, also reverse-complement of fragment was used to increase entropy. In the second stage, DNA sequence S was converted into overlapping fixed length k-mer by sliding a window of length k across S. To determine the size of each window, the k was sampled from the discrete uniform distribution $Uniform(k_{low}, k_{high})$. In stage three, a shallow two-layer neural network based on the skip-gram model of the Word2Vec method [13] was used to train an aggregate DNA k-mer embedding. Context size 10 was used before and after the targeted word, which makes a total of 20 k-mers. During training, negative sampling is used to optimize the update procedure over all words. In the last stage, the model by decomposed by k-mer length to form $k_{high} - k_{low} + 1$ models. For experiments, authors used the hg38 dataset [17].

2.3 Continuous Embeddings of DNA Sequencing Reads and Application to Metagenomics

Proposed by Menegaux and Vert, fastDNA [20] is a method that utilizes the fastText library. To increase the robustness, fastDNA uses random base mutations and reverse complements of sequences. Embedding vectors of k-mers of sequence are used as a vector of dimension d and size k. In experiments, authors tested the model on two benchmarks from [18]. One of those two datasets is smaller and more useful for parameter tuning than the other. The smaller database contains 356 complete genomes, belonging to 51 species of bacteria; its validation set is composed of 52 genomes, that belong to the same 51 species. The large database contains 2,961 genomes belonging to 774 species, while the validation set is composed of 193 genomes, each from a separate species. Authors compared the model with Support Vector Machines (SVM) [4] and Burrows-Wheeler Aligner (BWA) [7] and outperformed both in both datasets.

2.4 BERT contextual embeddings for taxonomic classification of bacterial DNA sequences

Helaly et al. [29] utilize BERT for the classification task of bacterial biological sequences for the first time. As the datasets, the authors used two different 16S rRNA datasets. One of the datasets is the GreenGenes dataset and the other is the RDP dataset. The former was used to pre-train BERT and the latter was used to train the CNN classifier. To train the BERT model, authors extracted k-mers from sequences, and then partitioned sequences. BERT was pre-trained using prepared pre-train data. Contextual embeddings of different datasets were extracted using the BERT model, which was then used to train a Convolutional Neural Networks (CNN) classifier to perform taxonomic classification. BioSeqBERT-CNN gets 93.5% accuracy with the original dataset and 99.9% accuracy on the augmented dataset on the most fine-grained taxonomic rank.

2.5 DNABERT-2: Efficient Foundation Model and Benchmark For Multi-Species Genome

Another method that utilizes BERT for genome embedding is DNABERT-2 [31], proposed by Zhou et al. in 2023. As the improved version of DNABERT [25], DNABERT-2 is a method that replaces k-mer tokenization with Byte Pair Encoding (BPE) [3] (https://www.semanticscholar.org/paper/A-new-algorithm-for-data-compression-Gage/1aa9c0045f1fe8c79cce03c7c14ef4b4643a21f8) and replaces positional embedding with Attention with Linear Bias (ALiBi) [27].

In the experiments, authors compared DNABERT-2 with DNABERT and Nucleotide Transformer (NT) [30]. While DNABERT employs the same architecture as BERT-base, it has a different vocabulary size, which is dependent on the chosen k-mer. DNABERT has four variants, which are DNABERT (3-mer), DNABERT (4-mer), DNABERT (5-mer), and DNABERT (6-mer), which utilize overlapping 3/4/5/6-kmer tokenization respectively. NT scales the data and model size to achieve state-of-the-art performance in 27 DNA analysis tasks. NT also has 4 variants, which are NT-500M-human, NT-500M-1000g, NT-2500M-1000g, and NT-2500M-multi. The authors compared these methods in the Genome Understanding Evaluation (GUE) benchmark they proposed. GUE benchmark includes 7 genome sequence classification problems with 28 datasets. DNABERT-2 outperformed other methods in 4 for 7 tasks tested, while NT methods performed better in 2 other tasks, and 5-mer DNABERT outperformed others in the human Carboxypeptidase D (CPD) dataset.

3 Methodology

In this section, the methods that will be used in the experiments will be discussed. The methods were discussed in three sections: vectorization methods, clustering methods, visualization, and measuring methods.

3.1 Vectorization Methods

3.1.1 Biovec

As the first vectorization method, a Python module named biovec [28] was used. the module utilizes the ProtVec method [15] proposed by Asgari and Mofrad. The method uses an n-gram model. Even though usually an overlapping window of three to six residues is used in the n-gram model in bioinformatics, 3 lists of shifted non-overlapping words are taken instead of overlapping windows. This can be seen in Figure 2. Evaluation of K-nearest neighbors classification [8] in 2-fold cross-validation [9] for different windows sizes for different window sizes, embedding vector sizes, and overlapping versus non-overlapping n-grams shows that a more consistent embedding training for a window size of 3 was obtained.

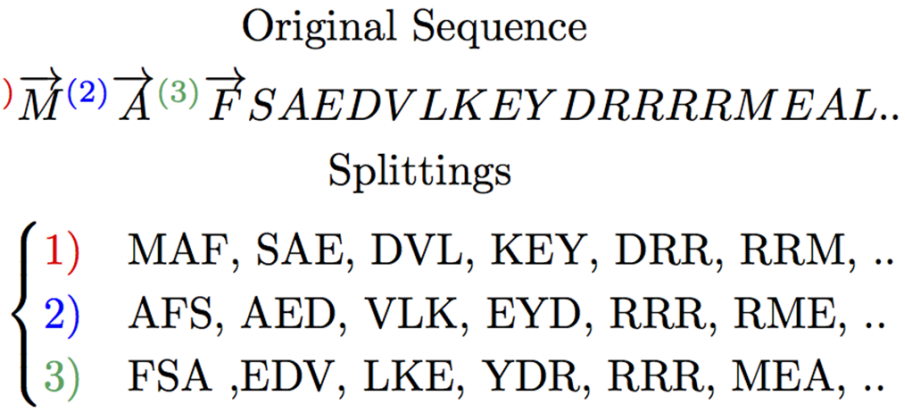


Figure 2

Data Preparation from [15]

Furthermore, Biovec uses Skip-gram neural network [13] to train the embeddings. Skip-gram is one of the two models of the Word2Vec method. The log loss function that the Skip-Gram model tries to minimize is written as

$$-\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} logP(w^{(t+j)}|w^{(t)}). \quad (1)$$

In this equation w_i is the center word in time step t in a text sequence T. Furthermore, we

can generate the equation

$$P(w_o|w_c) = \frac{exp({\mathbf{u}_o}^\top \mathbf{v}_c)}{\sum_{i\in\mathcal{V}} exp({\mathbf{u}_i}^\top \mathbf{v}_c)}. \tag{2}$$

.

In this equation, $P(w_o|w_c)$ represents the conditional probability of generating the word w_o, which is any context word, by using w_c, which is the center word. The vectors used in this probability are the vectors of those words. Finally, the likelihood function for the Skip-Gram model for context window size m is

$$\prod_{t=1}^{T} \prod_{-m\leq j\leq m, j\neq 0} P(w^{t+j}|w^t). \tag{3}$$

The module named models module in Biovec has a class called ProtVec, which helps to train the model based on input Fasta file. After training, the model can generate vectors using the to_vecs function.

3.1.2 Word2Vec

One of the most popular methods in word embedding, Word2Vec was compared with its biological counterpart ProtVec. Two model architectures can be used in Word2Vec, which are called Skip-Gram and Continuous Bag of Words (CBOW). While Skip-Gram generates the surroundings of the word by using it, CBOW generates the word by using its surroundings. The properties of the Skip-Gram model are given in the Biovec section, and for CBOW, the log loss function is

$$-\sum_{t=1}^{T} logP(w^{(t)}|w^{t-m}, ..., w^{(t-1)}, w^{(t+1)}, ..., w^{(t+m)}), \tag{4}$$

where

$$P(w_c|w_o, ..., w_{o_{2m}}) = \frac{exp(\frac{1}{2m} u_c^\top (\mathbf{v}_{o_1} + ... + \mathbf{v}_{o_{2m}}))}{\sum_{i\in\mathcal{V}} exp(\frac{1}{2m} u_c^\top (\mathbf{v}_{o_1} + ... + \mathbf{v}_{o_{2m}}))}, \tag{5}$$

and finally, the context window size of m is

$$P(w_c|w_o,...,w_{o_{2m}}) = \frac{exp(\frac{1}{2m}u_c^\top(\mathbf{v}_{o_1}+...+\mathbf{v}_{o_{2m}}))}{\sum_{i\in\mathcal{V}} exp(\frac{1}{2m}u_c^\top(\mathbf{v}_{o_1}+...+\mathbf{v}_{o_{2m}}))}. \tag{6}$$

.

All the variables are the same as Skip-Gram. Gensim [12] library in Python was used to access Word2Vec architectures.

3.2 Clustering Methods

For clustering, Hierarchical clustering was used. By sequentially pairing variables and clusters, the hierarchical clustering method produces a unique set of clusters (or nested categories). All clusters and not clustered variables are tried in all possible pairs, and the best pair producing the highest average intercorrelation is chosen as the new cluster. This proceeds sequentially from less inclusive clusters through larger more inclusive clusters and is continued until all variables are clustered in a single group [2].

The linkage method is used to calculate the distance $d(s,t)$ between clusters s and t. The linkage algorithm used (https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html) begins with forming a forest of clusters that are not yet used in the hierarchy. Cluster u is formed by combining clusters s and t, and then removing them from the forest. The algorithm continues until there remains only one cluster in the forest. The final cluster in the forest becomes the root. Ward variance minimization algorithm [1], which is also known as an incremental algorithm, is used to get linkages. The distance $d(u,v)$ is computed as

$$d(u,v) = \sqrt{\frac{|v|+|s|}{T}d(v,s)^2 + \frac{|v|+|t|}{T}d(v,t)^2 - \frac{|v|}{T}d(s,t)^2} \tag{7}$$

.

In this equation, u is the combined cluster formed of s and t, v is an unused cluster in the forest, and T $= |v| + |s| + |t|$ is the cardinality of its argument.

3.3 Visualization and Measuring Methods

For visualization of the hierarchical clustering, the dendrogram method is used. A dendrogram is a diagram that illustrates a tree or hierarchical clustering. For hierarchical clustering, the dendrogram illustrates how each cluster is composed by drawing a U-shaped link between a cluster and its children. The length of the two legs of the U-link represents the distance between the child clusters (https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.dendrogram.html.

The tanglegram is a method to compare two dendrograms. It allows the visual comparison of two dendrograms, by facing them one in front of the other and connecting their labels with lines [16].

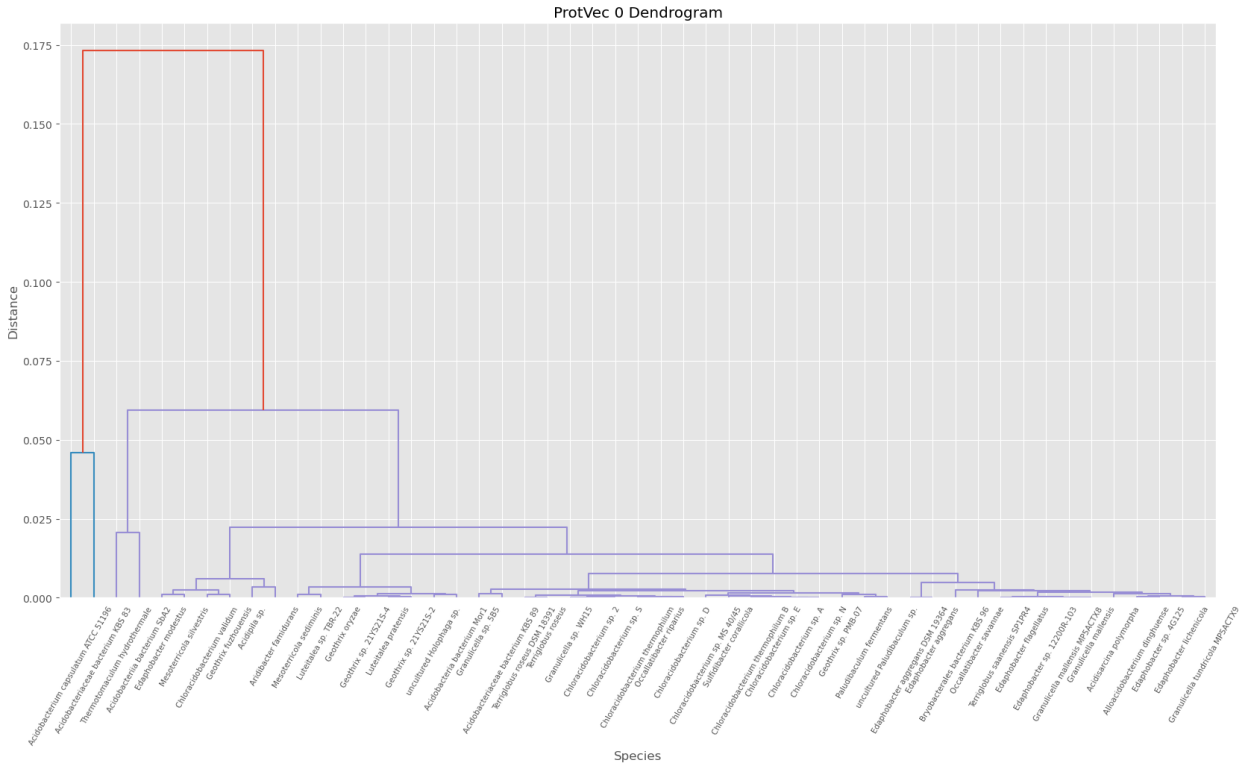


Figure 3

The dendrogram generated by the vectors of ProtVec 0.

4 Experiments

4.1 Description of the Experiment

As an application, a taxonomy of acidobacteriota is made using embedding methods. Acidobacteriota is a phylum of bacteria superkingdom and it has five children: Blastocatellia, Holophagae, Terriglobia, Thermoanaerobaculia, and Vicinamibacteria. The main idea behind the experiment is that the similarity of the organisms can be measured using the semantic similarity of their genome sequences. As the dataset, the NCBI genome dataset is used.

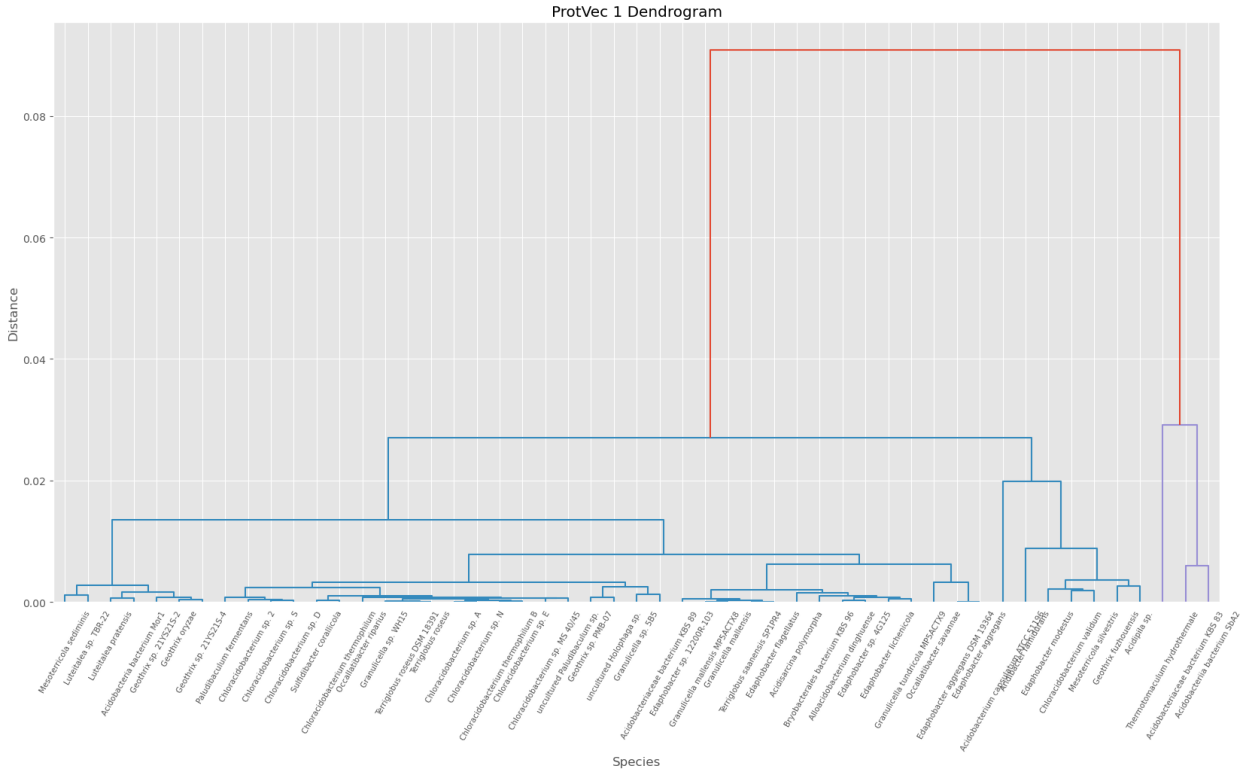


Figure 4

The dendrogram generated by the vectors of ProtVec 1.

4.2 Dataset Description

The Data was downloaded from Genome section of NCBI dataset (https://www.ncbi.nlm.nih.gov/datasets/genome/?taxon=57723&reference_only=true&typical_only=true&assembly_level=3:3). As selected taxa, *Acidobacteriota* was

chosen. Due to the aim of constructing a taxonomy, only reference genomes and genomes with complete assembly levels were chosen. Also, atypical genomes were excluded. A table containing columns Assembly for Assembly name, RefSeq for Assembly accession number, the scientific name for the name of the organism, the annotation for the submitter of the annotation, size (MB) for the size of the annotation, genes for gene count of the annotation, and protein-coding for gene protein-coding of the annotation.

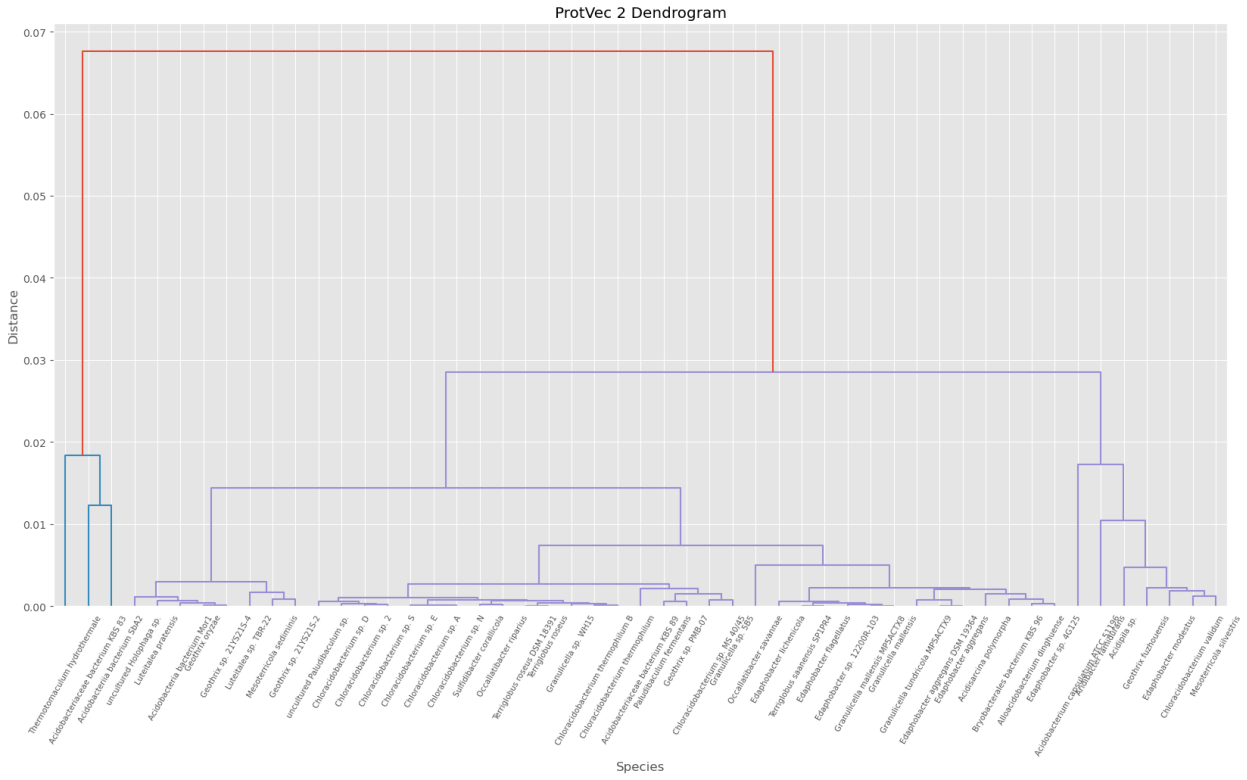


Figure 5

The dendrogram generated by the vectors of ProtVec 2.

4.3 Dataset setup

The table downloaded from NCBI Genome was opened as a pandas data frame [22]. While clearing the dataset, only the annotations from the NCBI Prokaryotic Genome Annotation Pipeline (PGAP) were used for consistency. Also, organisms that contain Candidate, Candidatus, or uncultured in their name are dropped because them being only candidates to their representing group. Finally duplicates in organism names are dropped. Following

that, taxonomy IDs of the species are obtained from the Entrez Molecular Sequence Database System (https://www.ncbi.nlm.nih.gov/Web/Search/entrezfs.html) by using Biopython's [6] Entrez module and saved as the "taxid" column. By using Bio.Entrez, genome sequences of each taxid are obtained and saved as the "sequence" column, while sequences that could not obtained are dropped.

Also, a control dataset is created by lineage information from the Entrez database. While creating the dataset, organisms in the base data frame were used. After obtaining lineage information, each section expanded into different columns of the data frame, and then domain, phylum, and class columns were dropped.

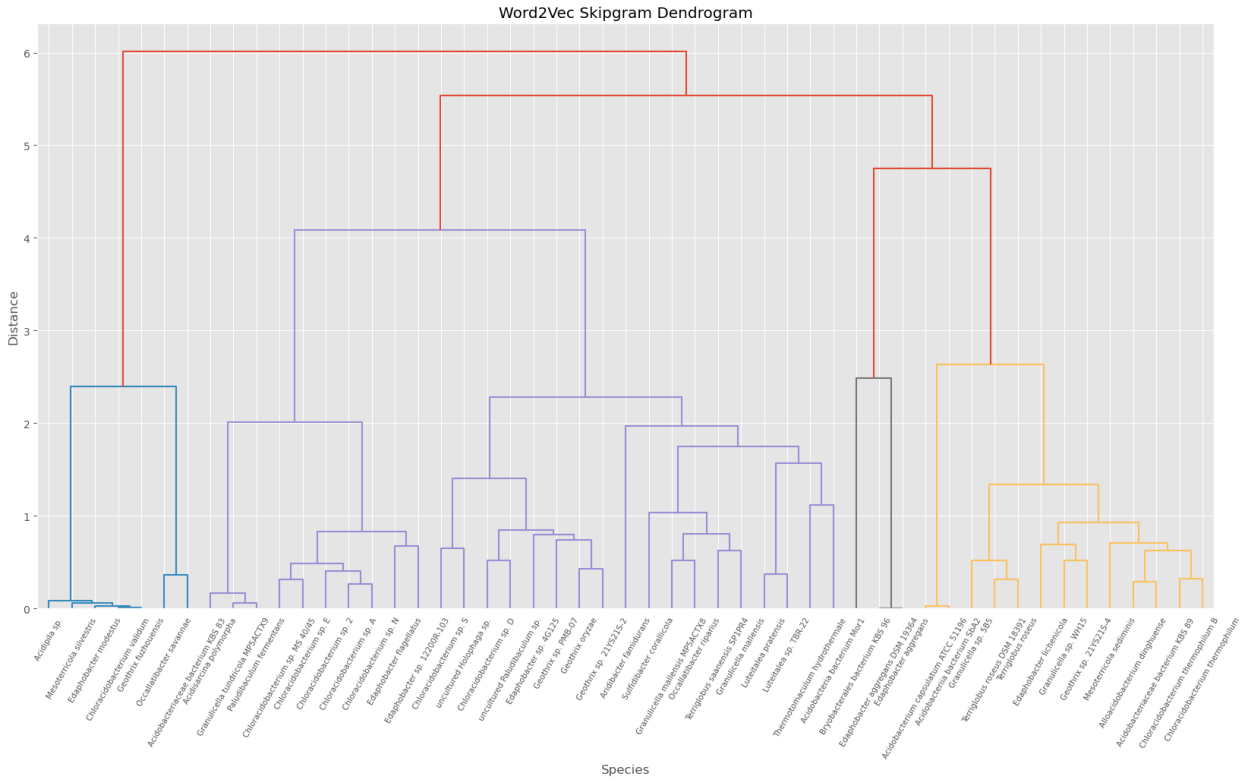


Figure 6

The dendrogram generated by the vectors of Word2Vec Skip-Gram.

4.4 Experimental Setup

A FASTA file version of the data frame is created, using Assembly Accession and sequence rows. After that, models.ProtVec function from the biovec module in Python. ProtVec

model is trained by using this FASTA file, then the model is saved as the "pv" variable. This was followed by generating flattened vectors of sequences. pv.to_vecs function generates three different vectors, and those vectors were flattened using the flatten() function of Python. Finally, three different columns of vectors were generated due to three ProtVec vectors. They saved as protvec_0, protvec_1, and protvec_2.

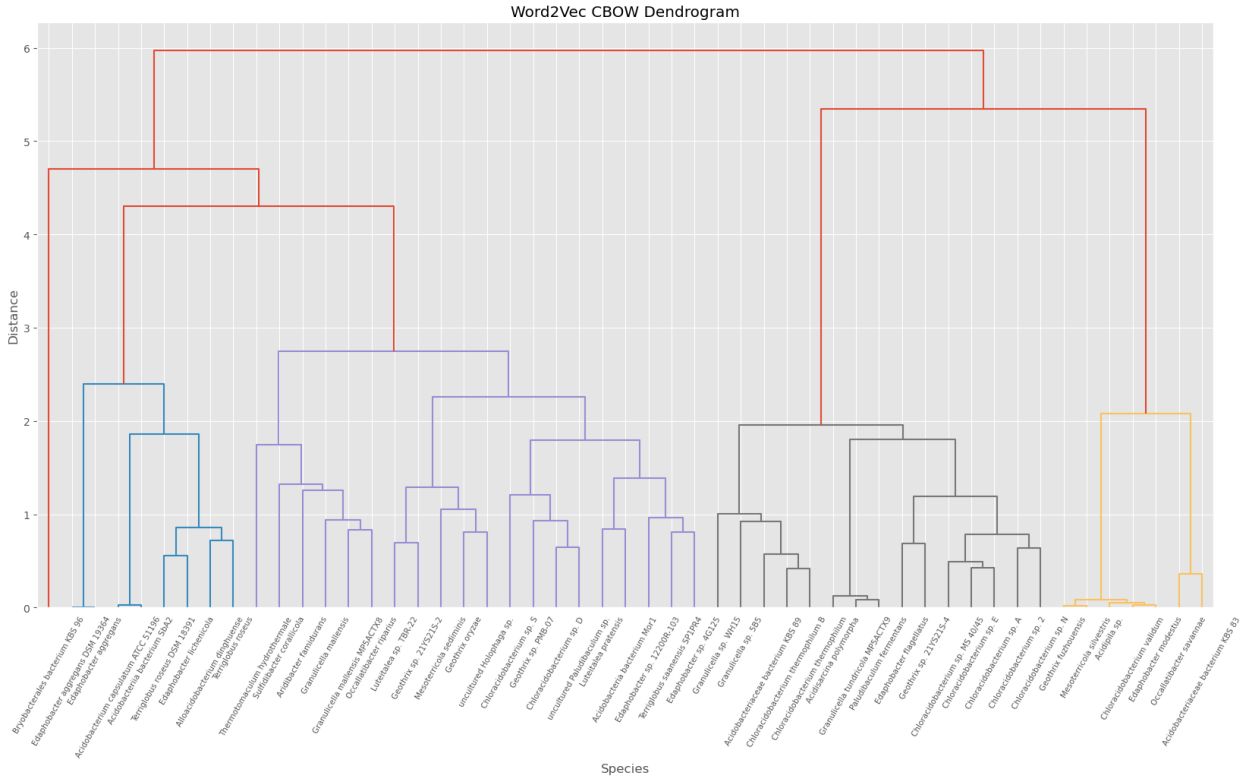


Figure 7

The dendrogram generated by the vectors of Word2Vec CBOW.

After generating the vectors from ProtVec, the popular Word2Vec library Gensim [12] was used to generate vectors. Firstly, sequences split into three-letter groups to form codon-like little sequences, which can be called biological words. After that, split sequences were used to train Gensim models. module library has a Word2Vec class to train the embedding and embeddings were generated with min_count 3, which ignores all words smaller than size 3. Also, the vector size was set as 100. Both Skip-Gram and CBOW models were trained. From those models, vectors from the wv variable (Word2VecKeyedVectors) were taken and

flattened. Flattened vectors are set as new columns of the data frame, where the column names are word2vec_skipgram and word2vec_cbow. Finally, Organism names were set as indexes.

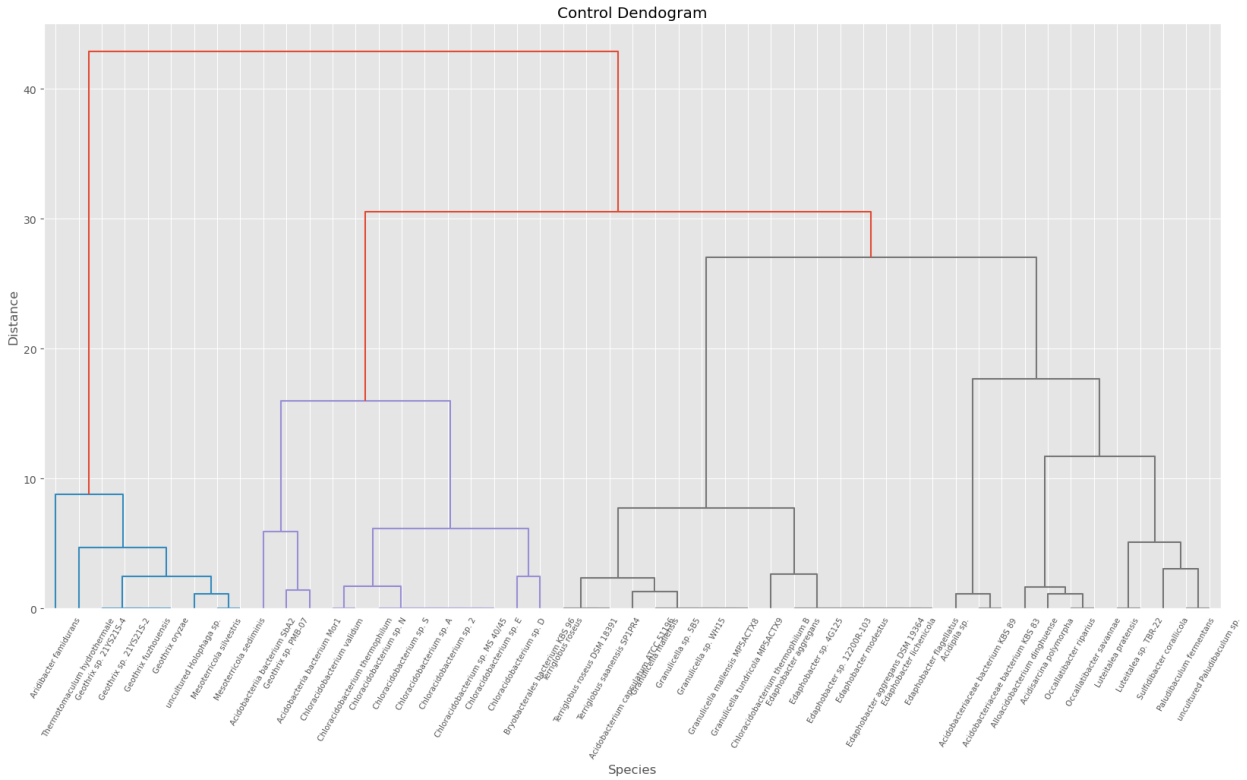


Figure 8

The control dendrogram.

The similarities of vectors were calculated using metrics.pairwise.cosine_similarity function from the Sklearn module [11] in Python. This function generated a similarity matrix of each organism for each vector. To get clusters, the cluster.hierarchy.linkage function from the Scipy module [23] from Python was used. As the method, the 'ward' method was used. This way the linkages of corresponding similarity matrices were obtained.

4.5 Visualization

For visualizations of the linkages, the dendrogram function of scipy.cluster.hierarchy and Matplotlib module [5] were used. For each dendrogram, leaf rotation was set as 180 degrees, leaf font size set as 2, and labels were set as indexes of the sequence data frame. Also for

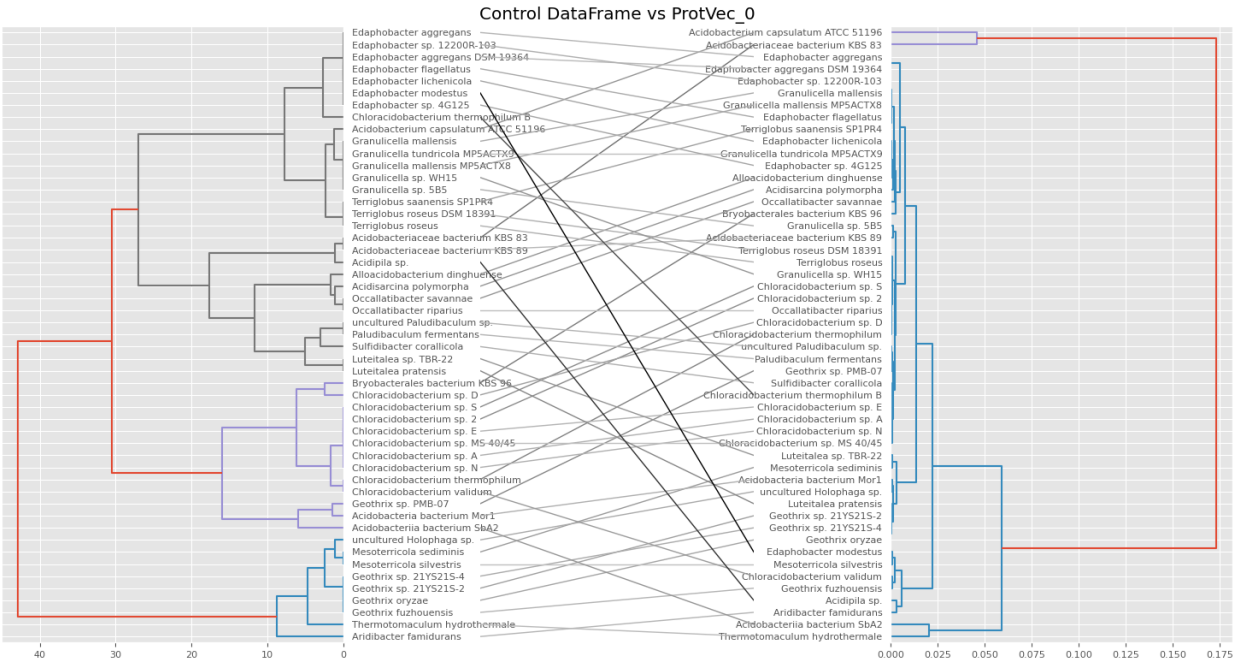


Figure 9

Tanglegram of control dendrogram vs dendrogram of ProtVec 0.

the plot, the x-axis rotation was set as 60 degrees, and the font was set as 7.5. Label of the y-axis set as "Distance", and label of the x-axis set as "Species". Figure size set as 20 to 10.

4.6 Measurement

Finally, the tanglegram method was used to see the performance of the dendrograms. To create tanglegrams, the Python module tanglegram (https://github.com/schlegelp/tanglegram) was used. The linkages of each dendrogram are compared with the control dataset. Also, each linkage was compared with each other to test their similarity.

5 Discussion

5.1 Hierarchial Clusterings

After vectorizations, 6 different dendrograms were obtained. 3 of them were generated using ProtVec, 2 of them were generated using Word2Vec (Skip-Gram and CBOW), and last one is the control data frame. The ProtVec method generated three different vectors,

which will be called as ProtVec 0, ProtVec 1, and ProtVec 2. The dendrogram generated by the vectors of ProtVec 0 can be seen in Figure 3.

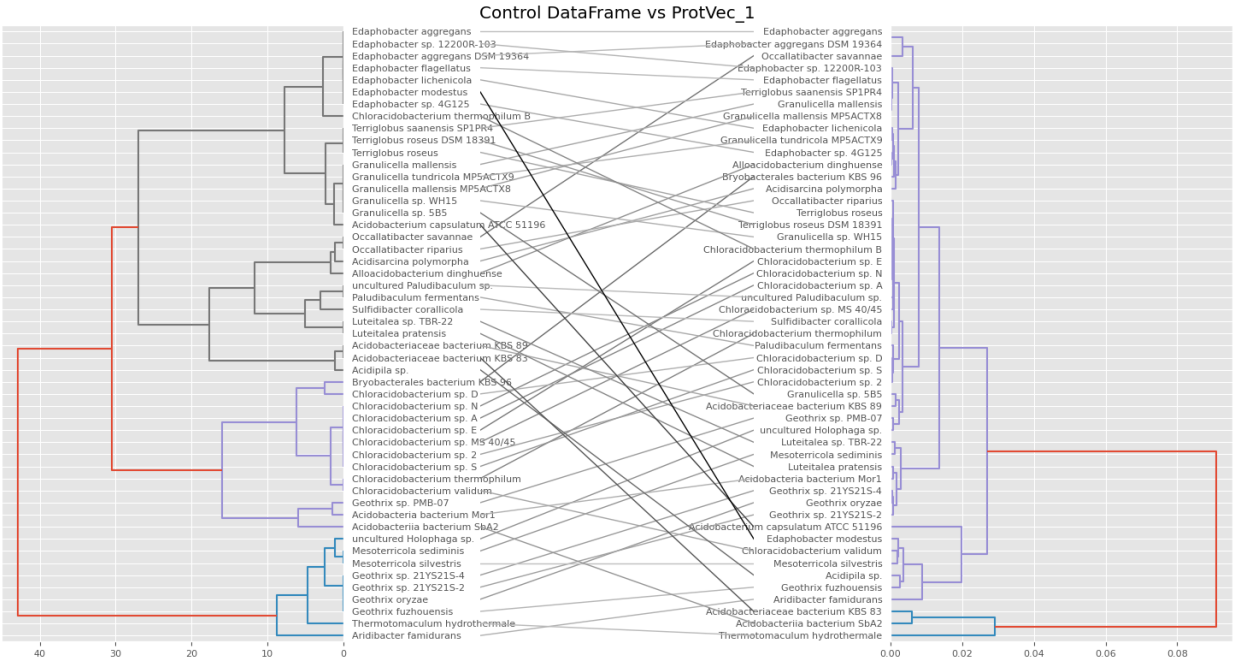


Figure 10

Tanglegram of control dendrogram vs dendrogram of ProtVec 1.

The dendrogram of the vectors of ProtVec 1 can be seen in Figure 4. Furthermore, the dendrogram generated by the vectors of ProtVec 2 can be seen in Figure 5. It can be seen that even though dendrograms generated by ProtVec have small differences between each, they follow the same structure. Up to three species clustered outside of the greater species. In ProtVec 1 and ProtVec 2, Acidobacteriia bacterium SbA2, Thermotomaculum hydrothermale, and Acidobacteriaceae bacterium KBS 83 are the ones that clustered distant from other species. Both Acidobacteriaceae bacterium KBS 83 and Acidobacteriia bacterium SbA2 are unclassified species (Acidobacteriaceae bacterium KBS 83 is unclassified at the species level, while Acidobacteriia bacterium SbA2 is unclassified at the family level) might indicate some relationship between organisms.

When it comes to Word2Vec methods Skip-Gram and Continous Bag of Words, the dendrogram generated by the vectors of the Skip-Gram model can be seen in Figure 6. As

can be seen, the dendrogram of skip-gram has different attributes than the dendrograms generated by ProtVec, such as it having three greater clusters instead of two. Also, clusters have more distance than the ProtVec method.

Dendrogram of the other Word2Vec method CBOW can be seen in Figure 7. Even though it shares similar structures with the dendrogram generated by skipgram vectors, the dendrogram generated by CBOW is formed differently than it.

Finally, the dendrogram of the control dataset can be seen in Figure 8. This dendrogram was generated by numeric values of Entrez data.

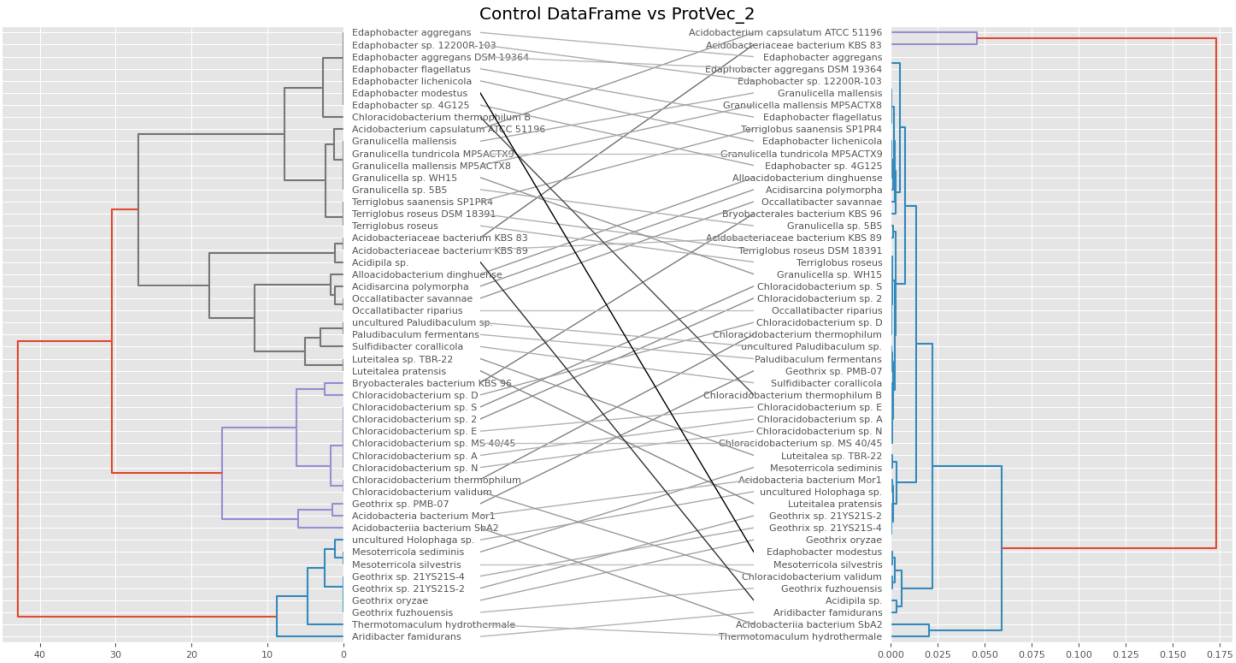


Figure 11

Tanglegram of control dendrogram vs dendrogram of ProtVec 2.

5.2 Tanglegrams

The tanglegram method is used to show the performance of the dendrograms. Each dendrogram was compared with the control dendrogram to see how it differs from it.

Firstly, the tanglegram generated by the control dendrogram and the dendrogram of ProtVec 0 can be seen in Figure 9. It can be seen that even though this tanglegram has low

matching between the control dendrogram and the ProtVec 0 dendrogram, it keeps some clusters of organisms, such as species of Chloracidobacterium sp. This issue continues with the tanglegram of the control dendrogram versus both the dendrogram of ProtVec 1 and ProtVec 2. The tanglegram of the control dendrogram and the dendrogram of ProtVec 1 can be seen in Figure 10. Also, the tanglegram of the control dendrogram and the dendrogram of ProtVec 2 can be seen in 11.

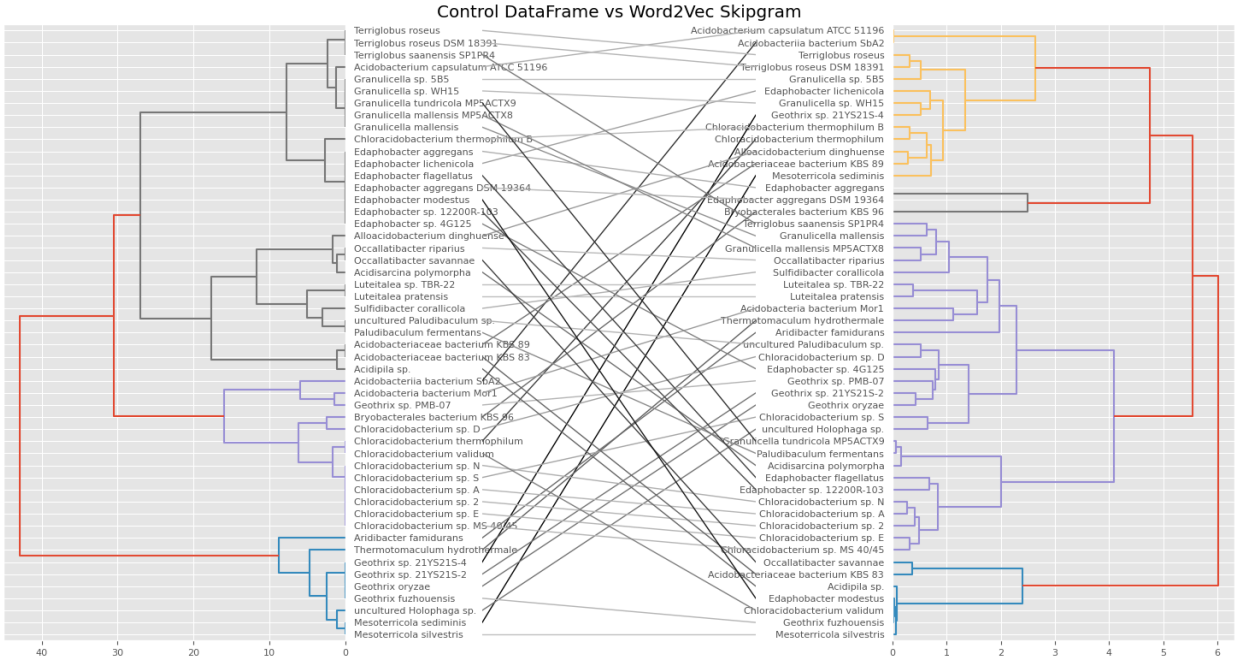


Figure 12

Tanglegram of control dendrogram vs dendrogram of Skip-Gram model.

When Tanglegrams are constructed using dendrograms of Word2Vec models checked, it can be seen that their performance is worse than the performance of ProtVec models. The tanglegram of the control dataframe and the dendrogram of the vectors of the skip-gram model can be seen in Figure 12. Even though it has some clusters right such as species of Chloracidobacterium sp., the dendrogram of the skip-gram model has less consistency with the control dendrogram, and distances of unmatching leaves are greater. This is also the case for the tanglegram of the control dendrogram and the dendrogram of the vectors of the CBOW model, which can be seen in Figure 13.

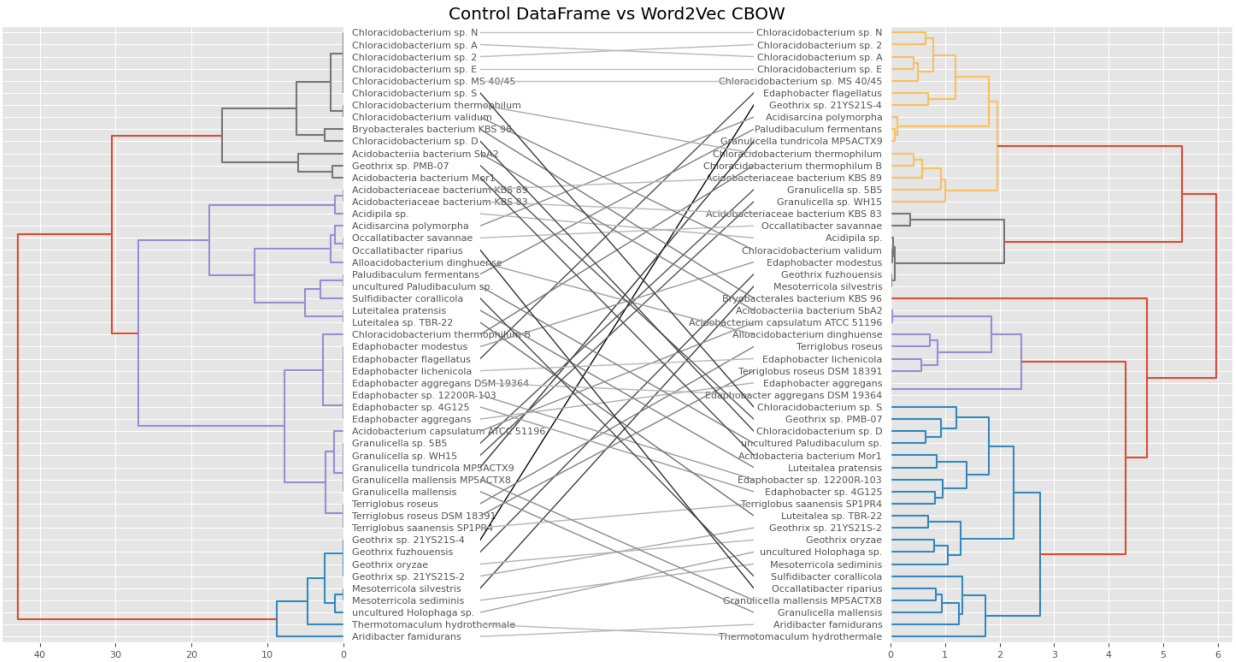


Figure 13

Tanglegram of control dendrogram vs dendrogram of Continous Bag of Words model.

When checked, the CBOW model seems to work worse with sequence data compared to both ProtVec models and the skip-gram model, which can be reasoned due to [19] and [15] have chosen the skip-gram architecture of Word2Vec to base their models instead of CBOw architecture.

6 Conclusion

All in all, ProtVec and Word2Vec are used to create a taxonomy of acidobacteriota by the genome sequences of the organisms. Various papers were studied and discussed as a literature review and an understanding of the subject, and methods used are given. To take the study further, a greater and more distinct dataset could be used to generate vectors and more distinct dendrograms.

References

[1] Joe H. Ward Jr. "Hierarchical Grouping to Optimize an Objective Function." In: *Journal of the American Statistical Association* 58.301 (1963), pp. 236–244. DOI: 10.1080/01621459.1963.10500845. eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1963.10500845. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500845.

[2] Jr. Cecil C. Bridges. "Hierarchical Cluster Analysis." In: *Psychological Reports* 18.3 (1966), pp. 851–854. DOI: 10.2466/pr0.1966.18.3.851. eprint: https://doi.org/10.2466/pr0.1966.18.3.851. URL: https://doi.org/10.2466/pr0.1966.18.3.851.

[3] Philip Gage. "A New Algorithm for Data Compression." In: *C Users J.* 12.2 (Feb. 1994), pp. 23–38. ISSN: 0898-9788.

[4] Corinna Cortes and Vladimir Vapnik. "Support-vector networks." In: *Machine learning* 20.3 (1995), pp. 273–297.

[5] J. D. Hunter. "Matplotlib: A 2D graphics environment." In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[6] Peter J A Cock et al. "Biopython: freely available Python tools for computational molecular biology and bioinformatics." In: *Bioinformatics* 25.11 (June 2009), pp. 1422–3. DOI: 10.1093/bioinformatics/btp163.

[7] Heng Li and Richard Durbin. "Fast and accurate short read alignment with Burrows–Wheeler transform." In: *bioinformatics* 25.14 (2009), pp. 1754–1760.

[8] Antonio Mucherino, Petraq J. Papajorgji, and Panos M. Pardalos. "k-Nearest Neighbor Classification." In: *Data Mining in Agriculture*. New York, NY: Springer New York, 2009, pp. 83–106. ISBN: 978-0-387-88615-2. DOI: 10.1007/978-0-387-88615-2_4. URL: https://doi.org/10.1007/978-0-387-88615-2_4.

[9] Payam Refaeilzadeh, Lei Tang, and Huan Liu. "Cross-Validation." In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M. TAMER ÖZSU. Boston, MA: Springer US, 2009, pp. 532–538. ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_565. URL: https://doi.org/10.1007/978-0-387-39940-9_565.

[10] "TF–IDF." In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 986–987. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_832. URL: https://doi.org/10.1007/978-0-387-30164-8_832.

[11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[12] Radim Rehurek and Petr Sojka. "Gensim–python framework for vector space modelling." In: *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic* 3.2 (2011).

[13] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[14] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation." In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162.

[15] Ehsaneddin Asgari and Mohammad R K Mofrad. "Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics." In: *PLoS One* 10.11 (2015), e0141287. DOI: 10.1371/journal.pone.0141287.

[16] Tal Galili. "dendextend: an R package for visualizing, adjusting and comparing trees of hierarchical clustering." eng. In: *Bioinformatics* 31.22 (Nov. 2015), pp. 3718–3720. ISSN: 1367-4811 (Electronic); 1367-4803 (Print); 1367-4803 (Linking). DOI: 10.1093/bioinformatics/btv428.

[17] Kate R Rosenbloom et al. "The UCSC Genome Browser database: 2015 update." eng. In: *Nucleic Acids Res* 43.Database issue (Jan. 2015), pp. D670–81. ISSN: 1362-4962 (Electronic); 0305-1048 (Print); 0305-1048 (Linking). DOI: 10.1093/nar/gku1177.

[18] Kévin Vervier et al. "Large-scale machine learning for metagenomics sequence classification." In: *Bioinformatics* 32.7 (Nov. 2015), pp. 1023–1032. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btv683. eprint: https://academic.oup.com/bioinformatics/article-pdf/32/7/1023/49018729/bioinformatics_32_7_1023.pdf. URL: https://doi.org/10.1093/bioinformatics/btv683.

[19] Patrick Ng. *dna2vec: Consistent vector representations of variable-length k-mers*. 2017. arXiv: 1701.06279 [q-bio.QM].

[20] Romain Menegaux and Jean-Philippe Vert. "Continuous Embedding of DNA reads and application to metagenomics." In: *bioRxiv preprint 335943* (2018).

[21] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[22] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[23] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[24] Aysun GÜRAN and Doğancan KINIK. "TF-IDF ve Doc2Vec Tabanlı Türkçe Metin Sınıflandırma Sisteminin Başarım Değerinin Ardışık Kelime Grubu Tespiti ile Arttırılması." In: *European Journal of Science and Technology* (Jan. 2021). DOI: 10.31590/ejosat.774144.

[25] Yanrong Ji et al. "DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome." In: *Bioinformatics* 37.15 (Feb. 2021), pp. 2112–2120. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btab083. eprint: https://academic.oup.com/bioinformatics/article-pdf/37/15/2112/50578892/btab083.pdf. URL: https://doi.org/10.1093/bioinformatics/btab083.

[26] Jaina Mistry et al. "Pfam: The protein families database in 2021." eng. In: *Nucleic Acids Res* 49.D1 (Jan. 2021), pp. D412–D419. ISSN: 1362-4962 (Electronic); 0305-1048 (Print); 0305-1048 (Linking). DOI: 10.1093/nar/gkaa913.

[27] Ofir Press, Noah A. Smith, and Mike Lewis. "Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation." In: *CoRR* abs/2108.12409 (2021). arXiv: 2108.12409. URL: https://arxiv.org/abs/2108.12409.

[28] Stevin Wilson. *Biovec*. July 2021. URL: https://github.com/kyu999/biovec/tree/master.

[29] Marwah A. Helaly, Sherine Rady, and Mostafa M. Aref. "BERT contextual embeddings for taxonomic classification of bacterial DNA sequences." In: *Expert Systems with Applications* 208 (2022), p. 117972. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2022.117972. URL: https://www.sciencedirect.com/science/article/pii/S0957417422012039.

[30] Hugo Dalla-Torre et al. "The Nucleotide Transformer: Building and Evaluating Robust Foundation Models for Human Genomics." In: *bioRxiv* (2023). DOI:

10.1101/2023.01.11.523679. eprint: https://www.biorxiv.org/content/early/2023/01/15/2023.01.11.523679.full.pdf. URL: https://www.biorxiv.org/content/early/2023/01/15/2023.01.11.523679.

[31] Zhihan Zhou et al. *DNABERT-2: Efficient Foundation Model and Benchmark For Multi-Species Genome*. 2023. arXiv: 2306.15006 [q-bio.GN].

Appendix

Code

Application

January 13, 2024

0.1 Downloading The Dataset

In this section, dataset will be downloaded and opened as a pandas dataframe

[ ]:
```
from Bio import Entrez
import pandas as pd
from io import StringIO
from Bio import SeqIO
import numpy as np
import biovec
import gensim
from sklearn.metrics.pairwise import cosine_similarity
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt
import tanglegram as tg
```

[ ]:
```
def get_taxid_by_name(organism_name): # gets taxid from organism name using␣
 ↪Entrez
    Entrez.email = "n@mail.com"

    search_handle = Entrez.esearch(db="taxonomy", term=organism_name)
    search_results = Entrez.read(search_handle)

    if len(search_results["IdList"]) == 0:
        print(f"No results found for {organism_name}")
        return 0

    # Get the taxid from the search results
    taxid = search_results["IdList"][0]

    return taxid
```

[ ]:
```
Acidobacteriota = "Acidobateriota.tsv"
base_df = pd.read_csv(Acidobacteriota, sep="\t")
```

[ ]:
```
df = base_df.dropna(subset=["Annotation Name"])
df = df[df["Annotation Name"].str.contains('NCBI Prokaryotic Genome Annotation␣
 ↪Pipeline')] # only organisms from NCBI pipeline
```

```
df = df[~df["Organism Name"].str.contains("Candidatus")] # to remove candidate␣
↪organisms
df = df[~df["Organism Name"].str.contains("Candidate")]
df = df[~df["Organism Name"].str.contains("uncultered")]
df = df.drop_duplicates(subset=["Organism Name"]) # to remove duplicates
```

[ ]:
```
df["taxid"] = df["Organism Name"].apply(get_taxid_by_name)
```

[ ]:
```
def get_nucleotide_sequence_by_taxid(taxid, num_records=1): # gets nucleotide␣
↪sequence from taxid using Entrez
    try:
        search_handle = Entrez.esearch(db="nuccore",␣
↪term=f"txid{taxid}[Organism:exp]", retmax=num_records)
        search_results = Entrez.read(search_handle)
        nuccore_uids = search_results["IdList"]
        fetch_handle = Entrez.efetch(db="nuccore", id=nuccore_uids,␣
↪rettype="fasta", retmode="text")
        nucleotide_records = fetch_handle.read()

        return str(SeqIO.read(StringIO(nucleotide_records), "fasta").seq)
    except:
        return 0
```

[ ]:
```
df["sequence"] = df["taxid"].apply(get_nucleotide_sequence_by_taxid)
df.drop(df[df["sequence"] == 0].index, inplace=True) # to remove organisms with␣
↪no sequence
```

[ ]:
```
df.reset_index(drop=True, inplace=True)
```

1 ProtVec

[ ]:
```
def df_to_fasta(df, fasta_file): # to create a fatsa file to train protvec
    with open(fasta_file, "w") as f:
        for index, row in df.iterrows():
            f.write(f">{row['Assembly Accession']}\n{row['sequence']}\n")
```

[ ]:
```
df_to_fasta(df, "Acidobacteriota.fasta")
```

[ ]:
```
pv = biovec.models.ProtVec('Acidobacteriota.fasta')
```

```
Generate Corpus file from fasta file…

corpus generation progress: 100%|      | 51/51 [00:14<00:00,  3.58it/s]
```

[ ]:
```
def get_protvec_vectors(seq, pv,vector):
    return pv.to_vecs(seq)[vector].flatten()
```

[ ]:
```
df["protvec_0"] = df["sequence"].apply(get_protvec_vectors, pv=pv, vector=0)
df["protvec_1"] = df["sequence"].apply(get_protvec_vectors, pv=pv, vector=1)
df["protvec_2"] = df["sequence"].apply(get_protvec_vectors, pv=pv, vector=2)
```

2 Word2Vec

[ ]:
```
def split_string_into_thirds(input_string):
    return [input_string[i:i+3] for i in range(0, len(input_string), 3)]

def get_word2vec_skipgram(seq):
    try:
        return gensim.models.Word2Vec(split_string_into_thirds(seq),␣
 ↪min_count=3, size=100, sg=1).wv.vectors.flatten()
    except:
        return np.zeros(300)
def get_word2vec_cbow(seq):
    try:
        return gensim.models.Word2Vec(split_string_into_thirds(seq),␣
 ↪min_count=3, size=100, sg=0).wv.vectors.flatten()
    except:
        return np.zeros(300)
```

[ ]:
```
df["word2vec_skipgram"] = df["sequence"].apply(get_word2vec_skipgram)
df["word2vec_cbow"] = df["sequence"].apply(get_word2vec_cbow)
```

3 Control DataFrame

[ ]:
```
df.set_index("Organism Name", inplace=True)
```

[ ]:
```
def get_lineage(taxid):
    try:
        handle = Entrez.efetch(db="Taxonomy", id=taxid, retmode="xml")
        records = Entrez.read(handle)
        lineage = records[0]["Lineage"]
        return lineage
    except:
        return 0
```

[ ]:
```
control_df = df.copy()
control_df["lineage"] = control_df["taxid"].apply(get_lineage) # control␣
 ↪dataframe
control_df['Domain'] = control_df.lineage.str.split(';',expand=True)[0]
control_df['Phylum'] = control_df.lineage.str.split(';',expand=True)[1]
control_df['Class'] = control_df.lineage.str.split(';',expand=True)[2]
control_df['Order'] = control_df.lineage.str.split(';',expand=True)[3]
control_df['Family'] = control_df.lineage.str.split(';',expand=True)[4]
```

```
control_df['Genus'] = control_df.lineage.str.split(';',expand=True)[5]
control_df['Species'] = control_df.lineage.str.split(';',expand=True)[6]
control_df.drop(['lineage',"Domain", "Phylum","Class","Assembly␣
 ↪Accession","Assembly Name","Annotation Name","Annotation Count Gene␣
 ↪Total","Annotation Count Gene Protein-coding","taxid","sequence",
                "protvec_0","protvec_1","protvec_2","word2vec_skipgram",␣
 ↪"word2vec_cbow"], axis=1, inplace=True)
control_df.fillna(value=np.nan, inplace=True)
```

[ ]:
```
def map_to_numeric(series):
    unique_values = series.unique()
    mapping = {value: i for i, value in enumerate(unique_values)}
    return series.map(mapping)

control_df_numeric = control_df.apply(map_to_numeric) # transforming control_df␣
 ↪to a numeric dataframe, inorder to use it in linkage
```

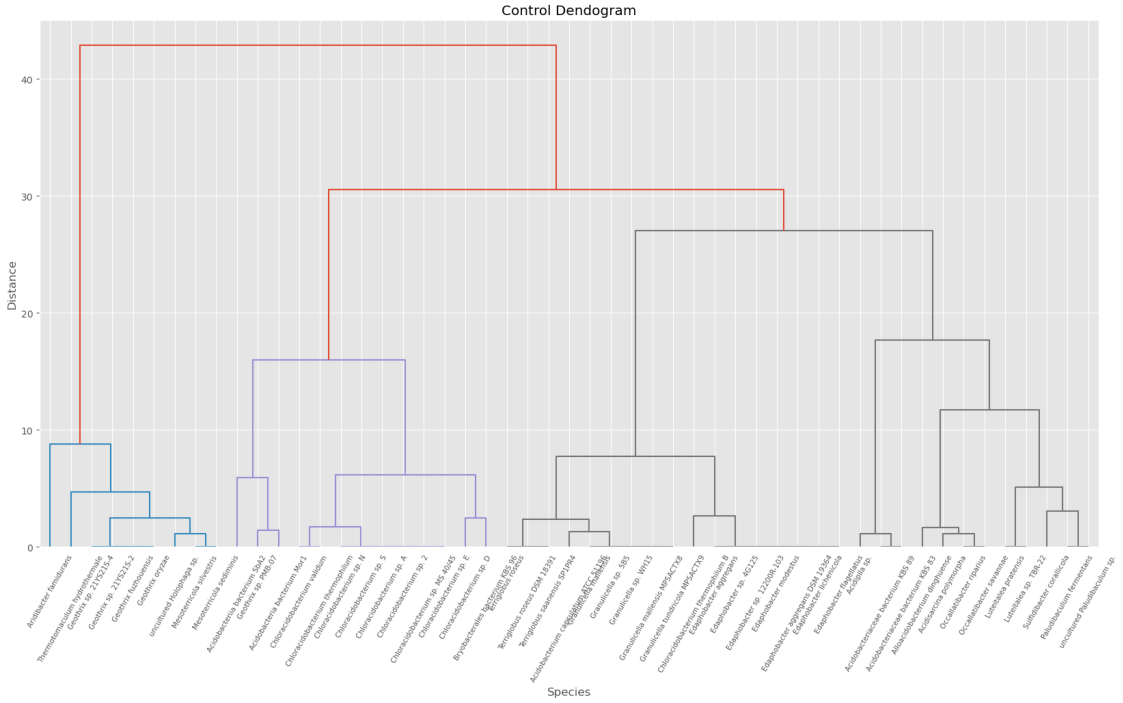
4 Dendrograms

[ ]:
```
def vector_dataframe(dataframe, vectors): #dataframe of vectors
    return pd.DataFrame([x for x in dataframe[vectors]], index= dataframe.
 ↪index).dropna(axis=1)

def similarity(vector_df): # cosine similarity matrix
    return cosine_similarity(vector_df)

def get_linkage(dataframe, vectors): #linkage matrix
    return linkage(similarity(vector_dataframe(dataframe,vectors)), 'ward')

def print_dendrogram(dataframe,vectors,title= "No Title"): # dendogram
    plt.figure(figsize=(20,10))
    plt.title(title)
    Z = get_linkage(dataframe, vectors)
    vector_df = vector_dataframe(dataframe, vectors)
    dend =dendrogram(Z, leaf_rotation=180, leaf_font_size=2,labels=vector_df.
 ↪index)
    plt.xticks(rotation = 60, fontsize = 7.5)
    plt.ylabel('Distance')
    plt.xlabel('Species')
    plt.figure(figsize=(20,10))

    plt.show()
    return dend
```

[ ]:
```
linkage_control = linkage(control_df_numeric, method = 'ward')
linkage_0 = get_linkage(df, "protvec_0")
```

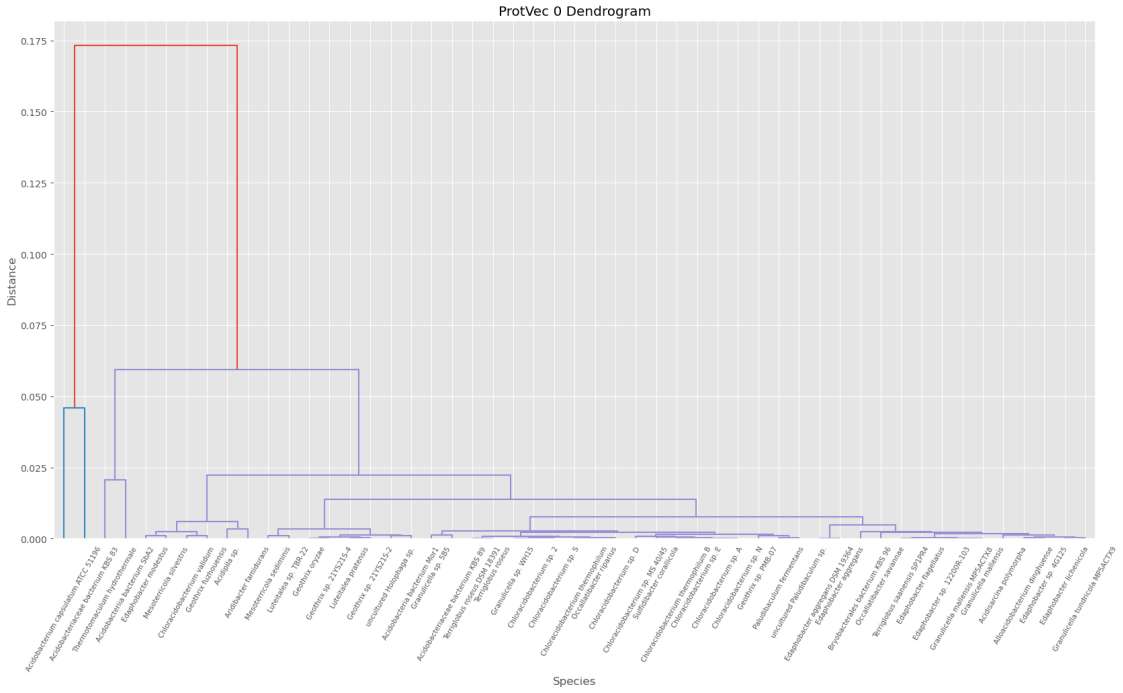
```
linkage_1 = get_linkage(df, "protvec_1")
linkage_2 = get_linkage(df, "protvec_2")
linkage_skipgram = get_linkage(df, "word2vec_skipgram")
linkage_cbow = get_linkage(df, "word2vec_cbow")
```

[ ]:
```
plt.figure(figsize=(20,10))
dend_control = dendrogram(linkage_control, leaf_rotation=180,␣
 ↪leaf_font_size=2,labels=control_df.index)
plt.xticks(rotation = 60, fontsize = 7.5)
plt.title("Control Dendogram")
plt.ylabel('Distance')
plt.xlabel('Species')
plt.figure(figsize=(20,10))
plt.show()
```



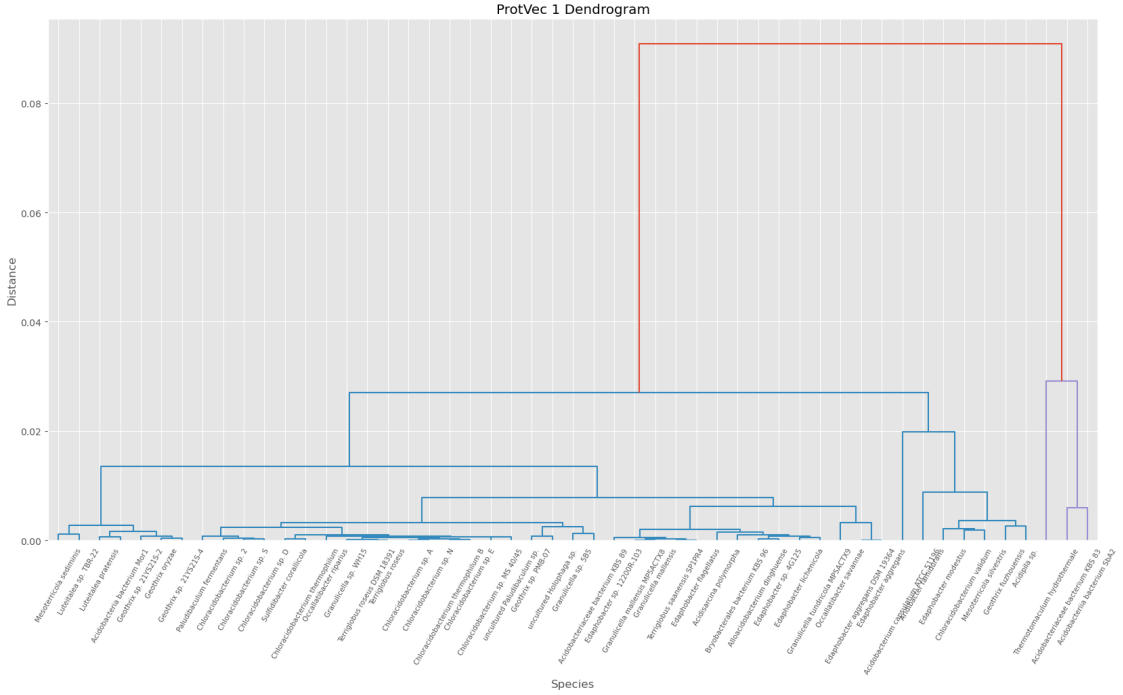
<Figure size 2000x1000 with 0 Axes>

[ ]:
```
dend_0 = print_dendrogram(df, "protvec_0", title="ProtVec 0 Dendrogram")
```



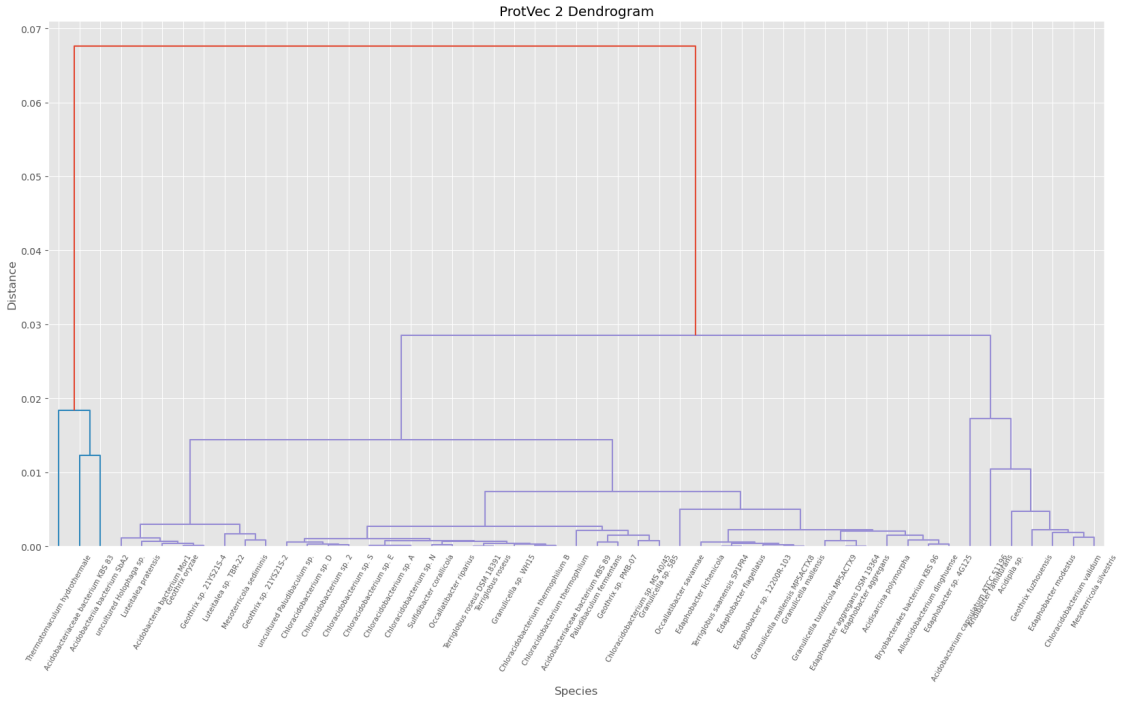
<Figure size 2000x1000 with 0 Axes>

[ ]:
```
dend_1= print_dendrogram(df, "protvec_1", title="ProtVec 1 Dendrogram")
```



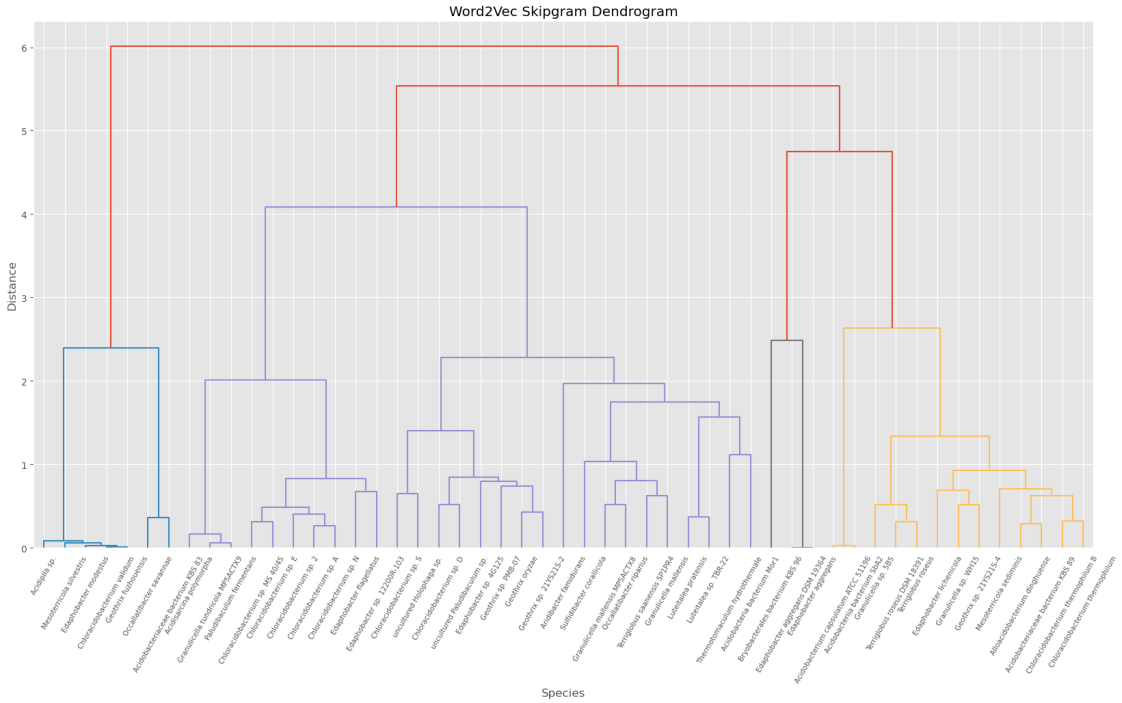
<Figure size 2000x1000 with 0 Axes>

[ ]:
```
dend_2 = print_dendrogram(df, "protvec_2", title="ProtVec 2 Dendrogram")
```



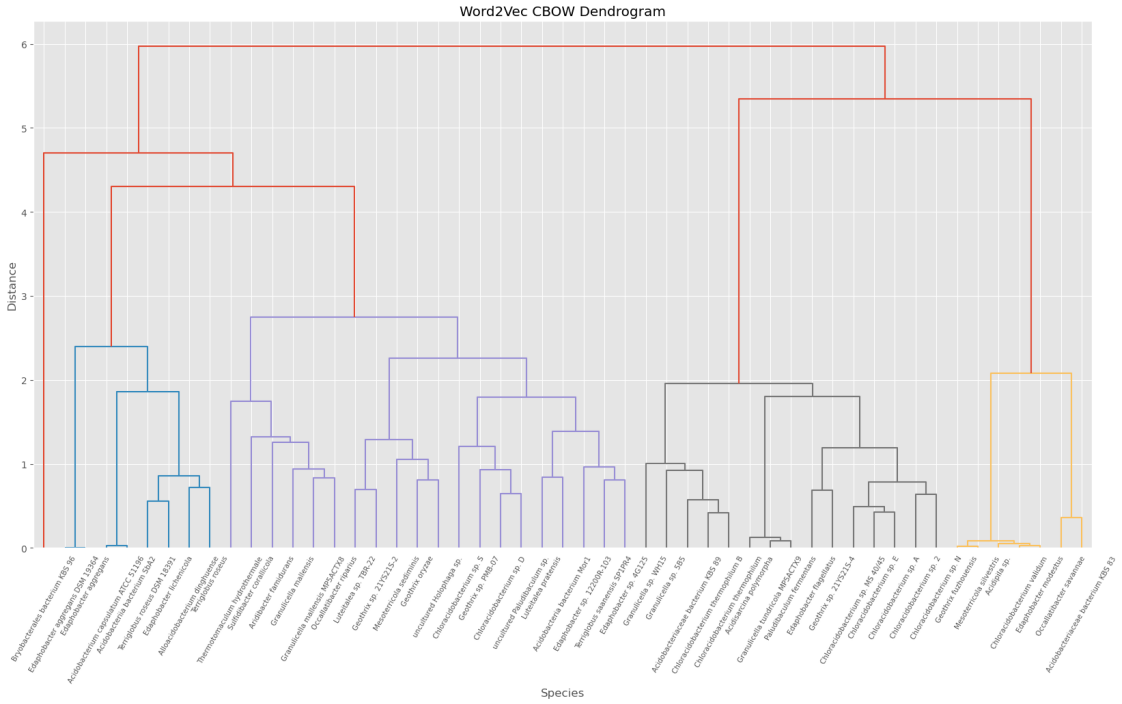
<Figure size 2000x1000 with 0 Axes>

[ ]:
```
dend_skipgram = print_dendrogram(df, "word2vec_skipgram", title="Word2Vec␣
↪Skipgram Dendrogram")
```



<Figure size 2000x1000 with 0 Axes>

[ ]:
```
dend_cbow = print_dendrogram(df, "word2vec_cbow", title="Word2Vec CBOW␣
↪Dendrogram")
```



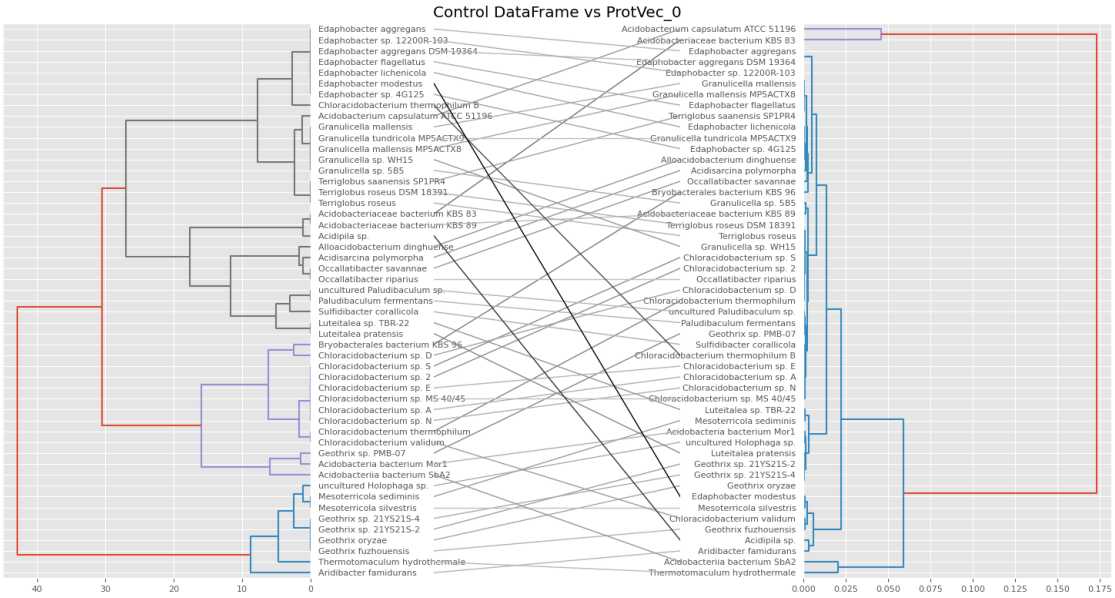
```
<Figure size 2000x1000 with 0 Axes>
```

5 Tanglegrams

[ ]:
```
def plot_tanglegram(l_1,l_2,index_1,index_2, title):
    tg.plot(l_1,l_2,index_1, index_2,figsize=(16,9),sort="step2side")
    plt.title(title)
    plt.show()
```

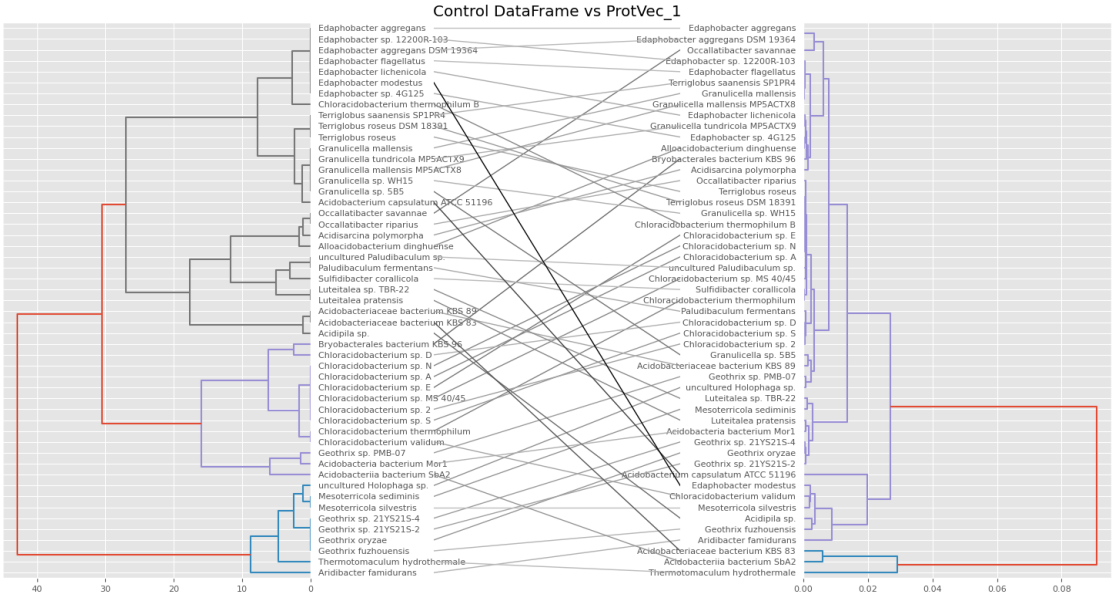
[ ]:
```
plot_tanglegram(linkage_control,linkage_0,control_df.index,df.index,"Control␣
↪DataFrame vs ProtVec_0")
```

```
INFO  : Finished optimising at entanglement 0.16 (tanglegram.tangle)
INFO  : Done. Use matplotlib.pyplot.show() to show plot. (tanglegram.tangle)
```



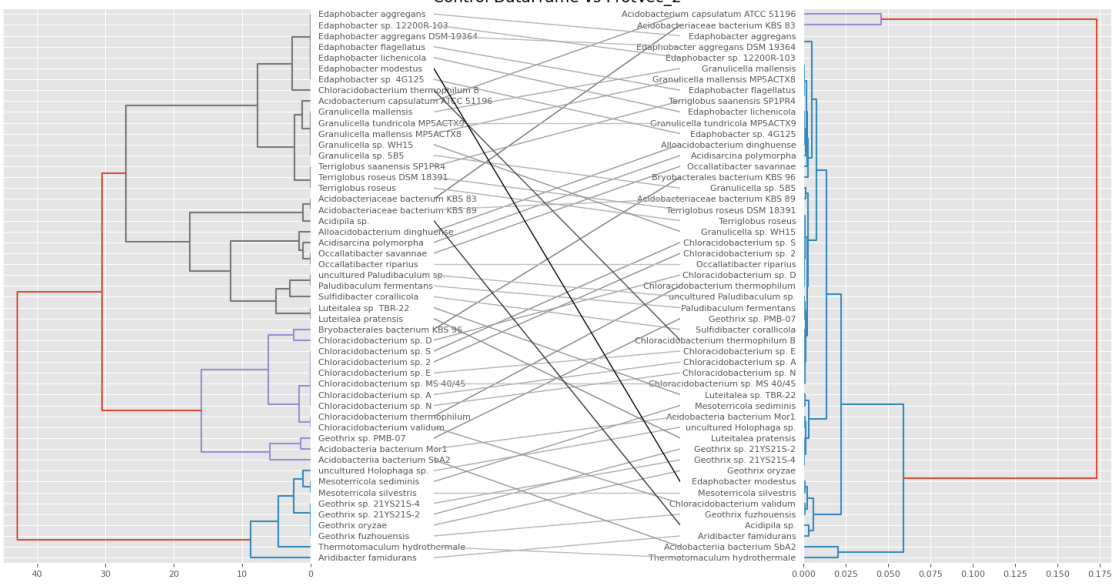
[ ]:
```
plot_tanglegram(linkage_control,linkage_1, control_df.index,df.index,"Control␣
↪DataFrame vs ProtVec_1")
```

```
INFO  : Finished optimising at entanglement 0.19 (tanglegram.tangle)
INFO  : Done. Use matplotlib.pyplot.show() to show plot. (tanglegram.tangle)
```



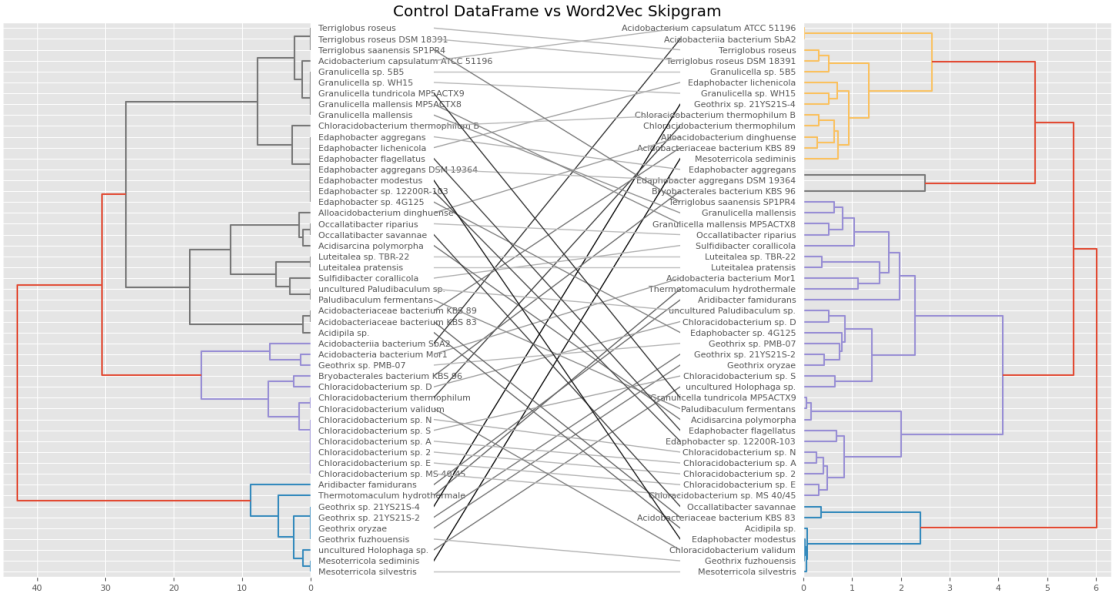
[ ]:
```
plot_tanglegram(linkage_control,linkage_0,control_df.index,df.index,"Control␣
↪DataFrame vs ProtVec_2")
```

```
INFO  : Finished optimising at entanglement 0.16 (tanglegram.tangle)
INFO  : Done. Use matplotlib.pyplot.show() to show plot. (tanglegram.tangle)
```



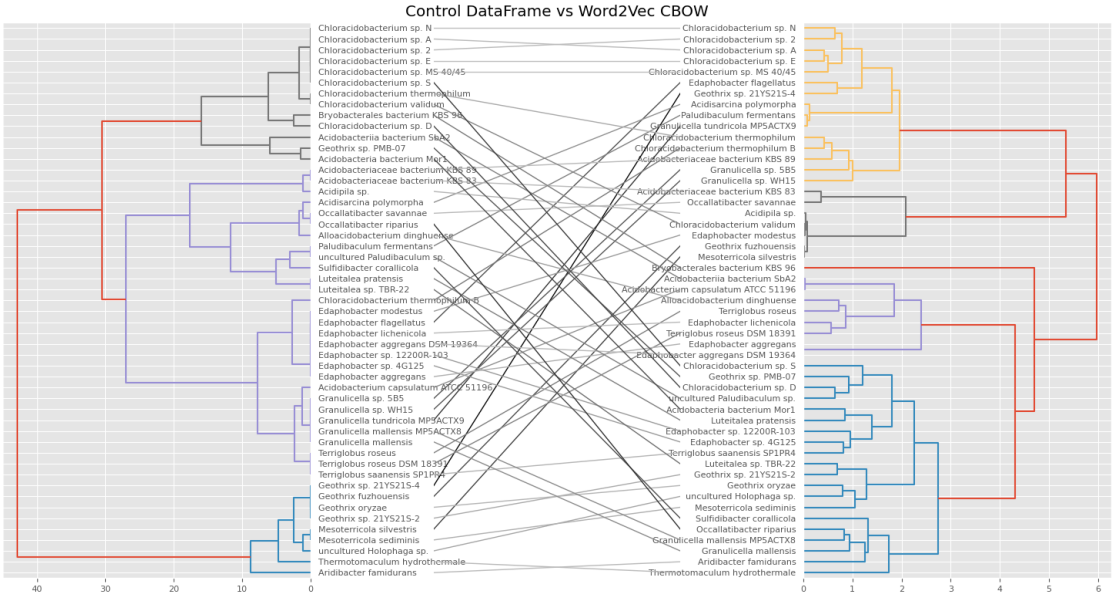
[ ]:
```
plot_tanglegram(linkage_control,linkage_skipgram,control_df.index,df.
↪index,"Control DataFrame vs Word2Vec Skipgram")
```

```
INFO  : Finished optimising at entanglement 0.33 (tanglegram.tangle)
INFO  : Done. Use matplotlib.pyplot.show() to show plot. (tanglegram.tangle)
```



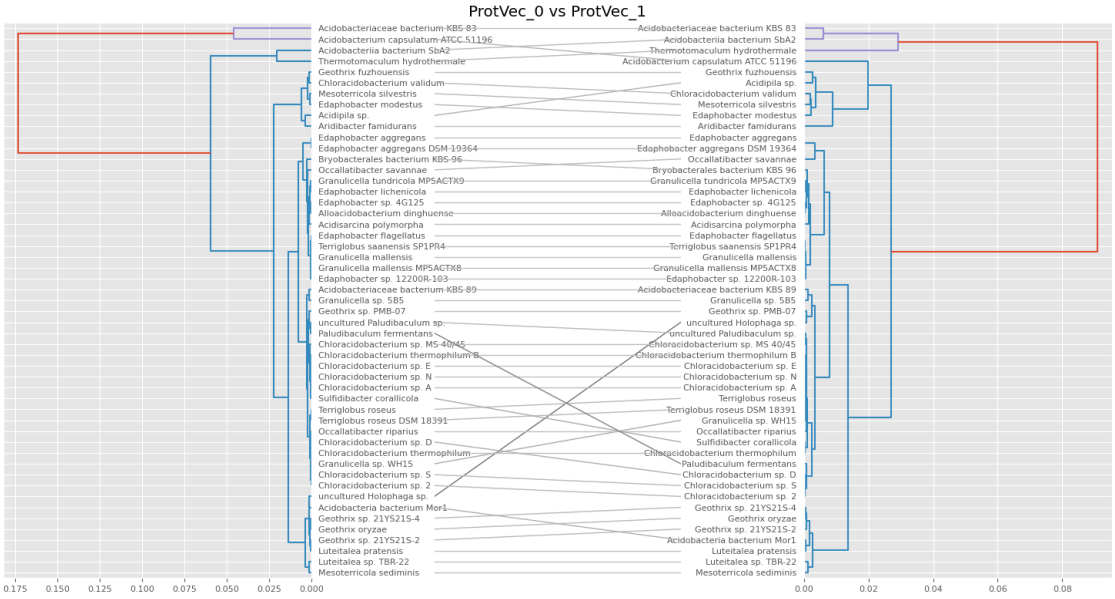
[ ]:
```
plot_tanglegram(linkage_control,linkage_cbow,control_df.index,df.index,"Control␣
↪DataFrame vs Word2Vec CBOW")
```

```
INFO  : Finished optimising at entanglement 0.31 (tanglegram.tangle)
INFO  : Done. Use matplotlib.pyplot.show() to show plot. (tanglegram.tangle)
```



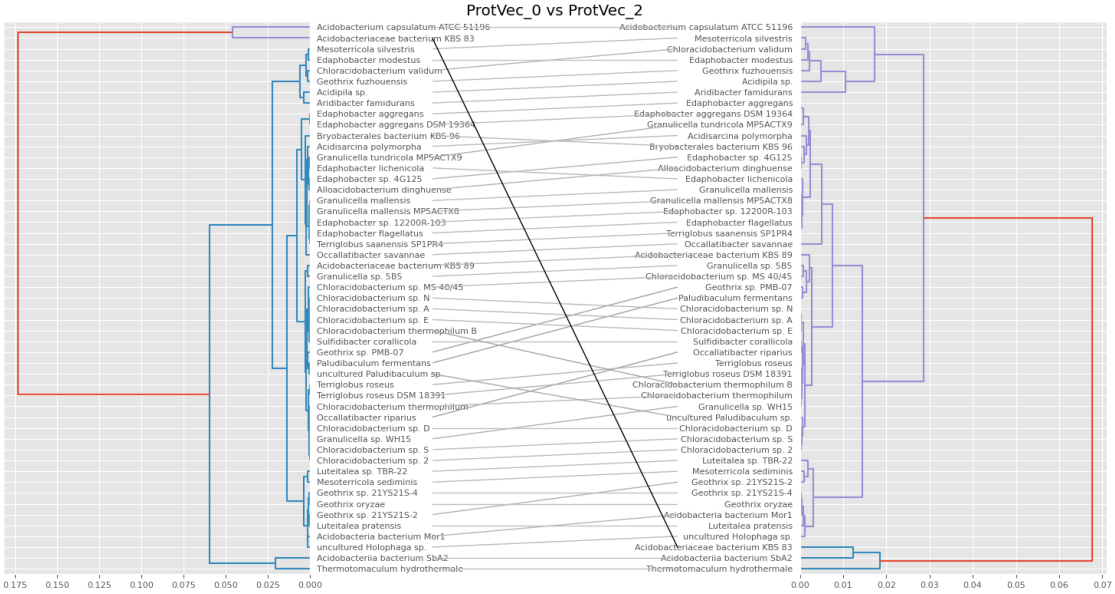
[ ]:
```
plot_tanglegram(linkage_0,linkage_1,df.index,df.index,"ProtVec_0 vs ProtVec_1")
```

```
INFO  : Finished optimising at entanglement 0.02 (tanglegram.tangle)
INFO  : Done. Use matplotlib.pyplot.show() to show plot. (tanglegram.tangle)
```



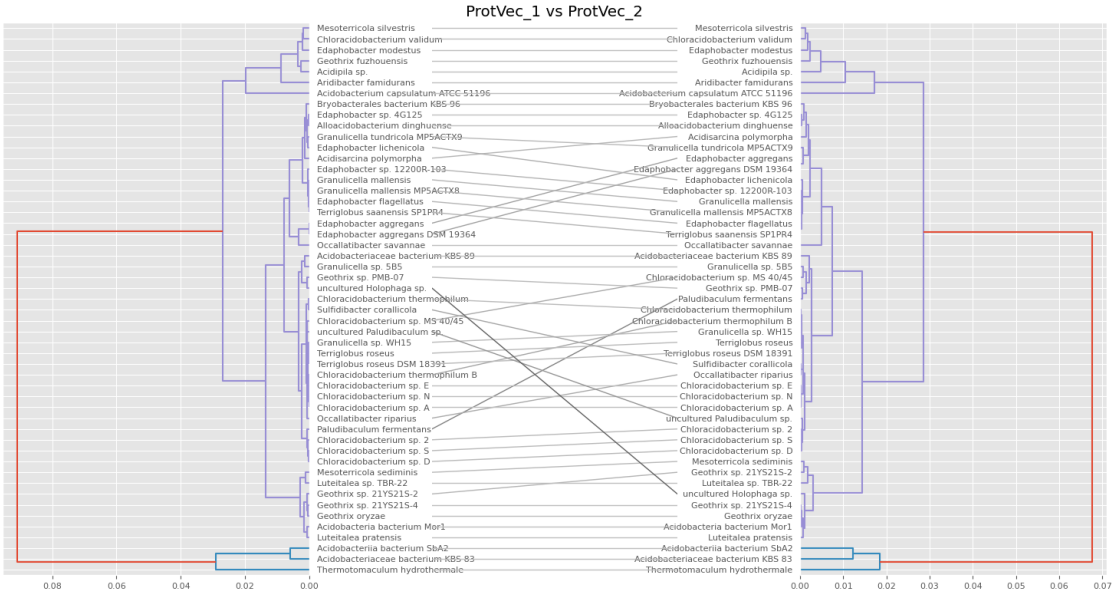
[ ]:
```
plot_tanglegram(linkage_0,linkage_2,df.index,df.index,"ProtVec_0 vs ProtVec_2")
```

```
INFO  : Finished optimising at entanglement 0.06 (tanglegram.tangle)
INFO  : Done. Use matplotlib.pyplot.show() to show plot. (tanglegram.tangle)
```



[ ]:
```
plot_tanglegram(linkage_1,linkage_2,df.index,df.index,"ProtVec_1 vs ProtVec_2")
```

```
INFO  : Finished optimising at entanglement 0.03 (tanglegram.tangle)
INFO  : Done. Use matplotlib.pyplot.show() to show plot. (tanglegram.tangle)
```



[ ]:
```
plot_tanglegram(linkage_0,linkage_skipgram,df.index,df.index,"ProtVec_0 vs␣
↪Word2Vec Skipgram")
```

```
INFO  : Finished optimising at entanglement 0.28 (tanglegram.tangle)
INFO  : Done. Use matplotlib.pyplot.show() to show plot. (tanglegram.tangle)
```

[ ]:
```
plot_tanglegram(linkage_0,linkage_cbow,df.index,df.index,"ProtVec_0 vs Word2Vec␣
↪CBOW")
```

```
INFO  : Finished optimising at entanglement 0.19 (tanglegram.tangle)
INFO  : Done. Use matplotlib.pyplot.show() to show plot. (tanglegram.tangle)
```