**CSDS 435 Project 2: Overview**

Health Tweet Cluster Analysis

Kyle Roncagli

Dept. of Electrical Engineering and Computer Science

Case Western Reserve University

Cleveland, Ohio

[kvr25@case.edu](mailto:kvr25@case.edu)

Peter Cork

Dept. of Electrical Engineering and Computer Science

Case Western Reserve University

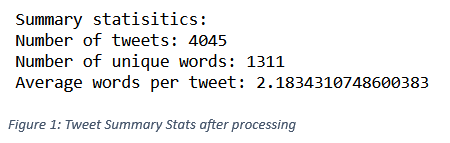
Cleveland, Ohio

phc23@case.edu

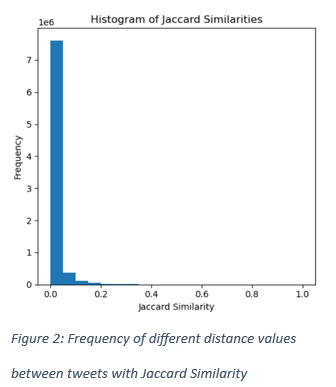
**Abstract**

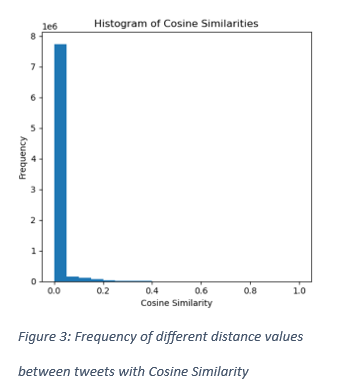
Cluster analysis, also known as unsupervised classification, is a process that attempts to group data based solely on attributes of the data itself. This is useful for data analysis when patterns within the data are not already recognized, in contrast with supervised classification, and is often a cursory step in the process of building classification models. While certain classification measures allow for data objects to belong to different clusters simultaneously, most generally the more similar that data objects are within an individual cluster and the less similar that those same data objects are with objects of another cluster, the better the distinctions that are drawn from the data will be. Our goal with this project was to explore the relationships between different health-focused tweets in the given dataset using differing distance measures, classification techniques, visualizations, etc. Using the Bag-O-Words approach, raw data was transformed into a feature matrix, composed of document vectors which represented each individual tweet. Representing the tweets in this way, with numerical data representing the frequency of terms within each tweet, then allowed us to use proximity measures to compare the tweets themselves and subsequently perform classification algorithms on the data.

**Distance Definitions**

****

Due to the sparse nature of our data, cosine and jaccard similarity were our proximity measures of choice. Cosine and jaccard proximity measures are a natural choice for document data due to the fact that they ignore shared non-existence of terms between documents. As shown in figure 1, after accounting for irrelevant words, punctuation, etc. we were left with 1311 unique words with each tweet only having three meaningful words on average. Based on this data, we can see how sparse our data really is and the number of non-existent terms we have per tweet (1308 on average). Cosine and Jaccard similarity give us the ability to ignore these terms in our proximity measures, allowing for us to compute distance or similarity based on the existing features, as typically it does not matter to us whether or not many of the same words do not exist within two particular tweets.

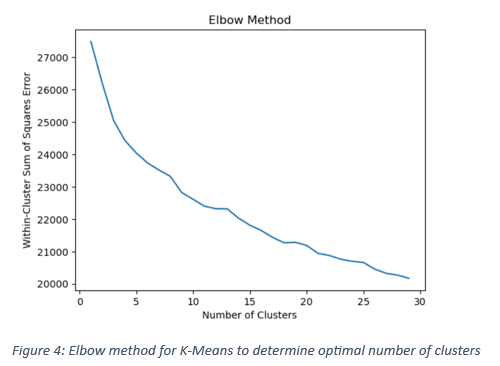


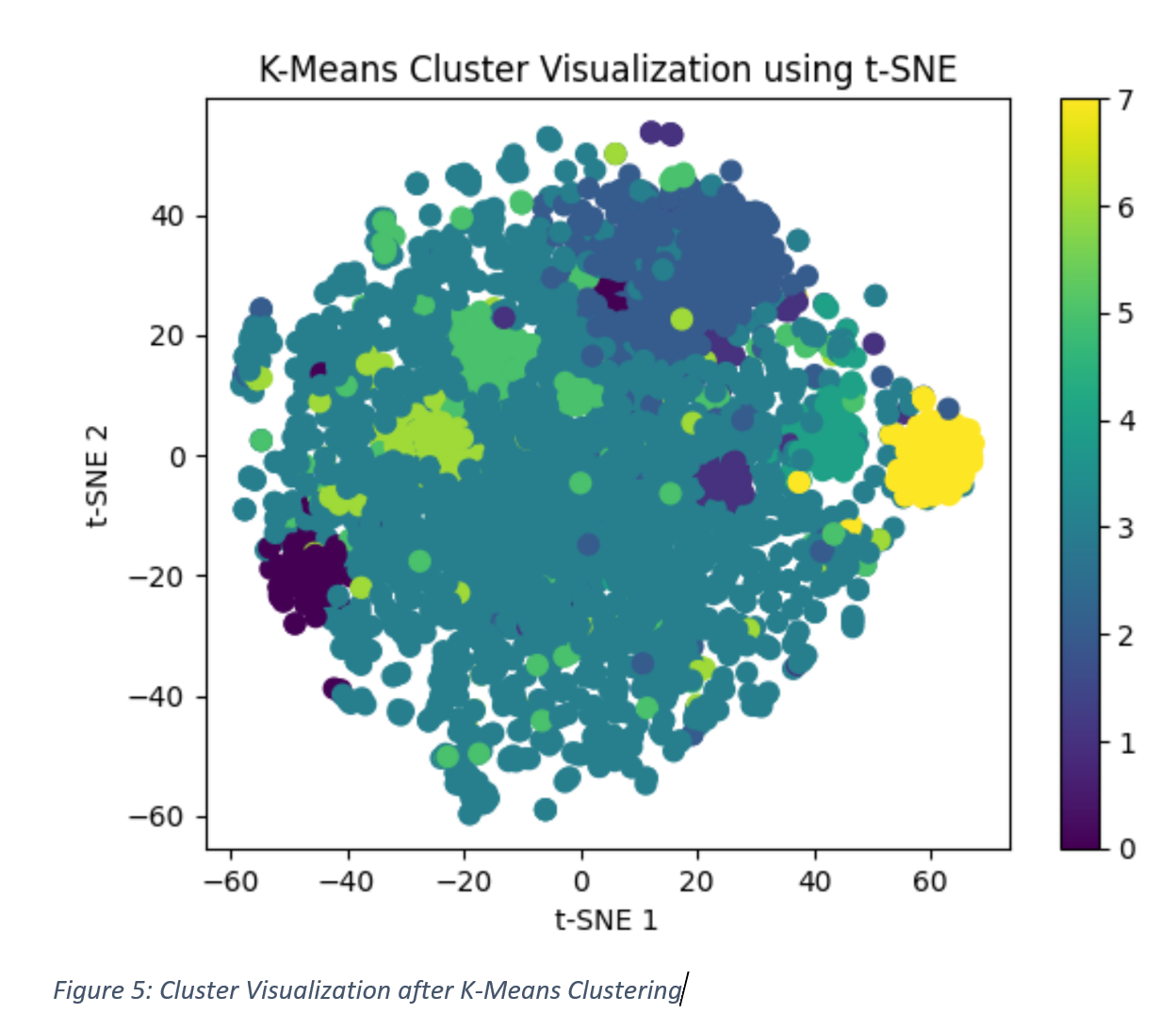


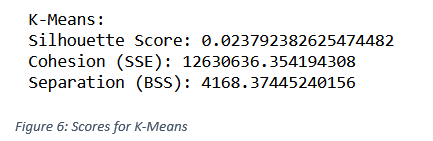
As you can see via the histograms of our distance measures, they seemed to have performed very similarly in calculating the distance measures. It was most common for tweets to be very dissimilar, with jaccard similarity accounting for a bit less of the similarity between tweets. This would make sense as jaccard similarity is not able to take into account non-binary values, it takes the existence of a word to be 1 and otherwise to be 0. Cosine similarity on the other hand, is able to account for multiple occurrences of the same word.

**K-Means Clustering**

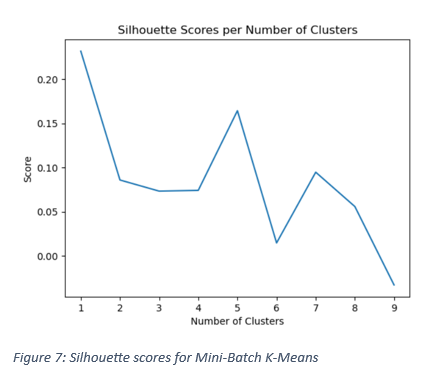
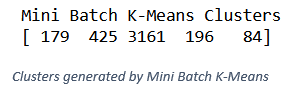
K-Means clustering was used in tandem with our Cosine similarity measures to perform cluster analysis on our dataset. To find the optimal number of clusters, we performed the elbow method, calculating the sse at each different n number of clusters.



Based on the elbow method graph above, we chose 8 as our optimal number of clusters for our k-means clustering. The idea is to choose a value for the number of clusters that corresponds to the “elbow” in the graph, i.e. where sse goes from reducing exponentially to more linearly. After this point, the gains in SSE are less and less so we would rather choose the smaller cluster size. Below you can see the visual result of our clustering after reducing dimensionality using t-SNE.Silhouette score was highest with k-means, indicating that clusters generated via this algorithm had better intra-cluster cohesion and inter-cluster separation. You can even make sense of this with the visualization above, while still not great, you can see more distinct clusters throughout the dataset.



**Mini Batch K-Means**

Mini Batch K-Means, an offshoot of K-Means, was also used on the dataset. Mini-batch k means is designed to perform better on larger datasets, so it seemed useful to try it on and compare our results. In choosing the number of clusters, we decided to use silhouette score. We ran the algorithm for a number of different clusters and calculated the results for each iteration, shown in the figure 7 below. Based on the results, we decided to go with 5 for our number of clusters. This result gives us a relatively high silhouette score while giving us a number of clusters that makes more sense for our data (choosing only one cluster would not be very informative). 

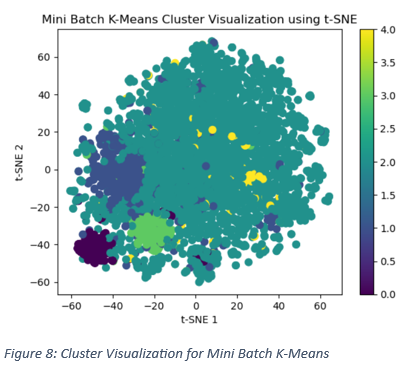
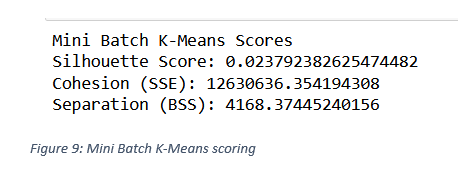


Figure 8 above shows the visualization for our Mini Batch K-Means clustering, again using t-SNE for our dimensionality reduction. The clusterings generated do not seem to be any better than that of K-Means, which was our hope due to the claim of Mini Batch K-Means working better on a larger dataset. Perhaps some more parameter tuning could have been implemented for a better result.



**Density Based Clustering (DBSCAN)**

Density-Based Spatial Clustering (DBSCAN) was also applied to the dataset using Jaccard similarity. DBSCAN is designed to find clusters of arbitrary shapes, making it an interesting alternative to try on and compare our results. It is also resistant to noise in the data set. In choosing the hyperparameters eps and min\_samples, we decided to use the silhouette score to try out different combinations of values. Eps is the maximum distance between two samples for them to be considered as in the same neighborhood, while min\_samples is the minimum number of samples in a neighborhood for a point to be considered as a core point. We ran the algorithm for various values of eps and min\_samples and calculated the results for each iteration, shown in Figure 10 below is the maximum value we achieved for the silhouette score. Based on the results, we decided to go with a combination of eps and min\_samples that produced more than ten clusters, as less than ten clusters didn't provide much insight into the data or make much logical sense. The clusters created seemed to be wildly arbitrary.

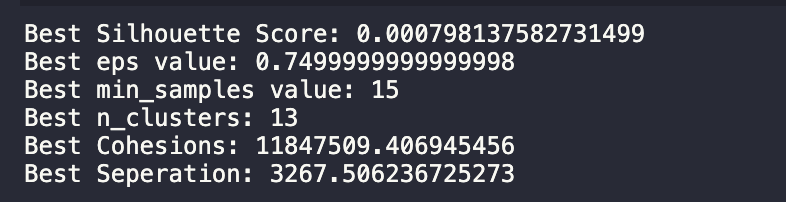


Figure 10 DBSCAN scoring for the best combination of min\_samples and eps

Figure 12 below shows the visualization for our DBSCAN clustering, again using t-SNE for dimensionality reduction. The clusterings generated do not seem to be significantly better than those of Mini Batch K-Means, which was our hope due to DBSCAN's ability to find clusters of arbitrary shapes. Perhaps some additional parameter tuning could have been implemented for a better result.



Figure 11 Clusters and Cluster Sizes for DBSCAN

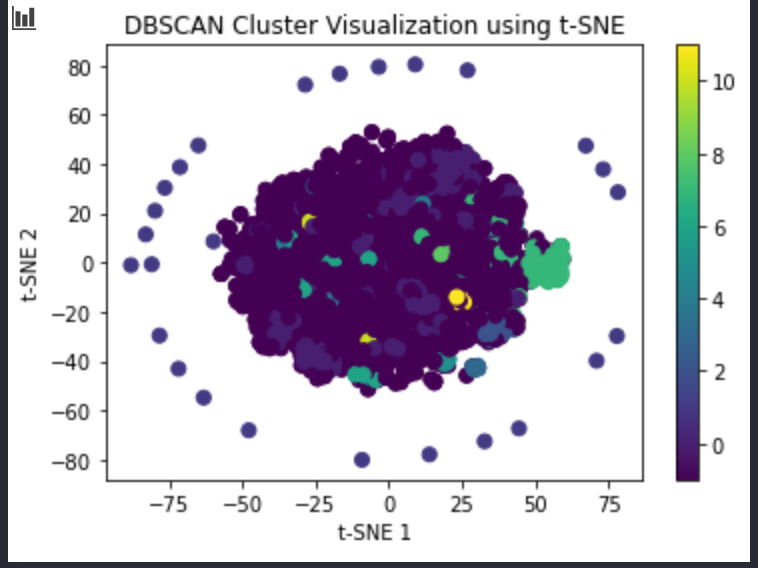


Figure 12 Cluster Visualization for DBSCAN

**Spectral Clustering**

Spectral Clustering, a graph-based clustering technique, was employed on the dataset, offering an appealing method to test and compare our results due to its ability to detect non-convex clusters. This approach is similar to K-Means in that we need to specify the desired number of clusters. However, both methods are susceptible to the presence of outliers, and preprocessing or post processing can help mitigate their influence.

Spectral Clustering constructs a similarity graph using pairwise similarities between data points, in our case, cosine similarity was used as the distance measure. It then generates eigenvectors of the dataset's Laplacian matrix and uses K-Means to cluster the eigenvalues. This process is essentially equivalent to graph partitioning.

In choosing the number of clusters, we decided to use the silhouette score, iterating through different cluster quantities, as we did with K-Means, to find the optimal number of clusters that generated a high silhouette score. We ultimately chose 8 clusters, as it seemed good enough, and we wanted to match the number of clusters chosen for K-Means so that we could compare their purity and entropy. As shown in Figure 13 below, we evaluated the results for each iteration using the silhouette score.

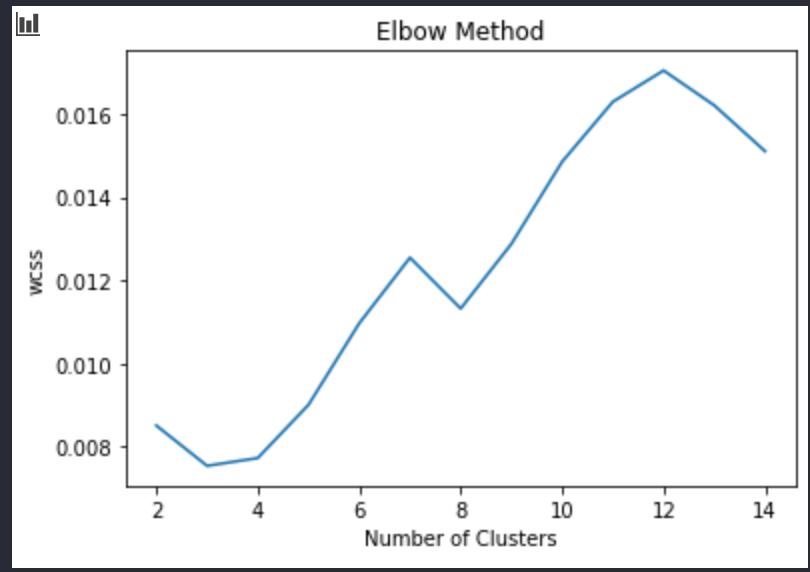


Figure 13 Silhouette Score for Spectral Clustering by Cluster amount

Figure 14 below displays the visualization for our Spectral Clustering, again using t-SNE for dimensionality reduction.

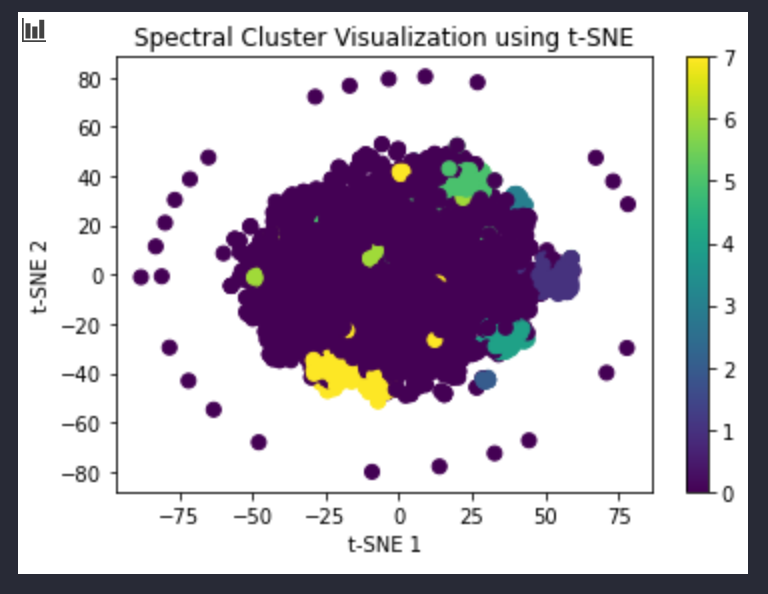


Figure 14 Cluster Visualization for Spectral Clustering

The clusterings generated appear to be similar to those of Mini Batch K-Means and K-Means, which was our expectation due to Spectral Clustering's ability to detect clusters through eigenvalue analysis and then K-means. Further parameter tuning and exploration of different affinity measures could potentially lead to even better results.

Below in figure 15 is our evaluation metrics for Spectral Clustering.

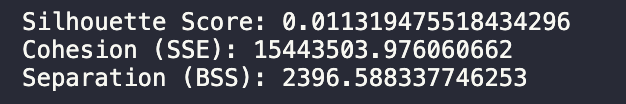


Figure 15 Metrics for Spectral Clustering

In Figure 16 is the number of clusters and their sizes after going through Spectral Clustering.



Figure 16 Clusters and Cluster Sizes for Spectral Clustering

**Entropy and Purity comparison**

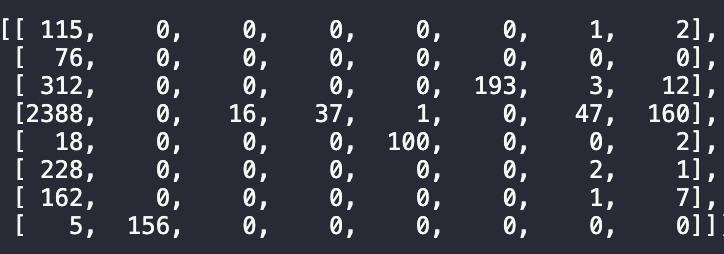
For determining the consistency of our clustering results we compared clusters generated by the K-means algorithm with those of the Spectral clustering algorithm. The first step was to generate a confusion matrix of the clusters. As we did not have the true labels of the clusters we created a confusion matrix using the K-means clusters as the true ones. See Figure 17 below for the computed confusion matrix. 

Figure 17 Confusion Matrix of K-means Clusters Versus Spectral Clusters.

Purity was calculated as the sum of the maximum values in each column of the confusion matrix divided by the total number of data points. Entropy was then calculated for each cluster, and then the weighted sum of the entropies was computed to obtain the total entropy.

As seen in the below figure purity was high for the comparisons between the two clusters as purity ranges from 0 to 1 with 1 being the values in the clusters are all of one class.



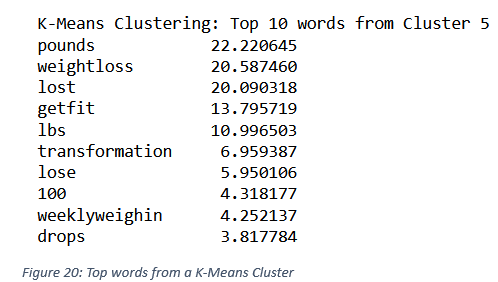
Figure 18 Purity of K-Means Clusters compared to Spectral Clusters

Entropy was also a good value for this comparison as the higher the entropy the more different classes are contained in a cluster. For clustering we want our cluster to be very similar to themselves but different from other clusters. With a lower entropy value it means that comparing the K-means cluster to a Spectral density cluster they were usually similar. Entropy goes as low as 0 when the clusters are completely homogenous.



Figure 19 Entropy of K-Means Clusters compared to Spectral Clusters

**Cluster Evaluation**

Figure 20 above lists the top ten most common words for one of the clusters generated by our K-Means implementation. The tweets within this cluster are clearly about weight loss. In examining our clusters, we found that they were coherent. We had clusters about eating healthy, allergies, and cancer to name a few. In analyzing the top words from each cluster we found that some words that should have been on our stop word list had made it through. For example, the abbreviation for the word retweet, displayed as “RT” showed up as a top word in many clusters. This would make sense as retweeting is a common occurrence in the twitter space, however did not add much meaning for us when interpreting the meaning of the clusters. After accounting for these and filtering them out we had more success in interpreting each cluster.

**\*\*All team members agree with the specified effort breakdown below\*\***

**Peter Cork: 50%**

**Kyle Roncagli: 50%**