Interpreting Clusters of World Cup Tweets

Daniel Godfrey, Caley Johns, Carol Sadek
UNC Charlotte, BYU-Idaho, Wofford College

Mentors: Carl Meyer, Shaina Race
NC State University

Abstract

Cluster analysis is a field of data analysis that extracts underlying patterns in data. One application of cluster analysis is in text-mining, the analysis of large collections of text to find similarities between documents. We used a collection of about 30,000 tweets extracted from Twitter just before the World Cup started. A common problem with real world text data is the presence of linguistic noise. In our case it would be extraneous tweets that are unrelated to dominant themes. To combat this problem, we created an algorithm that combined the DBSCAN algorithm and a consensus matrix. This way we are left with the tweets that are related to those dominant themes. We then used cluster analysis to find those topics that the tweets describe. We clustered the tweets using k-means, a commonly used clustering algorithm, and Non-Negative Matrix Factorization (NMF) and compared the results. The two algorithms gave similar results, but NMF proved to be faster and provided more easily interpreted results. We explored our results using two visualization tools, Gephi and Wordle.

Contents

1. Background Information

Cluster analysis is the process of grouping data points together based on their relative similarities. Text mining is the analysis of large collections of text to find patterns between documents.

1.1. **Term Document Matrix.** Text datasets consist of a collection of documents and a dictionary of words used in those documents. These documents can be in the form of verses, tweets, sentences, paragraphs, or books. A term document matrix is created where the columns correspond with the documents and the rows correspond with all the words used in the dataset such that

$$\begin{array}{c} \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{array} \begin{array}{c} \begin{array}{cccc} d_1 & d_2 & \cdots & d_m \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{bmatrix} \end{array}.$$

Position ij in the term document matrix is the number of times word i appears in document j. This is the matrix that we will use in our clustering algorithms.

1.2. **Feature Selection.** Feature selection is choosing to use a subset of more important features from the set of existing ones. We do this because there are some features, in our case words, that will not help us find patterns in the data. This process also helps reduce the dimensions of our dataset. In text analysis, we use three main processes for feature selection.

1.2.1. *Stop Listing.* In text data, there are many function words such as 'the', 'and', 'of' which appear frequently but do not contribute valuable information to the text. Stop listing removes these function words from the vocabulary so that the remaining words hold more meaning. There are many different stop lists available that can be applied to any text. Some texts requires specialized stop lists. For example, a Shakespearean play would require a stop list with more old English words than a more modern text. Other texts, such as Twitter data, require stop lists from multiple languages.

1.2.2. *Stemming.* Stemming is the process of reducing words to their base form. There are several algorithms to do this; we used the Snowball Algorithm. For example a stemming algorithm would remove the '-ing' ending of the word 'jumping' so that it was simply 'jump'. Thus, any two words that have the same stem but different endings are considered the same word. This process makes the root word more frequent in our term document matrix and could potentially increase its importance to the document. However, this algorithm has some problems. One such problem is that it does not recognize irregular verbs. For example, it does not recognize 'ran' to be the past tense of 'run'. The '-ing' ending can create a problem as 'running' would become 'runn'. Similarly, when plural words ending in '-ies' are stemmed, the algorithm does not put a '-y' at the end of the word. An example of this problem is the word 'penalties' which should be stemmed to 'penalty' but is instead output as 'penalti'.

1.2.3. *Term Frequency-Inverse Document Frequency.* In a collection of documents term frequency is not the only factor in deciding importance. For example, in a collection of documents of Disney movies the word 'Disney' will have a high frequency but this does not mean that the word 'Disney' is important to finding clusters between the documents. The term frequency-inverse document frequency (TF-IDF) matrix is a metric for weighting words so that word frequency is not the only factor in deciding the importance of the word. The inverse document frequency matrix is calculated such that

$$\text{idf}_{ij} = \log\left(\frac{n}{\text{tf}_j}\right)$$

where n is the number of documents in the collection and tf_j is the frequency of word i in document j. To create the TF-IDF matrix, we multiply the term document matrix with the inverse document frequency matrix. Therefore words that are frequently used but only in a small number of documents are weighted the most heavily.

2. Datasets

2.1. **Twitter Data.** We extracted tweets from Twitter containing the words 'world cup'; this was before the World Cup games had started. In the beginning we had 29,353 tweets. The tweets consisted of English and Spanish words. After working with the data we kept 17,023 tweets that still contained the important information.

Twitter can be a useful tool to gather information about its users' demographics and their opinions about certain subjects. For example, a political scientist might find it useful to see what a younger audience feels about certain news stories, or an advertiser might like to know what Twitter users are saying about their products. With security it is important to be able to discern between threats and non threats. Search engines also use this to discern between the various topics that can apply to one word. For example, 'Jordan' could apply to Micheal Jordan, the county Jordan, or the Jordan River.

2.2. **Other Datasets.** These are the datasets that we used to develop our understanding of the algorithms:

- Bible
  - Rafael Banches *Text Mining with MATLAB*
  - 12,224 words x 31,103 verses
  - Learning k-means and Wordle
- Abstracts from 86 different articles online
  - Created by Dr. Race
  - 1,819 words x 86 abstracts
  - Learning Non Negative Matrix Factorization
- Medlars-Cranfield-CISI abstracts
  - From Micheal Berry's LSI page
  - 11,001 words x 3,891 abstracts
  - Learning Non Negative Matrix Factorization
- Iris dataset
  - From UCI repository
  - 150 flowers x 4 measurements
  - Learning Graph Partitioning
- Seeds dataset

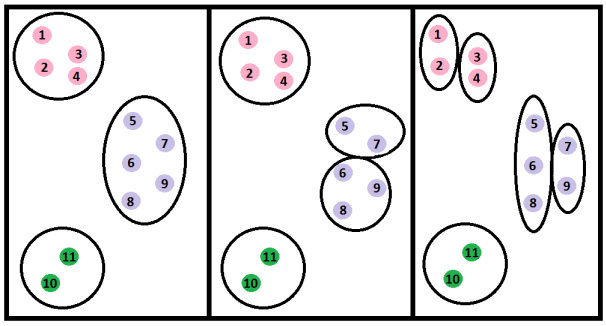
  - From UCI repository
  - 210 wheat seeds x 7 measurements
  - Learning Graph Partitioning and DBSCAN
- *E. coli* dataset
  - From UCI repository
  - 336 instances x 8 measurements
  - Learning Graph Partitioning
- Thyroid dataset
  - From UCI repository
  - 215 people x 5 protein levels
  - Learning and creating noise removal algorithms

3. Algorithms

3.1. **k-Means.** One way to cluster data points is through an algorithm called k-means, the most widely used algorithm in the field. This algorithm groups data points into clusters based on their distances. Initially k-means chooses k random points from the data space, not necessarily points in the data, and assigns them as centroids. Then, each data point is assigned to the closest centroid to create k clusters. After this first step, the centroids are reassigned to minimize the distance between them and all the point in their cluster. Each data point is reassigned to the closest centroid. This process continues until convergence is reached.

One of the disadvantages of k-means is that it is highly dependent on the initializations of the centroids. Since these initializations are random, multiple runs of k-means produce different results. Another disadvantage is that the value of k must be known in order to run the algorithm. With real-world data, it is sometimes difficult to know how many clusters are needed before performing the algorithm.

3.2. **Consensus Clustering.** Consensus clustering combines the advantages of many algorithms to find a better clustering. Different algorithms are run on the same dataset and a consensus matrix is created such that each time data points i and j are clustered together, a 1 is added to the consensus matrix at positions ij and ji. In the case of text mining, the consensus matrix is then used in place of the term document matrix when clustering again. Figure 1a shows the results of three different clustering algorithms. Note that data points 1 and 3 cluster together two out of three times. Thus in Figure 1b there is a 2 at position $C_{1,3}$ and $C_{3,1}$.



	1	2	3	4	5	6	7	8	9	10	11
1	3	3	2	2	0	0	0	0	0	0	0
2	3	3	2	2	0	0	0	0	0	0	0
3	2	2	3	3	0	0	0	0	0	0	0
4	2	2	3	3	0	0	0	0	0	0	0
5	0	0	0	0	3	2	2	2	1	0	0
6	0	0	0	0	2	3	1	3	2	0	0
7	0	0	0	0	2	1	3	1	2	0	0
8	0	0	0	0	2	3	1	3	2	0	0
9	0	0	0	0	1	2	2	2	3	0	0
10	0	0	0	0	0	0	0	0	0	3	3
11	0	0	0	0	0	0	0	0	0	3	3

(A) Clusters (B) Matrix C

FIGURE 1. Consensus Clustering

3.3. **Singular Value Decomposition.** In high dimensional datasets, distance metrics become less reliable. This creates a problem as many clustering algorithms are highly dependent on distance. Singular Value Decomposition (SVD) is one way to reduce the dimensions of a dataset without losing all meaning of the data. Given an m x n matrix, A, we can decompose it such that

$$A = USV^T$$

where U is an m x m matrix, S is an m x n matrix, and V^T is an n x n matrix. U contains the eigenvectors of AA^T, V contains the eigenvectors of A^TA, and S is a diagonal matrix of singular values of A. These singular values are the square roots of eigenvalues of AA^T or A^TA. After decomposing our data matrix A, we can rewrite it as

$$A = \sigma_1 U_1 V_1^T + \sigma_2 U_2 V_2^T + \cdots + \sigma_R U_R V_R^T$$

where $\sigma_1 > \sigma_2 > \cdots > \sigma_R$.

If we assume that irrelevant information is uniformly distributed in the data, then smaller σ's have an equal amount of noise as larger values. When we graph the singular values we are able to see where the irrelevant to relevant information ratio changes as it will create an 'elbow' in the graph. This 'elbow', the r^{th} singular value, is where we can truncate our singular values so that we retain the most amount of information while still reducing the dimension of the data. We can project our data onto the first r singular vectors of A for better distance metrics. However, singular vectors have no meaning in terms of data analysis, so there are no interpretations to get from the decomposition.

3.4. **Non-Negative Matrix Factorization.** One disadvantage to the SVD algorithm in regards to text mining is its inability to detect themes for each cluster. Non-Negative Matrix Factorization (NMF) resolves this issue by decomposing the term-document matrix into two matrices: a term-topic matrix and a topic-document matrix with k topics. The term document matrix A is decomposed such that

$$A \approx WH$$

where A is an m x n matrix, W is an m x k non-negative matrix, and H is a k x n non-negative matrix.

Each topic vector is a linear combination of words in the text dictionary. Each document or column in the term document matrix can be written as a linear combination of these topic vectors such that

$$A_j = h_{1j}w_1 + h_{2j}w_2 + \cdots + h_{kj}w_k$$

where h_{ij} is the amount that document j is pointing in the direction of topic vector w_i.

3.5. **DBSCAN.** Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a common clustering algorithm. DBSCAN uses some similarity metric, usually in the form of a distance, to group data points together. DBSCAN also marks points as noise, so it can be used in noise removal applications.

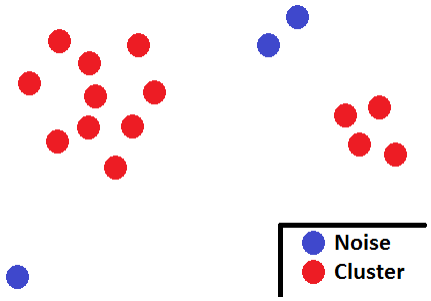


FIGURE 2. Noise Points in DBSCAN

DBSCAN requires two inputs: c minimum number of points in a dense cluster, and ϵ distance. DBSCAN visits every data point in the dataset and draws an ϵ radius around the point. If there is at least c number of points in the ϵ radius, we call the point a dense point. If there are not the c minimum number of points in the ϵ radius, but there is a dense point, then we call the point a border point. Finally, if there is neither the c number of points nor a dense point in the radius, we call the point a noise point. In this way, DBSCAN can be used to remove noise.

DBSCAN has a few weaknesses. First, it is highly dependent on its parameters. Changing ϵ or c will drastically change the results of the algorithm. Also, it is not very good at finding clusters of varying densities.

3.6. **Other.** The following algorithms are ones that we explored but we did not further investigate due to time constraints.

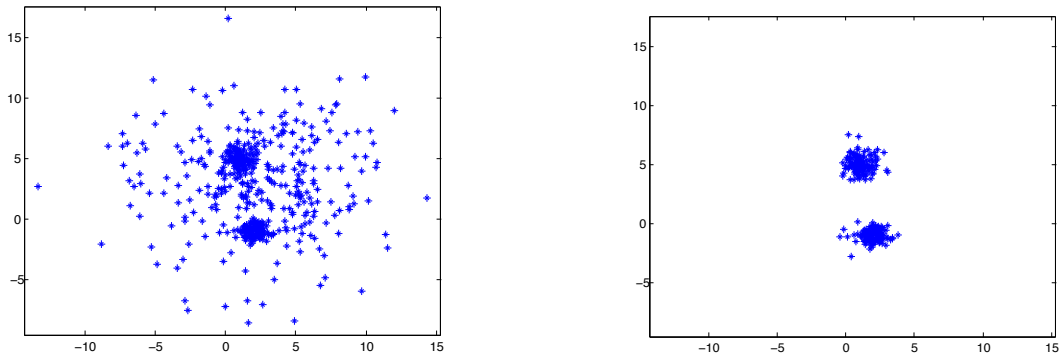
- Graph Partitioning:
  - Looks at the smallest eigenvalue and corresponding eigenvector of the consensus matrix.
  - Partitions dataset into two clusters such that cluster 1 contains the data points that correspond with the positive values in the eigenvectors and cluster 2 contains the rest.
  - Repeats until specified number of clusters is reached.
  - Problem: There is no way to know if we have partitioned enough.
- Hierarchical Clustering:
  - Looks at individual points.

  - Data points are paired based on smallest Euclidean distance.
  - Pairs are then grouped together in the same manner.
  - Continues until all points are connected.
  - Problem: No way to tell optimal number of clusters.
- Stochastic Method:
  - Balances data and creates doubly stochastic consensus matrix (P).
  - Initial vector (V_0): random probabilities for each data point.
  - $V_k^T P = V_{k+1}^T$
  - Observes the evolution of the probabilities to identify clusters.
  - Problem: In unsorted data the clusters are difficult to find.

4. Methods

4.1. **Removing Retweets.** The data we analyzed were tweets from Twitter containing the words 'world cup'. Many of the tweets were the same; they were what Twitter calls a 'retweet'. Since a retweet does not take as much thought as an original tweet we decided to remove the retweets as to prevent a bias in the data. We compared each column of the term document matrix. If columns were exactly the same one would be kept and the others would be removed. This process removed about 9,000 tweets from the data.

4.2. **Removing Noise in World Cup Tweets.** Tweets are written and posted without much revision. That is to say that tweets will contain noise. Some of that noise in the vocabulary of tweets can be removed with a stop list and by stemming. When we look at a collection of tweets we want the tweets that are the most closely related to one specific topic. Tweets on the edges of clusters are still related to the topic just not as closely. Therefore, we can remove them as noise without damaging the meaning of the cluster. Figure 3 shows a simple two-dimensional example of how the noise is removed while still keeping the clusters. We created four algorithms for noise removal.



(A) Before Noise Removal (B) After Noise Removal

Figure 3. Noise Removal

The first algorithm that we created used only the consensus matrix created by multiple runs of k-means where the k value varied. We wanted to vary k so that we could see which

tweets clustered together more frequently. These were then considered the clusters and other points were removed as noise. We did this by creating a drop tolerance on the consensus matrix. We decided that if two tweets did not cluster together more than 10% of the time that it would be like they never clustered together. Then we looked at the row sums for the consensus matrix. Again, we employed a drop tolerance and said that tweets whose row sum was less than the average for all entries in the matrix would be marked as a noise point. The problem with this algorithm is that the clusters must be of similar density. When there is a variation in the density of clusters the less dense cluster is removed as noise.

The second algorithm used multiple runs of DBSCAN that helped us decide if a tweet was a true noise point or not. The distance matrix that we used was based on the cosine distance between tweets. We used the cosine distance because it is standard when looking at the distance between text data. Since our first algorithm removed less dense clusters, we wanted to make sure that they were still included and not removed as noise. As DBSCAN is so dependent on the ϵ we thought that if we used a range of ϵ we would be able to include those clusters. Through experimentation we found that larger data sets required more runs of DBSCAN. We created a matrix that was the number of tweets by the number of runs of DBSCAN where each entry ij in the matrix was the classification: dense, border, or noise point, for the i^{th} tweet on the j^{th} run. If the tweet was marked as a border point or noise point by more than 50% of the runs it was considered a true noise point. We also looked into varying c. However, this created problems as the algorithm then marked all the tweets as noise points. Therefore, we decided to keep the c value constant. While this algorithm kept clusters of varying density it was more difficult to tell the clusters apart.

Since the consensus matrix is a similarity matrix, we decided to use DBSCAN on that matrix instead of a distance matrix. The idea is similar to the second algorithm; we still varied ϵ and kept c constant. The ϵ value in this algorithm was now the number of times tweet i and j clustered together. We performed DBSCAN multiple times on the consensus matrix and created a new matrix of classification as described before. Again we decided that if the tweet was marked as a border point or noise point by more than 50% of runs it was considered a true noise point. This algorithm is unique because it removes noise points between the clusters. We found that this is because the points between clusters will vary more frequently in which cluster they belong.

We wanted use all the strengths from the previous algorithms so we created this one. It looks at the classification from each algorithm where a noise point is represented by a 0. Then if two of the three algorithms marked a tweet as a noise point it would be removed from the data. This allows us to remove points on the edge and between clusters but still keep clusters of varying density. This process removed about 3,000 tweets from the data.

4.3. **Choosing a number of Topics.** We then had to decide how many topics we would ask the algorithms to find. To do that we created the Laplaian matrix (L) which is

$$L = D - C$$

where D is a diagonal matrix with entries corresponding to the sum of the rows of the consensus matrix, C. We looked at the 50 smallest eigenvalues of the Laplacian matrix to identify the number of topics we should look for. A gap in the eigenvalues signifies the

number of topics. There are large gaps between the first 6 eigenvalues, but we thought that a small number of topics would make them too broad. Since we wanted a larger number of topics we chose to use the upper end of the gap between the 8^{th} and 9^{th} eigenvalue as shown in Figure 4.

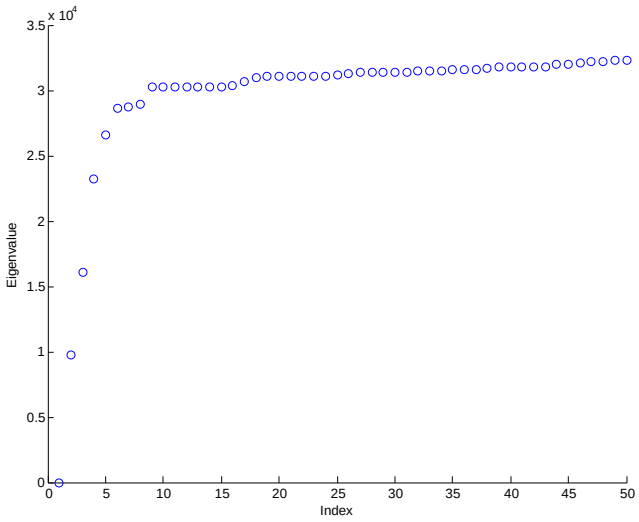


FIGURE 4. Eigenvalues of Laplacian Matrix

4.4. **Clustering World Cup Tweets with Consensus Matrix.** We clustered the remaining tweets in order to find major themes in the text data. Since k-means is the most widely used clustering algorithm and since its results are highly dependent on the value of k, we ran k-means on our Twitter data with $k = 2$ through 12. We then ran k-means a final time with the consensus matrix as our input and $k = 9$. The algorithm gave us the cluster to which each tweet belonged and we placed each tweet in a text file with all the other tweets from its cluster. Then we created a word cloud for each cluster in order to visualize the overall themes throughout the tweets.

4.5. **Clustering World Cup Tweets with Non-Negative Matrix Factorization.** The problem with using the k-means algorithm is that the only output from k-means is the cluster number of each tweet. Knowing which cluster each tweet belonged to did not help us know what each cluster was about. Although it is possible to look at each tweet in each cluster and determine the overall theme, it usually requires some visualization tools, such as a word cloud, in order to discover the word or words that form a cluster. Thus, we used a Non-Negative Matrix Factorization algorithm, specifically the Alternating Constrained Least Square (ACLS) algorithm, in order to more easily detect the major themes in our text data. The outline for the ACLS algorithm is below.

The algorithm returns a W term-topic matrix and an H topic-document matrix. The rows in W are sorted in descending order such that the first element in column j corresponds with the most important word to topic j. Thus, it is possible to see the top 10 or 20 most important words for each of the topics.

Once we found the most important words for each topic, we were curious to see how these words fit together. We created an algorithm that picked a representative tweet for each topic

Algorithm 1 Alternating Constrained Least Square

Input: A term document matrix (m x n), k number of topics
W = abs(rand(m,k))
for i = 1:maxiter **do**
    solve $W^T W H = W^T A$ for H
    replace all negative elements in H with 0
    solve $H H^T W^T = H A^T$ for W
    replace all negative elements in W with 0
end for

such that the representative tweet had as many words from the topic as possible. We called these tweets topic sentences.

5. Results

In the visualizations from Gephi we are able to see how close topics are to one another. In the graph of the consensus matrix, two tweets are connected if they are clustered together more than 8 times. If the tweets are clustered together more frequently, they are closer together in the graph and form a topic, represented by a color in the graph. The unconnected nodes are tweets that are not clustered with any other tweet more than 7 times. Because of the way k-means works we see that some of the topics are split in Figure 5. The most obvious split is the 'Falcao/Spanish/Stadium' topic.

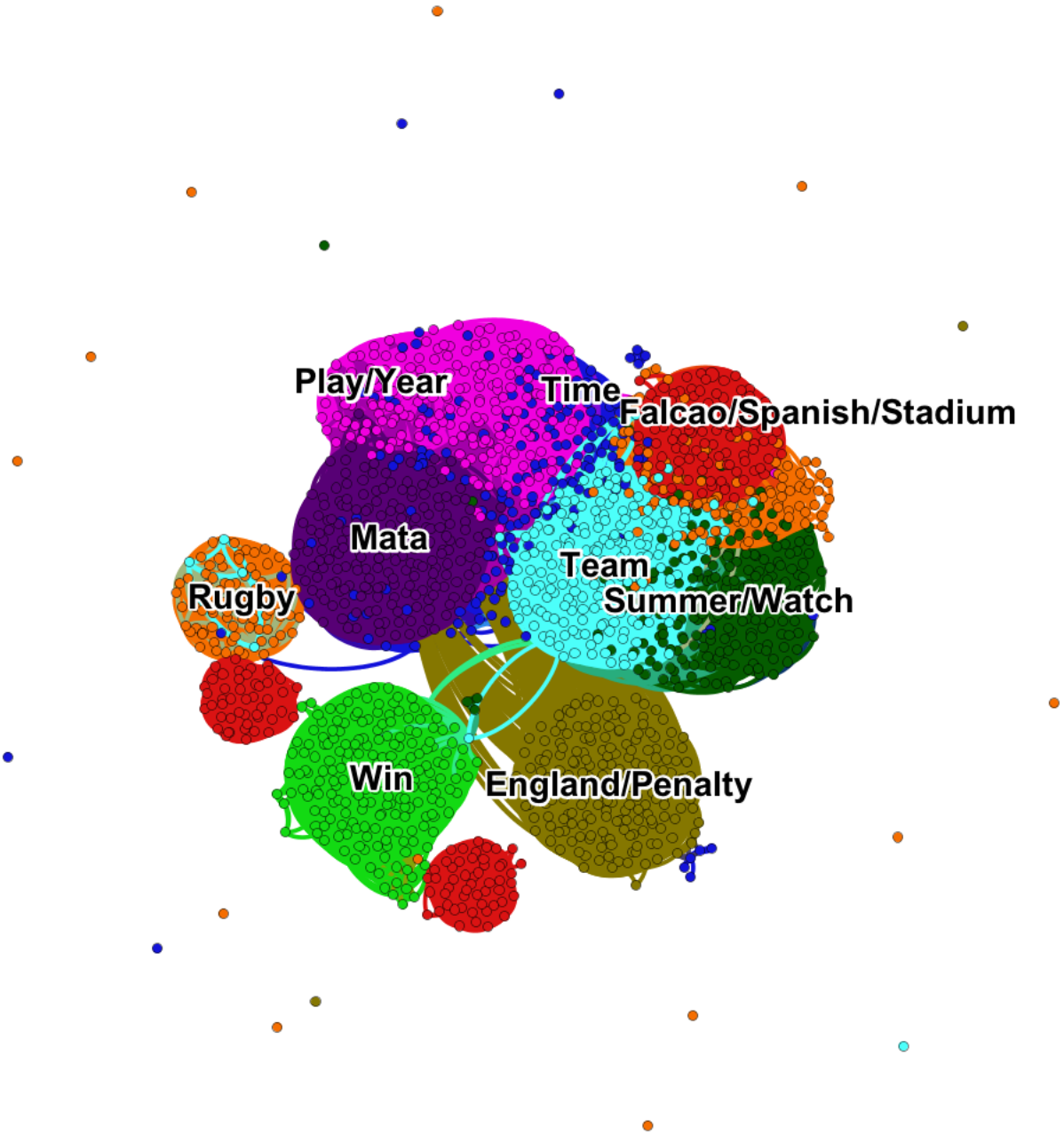


FIGURE 5. Topics found by k-means

In the graph for NMF, the colored nodes represent the topic that the tweet is most closely related to. The edges emanating from a node represent the other topics that the tweet is only slightly related to. We created the graph in such a way that distance between higher weights is shorter. This pulls topics that are similar towards each other. For example, the 'FIFA' and 'Venue' topics are right next to each other as seen in Figure 6. This means that there are tweets in the 'FIFA' topic that are highly related to the 'Venue' topic. When we further examined these two topics, we found that they both shared the words 'stadium' and 'Brazil' frequently.

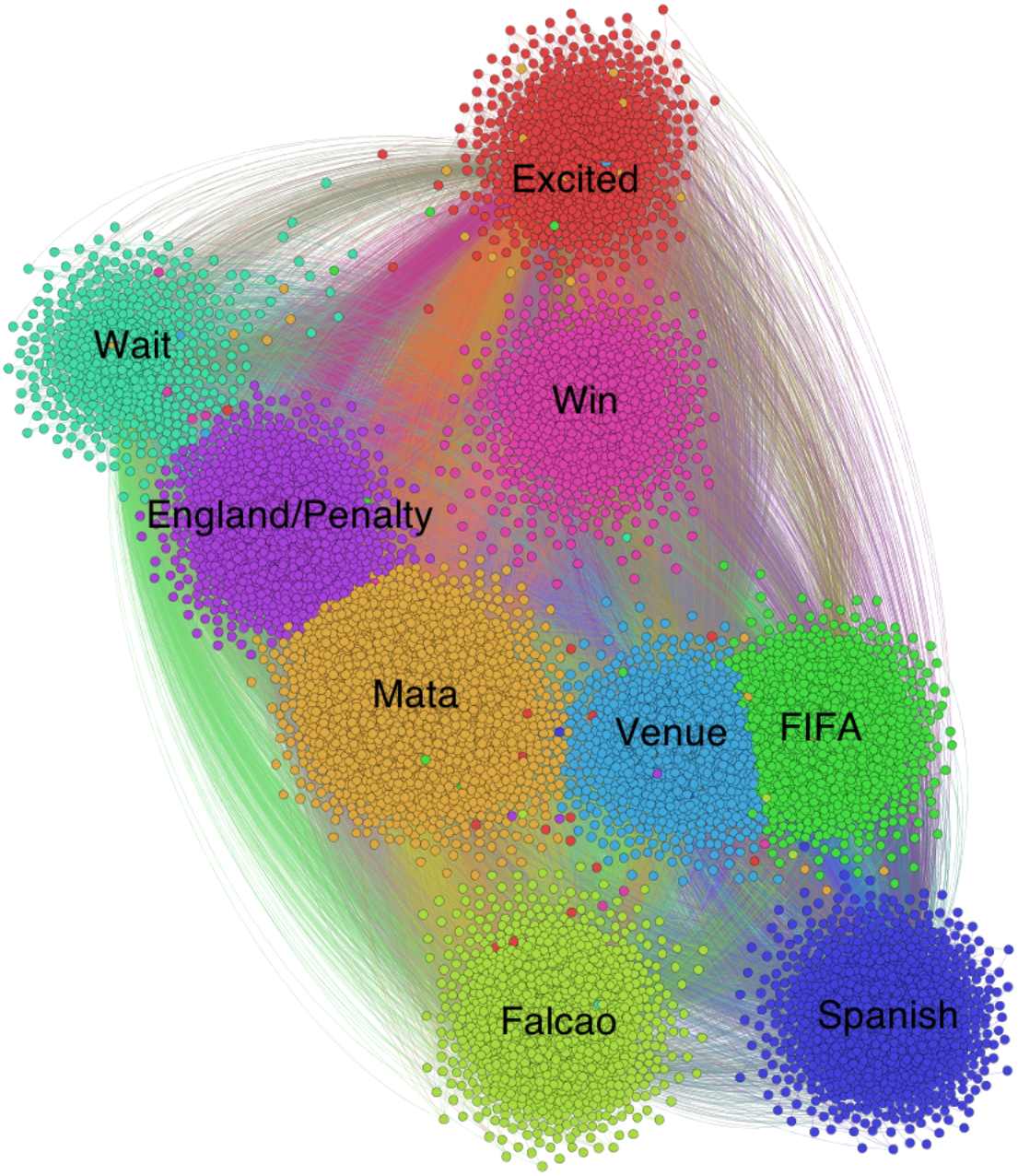


FIGURE 6. Topics found by NMF

We wanted to compare the results from k-means and NMF so we created visualizations of the most frequent words with software called Wordle. For example, NMF selected the Spanish tweets as their own topic. When we looked for the same topic in the results of k-means we found that it created one cluster that contained the Spanish topic, a topic about the player Falcao, and a topic about stadiums. From this we thought that NMF was more apt at producing well defined clusters.

FIGURE 7. Falcao/Spanish/Stadium Topic from k-means

(A) Spanish Topic from NMF

(B) Falcao Topic from NMF

(C) Venue Topic from NMF

(D) FIFA Topic from NMF

FIGURE 8. NMF topics that create k-means topic

In the end we decided that NFM was a better algorithm for clustering these tweets. It worked faster and gave more specific topics than k-means.

6. Conclusion

We used cluster analysis to find topics in the collection of tweets. NMF proved to be faster and provided more easily interpreted results. NMF selected a single tweet that represented an entire topic whereas k-means can only provide words in the topic.

There is still more to explore with understanding text data in this manner. We only looked at NMF and k-means to analyze these tweets. The other algorithms that we did not use could prove to be more valuable. Since we only looked deeply into text data, further research could prove that other algorithms are better for different types of data. We explored our results using two visualization tools, Gephi and Wordle. There is still much to be done in this aspect. In retrospect we would perform Singular Value Decomposition on our consensus matrix before running k-means. This way noise would be removed and the clustering would be more reliable.

References

[1] Martin Ester, Hans peter Kriegel, Jrg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[2] A. G. K. Janecek and W. N. Gansterer. *Utilizing Nonnegative Matrix Factorization for Email Classification Problems in Text Mining: Applications and Theory*. John Wiley and Sons, 2010.

[3] Tao Li and Chris Ding. Nonnegative matrix factorizations for clustering: A survey. pages 149–179, 2013.

[4] Carl D. Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM, 2001.

[5] Carl D. Meyer and Charles D. Wessell. Stochastic data clustering. *SIAM Journal on Matrix Analysis and Applications*, 33(4):1214–1236, 2012.

[6] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[7] Shaina L. Race. *Iterative Consensus Clustering*. PhD thesis, North Carolina State University, 2014.

[8] Shaina L. Race. Parts of a whole: Matrix factorization and dimension reduction. Presentation, 2014.

[9] Andrey A. Shabalin. k-means animation. Web.

[10] Farial Shahnaz. Document clustering using nonnegative matrix factorization. *Information Processing and Management*, 42(2):373–386, 2006.