Into the Twitter-verse: dimensionality reduction and clustering with tweets

Michael Young & Daniel Merrel

Problem Description

It's hard to imagine a world pre-Twitter. A world where people's random thoughts were expressed only to their distracted spouses or disinterested pets. A world where the president of the United States only communicated with the public through thoughtful and painstakingly planned press conferences. A world where this video: https://twitter.com/businesspastel/status/1254392896138227712?s=21 didn't exist, in fact, probably *couldn't* exist. For better or worse, that world is no more and what we have instead is a micro-blogging platform that is essentially one gargantuan dataset.

There are many potential ways to explore this dataset, but here we're primarily interested in seeing ***how the tweets of well known accounts are related to each other.*** We want to explore this question using only the raw text of the tweets themselves. We have over 88 thousand tweets from 11 well-known accounts which we'll use in our analysis. We'll use various dimensionality reduction techniques paired with clustering methods discussed in class to probe for any kind of interesting structure in the tweets that could help answer our question.

One of our main challenges is defining what makes tweets similar/dissimilar. We rely primarily on two natural language processing (NLP) methods of converting tweets into vectors that can be clustered: TF-IDF and Word2Vec (discussed below).

With these tools, we hope to gain insight into the relationships between the twitter feeds of these prominent figures. We expect to find relationships between tweets by figures that work in the same field. For instance, we expect that Barack Obama, Hillary Clinton, and Donal Trump's tweets cluster together as they are probably more related to politics than someone like Kim Kardashian. It would be surprising to us if this weren't the case. We also expect tweets from a single author to cluster together. If we set the number of clusters to the number of unique authors, will each cluster just contain all the tweets from that author? Or will tweets be more likely to cluster on topics that are shared by many authors? These are some of the things we hope our exploration will provide some insight on.

Data Source

We have a dataset of tweets that consists of 6 features:

- Date: the date the tweet was published
- Id: 18+ digit identifier (probably used by twitter)
- Link: a URL to the tweet
- Retweet: boolean indicating whether or not it was a retweet
- Text: the actual tweet
- Author: Barack Obama, Hillary Clinton, Donald Trump, Adam Savage, Kim Kardashian, Richard Dawkins, Neil DeGrasse Tyson, Scott Kelley, NASA, Fivethirtyeight, kdNuggets

Since we are mostly curious about the similarity/dissimilarity between the content of the tweets, the only features we rely on are Author and Tweet. The author data is used only for labeling the tweets in our visualizations and for analyzing cluster content. The raw text of the tweets was the only data we actually used for our NLP, dimensionality reduction, and clustering techniques. We could take our project further by using the Date feature to analyze the habits of authors (e.g. how often they tweet, when during the day/night they tweet) but we leave this for future work.

Our dataset consists of 88 thousand rows. Since our computational resources are limited, we have selected two random subsets used for our TF-IDF and word2vec techniques respectively: one which consists of approximately 500 tweets from each author, giving us about 5,000 rows, and another which consists of 1000 tweets from each author - giving us 11,000 rows.

Materials and Methods

NLP Methods:

As a pre-processing step, we made use of the Natural Language Toolkit (NLTK). This library contains a list of stopwords that we removed from the tweets. Stopwords are words that fill our speech but are not significant in meaning (words like ‘and’, ‘the’, ‘of’, etc...). In order to find more meaningful clusters, it’s important to remove stopwords. We also use a technique called ‘Stemming.’ Stemming takes a word and breaks it down to its root, or ‘stem’. It will take words like ‘fighter’, ‘fighting’, and ‘fought’, and reduce them to ‘fight’. We used NLTK’s stemming algorithm in hopes of finding more similarities between tweets. We also leveraged some of NTLK’s other tools to clean our tweets, like TweetTokenizer, that was designed specifically for reducing the sentences of tweets to an array of words.

We relied on two NLP techniques in order to get our tweets into a numerical form that we could use for clustering: TF-IDF (Daniel) and word2vec (Michael).

TF-IDF stands for Term Frequency - Inverse Document Frequency. It is a method of assigning weights to each word in a document based on their frequency. Each weight is offset by the occurrence of the word in all of the documents. We used Scikit Learn's implementation of TF-IDF for this project. It takes a corpus of documents and outputs a sparse matrix containing the TF-IDF weights for each document. From there, we were able to use various clustering techniques.

Word2vec is a fairly recent (2013) approach to natural language processing that seeks to solve some of the shortcomings of a traditional "bag-of-words" approach by leveraging the power of neural networks. Traditional approaches can suffer because they fail to learn meaning between words. As a result, distances between words don't always indicate a distance between the words' meaning. Word2vec uses shallow neural nets to embed words into a vector space in which distance between vectors do actually indicate some sort of linguistic relationship. The results of this technique can be quite stunning. For instance, it can produce results like: King - Man + Women = Queen (where each word here represents a high dimensional vector in embedded space).

Rather than train our own word2vec model on our dataset (which would have been cool, but perhaps more relevant for a machine learning course), we used a pretrained network trained on 2 billion tweets (https://nlp.stanford.edu/projects/glove/). This network trained using a different but highly similar method to word2vec (GloVe), which we converted to a word2vec file format so that we could use it with the popular NLP python library genism (https://radimrehurek.com/gensim/). After some trial and error (discussed in the evolution section), we chose to use the 200 dimension vector word embeddings with our tweets. In this framework, each word in a tweet was represented by a 200-dimensional vector. To then represent a tweet, we averaged all of the 200-dimensional word vectors in the tweet to get a 200 dimensional 'tweet' vector. It was with this set of 200 dimensional tweet vectors that we ran all of our analysis.

Dimensionality Reduction:

Because of what is termed 'the curse of dimensionality' we figured it may be a good idea to reduce the dimensions of our tweet vectors before running clustering on them (although in one case, clustering on the high dimensional vectors wasn't too bad). To do this, we relied on some dimensionality reduction techniques discussed in class: t-SNE (word2vec only), and PCA (both). Both of these techniques were implemented with scikit-learn.

t-SNE (t-distributed stochastic neighbor embedding) is a non-linear dimensionality reduction technique aimed at preserving local neighborhoods in a projection from high dimensional space to low dimensional space. The general flavor of this looks

like constructing a probability distribution between pairs of points in both high dimensional and low dimensional spaces, then minimizing the KL divergence between the two distributions. This is often very effective for reducing high dimensional data to low dimensional data (2-3 dimensions) for visualization purposes, and sometimes clustering. We used it for both.

PCA (principal component analysis) is a linear dimensionality reduction technique which seeks to find a basis coordinate system that best captures the variance in the data. It does this by eigen-value decomposition of the covariance matrix (can also be done with singular value decomposition). We used this primarily to cluster on more dimensions than t-SNE would allow us. We visualized the results of these clusterings on t-SNE.

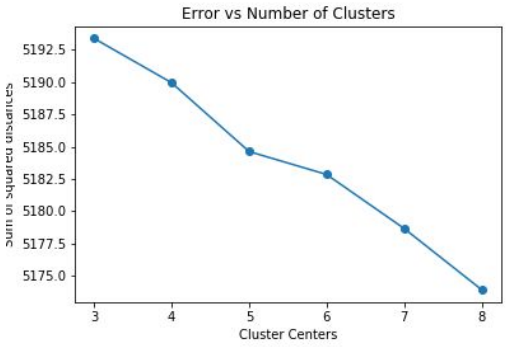
Clustering:
We used two clustering methods to examine our data: k-means (for both) and density based spatial clustering of applications with noise (DBSCAN) (for word2vec). Both of these were implemented using scikit-learn.

K-means using Lloyd's algorithm essentially finds a predetermined number of clusters by iteratively updating the centroids of the clusters to be the mean of all of the points within the cluster. It begins with an initial selection of k cluster centers. The algorithm then assigns all of the points to their nearest cluster center. The new centers are then chosen to be the averages of all of the points that belong to each cluster. This process can occasionally get 'stuck' in 'bad' clusterings if the initial points are chosen unluckily. This can be helped by initializing the points using k-means++ (essentially the gonzalez algorithm), which we used in our implementations.

DBSCAN is an interesting clustering approach in the sense that it doesn't require a predetermined number of clusters beforehand, like k-means. It detects clusters based purely on density - that is, points with a lot of nearest neighbors within some sort of distance get paired together, and points that lie in low density regions without enough neighbors designated as close enough get dubbed as 'noise'. We wanted to use this as a counter-weight to k-means and see what difference (if any) there was in the kinds of clusterings they produced.

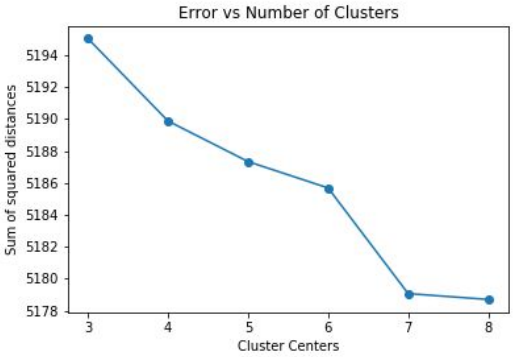
Evolution

We used Scikit Learn's implementation of TF-IDF to convert our tweets into a sparse matrix of size ~5000 x 80,000. We ran k-means on this matrix, and unsurprisingly,

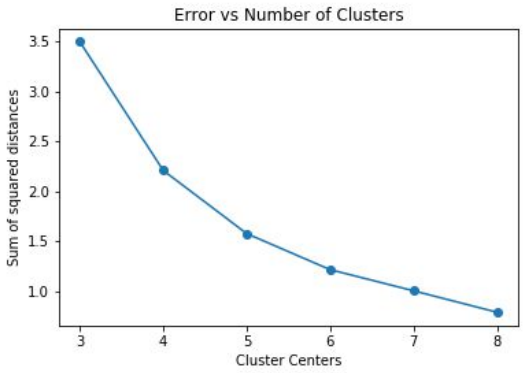


we were not able to find a clear number of clusters that reduced the cost significantly.

Since cluster centers are initialized randomly with k-means, we used k++ to initialize centers and then ran k-means. This seemed to perform much better and from the cost plot, it was clear that 7 clusters was the right number to use.



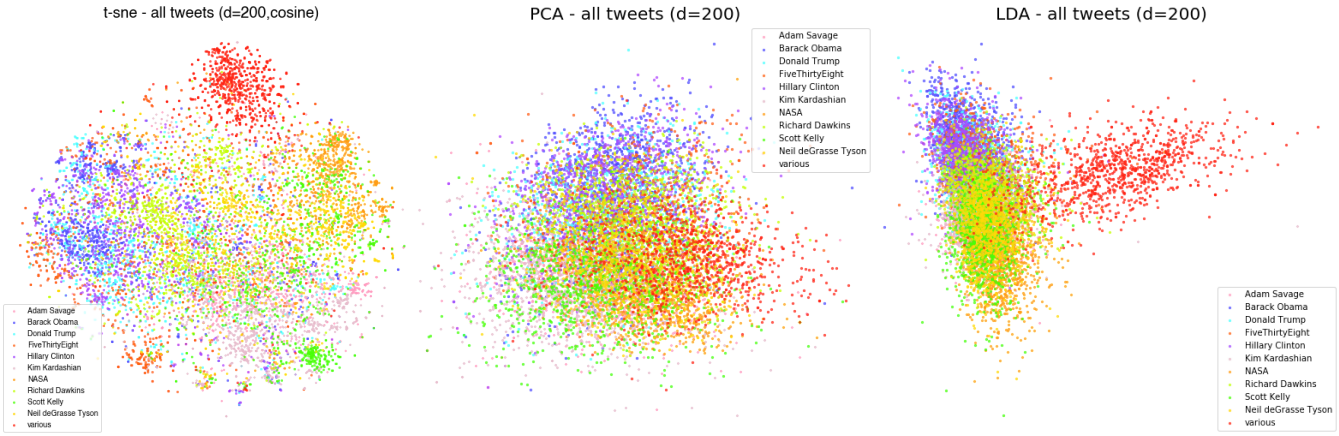
Running k-means on large matrices is not always reliable because it's possible that the algorithm finds local minima. To avoid this, we tried out some dimensionality reduction methods. First, we tried Multi-Dimensional Scaling, but it performed poorly due to the size of the matrix. After several hours, it still had not finished. Next, we tried to find the principal components using Singular Value Decomposition. At 200, 100, and 50 dimensions, we still had linear-looking cost plots. We had the best results clustering on a tweets that were reduced to 2 dimensions. We generated the following cost plot:

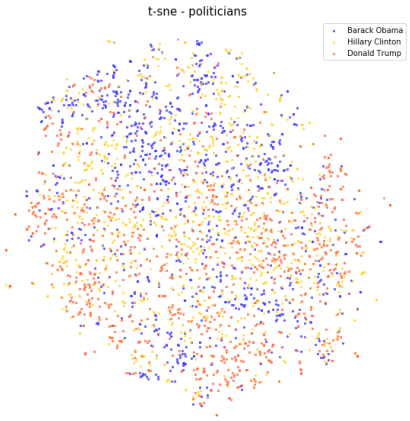


We were able to recover interesting results using 5 clusters and with 2 dimensions, we were able to visualize the results in a scatter plot. The scatter plot and results are discussed in the next section.

Things were pretty smooth-sailing in terms of converting tweets into vectors using the word2vec implementation from genism. There was some tinkering with the dimensionality of the vectors used, but between 50, 100, and 200 dimensions, 200 showed perhaps the 'tightest' results on a t-SNE plot, meaning the tweets seemed to gather together in the 'clumpiest' formation.

Next came the decision about how best to display all of the tweets. t-SNE using the cosine distance showed the most desirable results (we also tried euclidean). We also tried PCA and even linear discriminant analysis (LDA) as well, but as seen these methods in 2D had too much overlap between tweets to be as successful visualization.



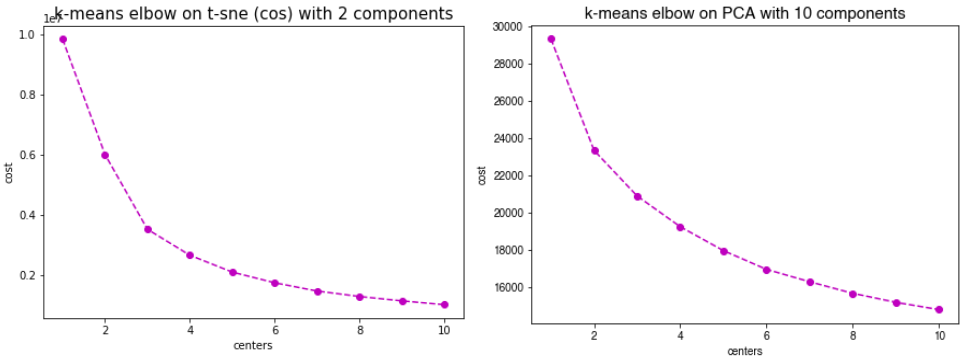


After seeing the t-SNE plot we noticed a conglomeration of mostly political figures' tweets in the left of the plot (the blues and purples). We wondered if we just ran t-SNE on those if there'd be some neater clumping between say Obama's and Clinton's tweets with Trump's. As seen to the right, there wasn't a super clear relationship to the degree we thought there might be, so we mostly abandoned this line of inquiry.

Once the visualization method was determined, it was time to decide what dim-reduction + clustering methods we wanted to explore. t-SNE and PCA paired with k-means and DBSCAN worked well. Additionally, I tried MDS again, but after 3+ hours my computer was still attempting to compute the similarity matrix, so I nixed that for scalability reasons.

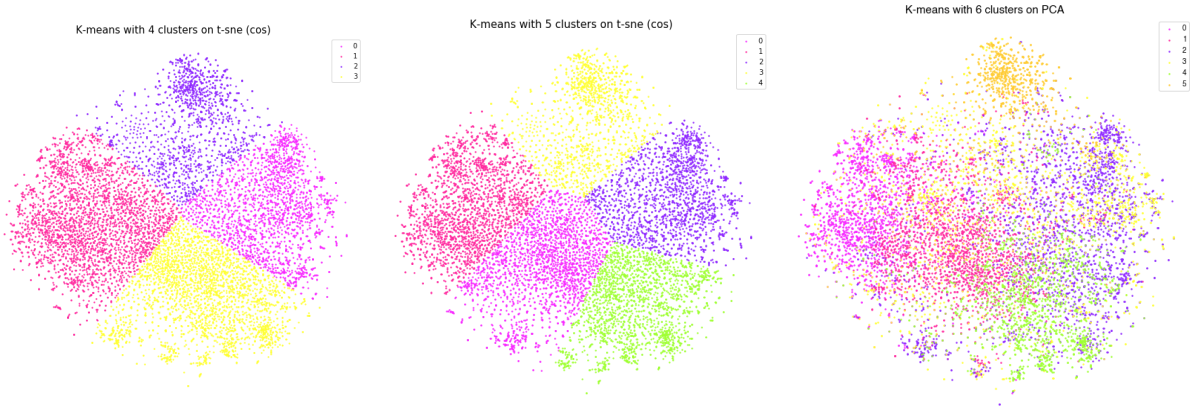
The next question was how many dimensions should we attempt to cluster on. t-SNE can only effectively reduce to 2-3 dimensions, so we arbitrarily chose to focus on 2. With PCA, we toyed with using 5, 10, and 20 dimensions for clustering. 10 seemed to work, so we stuck with that for all of the analysis.

We ran some cost plots to determine which k's to examine for k-means for both 2 t-SNE dimensions and 10 PCA dimensions. Based on these plots, we



decided that for t-SNE, k's of 4,5, and 6 might be good to explore. For PCA on the other hand, it looked like 6,7, and 8 would be more appropriate.

We clustered and visualized these results on top of a 2D t-SNE plot. We mostly judged the quality of these clusterings based on how well they compared to our 'ground truth' plot, which was just the t-SNE labeled by the authors of the tweets. As discussed in the analysis section, this may not have been the best way to compare, because some clusters may be composed of very similar tweets in terms of topic by a variety of different authors. Based on these plots, we chose to do some deeper dives into t-SNE with 6 clusters and PCA with 8 clusters.

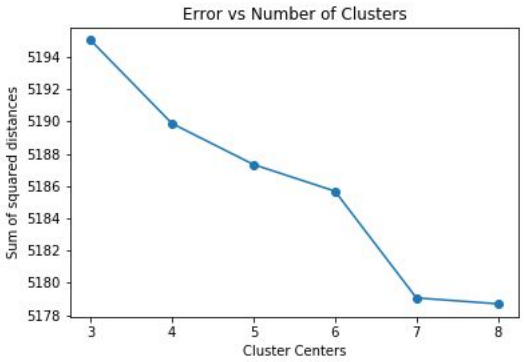


Finally we ran DBSCAN on both t-SNE with 2 dimensions and PCA with 10 and visualized the results on a t-SNE plot (shown in analysis sections). This took a lot of parameter tuning to achieve desirable results (i.e. results with minimal noise). This was especially difficult for PCA with 10 dimensions. DBSCAN seems to work much better in lower dimensions.

Analysis

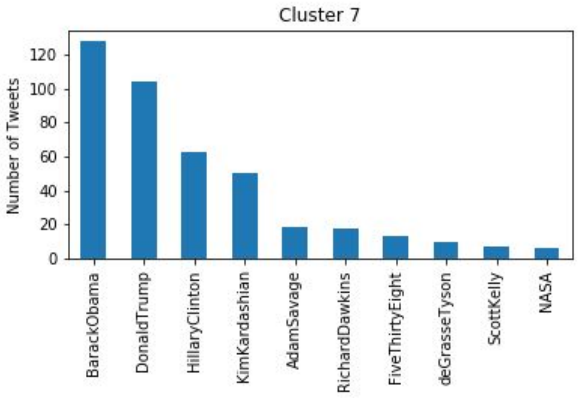
TF-IDF

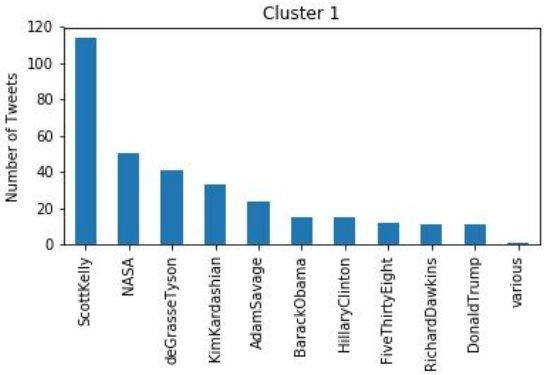
We saw the best results using k-means using initialized points from k++. Running this algorithm directly on the large sparse matrix without performing any kind of dimensionality reduction turned out surprisingly well. We can see in the plot below that there was a notable drop in cost at 7 clusters.



We found interesting results in many of the clusters. For example, the top words in cluster 7 were "president", "lovely", "need", "obama", "run", and "realdonaldtrump". The bar chart below (labeled "Cluster 7") shows the author contribution to that clusters.

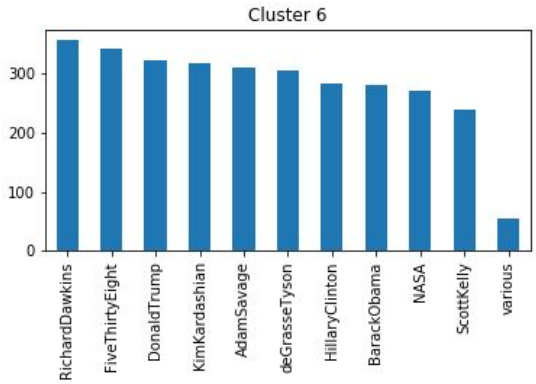
Most of those words seem related to politics so it is unsurprising that most of the tweets in that cluster came from Obama, Trump, and Clinton. We found it surprising that so many of Kim Kardashian's tweets were part of the cluster.

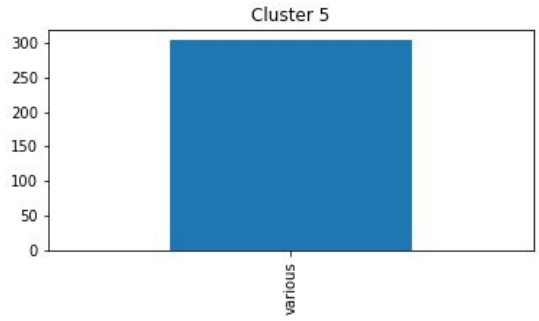




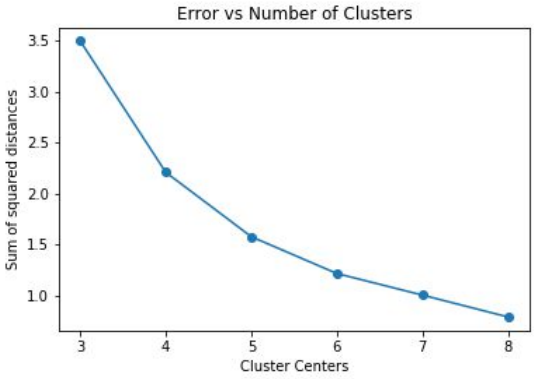
In the first cluster, we found words like "day, good, night, earth, space, station, goodnight". It seems reasonable that the top contributors to this cluster come from science and space related accounts. Again, we were surprised to see that Kim Kardashian's contributions came in at #4.

The results of other clusters are harder to make sense of. For example, in cluster 6, everyone's contributions were evenly spread with generic-sounding words like "new, just, makes, today, thanks, year, time". However, it was interesting to see that while most accounts had an even spread of contribution, the machine learning/data science account called kdNuggets (labeled "various" in the plots below) had hardly any tweets in the cluster. In fact, cluster 5 was composed of tweets *only* from kdNuggets.





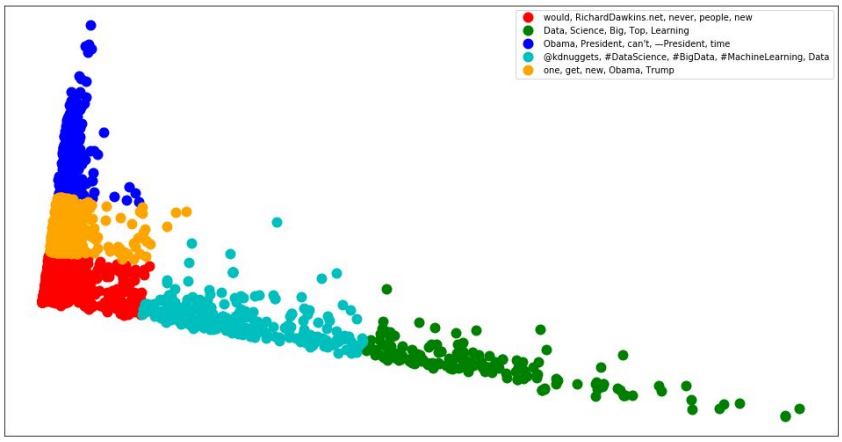
As mentioned above, we also used SVD to reduce the dimension of the large sparse matrix to 2. We chose to use 5 clusters based on this cost plot we generated:

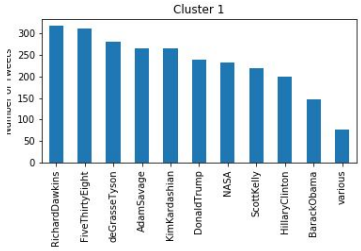


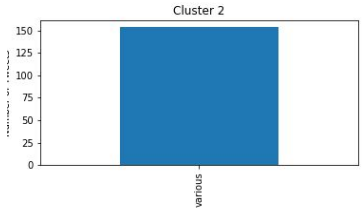
We plotted the results along with the top terms from each cluster. The clusters looked surprisingly similar to the clusters that were found using the data before dimensionality reduction. We found one cluster (cluster 3) with political-sounding words with highest author contribution from Obama, Clinton, and Trump. And again, the data science account kdNuggets formed its own cluster, and the majority of another cluster (clusters 2 and 4). The results are shown below, along with the top terms for each cluster:

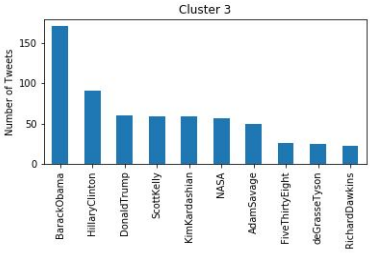
Top Terms:

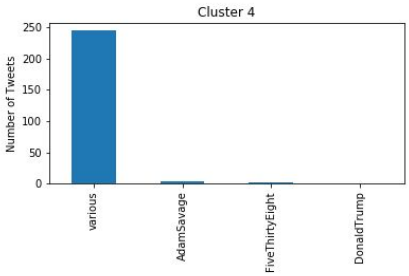
Cluster 1: would, RichardDawkins.net, never, people, new
Cluster 2: Data, Science, Big, Top, Learning
Cluster 3: Obama, President, can't, —President, time
Cluster 4: @kdnuggets, #DataScience, #BigData, #MachineLearning, Data
Cluster 5: one, get, new, Obama, Trump

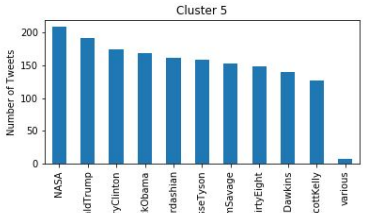










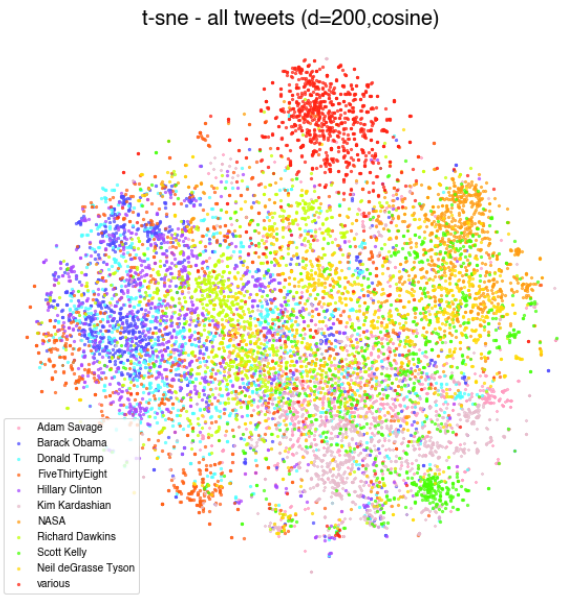


We were hoping to find more separation and distinction between some of the non-political and non-data science accounts. They seemed to blend in evenly between clusters 1, 3, and 5. We found better results for these accounts using Word2Vec.

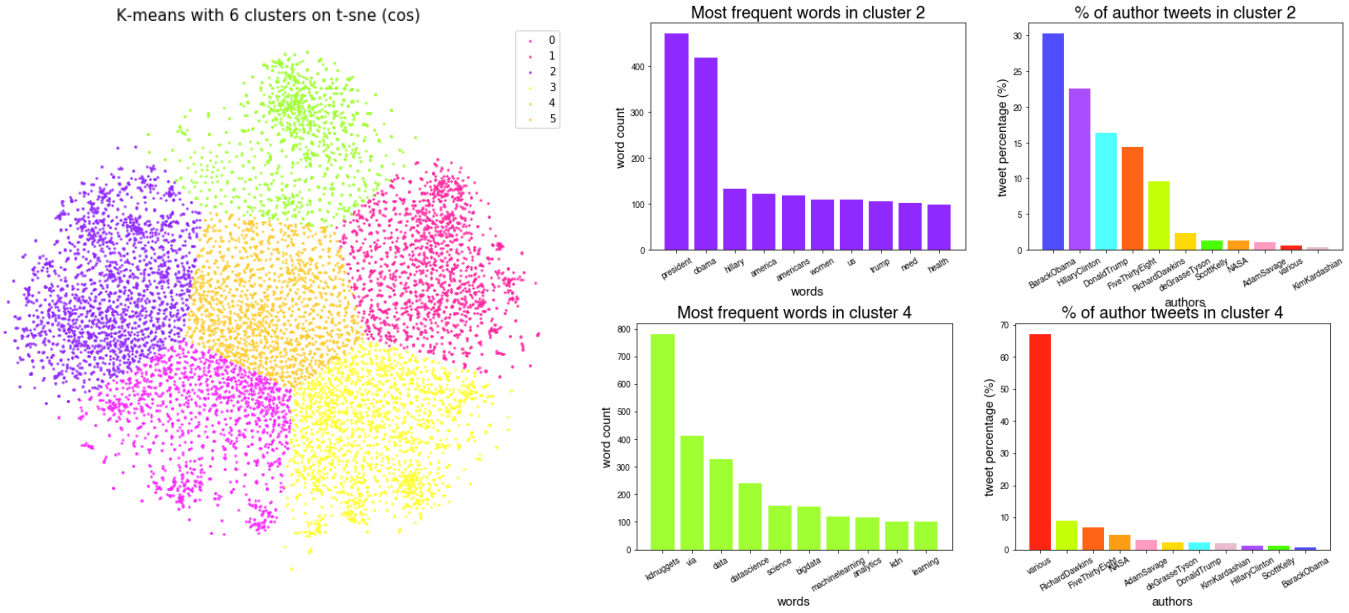
Word2Vec

t-SNE

The plot to the right shows the results of running t-SNE (with the cosine distance) on our tweets. Immediately some structure was apparent. The red blob near the top of the plot consists of only tweets from kdNuggets - more evidence on how distinct these tweets are from all others in the set. The chromatically cool colors (blues and purples) correspond to the political accounts (obama, clinton, and trump) and seem to be concentrated to the left of the plot. Fivethirtyeight (orange) was mixed in there as well, which makes sense because they do a lot of political-themed data work. The longer one stares at this plot the more relationships that make sense come to light.

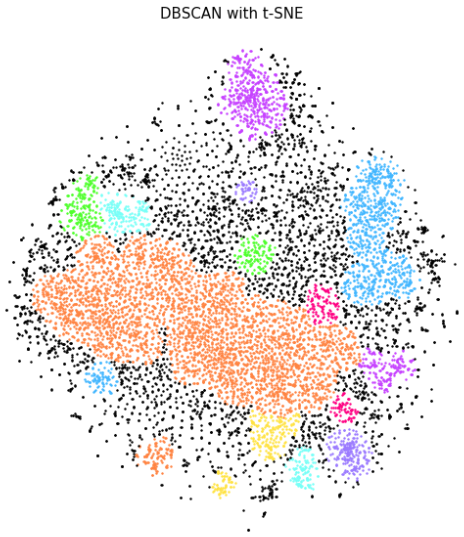


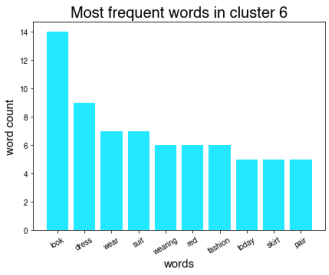
From here we did some clustering (this and following clustering effort's parameter rational discussed in the evolution section). Seen below is a k-means plot with k=6.

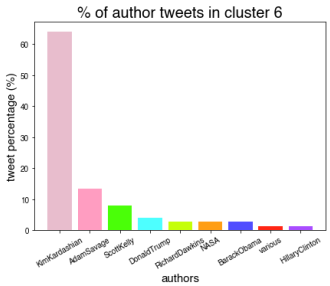


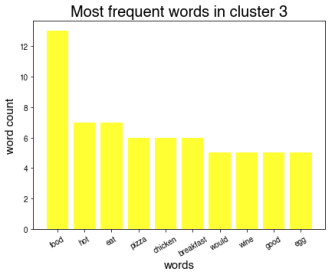
Some exploration of the cluster statistics supports many of the qualitative insights about the tweet relationships discussed above. For instance cluster 2 (purple) is in the location that appeared to be composed of mostly political tweets, and the cluster results report this, with ‘president’, ‘obama’, ‘hillary’, and ‘america’ being the most common words in the cluster. Additionally, cluster 4 (green) confirms our visual insight that that top clump of tweets all belong to kd-nuggets. Interestingly, the next highest contributing authors to this cluster were Richard Dawkins, FiveThirtyEight, and NASA. This makes some sense as all three of these accounts have something to do with science. The other clusters also showed interesting results indicating clear relationships between the tweets within the cluster.

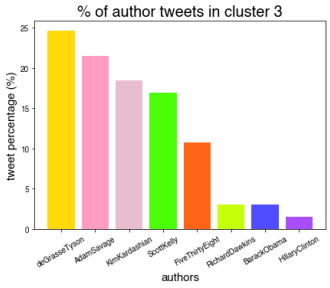
Next we ran DBSCAN on t-SNE. As discussed in the evolution section, there was significant interactive parameter tuning involved to get a clustering we thought looked ‘good’, yet our final result still involved a lot of noise. An examination of these 16 clusters reveals some exciting structure. For instance, cluster 6’s most frequent words are ‘look’, ‘dress’, and ‘wear’, and the most frequent author by far is Kim Kardashian. This is clearly a “fashion” cluster. Cluster 0’s most frequent words are ‘nba’, ‘nfl’, and ‘win’, and the most frequent author is FiveThirtyEight. Clearly this is a ‘sports’ cluster. Cluster 3’s most popular tweets are ‘food’, ‘hot’, ‘eat’, and ‘pizza’, and the most common authors are Neil Degreasse Tyson, Adam Savage, Kim Kardashian, and Scott Kelly - all tweeters with individual accounts who may be more likely to tweet about their personal lives. This is clearly a ‘food’ cluster. These clusters are more specific and topic-oriented than those found by our k-means 6-clustering. One reason is certainly because with only 6 clusters, k-means necessarily lumps in a lot more tweets that

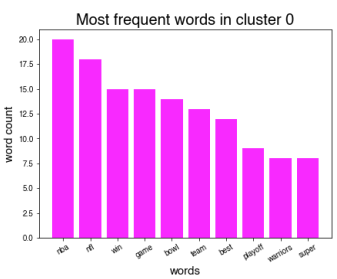


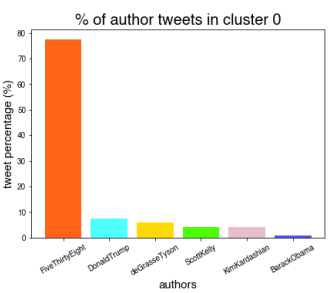








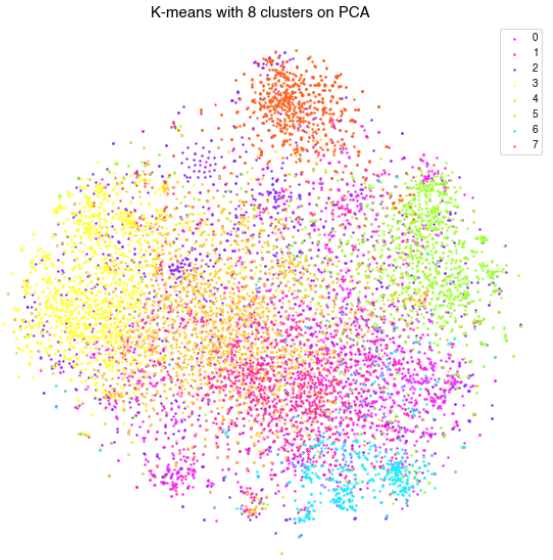


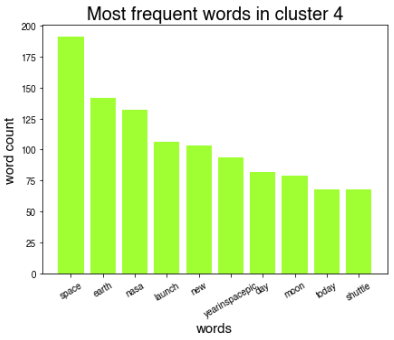


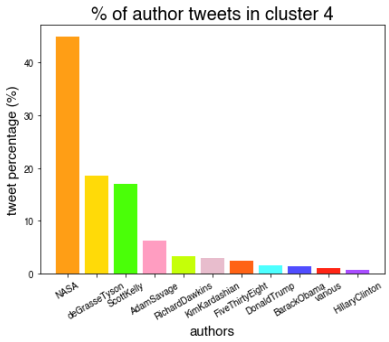
may not be too closely related, thus making the clustering content more general. This generality may be a desirable characteristic. However, a higher k may have made the clusterings more granular and specific in a way similar to these DBSCAN clusters. Another reason these clusters seem to be more topic-specific is that DBSCAN arguably clusters things in a more 'natural' way in the sense that it doesn't require a predetermined number of end clusters. This quality makes this clustering technique desirable, especially in exploratory settings. An additional clustering consideration that may capture a happy medium between k-means and DBSCAN would be hierarchical clustering, which could be explored in a future work.

PCA

Next we explored our PCA clusterings. As discussed in the evolution section, we chose to focus on k-means with 8 clusters on 10 dimensions from PCA. The resulting plot looks very similar to our 'ground truth' t-SNE plot. For instance, the political tweets appear to mostly be in yellow here, with the data science tweets in orange. This suggests that k-means on 10 dimensions of PCA results in a fairly sophisticated clustering. It found many similar clusters with the k-means + t-SNE clustering (i.e. political clusters, data science clusters, and pop culture clusters). Additionally, cluster 4 (green) seemed to be a 'space' oriented cluster with words like 'space', 'earth', and 'nasa' as top words and with NASA, NDT, and Scott Kelly as the top contributors.

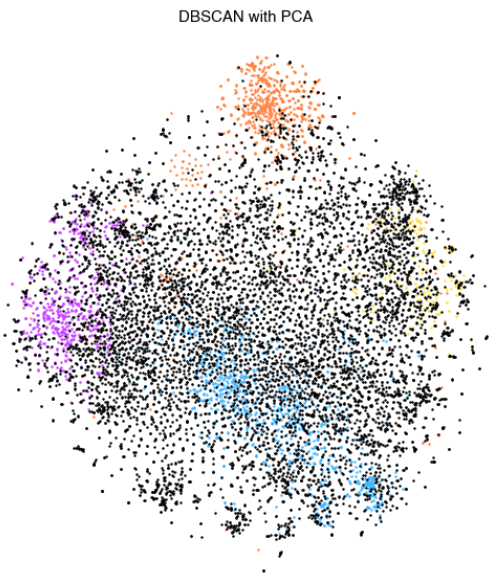






Next we ran DBSCAN with 10 dimensions of PCA. Obviously this didn't work very well. As shown, the majority of the points in the plot were considered noise, with only 4 true clusters. Much of this may be due to poor parameter tuning on our part, but some of it may be due to the inherent challenge of performing DBSCAN in higher dimensional spaces. Even though much of the data was missing, there were still some interesting cluster results. It was able to recognize the kdNuggets (orange), political (purple), and space (yellow) clusters. The blue cluster consists mainly of Kim Kardashian, Scott Kelly, and Adam Savage, with words like 'day', 'today', 'night', and 'people' among the top

words. This suggests these may be tweets from personal accounts about everyday, mundane topics.



Returning back to the problem posed at the beginning: can we probe a corpus of tweets and identify interesting relationships between them? We've demonstrated that the answer is clearly yes. Both TF-IDF and word2vec showed to be effective to differing degrees at establishing a way to create distances between tweets. t-SNE and SVD proved to be more than capable of presenting a way to visualize these tweets. Clustering on t-SNE and PCA output using k-means and DBSCAN produced meaningful clusters that revealed insights into how the tweets in our sample were related to one another. Regardless of the NLP method we used, the tweets seemed to cluster well based on topic. Where there are topics that are more general (such as food) we see a lot of authors contributing. Where there are topics that are more specific (such as space or politics) we see a smaller range of authors contributing. In what is a rare case in the domain of science, what we thought we would see is actually what we observed.