ANLY 501 Project: Part 2

November 8, 2016

Joo Chung and Jingshi Sun

Introduction

Revisiting the Data Science Problem

The first part of this project was motivated by the possibility of predicting the music listening behaviors in the United States through the general emotional state of the country. Specifically, we aimed to collect aggregated music listening behaviors through Last.fm, integrate lyrics data from LyricWikia, with the intention of further integrating emotional data aggregated at the national level through Twitter at a later time. We left Part 1 of the project having collected the 5656 ranked song data (some of which were found to be duplicates) and the appropriate lyric data for approximately 92.4% of the data.

State of the Data

While the data science problem has remained unchanged, the data sources have changed somewhat between Part 1 and the current report. The song and lyric data already collected is now supplemented with some additional song data (i.e., most popular tags associated with the song, a better populated song duration variable. and the play-count of each song for a particular week; See Table 1). Further, initial examinations of Twitter data (through Twitter's REST API) yielded what we determined to be problematic results. Specifically, Twitter's REST API allowed for the search of the most popular posts, but the search required a specific search topic (e.g., the most popular posts pertaining to the election). Since we were primarily interested in blanket searches of popular posts across all topics, this yielded the issue of determining what the most relevant topics were for a given week. A second, and more problematic, issue was the fact that Twitter posts were generally very word-sparse due to the 1) the general 140 character limit, 2) prevalence of images (and comments on images, which are highly contextual), and 3) the use of words that are not typically present in dictionaries (e.g., slang) which caused issues in regard to the determination of emotional content through words.

Table 1: Additional Attributes

Attribute Name	Description
duration	A new duration from Last.fm that is better populated than the one included previously (and now in milliseconds instead of seconds).
playcount	This attribute is the number of times the song was played by all listeners during the past 7 days.
toptags	This attribute is the top five most used tags (e.g., “rock”, “indie”, “best of 2015”) during the past 7 days. These tags are user-generated.
negative	The number of negative sentiment words in the song.
positive	The number of positive sentiment words in the song.
anger	The number of words associated with anger in the song.
fear	The number of words associated with fear in the song.
anticipation	The number of words associated with anticipation in the song.
surprise	The number of words associated with surprise in the song.
trust	The number of words associated with trust in the song.
sadness	The number of words associated with sadness in the song.
joy	The number of words associated with joy in the song.
disgust	The number of words associated with disgust in the song.
wordcount	The total number of words identified through the NRC database.
negative_percent	The percent of negative sentiment words in the song (as determined by wordcount).
positive_percent	The percent of positive sentiment words in the song (as determined by wordcount).
anger_percent	The percent of words associated with anger in the song (as determined by wordcount).
fear_percent	The percent of words associated with fear in the song (as determined by wordcount).
anticipation_percent	The percent of words associated with anticipation in the song (as determined by wordcount).
surprise_percent	The percent of words associated with surprise in the song (as determined by wordcount).
trust_percent	The percent of words associated with trust in the song (as determined by wordcount).
sadness_percent	The percent of words associated with sadness in the song (as determined by wordcount).
joy_percent	The percent of words associated with joy in the song (as determined by wordcount).
disgust_percent	The percent of words associated with disgust in the song (as determined by wordcount).
cohesiondistance	The Euclidean distance between percentages of anger, fear, surprise, sadness, joy, and disgust of each song and the percentages of the same emotions found in the news articles.

As an alternative to Twitter data, we used the news article database present in *eventregistry.org*, which allows the search of news articles by date, location, language, and the number of times the article is shared on social media. Through the use of the *Selenium* and *BeautifulSoup* Python modules, we were able to scrape the news article headers and first several sentences of the 25 most popular news articles on social

media for the week of September 18, 2016 to September 26, 2016, which is the same week the song popularity data was collected during Part 1 of the project[1]. The articles for the selected week were highly emotional (see Table 2). The words contained within the article headers and the first few sentences of each article were examined for emotional content by comparing them to the NRC Word-Emotion Association Lexicon database[2], which has categorized words from The Macquarie Thesaurus (Bernard, 1986) by the emotions they elicit (anger, fear, anticipation, surprise, trust, sadness, joy, and disgust) and sentiment (negative or positive). It is important to note that a particular word may elicit more than one emotion, but can only have a maximum or one sentiment (i.e., no sentiment is also possible). The number of times each emotion was elicited was collected and then summed across all 25 articles after multiplying a particular article's emotion counts by the number of social media shares (i.e., a weighted sum of emotions). The weighted sums were then converted to percentages based on the weighted sum of all words that were matched to an emotion.

The same emotion-counting procedure was completed for all of the lyrics for each song (both raw count and percentage), which added an additional 21 attributes to song dataset (i.e., 20 for raw count and percentages of emotions, and one for the total number of words). Lastly, a "cohesion distance" score was produced for each song. This attribute is simply the Euclidean distance between the percentages of anger, fear, surprise, sadness, joy, and disgust (i.e., the six "basic emotions"; Ekman, 1992) of each song to the same emotion percentages from the news articles. Since this attribute is a measure of distance, a cohesion score of zero indicates that the song is perfectly mapped to the emotional content of the news articles from the week while the score will be larger for songs with emotion percentages that are very different from the emotional content of the news articles[3]. Table 1 shows the full list of attributes added since the completion of Part 1.

[1] While eventregistry.org does have a Python scripts to access news articles directly, the scripts were based on Python 2 and attempts at converting the code to Python 3 (through the *2to3* and *future* modules) were generally unsuccessful. Further, attempts to scrape source code through *urllib* and *requests* were unsuccessful since the website loads news articles dynamically. Thus, the *Selenium* module and the Firefox browser were used to pull the appropriate data (though the process is not particularly fast). Since the process only needed to be completed once for a particular week, the process of scraping source code through *Selenium* was not particularly problematic.

[2] See http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm and an article on a particular application of the database in https://arxiv.org/pdf/1308.6297.pdf. Note that words were first cleaned through the replacement of contractions to their constituent words (e.g., "don't" becomes "do not"), lemmatization to their stem (e.g., "dogs" becomes "dog"), marked for negation if the word is preceded by a negation word (e.g., "not happy" should not count as a positive emotion), and with stop-words filtered out (e.g., "or", "and", and "if" are the most common words but have no emotional meaning). Note that negations prevented a word being counted for a particular emotion (e.g., "not sad" was not counted for sadness) and it was also not counted for a different "opposite" emotion (e.g., "not sad" was not counted as "joy"). This generally prevented ambiguities for what constituted an "opposite" emotion.

[3] We recognize that this attribute is conceptually strange given that we generally don't expect, for example, a week of 50% sadness to be motivate listeners to listen to a song comprised of 50% sadness words. However, this was a preliminary method of calculating a quantifiable relationship between the news articles and the song lyrics. Part 3 of the project will likely adopt a time-series method examining the changes in listening preferences for just a handful of songs over time while highlighting key emotional events that have occurred on specific days.

Table 2: News Articles

Header	Social Media Shares
'Disturbing' Helicopter Footage Shows Tulsa Police Kill Unarmed Man - NBC News	121,073
One person shot during violent Charlotte protest; officer hurt	45,307
Angelina Jolie files for divorce from Brad Pitt	45,013
US boy offers home to Syrian refugee in letter to Obama - BBC News	43,556
BREAKING NEWS: Golfing legend Arnold Palmer has died aged 87	40,443
Joss Whedon Returns with Star-Studded, Anti-Trump Video to Get Out the Vote	34,572
Miami Marlins ace Jose Fernandez dies in boating accident	26,267
Zuckerberg, Chan Pledge $3B to End Disease	23,380
Trump threatens to bring Gennifer Flowers to debate	20,595
Trump Foundation starts to look like a slush fund	20,049
What's Worrying Clinton and Trump's Campaigns Ahead of First Debate	19,563
The Latest: Doubts follow release of police shooting video	17,555
Police shoot 'active shooter' at strip mall in Houston	10,329
Israeli and Palestinian leaders clash at UN general assembly	7,784
Trump tells Netanyahu he'll recognize Jerusalem as Israel's 'undivided' capital	6,953
Mylan CEO's mother used position with education group to boost EpiPen sales nationwide	6,784
Rick Porcello's dominance has transformed the Red Sox -- and the AL East race	6,484
Apple is in talks with McLaren for a potential acquisition, report...	6,174
US authorities investigating New Jersey explosive devices - live updates	6,092
[Newsmaker] New evidence of water plumes on Jupiter's moon Europa	5,975
Theresa May urges new approach to migrant crisis at UN summit - BBC News	5,185
The iPhone 7 costs Apple more than you think	4,833
Patriots tear through Texans in Jacoby Brissett's first start	4,746
Curtis Hanson, Director of 'L.A. Confidential,' Dies at 71	4,436
Marion Cotillard confirms she is PREGNANT	4,259

Exploratory Analysis

Basic Statistical Analysis and Data Cleaning

Missing Data. Prior to exploratory analyses, the attributes were first adjusted for missing values. During Part 1 of the project, several attributes were already identified as missing values, and, among these, a smaller subset of attributes were identified as more central to tackling the data science question. These attributes were 1) duplicate songs, 2) songs with no matching lyrics, 3) songs where the lyrics text were extremely short, and 4) songs with zero duration. The counts for each of these are shown in Table 3.

Table 3: Missing Values

Description	Count
Duplicate songs	286
Lyrics are 'Not found'	487
Number of characters in lyrics is less than 35	650
Song duration is zero	283
Total Removed (not a sum of the above)	1056
Total Remaining (original = 5656)	4600

Given the fact that analyses were conducted on aggregated data (i.e., the data is generally more noisy than individual-level observations), we were generally motivated to create a "purified" dataset that is as free from anomalies as possible rather than attempt to retain as much data as possible through imputation, which had the potential of either adding noise to the attributes or decreasing the variance. Cases of duplicate songs

were removed from the dataset with little debate since these were known anomalies that were produced from the use of the Last.fm API itself (i.e., the raw data returned from the API contained duplicates). There was also little debate in the removal of cases where the lyrics text was extremely short (<35 characters without whitespaces). These cases were indicative of songs where their lyrics actually didn't exist (i.e., lyrics was "instrumental") or in a different language (and were converted to whitespaces in UTF-8). In such cases, the imputation of the attributes related to the attribute (i.e., emotion count and percentiles) was not reasonable (e.g., a song without lyrics cannot reasonably be considered the average number of sadness words in the dataset). The remaining two conditions (song duration is zero and lyrics were not found) were more problematic, since there was the possibility of applying mean imputation. However, in the end, we decided to remove these cases from the dataset as well due to 1) the relatively large number of attributes that would've needed imputation for missing lyrics (since each of the emotion counts and percentages also needed imputation) and 2) relatively low number of cases where the song duration was zero. After cases with missing values were removed, the final "cleaned" dataset was comprised of 4600 cases.

Table 4: Descriptive Statistics for Quantitative Attributes

Attribute	Mean	Median	SD
fullrank	2607.77	2568.50	1561.96
artistfreq	15.20	8.00	18.67
lyricslength	1146.52	991.00	688.27
duration	242262.00	233000.00	70810.10
listeners	309986.00	238910.00	268585.00
playcount	1981920.00	1335710.00	2066100.00
anger	3.30	2.00	4.98
fear	3.73	2.00	5.18
anticipation	4.67	3.00	5.48
surprise	2.58	1.00	3.87
trust	4.49	3.00	5.54
sadness	3.75	2.00	5.02
joy	5.35	3.00	6.85
disgust	2.44	1.00	4.24
wordcount	18.12	14.00	15.28
anger_percent	0.16	0.13	0.18
fear_percent	0.20	0.16	0.19
anticipation_percent	0.26	0.23	0.21
surprise_percent	0.14	0.10	0.16
trust_percent	0.24	0.21	0.20
sadness_percent	0.20	0.17	0.19
joy_percent	0.28	0.25	0.23
disgust_percent	0.12	0.08	0.15
cohesiondistance	0.57	0.53	0.22

Descriptive Statistics. After the removal of missing data, descriptive statistics were produced for all variables were analyses could be reasonably conducted (see Table 4). Among these statistics, there are several that are of particular interest. On average, the emotional content of the songs appear to be predominantly joy, followed by trust and anticipation (i.e., predominantly positive emotions). The least frequent emotion appears to be disgust. On average the number of listeners in the current dataset is

approximately 310,000 and the average play-count is about 2 million. In short, these songs are listened to very frequently by a fairly large number of people.

Based on the frequency tables in Table 5, we can see that the most popular tags are "rock", "indie", and "alternative". Not surprisingly, indie and alternative are typically sub-genres of rock so their common frequencies are unsurprising. It is somewhat surprising that the songs categorized as "pop" only comprise about 18% and "hiphop" comprising only about 16% of the songs in the dataset given that these two genres traditionally comprise the majority of the highest ranking songs. The most frequent artists in our dataset are Radiohead, Kanye West, Drake, and The Beatles. The mainstay of both The Beatles and Beach House are somewhat surprising given their general absence in the public sphere.

Table 5: The 10 Most Common Tags and Artists

Tags	Count		Artists	Count
rock	1674		Radiohead	83
indie	1359		Kanye West	76
alternative	1212		Drake	58
pop	829		The Beatles	51
hiphop	755		Beach House	40
indierock	641		Lana Del Rey	39
electronic	641		Arcade Fire	38
alternativerock	507		Kendrick Lamar	37
classicrock	484		Led Zeppelin	37
femalevocalists	402		The National	33

Outliers. There were no obvious outliers in the data at this stage of analyses, as some of the values that could be classified as an outlier (e.g., zero duration or no lyrics) have generally already been accounted for during the missing data stage. There are no obvious distributional outliers given that we currently have no reason to expect, for example, songs that are particularly short are indicative of poor regression fit. (Of course, this does not exclude the possibility of outliers appearing in later stages of the project.)

However, if there were outliers present that were more obviously problematic, there are several approaches that can be implemented to identify and deal with the outlier. Several possible statistical approaches include the use of the inter-quartile range (IQR) which is simply the difference between the 3rd quartile and the 1st quartile. Points that are lower than 1.5 times the IQR or higher than 1.5 times the IQR can be designated as a univariate outlier. Another possible way to detect possible outliers is through *k* nearest neighbors (KNN) and determining the *n* number of points in which the k^{th} nearest neighbors are largest (Ramaswamy et al., 2000). A density-based approach is the Local Outlying Factor (LOF; Breunig et al., 2000) which determines the average density of a particular point and the density of its nearest neighbors. Common ways of dealing with outliers is by treating the point essentially the same as missing data (and using mean imputation, regression smoothing, or simple removal from the dataset), the changing of statistical or machine learning model (e.g., ordinary least squares regression to Poisson regression if variance appears to increase as the outcome variable increases), or by simply including the outlier in the analysis.

Attribute Binning. Binning was applied to the attribute *cohesiondistance* for two main purposes. First, we recognize how right-skewed the distribution of *cohesiondistance* is as based on its histogram (see Figure 2). Thus, binning is used in this context to smooth out the extreme values that are occurring and prevent the any analyses from being highly influenced in the same manner. Secondly, binning was applied to *cohesiondistance* in order to discretize it for further analysis with classifiers. Binning was applied to split the attribute into three levels: high cohesion (0 to 0.5), medium cohesion (0.5 to 1), and low cohesion (1 and above). These are intuitive binning borders that are neither equal-density nor equal-distance. The first five rows of the newly binned are shown in Table 6.

Table 6: Binned Cohesion Distance

cohesiondistance	cohesiondistance_binned
0.575	(0.5, 1]
0.824	(0.5, 1]
0.676	(0.5, 1]
0.277	[0, 0.5]
0.446	[0, 0.5]

Histograms and Correlations

Histograms were generated for three attributes: *anger_percent*, *sadness_percent*, and *joy_percent*. Recall that these attributes pertain to the percentage of words designated for emotional content (anger, sadness, and joy) among all matched words as based on the NRC database. These three attributes will be explored in greater detail in the following sections since these emotions can be seen as thematically common in song lyrics (i.e., it is not difficult to find songs pertaining to anger, sadness, or happy, but somewhat more difficult to find songs pertaining to fear or disgust). Further, it was important to examine these three particular attributes for their clustering characteristics for possible discretizing.

As shown in Figure 1, all three histograms were heavily right-skewed, which generally indicates that lower percentages were more common than higher percentages. These plots generally suggest that songs that are dominated by a particular emotion are generally quite rare, and a more even mix of emotions is more common. A correlation matrix was generated for the same three attributes. As shown on Table 7, the Pearson correlation between *anger_percent* and *sadness_percent* was approximately 0.40, which is small but not trivial. Thus, there is some evidence to suggest that the negative emotions tend to occur simultaneously (and, hence, more likely to form a cluster). In contrast, *joy_percent* showed a very small negative correlation with the other two attributes, which provides some evidence that positive emotions do not co-occur with negative emotions.

Figure 1: Histograms

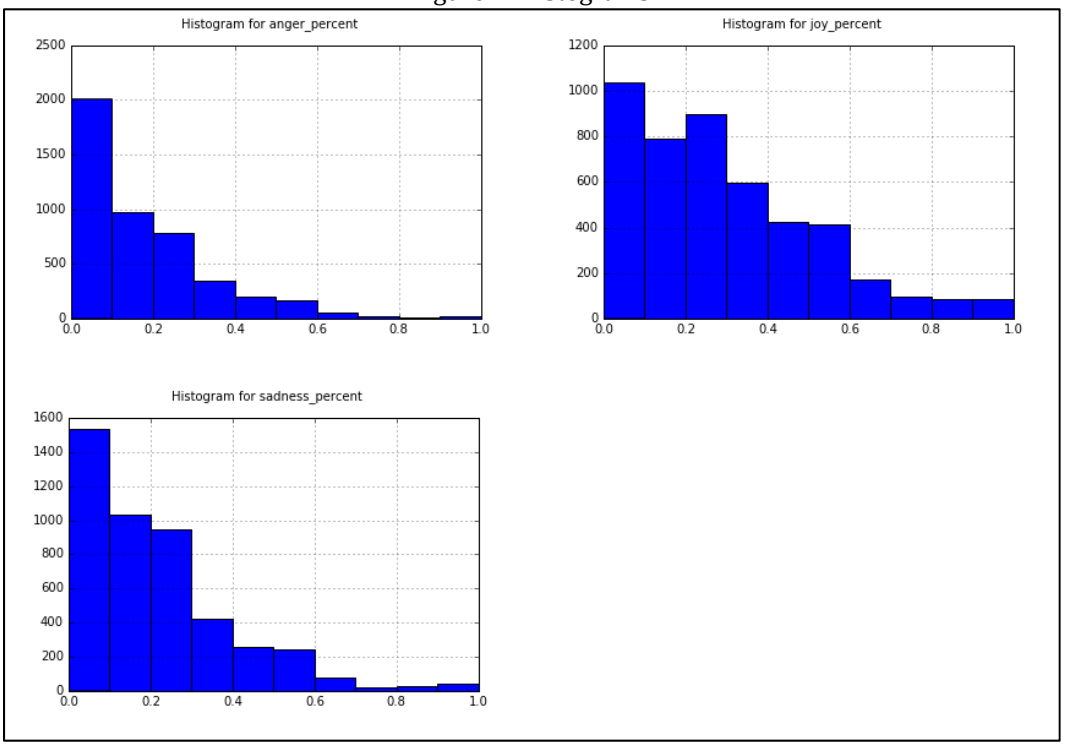
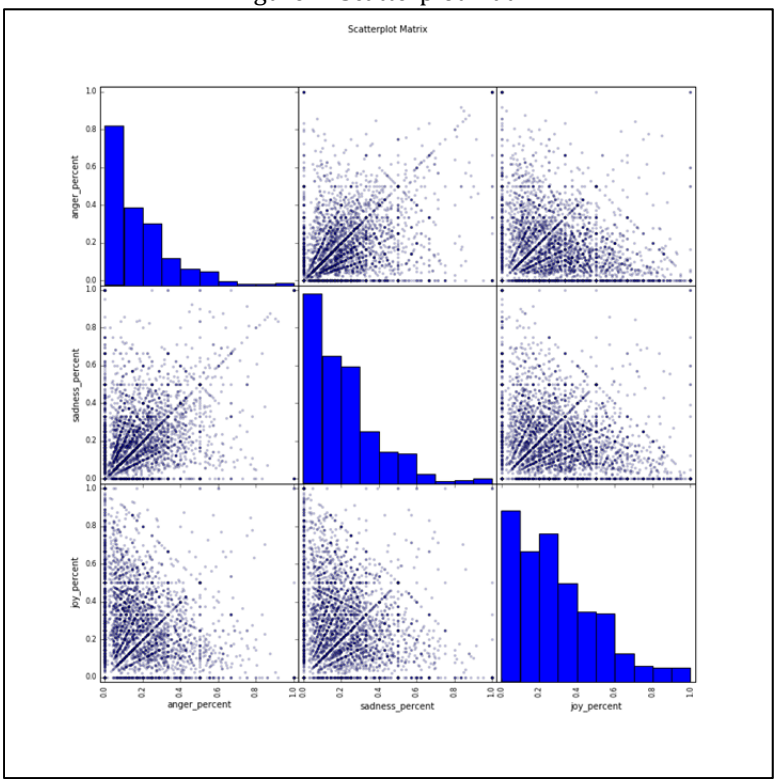


Table 7: Correlation Matrix

	anger_percent	sadness_percent	joy_percent
anger_percent	1	0.4037	-0.1659
sadness_percent	0.4037	1	-0.1587
joy_percent	-0.1659	-0.1587	1

A scatterplot matrix of the same three variables was generated for the same three variables. Based on Figure 2, the points are generally concentrated on the bottom-left portions of each of the plots. The higher positive correlation between *anger_percent* and *sadness_percent* is evident from what appears to be a near straight line of points extending from the bottom-left to the upper-right of these plots. This straight line is indicative of songs where the percentage of anger and sadness are exactly the same. In contrast, we see that for plots between *joy_percent* and *anger_percent*, and also *joy_percent with sadness_percent*, a clear negative trend is shown as a straight line of points extending from the upper-left to the lower-right of these plots. These are indicative of occasions where the percent of joy increases while the percent of sadness or anger decreases (or vice versa). These results further suggest a clear separation between the positive and negative emotions.

Figure 2: Scatterplot Matrix



Cluster Analysis

The following section will detail the application of three different clustering algorithms on the same three variables explored above (*anger_percent*, *sadness_percent*, and *joy_percent*). Since these attributes were already percentages that ranged from 0 to 1, no further standardization was deemed necessary.

K-Means Clustering. K-means clustering is one of the simplest clustering algorithms which attempts to determine how points are grouped through the use of arbitrary "centroid" points and determining the distance of all data points to these centroids. Quite simply, the points are assigned to their closest centroid (in Euclidean distance), and the centroid is then recalculated based on the mean of the data points that were assigned the centroid (and repeating the process until the centroid locations stop changing). The cluster centroids can then be interpreted as the representative point for each of the clusters. The `cluster.KMeans()` method from the *sklearn* Python module was utilized.

As the number of clusters (*K*) needs to be pre-defined, the optimal number of clusters was first explored through the use of silhouette scores[4]. The silhouette score of a particular point is calculated by determining the average distance between the point and all other points within the same cluster (*A*), and the average distance between the point and the points of the closest neighboring cluster (*B*). The calculation for the silhouette score for point *i* is:

[4] See http://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

$$s_i = \frac{(B - A)}{\max(A, B)}$$

The silhouette score of the entire model is simply the average of the silhouette scores of all the points, with values closer to 1 indicating better cluster cohesion. We utilized the `metrics.silhouette_score()` method available in the *sklearn* Python module, and was repeated for varying values of *K*. For the application of the K-means clustering algorithm in the current three attributes, the *K*'s were set to integer values from 2 to 8. The resulting silhouette scores are shown on Table 8.

Table 8: K-Means Silhouette Scores

K	Silhouette Score
2	0.304
3	0.308
4	0.266
5	0.277
6	0.293
7	0.297
9	0.281

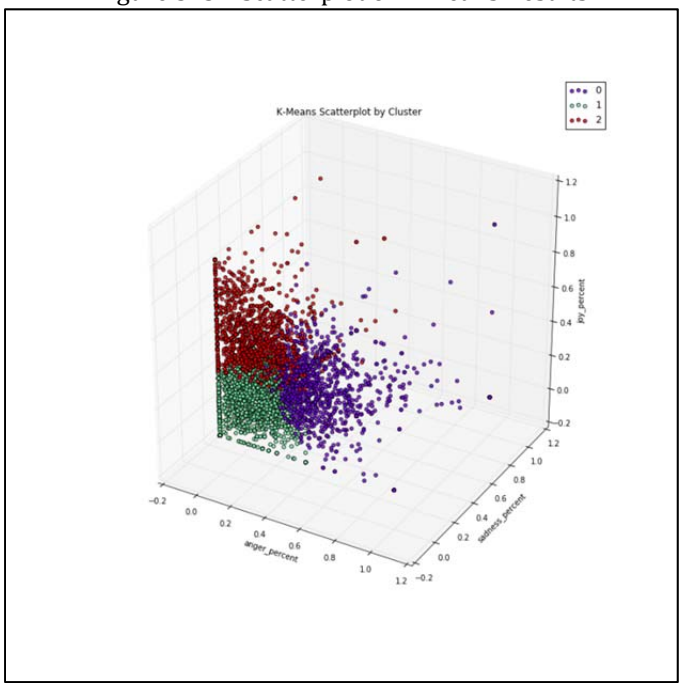
Table 9: K-Means Centroids by Cluster Label

Cluster	anger_percent	sadness_percent	joy_percent	Counts
0	0.333	0.408	0.190	1252
1	0.102	0.119	0.162	2074
2	0.099	0.137	0.573	1274

As shown on Table 8, the highest average silhouette score was achieved for *K*=3, thus, the K-means results for three clusters will be presented. The final centroid positions are shown for each of the three clusters is shown in Table 9. The centroid positions suggest that the clusters are separated by high percentages of anger and sadness (cluster 0), high percentages of joy (cluster 2), and a third cluster that appears to have nominal percentages in all three attributes (cluster 1). Based on the cluster frequencies, we can see that about 45% of the sample belongs in cluster 1 (i.e., the nominal cluster), while virtually equal portions of the remaining sample are assigned to both cluster 0 (i.e., the negative emotion cluster) and cluster 2 (the positive emotion cluster).

A 3D scatterplot of the cluster pattern and labels is shown in Figure 3. This plot gives insight into the varying levels of density for the three clusters. Specifically, we can see that the density of cluster 1 (i.e., the nominal cluster) is fairly high given the number of points in the relatively small space. In contrast, the remaining two clusters appear to be far more dispersed, particularly as the points become increasingly sparse near the extreme ends of the plot.

Figure 3: 3D Scatterplot of K-Means Results



Ward's Clustering. To follow up the results of the K-means clustering algorithm, the Ward hierarchical agglomerative clustering algorithm was also applied to the three attributes. In contrast to K-means, which uses partitioned clustering (i.e., the clusters never overlap), the Ward algorithm applies nested clustering in order to determine cluster labels. In a nutshell, the Ward algorithm first treats all individual points as individual clusters. Any two closest clusters are then combined to form a bigger cluster. This process is then repeated until one giant cluster is formed or some other stopping criterion is applied (e.g., the number of final clusters). We define "closeness" between two cluster's using Ward's minimum variance method, which essentially attempt to match two clusters which lead to the minimum total within-cluster variance after the cluster merging is completed[5]. The `cluster.AgglomerativeClustering()` method from the *sklearn* Python module was utilized.

In order to determine a stopping criterion for Ward's clustering algorithm, the same silhouette score method from K-means was applied. As before, the number of clusters varied between integer values of 2 through 8, and the average silhouette score was determined for each cluster count. The resulting average silhouette scores are shown in Table 10. As shown in Table 10, the largest average silhouette score was for *K*=3, thus, the Ward's clustering results for three clusters will be presented. Unlike K-means, the results of agglomerative clustering algorithms typically do not return information about centroids (although Ward's method does use centroids in the determination of minimum variance), so grouped means for each of the three attributes by cluster label is shown in Table 11. As the means suggest, the clustering patterns from Ward's clustering were generally very similar to the results of the K-means algorithm, with one of the clusters

[5] See https://en.wikipedia.org/wiki/Ward%27s_method and http://www-users.cs.umn.edu/~kumar/dmbook/ch8.pdf

(cluster 1) showing high mean percentages of anger and sadness relative to the mean percentage of joy, one cluster (cluster 2) showing a very high mean percentages for joy compared to mean percentages for anger and sadness, and a nominal cluster (cluster 0) that possess low mean percentages for all three attributes.

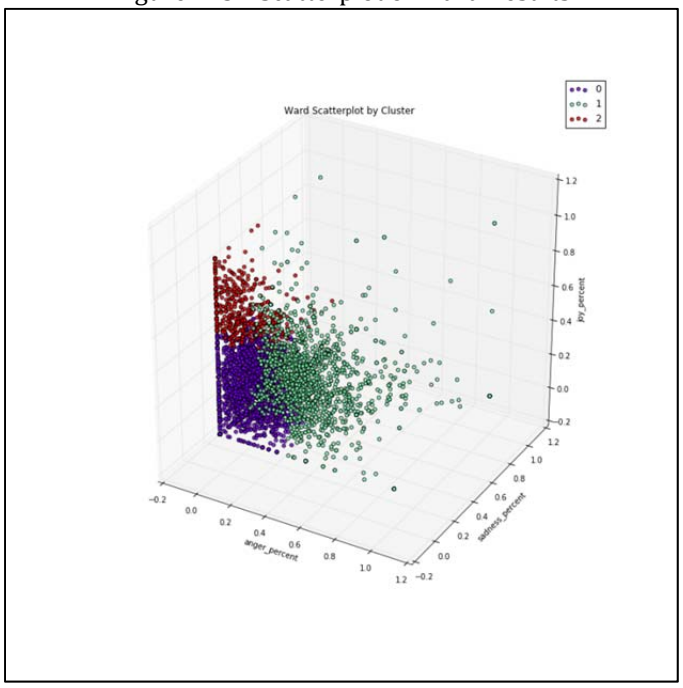
Table 10: Ward Silhouette Scores

K	Silhouette Score
2	0.261
3	0.268
4	0.207
5	0.217
6	0.227
7	0.225
9	0.234

Table 11: Ward Group Means by Cluster Label

Cluster	anger_percent	sadness_percent	joy_percent	Counts
0	0.092	0.114	0.225	2455
1	0.308	0.376	0.224	1590
2	0.074	0.098	0.709	555

Figure 4: 3D Scatterplot of Ward Results



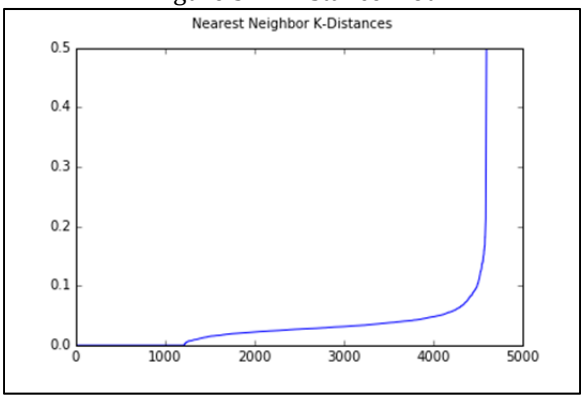
A 3D scatterplot of the cluster pattern and labels is shown in Figure 4. In general, this plot is quite similar to the scatterplot generated by the K-means algorithm with the exception that cluster 0 (i.e., the negative emotions cluster) appears to be the dominant cluster with more than half the points being in its membership (as opposed to the nominal cluster being dominant in K-means).

DBSCAN. The final clustering algorithm applied to our data is DBSCAN, which is a density-based clustering method. Unlike the previous two methods, DBSCAN categorizes points as either a core point (i.e.,

within the interior of a dense cluster), border point (i.e., near the edge of the dense cluster), or noise point i.e., neither a core nor a border point) based on the specified radius (known as *Eps*) and the minimum number of points to be considered a core point (*MinPts*)[6]. Essentially, a point is considered a core point if it has a sufficient number of other points (as determined by *MinPts*) within its *Eps* radius. Points that are not core points but are within the radius of a core point are categorized as border points. The method `cluster.DBSCAN()` from the *sklearn* Python module was utilized.

Since the *MinPts* and *Eps* parameters need to be assigned to the clustering program beforehand, these were estimated using the *K* nearest neighbors (KNN) algorithm[7]. This procedure entails the use of the KNN algorithm in order to determine the K^{th} nearest neighbor distances after sorting from smallest to largest. We should observe a sharp increase in the distance sizes (i.e., the "knee" of the plot) which is essentially the estimated of the *Eps* (i.e., the *Eps* should be large enough to where clusters are distinguished from the high noise outliers while also being small enough not to collapse into a single cluster by default). The value of *K* (which is analogous to *MinPts* in this context) was determined to be the number of attributes plus one[8].The resulting "knee plot" of the 4^{th} nearest neighbor is shown in Figure 5. The plot shows a sudden increase in distances after a distance of approximately 0.08, so this was determined as our *Eps* estimate. As such, the results of DBSCAN with an *Eps* of 0.08 and *MinPts* of 4 will be presented.

Figure 5: K-Distance Plot



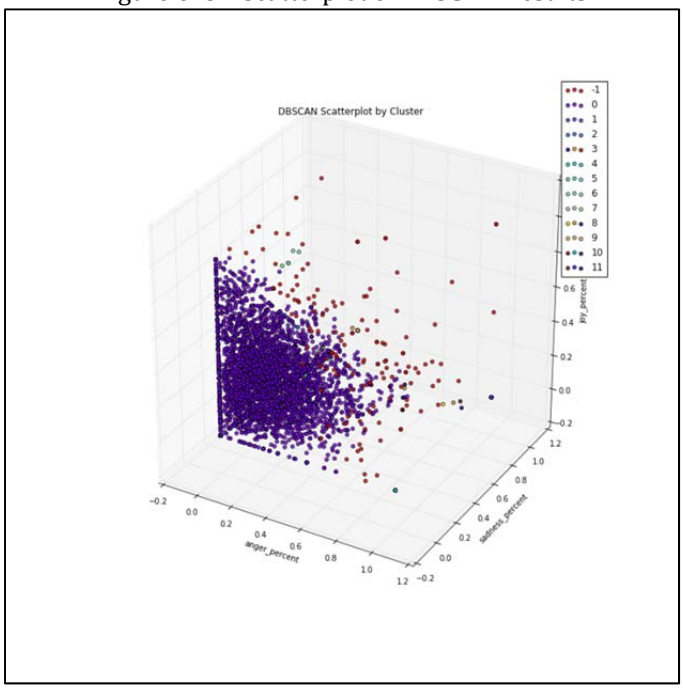
First, grouped means for each of the three attributes by cluster label is shown in Table 12. Unlike both K-means and Ward's clustering, the DBSCAN algorithm determined a total of 13 clusters (including a noise cluster) based on the three attributes. However, the vast majority (95%) of the points belongs to a single cluster (cluster 0) and an additional 139 points were categorized as noise points (cluster -1). Each of the remaining substantive clusters had a point count of less than 20. The dominant cluster (cluster 0) had grouped mean values that were generally quite low for the negative emotions (sadness and anger) and somewhat larger for the percentage of joy. The noise cluster means (cluster -1) can be characterized by large values for all three attributes.

[6] See http://www-users.cs.umn.edu/~kumar/dmbook/ch8.pdf
[7] This is the method recommended in http://www-users.cs.umn.edu/~kumar/dmbook/ch8.pdf
[8] This is known as a rule of thumb for determining *minpts* (see https://en.wikipedia.org/wiki/DBSCAN)

Table 12: DBSCAN Group Means by Cluster Label

Cluster	anger_percent	sadness_percent	joy_percent	Counts
-1	0.486	0.511	0.414	139
0	0.148	0.185	0.281	4382
1	1.000	1.000	0.000	13
2	0.000	0.666	0.334	8
3	0.532	0.853	0.000	4
4	1.000	0.000	0.000	8
5	0.500	0.000	0.485	5
6	0.011	0.668	0.693	5
7	0.000	0.997	0.000	15
8	0.873	0.840	0.006	4
9	0.498	0.498	0.502	9
10	0.579	0.083	0.594	4
11	0.664	0.677	0.031	4

Figure 6: 3D Scatterplot of DBSCAN Results



A 3D scatterplot of the cluster pattern and labels is shown in Figure 6. This plot is very different from the previous 3D scatterplots due the dominant single cluster than essentially spans the range from 0 to about 0.5 for all three attributes. Despite the difference in cluster labeling, this result was not surprising given the gradual dispersion of points as the percentages drifted toward 1. While this plot is interesting, the results unfortunately provide little information about how these clusters can be categorized (as such, the K-means or Ward's clustering results will be used for future analyses).

Association Rules

Association rules analysis was conducted on the *toptags* attribute, which is comprised of the top most popular tags that users of Last.fm have used for each of the songs. Tags are most often the genre of the song,

though it can also include characteristics of the song (see Table 5, which had "female vocalist" as the 10th most common tag). We were primarily interested in association rules to predict a particular tag given the occurrence of other tags. For illustrative purposes, we are currently interested in 1) the probability of a song being tagged as "rock" given that it is also tagged as "indie", 2) the probability of a song being tagged as "alternative" given that it is also tagged as both "rock" and "indie", and 3) the probability of a song being tagged as "pop" given that the song is also tagged as "female vocalist". These questions will help to determine in future analyses if there exists redundancies with these tags, and, thus, lead to the use of the fewer tags that capture a similar amount of information as predictors.

The association rules analysis was conducted using a custom `associationrules` class in Python. Constructing an `associationrules` object requires a list of lists (e.g., a list of items per row) and a minimum support (a positive integer for how many times item sets must occur to be included in the analysis). Based on these two arguments, the constructor will produce 1) a list of items with a support greater than the specified amount, and 2) an occurrence matrix (comprised of Boolean values) with columns being the items that are valid by the minimum support criterion. This utilizes the *apriori* principle in the sense that if the singleton items do not occur frequently, we also know that any set containing the item will also not occur frequently[9]. The `confidence()` method then can be applied to the `associationrules` object which takes in the predictor item set and predicted item set arguments (both being lists), and produces a confidence value (which is the number of simultaneous occurrences of all items in both predictor and predicted divided by the number of occurrences for the predictor list only). Lastly, the `associationcombtest()` method can then be used to compute all possible combinations of the predictor item set for set sizes from 1 through *maxitems* (which must be specified) and have the confidence computed predicting each single item that fulfills the minimum support and the minimum confidence. In general, this program is not particularly fast.

[9] See Gates, Ami (2016) ANLY-501 Lecture 5

Table 7: All Association Rules with Confidence ≥ .20 (Minimum Support ≥ 400)

Association Rule	Confidence	Numer.	Denom.
['rock'] --> indie	0.306	513	1674
['rock'] --> alternative	0.491	822	1674
['rock'] --> indierock	0.224	375	1674
['rock'] --> alternativerock	0.269	451	1674
['rock'] --> classicrock	0.268	448	1674
['indie'] --> rock	0.377	513	1359
['indie'] --> alternative	0.442	600	1359
['indie'] --> indierock	0.411	559	1359
['alternative'] --> rock	0.678	822	1212
['alternative'] --> indie	0.495	600	1212
['alternative'] --> indierock	0.281	340	1212
['alternative'] --> alternativerock	0.346	419	1212
['pop'] --> rock	0.261	216	829
['pop'] --> femalevocalists	0.228	189	829
['indierock'] --> rock	0.585	375	641
['indierock'] --> indie	0.872	559	641
['indierock'] --> alternative	0.530	340	641
['electronic'] --> indie	0.303	194	641
['electronic'] --> pop	0.209	134	641
['alternativerock'] --> rock	0.890	451	507
['alternativerock'] --> indie	0.308	156	507
['alternativerock'] --> alternative	0.826	419	507
['alternativerock'] --> indierock	0.203	103	507
['classicrock'] --> rock	0.926	448	484
['classicrock'] --> pop	0.223	108	484
['femalevocalists'] --> indie	0.311	125	402
['femalevocalists'] --> pop	0.470	189	402
['rock', 'indie'] --> alternative	0.686	352	513
['rock', 'indie'] --> indierock	0.661	339	513
['rock', 'indie'] --> alternativerock	0.265	136	513
['rock', 'alternative'] --> indie	0.428	352	822
['rock', 'alternative'] --> indierock	0.291	239	822
['rock', 'alternative'] --> alternativerock	0.474	390	822
['rock', 'alternativerock'] --> indie	0.302	136	451
['rock', 'alternativerock'] --> alternative	0.865	390	451
['rock', 'classicrock'] --> pop	0.217	97	448
['indie', 'alternative'] --> rock	0.587	352	600
['indie', 'alternative'] --> indierock	0.543	326	600
['indie', 'alternative'] --> alternativerock	0.202	121	600
['indie', 'indierock'] --> rock	0.606	339	559
['indie', 'indierock'] --> alternative	0.583	326	559
['alternative', 'alternativerock'] --> rock	0.931	390	419
['alternative', 'alternativerock'] --> indie	0.289	121	419

The `associationcombtest()` method was utilized with the minimum confidence was set at 0.20 (thus, any association rule must have confidence ≥ 0.20 to be listed). Three different levels of minimum support values were selected: 400, 500, and 600. The resulting association rules after completing the `associationcombtest()` method for each of the three minimum confidence levels are shown in Tables 7 through 9. We can see from the denominator column that the most frequent singleton item set is “rock” (with a frequency of 1674) while the most frequent double item set is “rock” and “alternative” (with a frequency of 513). There are no item sets with more than two items that fulfill the minimum support levels we have set. In

regard to the singleton frequencies, these findings are somewhat surprising, given that some of the highest ranking songs are typically tagged as “hiphop” or “pop”. The double item lists are generally combinations of “rock” and sub-genres of rock (e.g., “alternative” and “indie”), so the relatively large frequencies observed are not particularly surprising.

Table 8: All Association Rules with Confidence ≥ .20 (Minimum Support ≥ 500)

Association Rule	Confidence	Numer.	Denom.
['rock'] --> indie	0.306	513	1674
['rock'] --> alternative	0.491	822	1674
['rock'] --> indierock	0.224	375	1674
['rock'] --> alternativerock	0.269	451	1674
['indie'] --> rock	0.377	513	1359
['indie'] --> alternative	0.442	600	1359
['indie'] --> indierock	0.411	559	1359
['alternative'] --> rock	0.678	822	1212
['alternative'] --> indie	0.495	600	1212
['alternative'] --> indierock	0.281	340	1212
['alternative'] --> alternativerock	0.346	419	1212
['pop'] --> rock	0.261	216	829
['indierock'] --> rock	0.585	375	641
['indierock'] --> indie	0.872	559	641
['indierock'] --> alternative	0.530	340	641
['electronic'] --> indie	0.303	194	641
['electronic'] --> pop	0.209	134	641
['alternativerock'] --> rock	0.890	451	507
['alternativerock'] --> indie	0.308	156	507
['alternativerock'] --> alternative	0.826	419	507
['alternativerock'] --> indierock	0.203	103	507
['rock', 'indie'] --> alternative	0.686	352	513
['rock', 'indie'] --> indierock	0.661	339	513
['rock', 'indie'] --> alternativerock	0.265	136	513
['rock', 'alternative'] --> indie	0.428	352	822
['rock', 'alternative'] --> indierock	0.291	239	822
['rock', 'alternative'] --> alternativerock	0.474	390	822
['indie', 'alternative'] --> rock	0.587	352	600
['indie', 'alternative'] --> indierock	0.543	326	600
['indie', 'alternative'] --> alternativerock	0.202	121	600
['indie', 'indierock'] --> rock	0.606	339	559
['indie', 'indierock'] --> alternative	0.583	326	559

We can see that for Question 1, the probability of a song being tagged as “rock” given that it is also tagged as “indie” is 0.377 for all three minimum support levels. For Question 2, the probability of a song being tagged as “alternative” given that it is also tagged as both “rock” and “indie” is 0.686, though this is only shown on Tables 7 and 8 (based on the support of 513 for the double set “rock” and “indie”). For Question 3, the probability of a song being tagged as “pop” given that the song is also tagged as “female vocalist” is 0.223 but only for Table 7 (which suggests that the support of “female vocalist” is between 400 and 500). The largest confidence values were achieved for the prediction of “rock” given that the song is also tagged as “classic rock” (0.926) and for the prediction of “rock” given that the song is also tagged as both “alternative” and “alternativerock”. As the minimum support value increases, the number of high confidence association

rules has a tendency to become lower, which indicates that most of the redundancies in the tagging behaviors occur in microcosms that are not likely to generalize to the entire dataset.

Table 9: All Association Rules with Confidence ≥ .20 (Minimum Support ≥ 600)

Association Rule	Confidence	Numer.	Denom.
['rock'] --> indie	0.306	513	1674
['rock'] --> alternative	0.491	822	1674
['rock'] --> indierock	0.224	375	1674
['indie'] --> rock	0.377	513	1359
['indie'] --> alternative	0.442	600	1359
['indie'] --> indierock	0.411	559	1359
['alternative'] --> rock	0.678	822	1212
['alternative'] --> indie	0.495	600	1212
['alternative'] --> indierock	0.281	340	1212
['pop'] --> rock	0.261	216	829
['indierock'] --> rock	0.585	375	641
['indierock'] --> indie	0.872	559	641
['indierock'] --> alternative	0.530	340	641
['electronic'] --> indie	0.303	194	641
['electronic'] --> pop	0.209	134	641
['rock', 'alternative'] --> indie	0.428	352	822
['rock', 'alternative'] --> indierock	0.291	239	822
['indie', 'alternative'] --> rock	0.587	352	600
['indie', 'alternative'] --> indierock	0.543	326	600

Predictive Analysis

The following section will provide detail about specific predictive analyses and general hypothesis tests that were conducted using the data at hand. Specifically, there are 3 hypotheses we want to test:

1) The mean playcount is different between the two clusters (positive and negative) suggested by the K-means clustering algorithm.
2) Song playcount can be predicted by cohesion distance and song duration.
3) Binned cohesion distance (3-levels) can be predicted by song duration, number of listeners, playcount, and song rank.

Note that Question 2 involves a continuous outcome variable, while Question 3, while similar, involves a categorical outcome variable. Thus, these two questions will need to be tackled using different techniques.

Hypothesis 1. For the first hypothesis, we are investigating whether or not average song playcount is different between the two clusters we observed during K-means clustering (i.e., cluster 0 and cluster 2, for negative and positive emotions, respectively). For this particular test, we test the null hypothesis is that the means of playcount are not different between the two clusters (positive and negative) versus the alternative hypothesis that the means of playcount are different between the two clusters. Since the analysis does not perform a prediction, there was no need to split the sample by training and test cases. This can be achieved fairly easily through the independent samples *t*-test. Given the large sample size, we believe it is reasonable to assume the central limit theorem and perform the parametric test. In Python, this can be completed using the

`ttest_ind()` function from the module *scipy.stats*. The following line will complete the test given two lists (*var1* and *var2*):

```
print(ttest_ind(var1, var2, axis = 0, equal_var = False))
```

Since we currently have no reason to expect the variances between the two independent samples are equal, setting the parameter `equal_var=False` will perform Welch's *t*-test (instead of the standard pooled variance method) which adjusts for the heterogeneity of variances. The means and standard deviations by cluster are shown in Table 10. As shown on the table, the means are actually very similar between clusters. Accordingly, the *t*-test was found to be non-significant (t=0.700, p=0.48). These results suggest that the means between the two clusters are not different which does not support our initial hypothesis.

Table 10: Means by Cluster

	Mean	SD
Cluster 0	2024967.19	2070218.38
Cluster 2	1972815.77	2085184.41

Hypothesis 2. The second hypothesis deals with the prediction of a single continuous variable (playcount) by two other continuous variables (cohesion distance and song duration). Given the relative simplicity of the data, the relatively simple technique of ordinary least squares (OLS) regression was applied to these attributes. In a nutshell, OLS regression attempts to fit a single line that passes through the data that minimizes the sum of squares errors between the outcome and predictor attributes[10]. The null hypothesis of this particular analysis involves two main components. For the overall model, we test the null hypothesis that all of the beta weights (i.e., the predictor slopes) are zero versus the alternative that at least one of the beta weights is not zero using the model *F*-test. For the contribution of the individual predictors, the *t*-test, which tests the null hypothesis that a given beta weight is zero (versus the alternative hypothesis that the weight is not zero). The procedure was completed using the `ols()` function in the *statsmodels.formula.api* module. The following script will perform the regression analysis:

```
Y = 'playcount'
X = ['cohesiondistance', 'duration']
data = pd.DataFrame([Y] + X)
model = ols("Y ~ X", data).fit()
print(model.summary())
```

While the analysis could be implemented after splitting the sample by training and test cases (and comparing the *R*-squared values between the two sets of data after fitting the parameters of the training cases into the test cases), we decided to forego this procedure given that 1) the regression analysis is parametric (i.e., the errors are expected to follow a normal distribution), and 2) there was virtually no risk of overfitting due to

[10] See http://statsmodels.sourceforge.net/devel/generated/statsmodels.regression.linear_model.OLS.html

the relatively low number of predictors (which is the main benefit of splitting the sample in this context). The results of the regression model are shown in Table 11. The resulting *F*-test was found to be significant (F=4.050, p<.05), although the amount of total variance explained was very low (i.e., the R-squared was only 0.002). For the individual attributes, cohesion distance was non-significant (B=1.32e+05, t=0.952, p=0.341) while song duration was found to be a significant predictor of playcount (B=1.1916, t=2.760, p<.01). This suggests that for every millisecond a song is longer, the playcount is expected to increase by 1.19. These results show limited support Hypothesis 2.

Table 11: OLS Regression Results

	Coef	SE	t	P>|t|
Intercept	1.618e+06	1.4e+05	11.532	0.000
Cohesion Distance	1.32e+05	1.39e+05	0.952	0.341
Duration	1.1916	0.432	2.760	0.006

Hypothesis 3. For this particular hypothesis, we aim to determine whether or not a set of predictor attributes (song duration, number of listeners, playcount, and song rank) can reliably predict the category levels of cohesion distance, which has been binned for analysis during exploratory analyses. Since this is a predictive categorization task, five different machine learning algorithms were applied to the same set of attributes in order to determine if there are differences in performance between the methods. For all five (supervised) machine learning algorithms completed their initial fitting task on training data (70%) and the prediction based on the fitted model was applied to the test data (30%). An accuracy score was assigned to each algorithm based on how many of the test categories were correctly predicted by the fitted training data. This was divided by the total number of test cases to produce an accuracy percentage. In addition, a chi-square test of goodness-of-fit was produced since we know the expected frequency of each classification from the test dataset as compared to the predicted frequency from each classifier[11]. The chi-square test of goodness-of-fit tests the null hypothesis that the frequencies observed in each class that is predicted (in the test data after fitting the model using training data) is consistent with the frequencies observed in the actual test outcomes. The alternative hypothesis is that the two sets of frequencies are from different distributions, thus, a significant *p*-value at the 5% level was indicative of poor performance from the classification algorithm[12]. While there are 3 different levels present in the binned cohesion distance attribute, the null error rate[13] of the classification procedure was actually 51.8% (i.e., the percentage of the highest frequency classification in the binned cohesion distance, which was the interval from 0.5 to 1.0). This represents the expected percentage of correct classification given that just the most frequent class is selected every time. Thus, a classification model that can show an accuracy score substantively greater than 51.8% was indicative of a good classification model.

[11] This method was found to be viable in http://mephisto.unige.ch/pub/publications/gr/Ritschard_Zighed_fit_ismis03.pdf
[12] We recognize that observing similar frequencies of each category between the test and predicted outcomes is a necessary but not sufficient indicator of a good prediction. For example, it is possible for the classification to be function poorly yet retain the expected frequencies (as will be seen the decision tree classifier). However, we felt that this test functioned well as a preliminary look into the performance of the classifier.
[13] See http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/

The first algorithm is the support vector machine (SVM). The algorithm works by attempting to separate data points in space by a linear hyperplane (e.g., a line splitting 2 dimensional points on a graph) which maximizes the distance between the hyperplane and data points near the hyperplane[14]. The script to complete the SVM using default settings (i.e., RBF kernel) is the following:

```
test_size = 0.30
seed = 7
X_train, X_validate, Y_train, Y_validate =
       cross_validation.train_test_split(X, Y,
       test_size=test_size, random_state = seed)
model = SVC()
model.fit(X_train, Y_train)
predictions = model.predict(X_validate)
print(accuracy_score(Y_validate, predictions, normalize=True))
```

In a nutshell, the script first divides the data into training and test cases. The `SVC()` object is constructed and the training data is fit to it. The `SVC()` object now contains the data and the parameters generated from fitting the training data, thus, these parameters are used to make predictions using the test data. The actual outcome of the test data and the predicted outcomes are now compared and an accuracy score is produced. For this particular model, the accuracy score was 0.520, which is effectively the same as the null error rate. Further, the chi-square goodness-of-fit test was found to be highly significant (i.e., p-value was effectively zero). Thus, these results do not support Hypothesis 3.

The second algorithm is the Naïve Bayes classifier (NB). The algorithm works by attempting to maximize what is known as the posterior probability of a given classification (i.e., $P(C|X)$). This follows from Bayes rule in the following fashion (where C is the classification and X is the vector of predictors)[15]:

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

Assuming complete independence between the predictors in X, the equation reduces to:

$$P(C|X) = \frac{P(X_1|C)P(X_2|C)\dots P(X_n|C)P(C)}{P(X_1)P(X_2)\dots P(X_n)}$$

We also recognize that in attempting to maximize the posterior probability the denominator does not contribute at all. Thus, this reduces the equation further into:

$$P(C|X) \propto P(X_1|C)P(X_2|C)\dots P(X_n|C)P(C)$$

[14] See Gates, Ami (2016) ANLY-501 Lecture 7
[15] See Gates, Ami (2016) ANLY-501 Lecture 7

For any new set of predictors (i.e., the test data), a classification with the highest posterior probability is assigned. In practice, this means performing the above equation for each possible classification outcome that is possible. If the predictor attributes are categorical, the probabilities can be determined easily by determining the number of times a particular category occurs in the training data. In the case with the current data, since the predictors are continuous, each of the $P(X_i|C)$ probabilities are determined by assuming the attribute is normally distributed (i.e., thus, the probability is determined by the normal cumulative distribution function with mean and variance provided by the training data). Applying this model in Python is fairly simple, as it requires only a single line change from the previous SVM script (i.e., `model=SVC()` is replaced with `model=GaussianNB()`). The accuracy score resulting from applying this model was 0.525, which is also nearly the same as the null error rate. Further, the chi-square goodness-of-fit test was also highly significant (i.e., *p*-value was effectively zero). As such, these results do not support Hypothesis 3.

The third algorithm is the decision tree classifier (DT). The algorithm works by using the predictor attributes to split the data into progressively purer subsets (i.e., purer in the sense that a particular subset contains all or a majority of a particular classification rather than an even mix of all classifications)[16]. This occurs in steps which utilize a single attribute and splitting the data based on some determination of purity (e.g., the Gini index). The split (unless it is completely pure already) is then split by another attribute to form an even purer subset. This occurs in a series of branching splits to form a tree (with an appropriate tree diagram). If the predictor attributes are dichotomous, the splits occur easily since the data is already split into natural subsets through the attribute. Categorical attributes with more than two levels can be dealt with using multiway splits or through some dichotomizing strategy. Continuous attribute, however, require testing of outcome purity on several places along the attribute in order to determine the best splitting condition. In our case, the classification algorithm is first applied to the training data, and the branching conditions are saved. The prediction is then applied to the test data by applying the saved branching conditions. In Python the script to apply the algorithm only requires a single alteration from the previous SVC script (i.e., `model=SVC()` is replaced with `model=DecisionTreeClassifier()`). The resulting accuracy score was found to be 0.462, which is substantively worse than the null error rate. Thus, the DT algorithm appears to be functioning relatively poorly compared to the previous two algorithms. Interestingly, the chi-square goodness-of-fit test was non-significant (i.e., p=0.63), which indicates that the expected proportions of each of the classifications were achieved. However, while the proportion of the predicted classification appears to be representative of the actual classification in the test dataset as an aggregate, the poor accuracy score indicates that the algorithm is incorrectly assigning the classification at the correct proportions. Thus, we can conclude that this analysis also does not support Hypothesis 3.

The fourth algorithm is the random forest classifier (RF). The RF classifier works nearly identically with the DT algorithm with the exception that a multitude of decision trees are being applied to a multitude of subsets of the data. Essentially, the results of the multitude of decision trees and branching conditions for

[16] See Gates, Ami (2016) ANLY-501 Lecture 6

each of the trees are saved. The saved branching conditions are then applied to the test data, and the most common predicted classification (i.e., the mode) is the one that is applied to each test case[17]. The application of this algorithm in Python also requires only a single line alteration from the previous SVC script (i.e., `model=SVC()` is replaced with `model=RandomForestClassifier()`). The resulting accuracy score of this algorithm was found to be 0.499, which is similar to the performance of the DT algorithm. The chi-square goodness-of-fit test was highly significant (i.e., *p*-value was effectively zero). Thus, these results unfortunately also do not support Hypothesis 3.

The last algorithm utilized is the *K* nearest neighbors algorithm (k-NN). You may recall that this algorithm was already used in this report in providing estimates for the *Eps* parameter in the DBSCAN clustering algorithm. This algorithm is what is known as a "lazy learner", which is distinguished from what are known as "eager learner" algorithms[18]. All of the previous classification algorithms used so far apply "eager" learning in the sense that the act of fitting training data to the model produces parameters that contain all that are needed for a prediction to be completed. In contrast, "lazy" learning algorithm such as k-NN essentially just holds onto the training data and does nothing until a specific prediction request is made using the test data[19]. In a nutshell, a prediction is made using the k-NN algorithm by simply inputting the necessary predictor attributes from a single case of the test dataset. The distances (typically Euclidean) between the point produced by the inputted predictor attributes and the existing points in the training data are computed and the *K* closest points are identified (where *K* is determined beforehand). The mode of the categories of the *K* nearest neighbors (or a sum weighted by distances) determines which category the test data point is assigned. The application of this algorithm in Python also requires only a single line alteration from the previous SVC script (i.e., `model=SVC()` is replaced with `model=KNeighborsClassifier()`). The resulting accuracy score was found to be 0.478, which is somewhere in between the DT and RF algorithms and the SVM and NB algorithms and still substantively lower than the null error rate. The chi-square goodness-of-fit test was also highly significant (i.e., *p*-value was effectively zero) which further indicates that these results do not support Hypothesis 3.

Conclusion and Future Directions

At this point in the project, it is quite clear that the use of cross-sectional methodology to predict music listening behaviors was unsuccessful. Descriptive statistics indicated no discernable pattern between possible predictor variables and several possible outcome variables. Regression analysis revealed association between song duration and song playcount, but our primary variable of interest (cohesion distance) was not found to be a significant predictor. Finally, the results of the classification algorithms unanimously indicated no predictive effect of the selected attributes for the discretized cohesion distance attribute.

[17] See https://en.wikipedia.org/wiki/Random_forest
[18] See https://en.wikipedia.org/wiki/Eager_learning
[19] See http://scikit-learn.org/stable/modules/neighbors.html#classification

We may elect to use a time series in the next iteration of this report. The results of the cluster analysis did indicate reasonable separation between songs characterized by negative and positive emotion (though they do appear to be a single cluster in respect to density). This information can be used in the following weeks to identify several "prototypical" songs to represent the negative and positive emotions and attempt to track the playcount of these songs across several months. From there, we can identify several key news events that occurred through the same time period and determine if playcounts respond accordingly.

Citations

Bernard, J.R.L. (1986). *The Macquarie Thesaurus: The Book of Words*. Sydney, Macquarie Library.

Breunig, M.M., Kriegel, H.P., Ng, R.T., and Sander, J. (2000). LOF: Identifiying density-based local outliers. In: Proceedings of SIGMOD '00, Dallas, Texas, pp.427-438.

Ekman, P. (1992). An argument for basic emotions. *Cognition and Emotion*, 6(3-4).

Ramaswamy, S., Rastogi, R., Kyuseok, S. (2000). Efficient algorithms for mining outliers from large datasets. In: Proceedings of SIGMOD '00, Dallas, Texas, pp. 93-104.