Jack Dong

Consultation and Communication for Statisticians

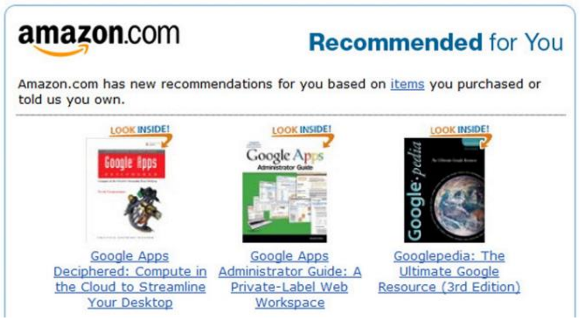
Dr. Barbara Kuzmak

December 19, 2020

Implementation of Spotify Recommender System using K-means Clustering

Introduction and Background

Many people know that companies track and record their userbases' information, but many do not know what they are used for and why. The short, easy answer is 'to sell more items', but it does not answer what specific data is collected and *how* it sells more items. Take TikTok for example: a new user sees basic, popular content for the first couple days. Then the contents start getting very specific; sports fans see more sports videos, gaming fans see more gaming videos, etc – it is built to cater to the user's interests. The TikTok community refers to this as "the algorithm". Amazon is the same; there are sections dedicated to recommending products other users buy based on purchasing history.


amazon.com
Recommended for You
LOOK INSIDE!
Google Apps Deciphered: Compute in the Cloud to Streamline Your Desktop
Google Apps Administrator Guide: A Private-Label Web Workspace
Googlepedia: The Ultimate Google Resource (3rd Edition)

Amazon.com has new recommendations for you based on items you purchased or told us you own.

Figure 1 Customer view of recommender system

Every modern, large company has similar features such as the following: Spotify for recommending tracks, Netflix for recommending movies, and Facebook for showing catered advertisements. Each company collects and uses different data for different implementations, but they all follow this model for one common reason of establishing customer loyalty.

The recommender system origins coincide with the rise of e-commerce and were only established after many trial and errors. As the internet base grew in the late 1990s, many businesses moved online to address the potential customer population. According to Goy et al. (2007), the first personalization attempts were developed to improve business-customer relationships and were met with several problems. There was

1. little to no established method of communicating to customers besides physical mail,
2. no way to update or delete inaccurate customer data in database (eg. cause businesses to deliver goods to wrong location or person),
3. no method of collecting customer digital interactions,
4. overall little to no infrastructure (application-database integration, etc) to uphold the previous three issues into one system.

In addition to lack of logistics, the demand for personalization was not there. A 2003 Jupiter Research study revealed "54% of respondents cited fast loading pages and 52% cited improved navigation was greater incentives" than personalized websites. Two years later, interests in developing e-personalization rose again to meet the growing business culture of appealing to customers, a "customer-first" mentality.

Goy et al. (2007) continues to describe the two approaches to recommender systems, and some advantages and disadvantages of them. Content-based methods use similarities between products for recommendation. Meaning if product A was purchased, recommendations will be similar to product A – exemplifying the high bias, low variance trade-off. It does not require user information, but computational power becomes problematic when number of items is large. Collaborative-filtering methods recommend products based on the interactions of the user. It does not require product information but requires a record of how all customers interact to every item. This runs into the sparse matrix issue because no user will interact with every item, and they rely on the customer interacting in a meaningful way (eg. rate products accurately). In summary, this textbook chapter details the emergence and approaches, and business-technology conflicts of recommender system – but little was said about the specifics of implementing them.

Introduction to Data Mining by Tan et al. (2006) lays out multiple unsupervised learning approaches, including k-means clustering. We focus on unsupervised learning algorithms due to the nature of our dataset, where the correct classifications are not known. Unsupervised learning also makes the evaluation process different, which we will cover later. K-means clustering partitions all datapoints into different groups based on proximity or similarity and defines the center of each group as the *centroid*, the average of the group. The algorithm, shown in Tan et al. (2006, p. 497) is as follows:

```
1: Select K points as initial centroids.
2: repeat
3:     Form K clusters by assigning each point to its closest centroid.
4:     Recompute the centroid of each cluster
5: until Centroids do not change
```

There are 3 assumptions under K-means: (1) that the cluster variance are spherical shapes, (2) clusters have similar sizes, and (3) the optimal K is chosen. The disadvantages are that k-means can only find local optima because there is not a way to find the global optimal centroids, the generated centroids vary each run due to randomness element in line 1, and it is sensitive to outliers – all considerations during evaluation. Overall, the chapter is very detailed on the framework of k-means and provides more than enough information (i.e., it continues onto extensions of K-means and “nested clustering”).

As previously mentioned, unsupervised learning cannot use cross-validation methods (eg. k-fold, leave-one-out) to evaluate the model. To understand the best evaluation for clustering, research by Brun et al. (2006) illustrates 3 possible validation categories. (1) Internal validation, which does not require additional data, uses characteristics (eg. size, shape, distance between clusters) of clusters to validate accuracy. Specifically, it illustrates the concept of cohesion, minimizing distance between points *within* clusters, and separation, maximizing distance between clusters. (2) Relative validation metrics assume the same model will perform similarly when trained using different parameters *or* using another sample set from the same source. (3) Lastly external validation, which requires the knowledge of true labels, includes cross validation and k-folds.

From this information, we choose internal validation is the best method of evaluation because we do not have pre-determined labels. There are three different internal methods mentioned by Brun et al. (2006): Dunn’s indices, the ratio of smallest inter-cluster distance to largest cluster size; Silhouette index, which is similar but uses *average* distance as ratio; and Hubert’s correlation with distance matrix, which calculates sums the similarities of each cluster. Overall, this research paper is very difficult to understand. The material is very dense with technical jargon and equations. Full comprehension requires knowing these equations beforehand, and 4 other more complicated cluster algorithms besides k-means were researched.

Nonetheless, the sources gathered provide us a great opportunity to create a recommender, which is very vibrant in the global market. This project allows me the chance to study and implement unsupervised learning instances, which we as students do not get many chances to learn; and understand the mechanics behind recommenders. We will be analyzing a subset of a large commercial Spotify dataset published for making recommenders. Therefore, this project will use k-means clustering and internal evaluation methods to create the best Spotify track recommendations.

Methods and Materials

The Spotify dataset originally had 1 million randomly selected playlists with over 2 million unique tracks. For the purpose of this project and due to computational power, only the first 1000 playlists are used (sourced from Vasylenko et al., 2018).

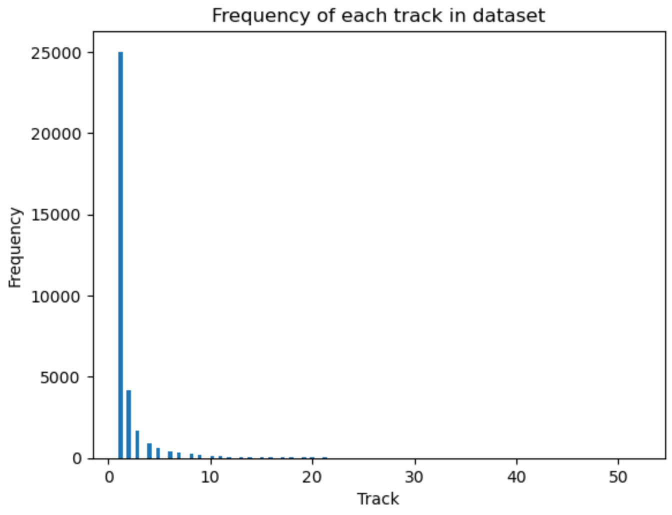


Figure 2 Frequency of each track in dataset. The total number of tracks exceed 34,000 with nearly 25,000 unique tracks. (Appendix B)

The raw data is a large nested json format where each playlist is an object containing a vector of tracks. Notable attributes are the following: *pid*, the playlist ID; *track_uri*, the track unique ID; and *artist_uri*, the artist ID. The last two identifiers can be used to retrieve audio data from Spotipy, the Spotify API for developers, such as the following:

Table 1
Description of audio features and what they measure. These variables are used for data mining.

Attribute	Definition provided by Spotify (all values 0.0 to 1.0, unless specified)
acousticness	Confidence measure of whether the track is acoustic. Higher value represents higher confidence it is acoustic.
danceability	How suitable a track is for dancing based on musical elements. Higher value means more danceable.
energy	Measures fastness, perceived loudness, onset rate, general entropy, and noisiness. Higher values are more energetic.
instrumentalness	Predicts if track has vocals. Symphonies have larger values, while rap and spoken words have low.
liveness	Probability of detecting live audience in audio. High values mean higher chances.
loudness	Average loudness throughout track in decibels. Ranges from -60 to 0.
speechiness	Describes the proportion of audio of spoken words; podcasts have high values; symphonies have low values.
valence	Overall presence of positivity, happiness, cheerfulness. Very happy has high value; depressed, sad have low
tempo	Average beats per minute

Note that most of these values are estimates and confidence intervals, not exact values throughout the track. These variables will make up the foundation of the cluster analysis.

In order to simplify the large dataset, we reduce the dataset's dimensionality. Principal Component Analysis (PCA) allows us to describe majority of the variance with less dimensions and select most important features in the process. After variable transformation, the dataset uses 5 components to describe 80% of the variance (see Appendix E). We continue onto modeling.

K-means assumes the following characteristics of the data: (1) clusters are spherical-shaped, (2) equal variance clusters, and (3) K is accurately established. The reasoning behind the one is that classification is based on a constant radius from the cluster's centroid. Non-spherical clusters are possible with K-means extensions, but not with basic K-means. Second, if one cluster is more spread out than another, the radii of each cluster are not equivalent. No feature allows clusters to be different sizes. Lastly, possibly the most ambiguous assumption is that the optimal model assumes the best K is already determined. Realistically it is rarely the case, but Brun et al. (2007) provides three methods to evaluate the goodness of K.

Once all assumptions are satisfied, we can model k-means. The objective is to minimize the sum of squares error (SSE), shown as:

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist(c_i, x)^2 \qquad (1)$$

where $dist$ refers to Euclidean distance between two objects, c_i is the centroid of cluster i, x is a datapoint within cluster i, and K number of clusters. Recall that the algorithm only finds the local SSE minimum given each cluster, because the final centroids vary depending on the initial positions. Global minimum is not guaranteed.

Now we apply the statistical techniques to the Spotify dataset. The first assumption on spherical clusters is undetermined. As each cluster represents a unique type of music users *prefer*, it is impossible to conclude the best cluster shape music preference. Though one alternative to address non-spherical clusters is converting to polar coordinates, but that is beyond this project's scope. Next, equal variance assumption can be resolved by standardizing all attributes to normal distribution (see Appendix D). Lastly, finding the right K-value is not intuitive so the Elbow, Silhouette score, and Dunn's index are used (Tan et al., 2007; Brun et al., 2006). Hubert's matrix method is not used due to sheer size of computational resource requirements.

While the task is to optimize SSE, K still has to be determined first. The intuition behind Elbow method is to explain as much SSE while minimizing K. The benefits of the Elbow method are its simplicity, interpretability, and trade-off mindset of minimizing SSE without creating abundant clusters. Next, Silhouette index is formally:

$$S = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{n_k}\sum_{x\in C_k}\frac{b(x)-a(x)}{max[b(x),a(x)]}, where\ S \in [-1,1] \qquad (2)$$

where x is a point within cluster C_k, n_k is number of points in cluster C_k, $b(x)$ is the minimum of all average separation distances, $a(x)$ is the average cohesion distance, and K is number of clusters. Higher values indicate better fit, and 0 indicates cluster overlap. Lastly Dunn's index is defined as:

$$V(C) = \frac{min_{h,k=1,\dots,K,h\neq k}\, d_C(C_k,C_h)}{max_{k=1,\dots,K}\,\Delta(C_k)} \qquad (3)$$

where $d_C(C_k,C_h)$ is the nearest distance between two clusters and $\Delta(C_k)$ is the size of cluster C_k (Brun et al., 2007). There are various methods for calculating d_C and Δ; we use Euclidean distances between points. Dunn's index measures how tightly compact the cluster is compared to its size. Higher values suggest better clustering.

To find the best model, we create 30 k-means models with K ranging from 50 to 1500 by 50s, then evaluate assumptions for each. Figure 3 displays 4 models to represent models similar to them on the scale. We see k-means maintain approximate spherical shapes across all clusters for all Ks. However, clusters with higher PC1 values appear to have less datapoints, suggesting slight imbalances in equal variance. There also are less clusters there to consider, so we continue with modeling. There is also an inverse relationship between number of clusters and cluster size; 50-means have many tracks per cluster, while 1200-means have fewer tracks.

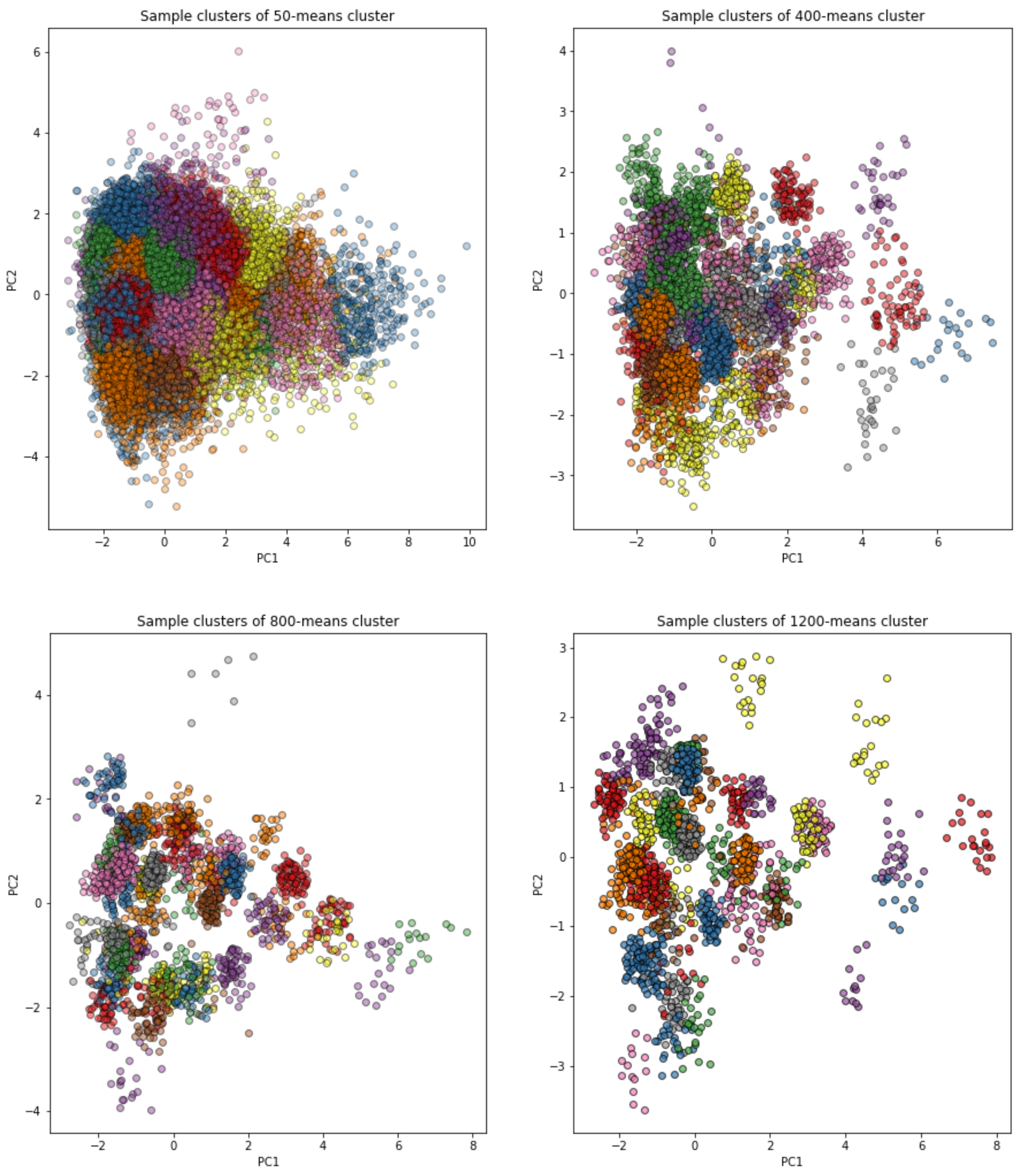


Figure 3 Comparison of low and high K-means clustering. Only 2 dimensions are visualized for simplicity. See Appendix G for more information.

Figure 5 shows the progression of total SSE, Silhouette, and Dunn’s Index over K. SSE exponentially decreases over K, possibly due to decrease in datapoints per cluster. Silhouette decreases until approximately K=400 to 600, then levels off around 0.16. Dunn’s Index widely varies between each model, but we see a general trend of increasing from 0.0075 to 0.0175.

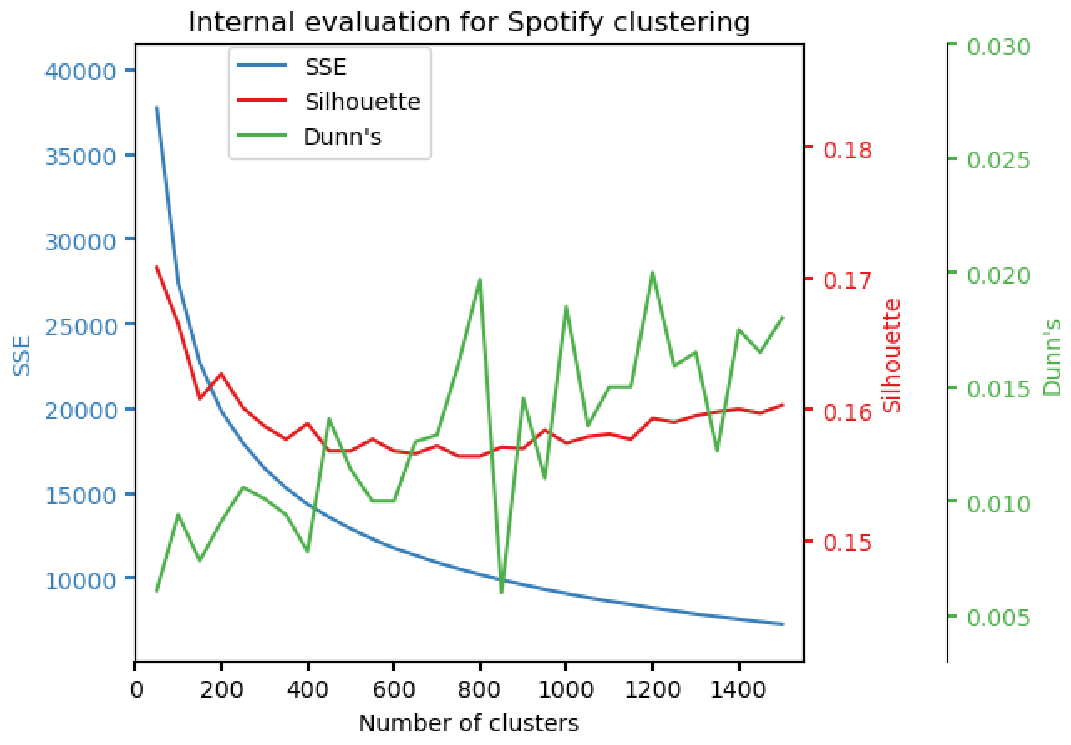


Figure 4 Silhouette and Dunn's scores are compared to overall SSE for each model. See Appendix H.

Silhouette’s line is smoother because it uses the *average* for each value, so all points are considered. This gets rid of any outliers’ effects on the overall metric. Meanwhile Dunn’s index uses minimums and maximums of clusters; if one cluster is abnormally large, it would affect the overall index. Nonetheless, both indices conclude all 30 models are poor models for the dataset; we want large values for both, but they are all approximately 0. The Elbow method concludes 400 as the best K, so we build the recommender using 400-means.

A sample run of the final recommender is shown below (see Appendix I).

```
You listened to: Loca by Alvaro Soler

We recommend (not ordered):
- City In A Snow Globe by Before Their Eyes
- Under The Flowers by The Orwells
- Light & Magic by Ladytron
- Leave It All To Me (Theme from iCarly) (feat. Drake Bell) by Miranda
Cosgrove, Drake Bell
- Dreamboat by The Walkmen
- Drunk Mouth Kitchen Smile by The Lawrence Arms
- Erase This by Lamb of God
- Rebel Love Song by Black Veil Brides
- Graveyard Dancing by Destroy Rebuild Until God Shows
- Blood Bubbles by The Orwells
- Halo / Walking On Sunshine (Glee Cast Version) by Glee Cast
- Epic by Faith No More
- Kinder Words by The Mighty Mighty Bosstones
- Young Lovers Go Pop! by This Many Boyfriends
- Parasite by Nine Shrines
```

Figure 5 Sample results of final recommender using "Loca by Alvaro Soler" as test track.

Discussion and Summary

We can conclude from the scores that k-means may not be the best clustering approach to the Spotify dataset. Although we ended up using Elbow method, it was a last resort and there was no visual "elbow" in Figure 4. Additionally, it is impossible to realistically evaluate the goodness of 400-means without a focus group to test the recommender. This is the most significant disadvantage of unsupervised learning – it is impossible to truly evaluate the model without involving test subjects.

There are a few different approaches to address bettering the model. One, use a different clustering algorithm. K-means proves to be a simple to understand clustering algorithm, but there was ambiguity in the dataset meeting the assumptions, specifically equal variance. There are other clustering methods such as hierarchical, agglomerative, and shared nearest neighbors to try. Second, use additional track information that could not previously be used. Mode (minor, major), key (A, A flat, etc), and genres are non-numeric variables that are retrievable but not suitable for clustering. Instead, they can be hot encoded to become numeric.

Third, incorporate the playlists into the analysis. We can treat each playlist as individual customers and the tracks as items, then use collaborative filtering for modeling. Four, enlarge the dataset. While 34442 is a large dataset, it is only a portion of the original published set. When considering the number of possible genres and niche music types, there may not be enough samples per cluster to accurately represent each type of music. K-means was just the best method to try first due to its quick, easy implementation. There are many different ways to improve our initial recommender model.

Nonetheless, this project allows us to learn the complexities of recommenders. They are a great business example of a data mining application. They can use any supervised or unsupervised learning algorithm depending on the dataset, and they are virtually on every major retail website. In general, it improves sales, boosts revenue, and motivates having a “customer-first” which overall benefits the business. It also provides better experiences for the customer. In the case of Spotify, it shows relevant tracks with less effort on the customer. The field of recommenders is always improving with data science, and what we see here is only a small snippet.

References

Brun, M., Sima, C., Hua, J., Lowey, J., Carroll, B., Suh, E., & Dougherty, E. R. (2007). Model-based evaluation of clustering validation measures. *Pattern Recognition Society*, *40*(3), 807-824. doi:10.1016/j.patcog.2006.06.026

Frequently bought together [Digital image]. (n.d.). Retrieved from https://www.mageplaza.com/blog/product-recomendation-how-amazon-succeeds-with-it.html

Goy, A., Ardissono, L., & Petrone, G. (2007). Personalization in E-Commerce Applications. *The Adaptive Web Lecture Notes in Computer Science, 4321*, 485-520. doi:10.1007/978-3-540-72079-9_16

Jupiter Research: Beyond the Personalization Myth: Cost-effective Alternatives to Influence Intent. (n.d.) Report. Jupiter Research Corporation.

Spotify. (2020). Tracks. Spotify for Developers. Retrieved from https://developer.spotify.com/documentation/web-api/reference/tracks/

Tan, P., Steinbach, M., & Kumar, V. (2006). Cluster Analysis: Basic Concepts and Algorithms. In *Introduction to Data Mining* (pp. 487-556). Boston, MA: Pearson Addison Wesley.

Vasylenko, J., Chitwankaudan, & Anithp. (2018, May 4). Spotify-Song-Recommendation-ML. GitHub repository. Retrieved March 11, 2020, from https://github.com/vaslnk/Spotify-Song-Recommendation-ML/tree/master/data

Appendix A

Raw Data and Data Formation

The original data is formatted as a very large json, with a vector of playlists and each playlist has a vector of tracks. The primary data extracted from the json is the “pid”, “track_uri”, and “artist_uri” because the Spotipy API contains variable information about each item. The data is read into a user-item data frame; assuming each playlist represents a unique user. The data frame is 34443 x 1000, total tracks by total playlists.

```
{
  "playlists": [
        {
          "name": "Throwbacks",
          "collaborative": "false",
          "pid": 0,
          "modified_at": 1493424000,
          "num_tracks": 52,
          "num_albums": 47,
          "num_followers": 1,
          "tracks": [
                {
                "pos": 0,
                "artist_name": "Missy Elliott",
                "track_uri": "spotify:track:0UaMYEvWZi0ZqiDOoHU3YI",
                "artist_uri": "spotify:artist:2wIVse2owClT7go1WT98tk",
                "track_name": "Lose Control (feat. Ciara & Fat Man Scoop)",
                "album_uri": "spotify:album:6vV5UrXcfyQD1wu4Qo2I9K",
                "duration_ms": 226863,
                "album_name": "The Cookbook"
                },...
                  ],
          "num_edits": 6,
          "duration_ms": 11532414,
          "num_artists": 37
      },...
          ],
}
```

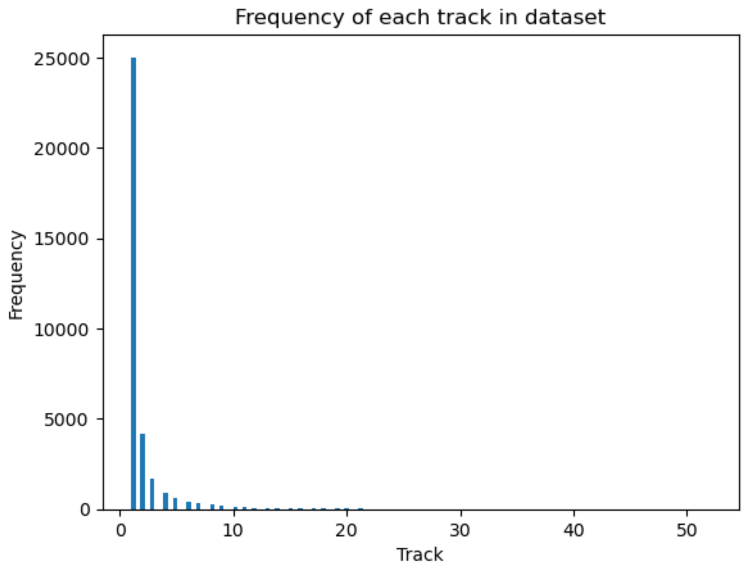
	0	1	2	...	997	998	999
track_uri							
spotify:track:0UaMYEvWZi0ZqiDOoHU3YI	1	0	0	...	0	0	0
spotify:track:6I9VzXrHxO9rA9A5euc8Ak	1	0	0	...	0	0	0
spotify:track:0WqIKmW4BTrj3eJFmnCKMv	1	0	0	...	0	0	0
...	...	...	...	...	...	...	...
spotify:track:2oM4BuruDnEvk59IvIXCwn	0	0	0	...	0	0	1
spotify:track:4Ri5TTUgjM96tbQZd5Ua7V	0	0	0	...	0	0	1
spotify:track:5RVuBrXVLptAEbGJdSDzL5	0	0	0	...	0	0	1

Appendix B

Frequency of Each Track

The sparsity of the user-item matrix can be illustrated by viewing the frequency of tracks. Sparse matrices are created when the majority of the matrix are zeros, meaning there is no interaction between a user and the majority of the items. This is problematic because it could be difficult to extract the fewer meaningful interactions between users and items, combatting the purpose of collaborative filtering – which finds similar users based on item interactions. As Figure 1 shows, most tracks appear only once in the dataset.

```
track_freq = user_item_matrix.sum(axis=1)
plt.rcParams.update({'figure.figsize':(7,5), 'figure.dpi':100})
params = {"ytick.color" : "w",
          "xtick.color" : "w",
          "axes.labelcolor" : "w",
          "axes.edgecolor" : "w",
         }
plt.rcParams.update(params)
plt.style.use("classic")
plt.hist(track_freq,bins=100,label="Unique tracks")
plt.gca().set(title = "Figure 1 Frequency of each track in dataset",
              ylabel = "Frequency", xlabel = "Track");
```



Appendix C

Audio Feature Information Retrieval

Each track’s audio features need to be collected through the Spotify API. The first code section is a helper function called `getAudioFeatures` that returns the feature details from Spotify. It requires the track_uri as parameter and returns a dictionary.

```
import requests
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials

SPOTIPY_CLIENT_ID = "client_id"
SPOTIPY_CLIENT_SECRET = "client_secret"

client_credentials_manager = SpotifyClientCredentials(SPOTIPY_CLIENT_ID, SPOTIPY_CLIENT_SECRET)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)

def getAudioFeatures(trackuri):
    return sp.audio_features(trackuri)[0]
```

The helper function gets called for each track on line 15, and relevant features are extracted and placed to their respective vectors. The final vectors comprise the `feature_stats` data frame.

```
1. danceability = []
2. energy = []
3. key = []
4. loudness = []
5. mode = []
6. speechiness = []
7. acousticness = []
8. instrumentalness = []
9. liveness = []
10.  valence = []
11.  tempo = []
12.
13.  for i in range(0,len(all_tracks)):
14.      try:
15.          feats = getAudioFeatures(all_tracks[i])
16.          danceability.append(feats['danceability'])
17.          energy.append(feats['energy'])
18.          key.append(feats['key'])
19.          loudness.append(feats['loudness'])
```

```
        mode.append(feats['mode'])
        speechiness.append(feats['speechiness'])
        acousticness.append(feats['acousticness'])
        instrumentalness.append(feats['instrumentalness'])
        liveness.append(feats['liveness'])
        valence.append(feats['valence'])
        tempo.append(feats['tempo'])
    except:
        print("Index ",i," had issues: ",all_tracks[i])

feature_stats = pd.DataFrame({'danceability':danceability,
                              'energy':energy,
                              'key':key,
                              'loudness':loudness,
                              'mode':mode,
                              'speechness':speechiness,
                              'acousticness':acousticness,
                              'instrumentalness':instrumentalness,
                              'liveness':liveness,
                              'valence':valence,
                              'tempo':tempo})
```

During data reading, one index errored out; the Spotify name `all_tracks[i]` return 0. That track was deleted from the data frame to mitigate outliers and missing data. The resulting data frame looks like:

```
feature_stats.head()
```

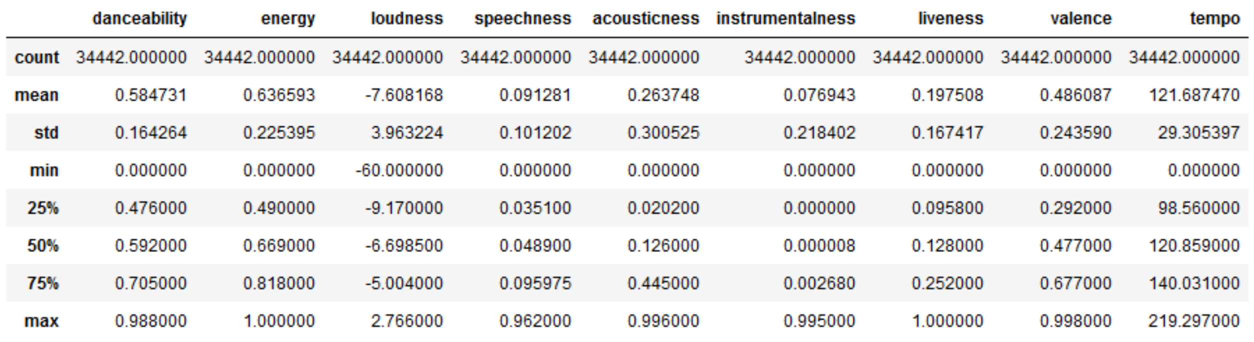
	danceability	energy	loudness	speechness	acousticness	instrumentalness	liveness	valence	tempo
0	0.904	0.813	-7.105	0.1210	0.03110	0.006970	0.0471	0.810	125.461
1	0.774	0.838	-3.914	0.1140	0.02490	0.025000	0.2420	0.924	143.040
2	0.664	0.758	-6.583	0.2100	0.00238	0.000000	0.0598	0.701	99.259
3	0.891	0.714	-6.055	0.1400	0.20200	0.000234	0.0521	0.818	100.972
4	0.853	0.606	-4.596	0.0713	0.05610	0.000000	0.3130	0.654	94.759

Appendix D

Feature Statistics Matrix and Standardization

The feature statistics matrix is the training data needed for modeling. The second assumption requires equal variance between all attributes. Pre-transformed data shows

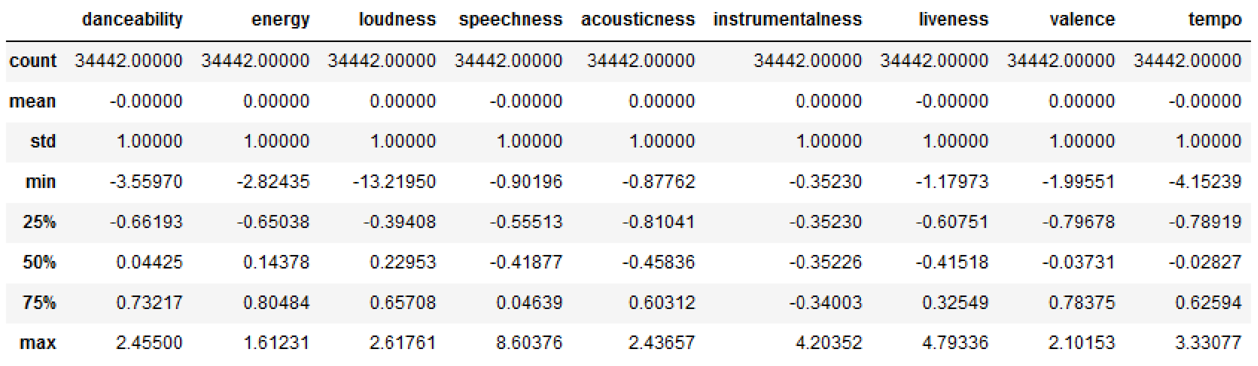
```
feature_stats.describe()
```



	danceability	energy	loudness	speechness	acousticness	instrumentalness	liveness	valence	tempo
count	34442.000000	34442.000000	34442.000000	34442.000000	34442.000000	34442.000000	34442.000000	34442.000000	34442.000000
mean	0.584731	0.636593	-7.608168	0.091281	0.263748	0.076943	0.197508	0.486087	121.687470
std	0.164264	0.225395	3.963224	0.101202	0.300525	0.218402	0.167417	0.243590	29.305397
min	0.000000	0.000000	-60.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.476000	0.490000	-9.170000	0.035100	0.020200	0.000000	0.095800	0.292000	98.560000
50%	0.592000	0.669000	-6.698500	0.048900	0.126000	0.000008	0.128000	0.477000	120.859000
75%	0.705000	0.818000	-5.004000	0.095975	0.445000	0.002680	0.252000	0.677000	140.031000
max	0.988000	1.000000	2.766000	0.962000	0.996000	0.995000	1.000000	0.998000	219.297000

Loudness and tempo, by far, have the largest variance. The data is transformed into normal distribution to fulfill equal variance assumption like so:

```
normalized_feature_stats = (feature_stats-
feature_stats.mean())/feature_stats.std()
normalized_feature_stats.describe().apply(lambda s:
s.apply('{0:.5f}'.format))
```



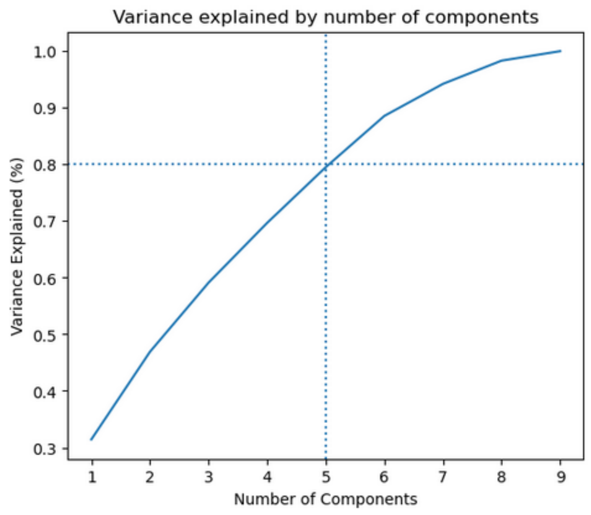
	danceability	energy	loudness	speechness	acousticness	instrumentalness	liveness	valence	tempo
count	34442.00000	34442.00000	34442.00000	34442.00000	34442.00000	34442.00000	34442.00000	34442.00000	34442.00000
mean	-0.00000	0.00000	0.00000	-0.00000	0.00000	0.00000	-0.00000	0.00000	-0.00000
std	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
min	-3.55970	-2.82435	-13.21950	-0.90196	-0.87762	-0.35230	-1.17973	-1.99551	-4.15239
25%	-0.66193	-0.65038	-0.39408	-0.55513	-0.81041	-0.35230	-0.60751	-0.79678	-0.78919
50%	0.04425	0.14378	0.22953	-0.41877	-0.45836	-0.35226	-0.41518	-0.03731	-0.02827
75%	0.73217	0.80484	0.65708	0.04639	0.60312	-0.34003	0.32549	0.78375	0.62594
max	2.45500	1.61231	2.61761	8.60376	2.43657	4.20352	4.79336	2.10153	3.33077

Appendix E

Dimension Reduction using Principal Component Analysis

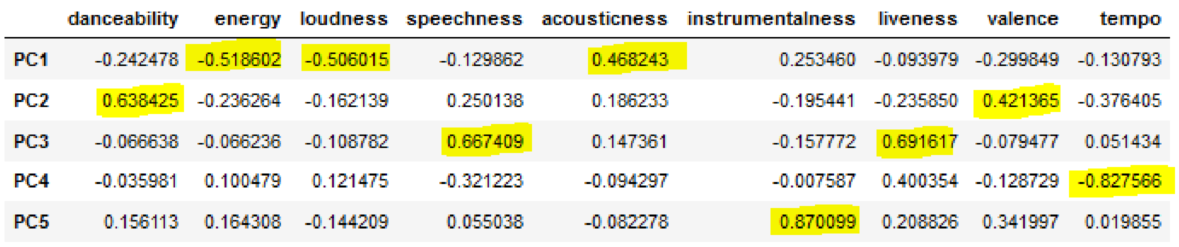
With a large dataset and with variables that have clear correlation with each other (eg. ‘energy’ and ‘loudness’), dimension reduction would improve the dataset’s describability and complexity. Reducing dimensions also improves computational time during modeling. One method is Principal Component Analysis, processed below.

```
pca = PCA(n_components=5)
train_pca = pca.fit_transform(normalized_feature_stats)
attributes  = ['danceability','energy','loudness','speechness',
 'acousticness','instrumentalness','liveness','valence','tempo']
PC_cols = ['PC1','PC2','PC3','PC4','PC5']
temp = pd.DataFrame(pca.components_,columns=attributes)
temp = temp.rename(index={0:'PC1',1:'PC2',2:'PC3',3:'PC4',4:'PC5'})
temp
```



PCA normally aims to have about 80 to 90% of the variance explained. In this dataset, 80% aligns with 5 components. After variable transformation, the components can be described as the following.

```
train_pca = pd.DataFrame(train_pca,columns=PC_cols)
train_pca
```



	danceability	energy	loudness	speechness	acousticness	instrumentalness	liveness	valence	tempo
PC1	-0.242478	-0.518602	-0.506015	-0.129862	0.468243	0.253460	-0.093979	-0.299849	-0.130793
PC2	0.638425	-0.236264	-0.162139	0.250138	0.186233	-0.195441	-0.235850	0.421365	-0.376405
PC3	-0.066638	-0.066236	-0.108782	0.667409	0.147361	-0.157772	0.691617	-0.079477	0.051434
PC4	-0.035981	0.100479	0.121475	-0.321223	-0.094297	-0.007587	0.400354	-0.128729	-0.827566
PC5	0.156113	0.164308	-0.144209	0.055038	-0.082278	0.870099	0.208826	0.341997	0.019855

PC1, for example, can be interpreted to be negatively correlated with energy and loudness, and positively correlated with acousticness – which makes sense, high energy tracks are not typically acoustic.

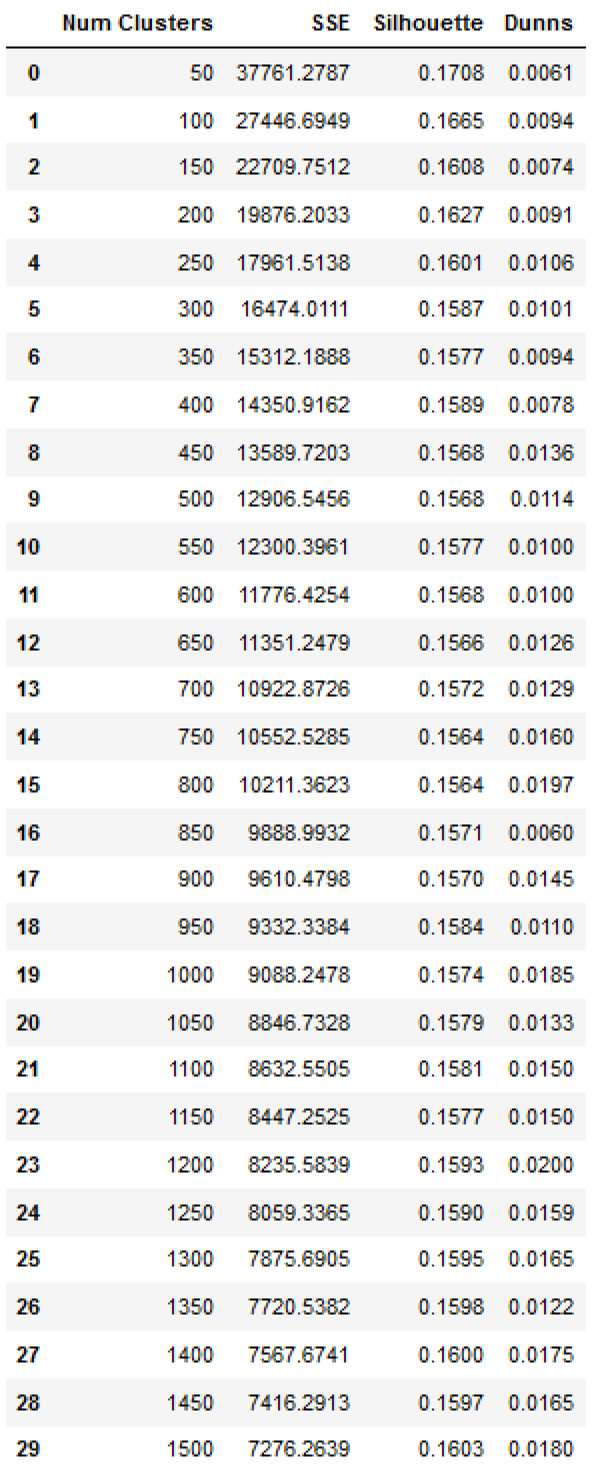
Appendix F

Internal Evaluations: Elbow, Silhouette, and Dunn’s

The objective is to optimize K based on several evaluations. Due to the sheer size of the data, it is expected to test large range of numbers. To shorten computation time, only 10 Ks are selected, see line 9. For each modeled K-means model, the Silhouette and Dunn’s score are calculated.

```
1. ## Finding optimal K
2. from sklearn.cluster import KMeans
3. from sklearn import metrics
4.
5. SEED = 4983
6. kmeans_inertia = []
7. silhouette = []
8. dunn = []
9. k_values = range(50,1550,50)
10.  for k in k_values:
11.     # Generate model
12.     temp = KMeans(n_clusters=k,random_state=SEED).fit(train_pca)
13.
14.     # Obtain Silhouette score
15.     sil = metrics.silhouette_score(train_pca,temp.labels_,
                                        metric='euclidean')
16.     # Obtain Dunn's index
17.     dun = base.dunn_fast(train_pca,temp.labels_)
18.
19.     # Add to arrays
20.     kmeans_inertia.append(round(temp.inertia_))
21.     silhouette.append(round(sil,4))
22.     dunn.append(round(dun,4))
```

The exact results of modeling K-means models of various cluster numbers are in the following table.



	Num Clusters	SSE	Silhouette	Dunns
0	50	37761.2787	0.1708	0.0061
1	100	27446.6949	0.1665	0.0094
2	150	22709.7512	0.1608	0.0074
3	200	19876.2033	0.1627	0.0091
4	250	17961.5138	0.1601	0.0106
5	300	16474.0111	0.1587	0.0101
6	350	15312.1888	0.1577	0.0094
7	400	14350.9162	0.1589	0.0078
8	450	13589.7203	0.1568	0.0136
9	500	12906.5456	0.1568	0.0114
10	550	12300.3961	0.1577	0.0100
11	600	11776.4254	0.1568	0.0100
12	650	11351.2479	0.1566	0.0126
13	700	10922.8726	0.1572	0.0129
14	750	10552.5285	0.1564	0.0160
15	800	10211.3623	0.1564	0.0197
16	850	9888.9932	0.1571	0.0060
17	900	9610.4798	0.1570	0.0145
18	950	9332.3384	0.1584	0.0110
19	1000	9088.2478	0.1574	0.0185
20	1050	8846.7328	0.1579	0.0133
21	1100	8632.5505	0.1581	0.0150
22	1150	8447.2525	0.1577	0.0150
23	1200	8235.5839	0.1593	0.0200
24	1250	8059.3365	0.1590	0.0159
25	1300	7875.6905	0.1595	0.0165
26	1350	7720.5382	0.1598	0.0122
27	1400	7567.6741	0.1600	0.0175
28	1450	7416.2913	0.1597	0.0165
29	1500	7276.2639	0.1603	0.0180

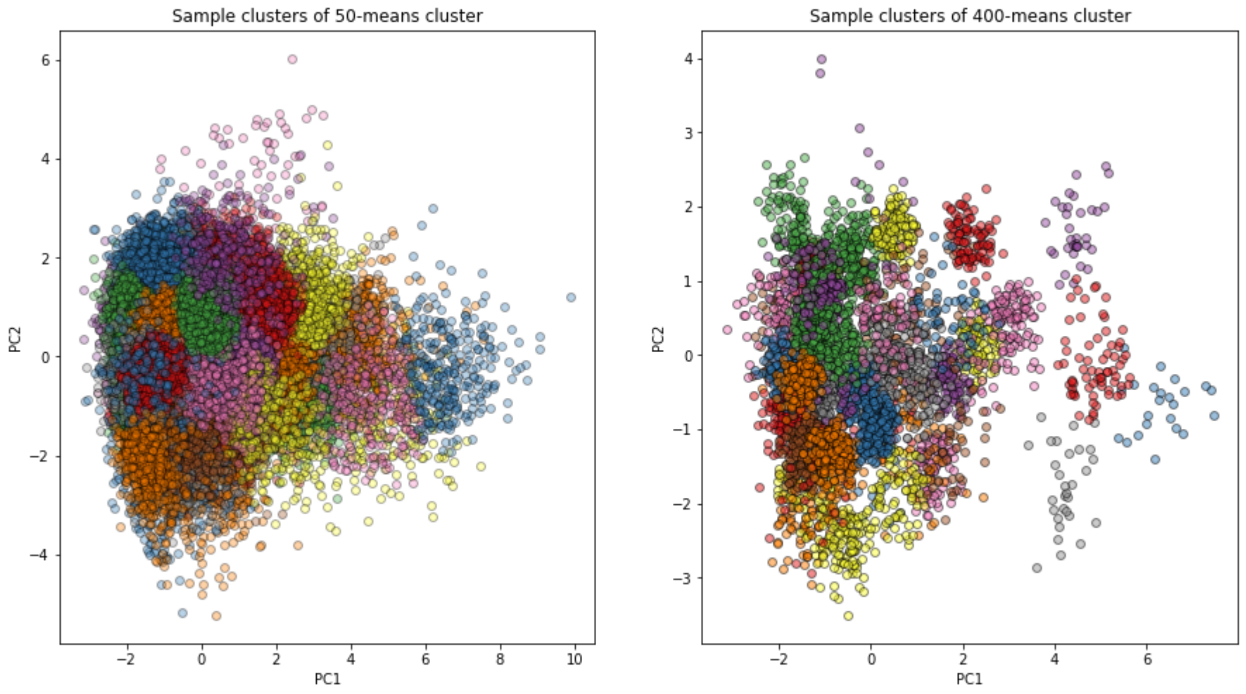
Appendix G

Visualization of Clusters

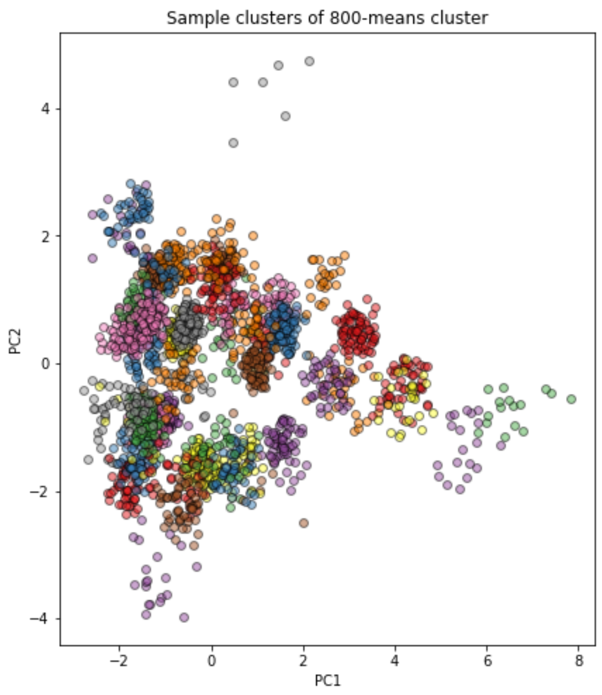
To clearly visualize the discrepancies between Silhouette and Dunn's, two opposing K-means models are generated (K=10,1440) to view cluster characteristics visually.

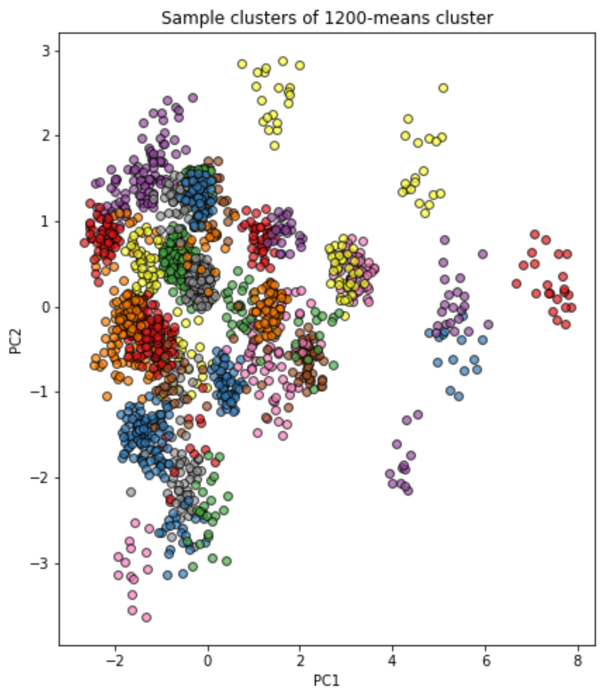
```
SEED = 4983
kmeans_50 = KMeans(n_clusters=50,random_state=SEED).fit(train_pca)
kmeans_400 = KMeans(n_clusters=400,random_state=SEED).fit(train_pca)
kmeans_800 = KMeans(n_clusters=800,random_state=SEED).fit(train_pca)
kmeans_1200 = KMeans(n_clusters=1200,random_state=SEED).fit(train_pca)
kmeans_1500 = KMeans(n_clusters=1500,random_state=SEED).fit(train_pca)

various_k_models = pd.DataFrame(index=range(0,34442),data = {
     'K=50':kmeans_50.labels_,
     'K=400':kmeans_400.labels_,
     'K=800':kmeans_800.labels_,
     'K=1200':kmeans_1200.labels_,
     'K=1500':kmeans_1500.labels_
})
various_k_models=pd.concat([various_k_models,train_pca.reset_index(
                           drop=True)], axis=1)
plt.subplot(1,2,1)

for i in range(50):
    plt.scatter(
        various_k_models[(various_k_models['K=50'] == i)]['PC1'],
        various_k_models[(various_k_models['K=50'] == i)]['PC2'],
        s=35, c=cols[i%9],
        marker='o', edgecolor='black',
        label='cluster '+str(i+1),
        alpha=0.35
    )

plt.title("Sample clusters of 50-means cluster")
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.subplot(1,2,2)
for i in range(50):
    plt.scatter(
        various_k_models[(various_k_models['K=400'] == i)]['PC1'],
        various_k_models[(various_k_models['K=400'] == i)]['PC2'],
        s=35, c=cols[i%9],
        marker='o', edgecolor='black',
        label='cluster '+str(i+1),
        alpha=0.50
    )
```

```

plt.title("Sample clusters of 400-means cluster")
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```



```
plt.figure(figsize=(15,8))
plt.subplot(1,2,1)

for i in range(50):
    plt.scatter(
        various_k_models[(various_k_models['K=50'] == i)]['PC1'],
        various_k_models[(various_k_models['K=50'] == i)]['PC2'],
        s=35, c=cols[i%9],
        marker='o', edgecolor='black',
         label='cluster '+str(i+1),
         alpha=0.35
     )

plt.title("Sample clusters of 50-means cluster")
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.subplot(1,2,2)
for i in range(50):
    plt.scatter(
        various_k_models[(various_k_models['K=400'] == i)]['PC1'],
        various_k_models[(various_k_models['K=400'] == i)]['PC2'],
        s=35, c=cols[i%9],
        marker='o', edgecolor='black',
        label='cluster '+str(i+1),
        alpha=0.50
    )

plt.title("Sample clusters of 400-means cluster")
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```





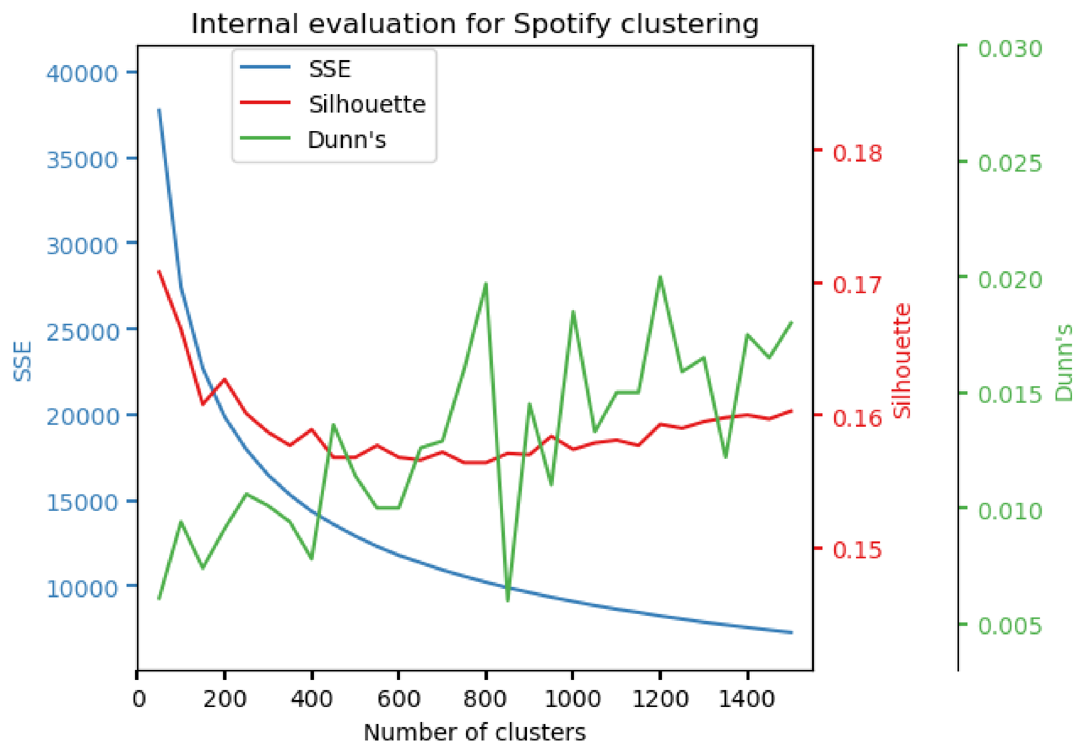
Appendix H

Visualization of Internal Evaluation Scores

The following Python code visualizes the table from Appendix F. Clear relationships of the scores to K can be observed then.

```
#https://colorbrewer2.org/?type=qualitative&scheme=Set1&n=9
cols = ['#377eb8','#e41a1c','#4daf4a','#984ea3','#ff7f00','#ffff33', '#a65628','#f781bf','#999999']

def make_patch_spines_invisible(ax):
    ax.set_frame_on(True)
    ax.patch.set_visible(False)
    for sp in ax.spines.values():
        sp.set_visible(False)

      k_values = range(50,1550,50)
      kmeans_inertia = eval_matrix['SSE']
      silhouette = eval_matrix['Silhouette']
      dunn = eval_matrix['Dunns']

      plt.style.use("default")
      fig, host = plt.subplots()
      fig.subplots_adjust(right=0.75)

      par1 = host.twinx()
      par2 = host.twinx()

      par2.spines["right"].set_position(("axes", 1.215))
      make_patch_spines_invisible(par2)
      par2.spines["right"].set_visible(True)

      p1, = host.plot(k_values, kmeans_inertia, cols[0], label="SSE")
      p2, = par1.plot(k_values, silhouette, cols[1], label="Silhouette")
      p3, = par2.plot(k_values, dunn, cols[2], label="Dunn's")

      host.set_xlim(0, 1550)

      # Auto calculate y axises
      ylow = min(eval_matrix['SSE'])*0.7
      yhigh = max(eval_matrix['SSE'])*1.1
      host.set_ylim(ylow, yhigh)

      ylow = min(eval_matrix['Silhouette'])*0.9
      yhigh = max(eval_matrix['Silhouette'])*1.1
      par1.set_ylim(ylow, yhigh)
```

```

        ylow = min(eval_matrix['Dunns'])*0.5
        yhigh = max(eval_matrix['Dunns'])*1.5
        par2.set_ylim(ylow, yhigh)

        host.set_xlabel("Number of clusters")
        host.set_ylabel("SSE")
        par1.set_ylabel("Silhouette")
        par2.set_ylabel("Dunn's")

        host.yaxis.label.set_color(p1.get_color())
        par1.yaxis.label.set_color(p2.get_color())
        par2.yaxis.label.set_color(p3.get_color())

        tkw = dict(size=4, width=1.5)
        host.tick_params(axis='y', colors=p1.get_color(), **tkw)
        par1.tick_params(axis='y', colors=p2.get_color(), **tkw)
        par2.tick_params(axis='y', colors=p3.get_color(), **tkw)
        host.tick_params(axis='x', **tkw)

        lines = [p1, p2, p3]

        host.legend(lines, [l.get_label() for l in lines],loc=(.14,.815))

        plt.title("Internal evaluation for Spotify clustering")
        plt.show()
```



Appendix I

Final Recommender

This is the program that consolidates the model generation and new track prediction process. We start with creating a 400-means model, then use the Spotify API to retrieve the closest track based on user input, predict the best cluster for that track, then return a playlist from that cluster.

```
from lib import audio_methods
import pandas as pd
from sklearn.cluster import KMeans

normalized_feature_stats = pd.read_csv("./data_frames/normalized_feature_stats.csv", usecols = range(0,10), index_col = 0)
mean_stats = normalized_feature_stats.mean()
std_stats = normalized_feature_stats.std()

# KMeans of K=400
kmeans = KMeans(n_clusters=400,random_state=4983).fit( normalized_feature_stats)

# DF of each track's clusters based on kmeans model
all_cluster_family = pd.DataFrame({
    'track':normalized_feature_stats.index,
    'cluster_id':kmeans.labels_
})

"""
Usage:

from lib import recommender

recommender.recommend('Alvaro Soler Loca')
"""
def recommend(description,n=10):
    """
    Uses Spotify search function to find closest related track to user's input description,
    Returns n number of recommended samples

    Inputs:
    - Description, user input string describing song title, artist
    - n, number of recommended songs desired, defaults to 10

    Returns:
    - Print statement of n recommends based on what Spotify found with user input description
    """

    try:
        # Find audio features of user input track
        base_track = audio_methods.findTrack(description)
        base_track_features = audio_methods.getAudioFeatures(base_track)
    except:
        return print("Error: -1. Track not found, try another track.")

    # Format features into df to prep for model prediction
    base_track_features_df = pd.DataFrame({
        'danceability':base_track_features['danceability'],
        'energy':base_track_features['energy'],
        'loudness':base_track_features['loudness'],
        'speechiness':base_track_features['speechiness'],
        'acousticness':base_track_features['acousticness'],
        'instrumentalness':base_track_features['instrumentalness'],
        'liveness':base_track_features['liveness'],
        'valence':base_track_features['valence'],
        'tempo':base_track_features['tempo']
    },index=[0])

    # Normalize user input track and predict using model
    normalized_base_track = (base_track_features_df-mean_stats)/std_stats
    base_track_cluster = kmeans.predict(normalized_base_track)[0]

    # Return N tracks in recommended cluster
    recommends = all_cluster_family[all_cluster_family.cluster_id == base_track_cluster].sample(n).track

    # Return string of title, artists of recommends
    final_string = "\r\nYou listened to: " + audio_methods.getNameOfTrack(base_track) + " by "
    final_string += audio_methods.getArtistOfTrack(base_track) + "\r\n\r\n"
    final_string += "We recommend (not ordered):\r\n"
    for sng in recommends:
        track_name = audio_methods.getNameOfTrack(sng)
        artist_name = audio_methods.getArtistOfTrack(sng)
        final_string += "- " + track_name + " by " + artist_name + "\r\n"
    return print(final_string)

recommend("loca alvaro soler",15)
```

```
You listened to: Loca by Alvaro Soler

We recommend (not ordered):
- City In A Snow Globe by Before Their Eyes
- Under The Flowers by The Orwells
- Light & Magic by Ladytron
- Leave It All To Me (Theme from iCarly) (feat. Drake Bell) by Miranda
Cosgrove, Drake Bell
- Dreamboat by The Walkmen
- Drunk Mouth Kitchen Smile by The Lawrence Arms
- Erase This by Lamb of God
- Rebel Love Song by Black Veil Brides
- Graveyard Dancing by Destroy Rebuild Until God Shows
- Blood Bubbles by The Orwells
- Halo / Walking On Sunshine (Glee Cast Version) by Glee Cast
- Epic by Faith No More
- Kinder Words by The Mighty Mighty Bosstones
- Young Lovers Go Pop! by This Many Boyfriends
- Parasite by Nine Shrines
```