Muziki[1] - Million Song Project

W251 - Scaling Up! Really Big Data

Hussein Danish, Younghak Jang, Sam Kabue, Saru Mehta, Stephane Truong

1. Introduction

The Million Song Dataset (MSD) is a 280GB collection of one million western popular music pieces that contains 55 music features provided by The Echo Nest. Examples of fields available in the dataset are tempo, loudness, timings of fade-in and fade-out, and more subjective attributes derived from previous classification algorithms, like danceability, song hotness, and energy. Along with the MSD data, we also used the MSD AllMusic Top Genre Dataset that maps over 400,000 music tracks to genre labels. Another dataset linked to the main MSD which we used for our exploratory analysis is the musiXmatch dataset, which provides lyrics for roughly 240,000 tracks in the MSD.

We chose to pursue this project because music is important to humans and acts as a universal outlet for humans to express themselves. However, since music is a creation of the mind which works in abstract ways, it is an inherent challenge for data scientists and music enthusiasts to classify, organize, and reveal patterns in music tracks. This is due to the nature of questions that must be answered in order to address this task: What makes a hit song; What are the similarities in music of a particular genre; How can music be auto-classified, besides via genre?

Our goal for the project was therefore to answer the following interesting questions:

- What are some trends of music over the last couple of decades?
- Can we cluster similar songs using ML algorithms?
- What is the best predictor for song hotness?
- How does the bag-of-words and sentiment of lyrics differ by genre?
- Can we predict the genre using song statistics and lyrics?

2. Tools/Architecture

In order to download and work with this data, we set up a CentOS-based cluster in SoftLayer which we configured as follows:

- **Spark Cluster**
  5 Nodes (1 master and 4 slaves) with a collective 20 cores, 40GB RAM

[1] Swahili: *music*

Each node has 100GB (OS) and 1000 GB San disk for the data

- **Apache Cassandra**
  The MSD data is presented in HDF5 format[2] which is good for storing and managing high volume and complex data, so in order to do our analysis we extracted the song data from each HDF5 file and stored this in Cassandra. We also used Cassandra to store aggregated track and lyric sentiment information.
- **DataStax OpsCenter for Apache Cassandra**
  This is a visual monitoring tool for Cassandra that we also used to deploy and configure the Cassandra cluster.
- **Apache Hadoop**
  We also used HDFS to store converted HDF5 files of each track and intermediate calculations and analysis of track data.
- **Elasticsearch**
  We leveraged the distributed, scalable, real-time search and analytics capabilities of Lucene Elasticsearch to process our aggregated music data.
- **Kibana**
  A user interface for Elasticsearch, Kibana was useful for visualizing the similarities and difference in attributes among song clusters (using K-means algorithm). We use Elasticsearch and Kibana as a UI to find similar songs given a user entered a particular song.
- **Tachyon**
  We used this to persist RDDs Off-heap that are used in our calculations.
- **HDF5 python library h5py**
  We used this library to convert each song track's HDF5 files to easily readable text files that we saved as .csv.

UIs for the above tools can be accessed as follows:

- Spark Cluster: http://169.53.141.8:8080
- OpsCenter: http://169.53.141.8:8888
- Kibana UI: http://169.53.141.8:5601

3. Exploratory Analysis

a. MapReduce

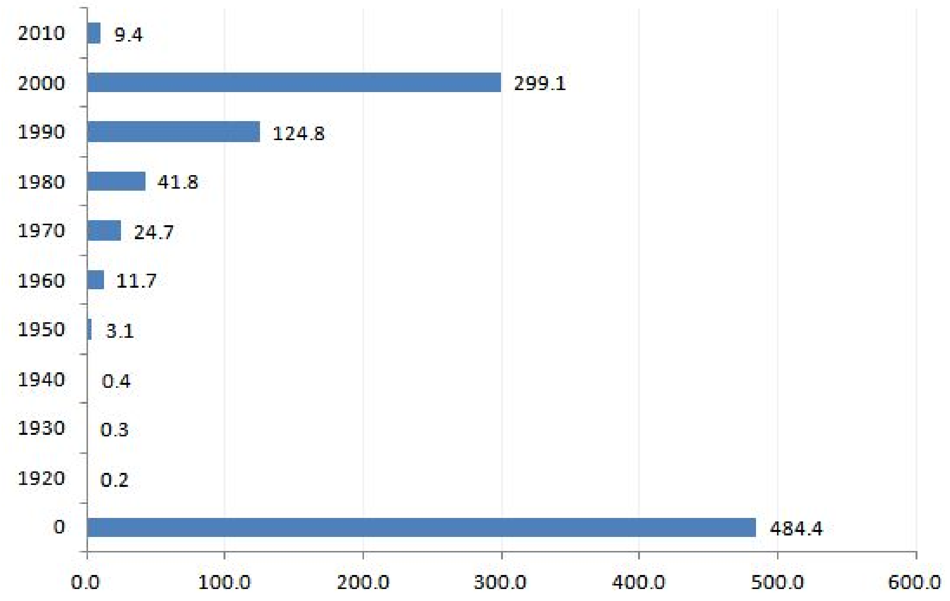
As the name implies, this dataset has exactly one million songs. The original data has the exact year of song publishing, but we grouped the songs by decades and used MapReduce to count the number of songs per decade. Roughly 40% of the songs are after 2000, and about 50% of the songs don't have the year data.

[2] http://www.hdfgroup.org/HDF5/

MapReduce Script:[3]

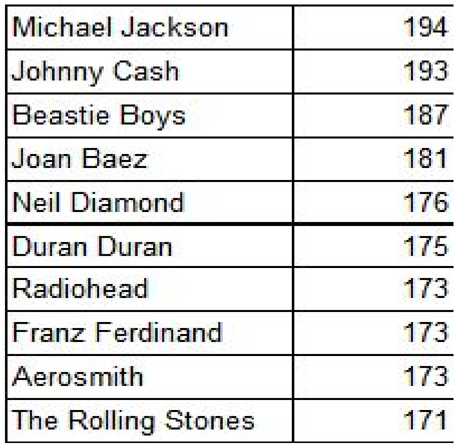
```
.map{case (year, title, art, ...) => (year.toFloat/10).toInt *10,1)}.reduceByKey(_+_)
```

Number of Songs by Decade
(1,000 songs)



Michael Jackson has the most number of songs (194), and all the top-ten artists have more than 170 songs.

Artist with the Most Number of Songs



Michael Jackson	194
Johnny Cash	193
Beastie Boys	187
Joan Baez	181
Neil Diamond	176
Duran Duran	175
Radiohead	173
Franz Ferdinand	173
Aerosmith	173
The Rolling Stones	171

[3] We have used pyspark for the project, but in this report used scala notation because it is more succinct.

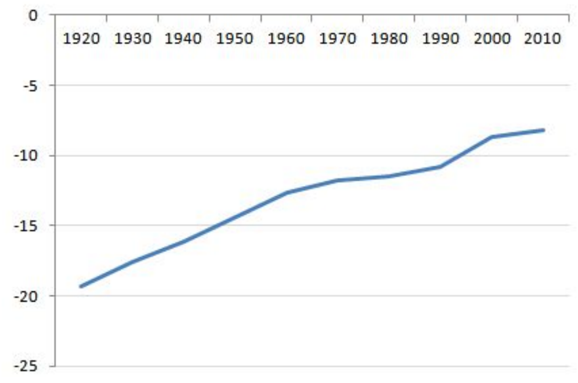
Our next goal was to find trends in song statistics by decade. We wanted to average the statistics by decade, but writing MapReduce scripts to get the means seemed challenging at first. After some iteration, the following simple script did the trick.

MapReduce Script:

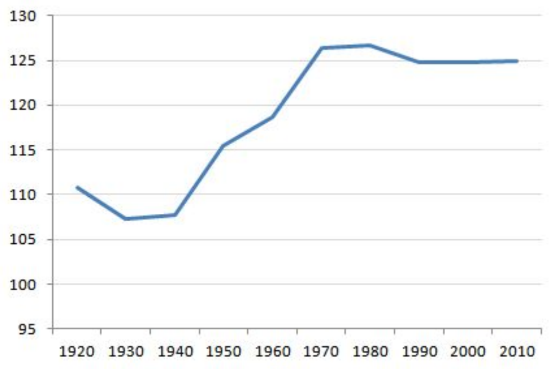
```
.map{case (k, v) => (k, v, 1)}   // assign counter
.reduceByKey((x, y) => (x._1+y._1, x._2+y._2))   // sum values and counter separately
.map{case (k, v, c) => (k, v.toFloat/c)}   // divide the sum by counter
```

Below is the average song statistics by decade. We can see that the songs have been getting louder, faster, and hotter. Average duration also has increased from less than three and a half minutes in 1950s to over four minutes after the 90s.

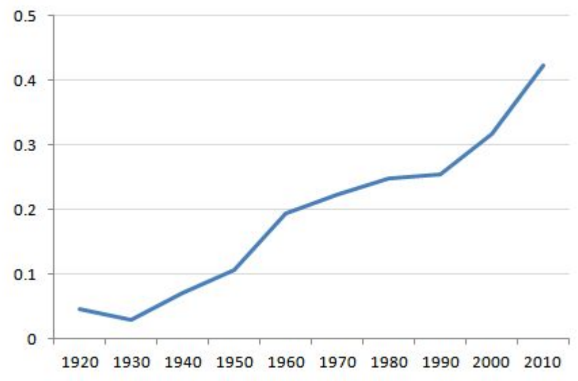
Average Loudness by Decade



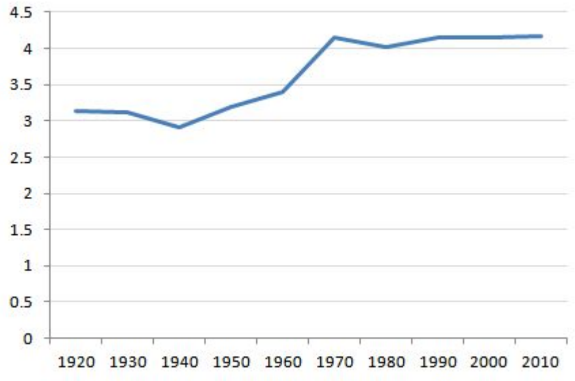
Average Tempo by Decade



Average Hotness by Decade

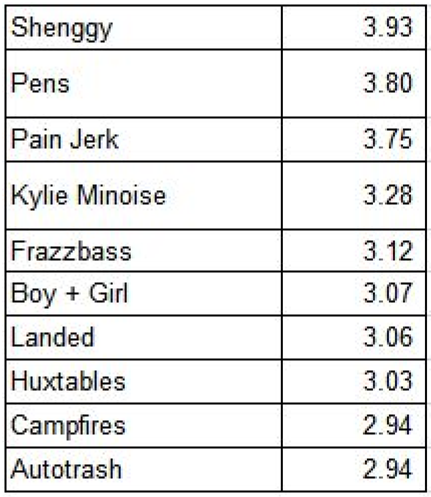


Average Duration by Decade



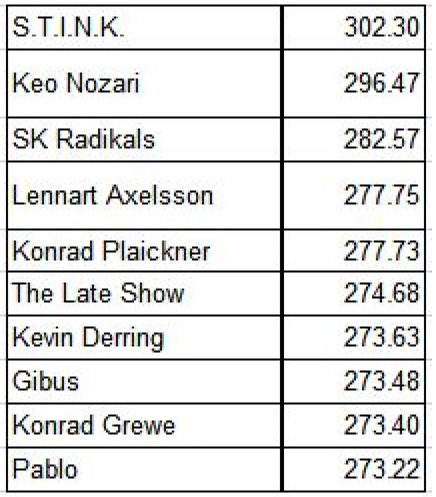
Our last exploration was to retrieve top-ten artists using various criteria. We leveraged the same MapReduce averaging method and then sorted the result to get the top-ten.

Top-Ten Artists by Avg. Loudness



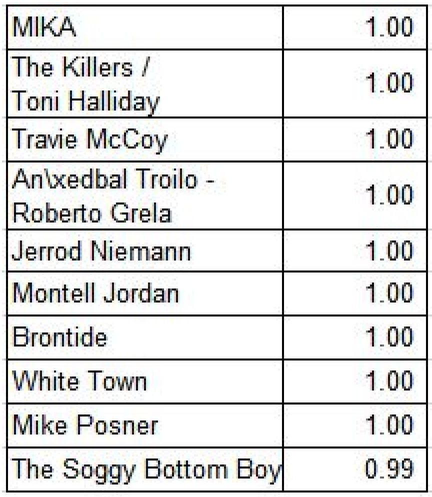
Shenggy	3.93
Pens	3.80
Pain Jerk	3.75
Kylie Minoise	3.28
Frazzbass	3.12
Boy + Girl	3.07
Landed	3.06
Huxtables	3.03
Campfires	2.94
Autotrash	2.94

Top-Ten Artists by Avg. Tempo



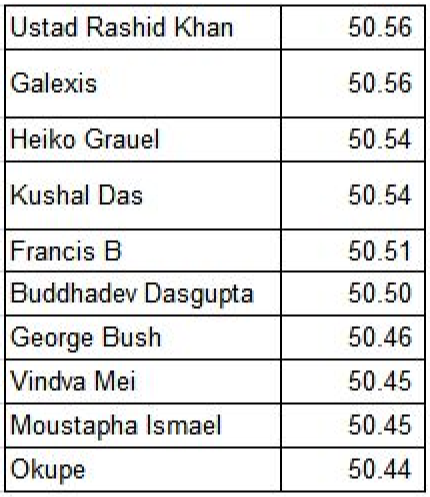
S.T.I.N.K.	302.30
Keo Nozari	296.47
SK Radikals	282.57
Lennart Axelsson	277.75
Konrad Plaickner	277.73
The Late Show	274.68
Kevin Derring	273.63
Gibus	273.48
Konrad Grewe	273.40
Pablo	273.22

Top-Ten Artists by Avg. Hotness



MIKA	1.00
The Killers / Toni Halliday	1.00
Travie McCoy	1.00
An\xedbal Troilo - Roberto Grela	1.00
Jerrod Niemann	1.00
Montell Jordan	1.00
Brontide	1.00
White Town	1.00
Mike Posner	1.00
The Soggy Bottom Boy	0.99

Top-Ten Artists by Avg. Duration

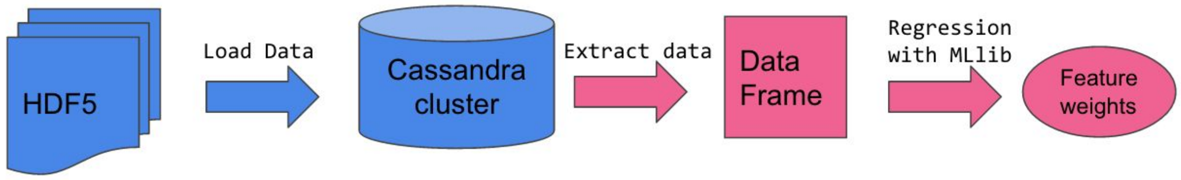


Ustad Rashid Khan	50.56
Galexis	50.56
Heiko Grauel	50.54
Kushal Das	50.54
Francis B	50.51
Buddhadev Dasgupta	50.50
George Bush	50.46
Vindva Mei	50.45
Moustapha Ismael	50.45
Okupe	50.44

b. Statistical Analysis with Cassandra and MLlib

Using the Spark environment and our powerful cluster configuration, we conveniently built a pipeline with Cassandra and MLlib to process and gain insights from the dataset. In this section we try to answer the question: What are the best predictors for song hotness? (Song hotness is a value from 0 to 1 and measures how popular the song was at the time of collection.)

Process diagram

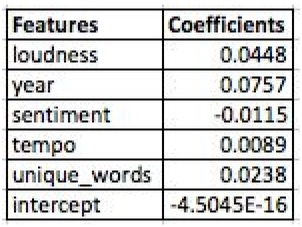


We first created a Cassandra database on our cluster. The database was split among 5 nodes. We used a Network topology strategy with a replication factor of 3. The data was then loaded with a Python script. The script opened each HDF5 file and used a CQL query to insert the data sequentially.

In a second step, we built a Pyspark script to load the Cassandra data into a RDD and converted it to a data frame. The selected fields were "loudness","year","sentiment","tempo" and "unique_words". We only kept data for rows with non empty values. We also dropped the entry with year=0. The resulting data set was about 130,000 songs.

We then took advantage of the MLlib library to manipulate the data frame. In order to have a good regression, we needed to scale the features. MLlib provided the Standard Scala library as a convenient tool to do this. We then initiated a model of class LinearRegressionWithSGD and we fit our data. The whole process took 32 seconds on our cluster.

MLlib Linear Regression model weights



Features	Coefficients
loudness	0.0448
year	0.0757
sentiment	-0.0115
tempo	0.0089
unique_words	0.0238
intercept	-4.5045E-16

c. Sentiment and Lexical Analysis

With the dataset having been pre-processed and loaded into a Cassandra database, along with the complimentary musiXmatch (MXM) dataset that is provided in a SQLite database, one of our goals was to perform some analysis on the lyrics of the dataset. The idea was to perform a sentiment analysis on all the songs with lyrics as well as to calculate the lexical diversity for those songs. Having those values available, we could try to see how they vary across different genres and whether there were any interesting trends.

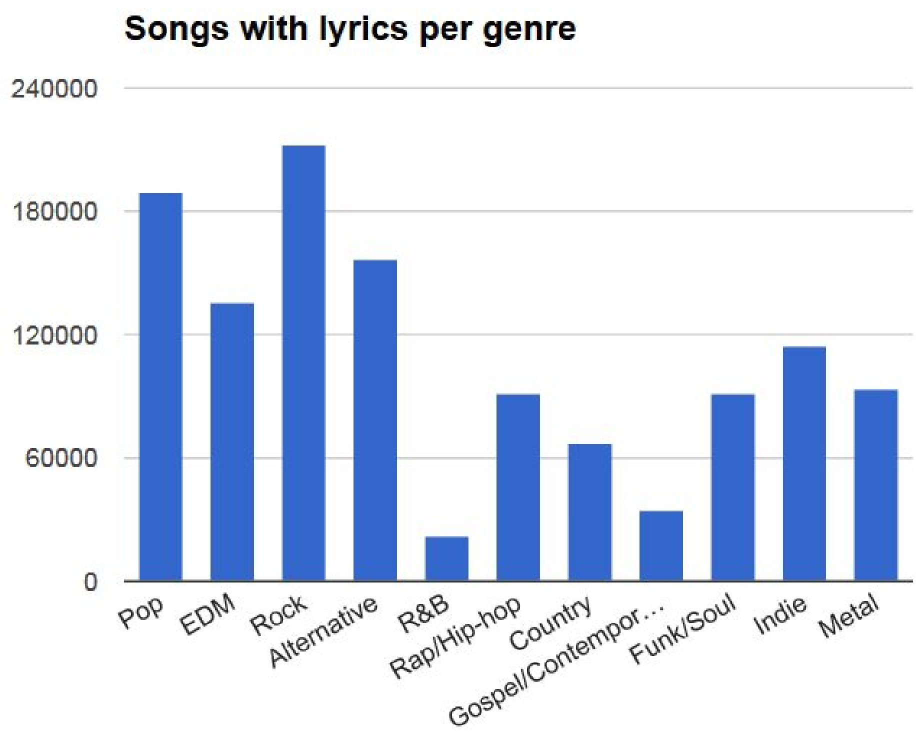
However, the nature of the lyrics dataset as well as some of our chosen technology provided us with some challenges. The MXM dataset only has lyrics available for 237,662 tracks from the MSD dataset. Although the MXM team was able to resolve lyrics for over 77% of the MSD, only about 24% are available for analysis. Some of this is due to restrictions such as copyright, or tracks being instrumental or duplicates. Another issue with the MXM dataset is that it does not provide full lyrics for the songs available, rather it provides lyrics in a bag-of-words format for the top 5000 words across the set. Moreover, the words used were all stemmed prior to their entry in the bag-of-words.

Another challenge was determining what genre to associate the song to. In fact, one of the open tasks on the MSD page for project ideas is for automatic tagging of songs, which can include genre/mood detection. One of the fields available in the MSD for each entry is called artist_terms. This field is an array that contains all the different genres that apply to this artist. In the “songs” table of the Cassandra database, the field was represented by comma separated fields. The full list of artist terms has over 7500 entries and can include very specific genres such as “zombie garage” or “acoustic folky stuff”. Given that, the genres used for analysis would need to be greatly simplified in order to get meaningful results. As such, a list of 11 “main genres” was arbitrary selected as follows:

Main Genre	**Artist Terms**
Pop	“pop”
EDM	“dance”, “electronic”
Rock	“rock”
Alternative	“alternative”
R&B	“rnb”
Rap/Hip-hop	“rap”, “hip-hop”
Country	“country”
Gospel/Contemporary	“gospel”, “contemporary”
Funk/Soul	“funk”, “soul”
Indie	“indie”
Metal	“metal”

Due to these choices, the distribution of songs across genres was certain to have duplicate entries because some songs could be considered as both rock and alternative or both pop and electronic and so forth.

In order to make the analysis through the database simpler, a boolean field was added to the main songs table for each of the genres and if the artist terms were a match for the genre then that flag is set to true otherwise it is set to false. The distribution of songs per genre ended up as follows:



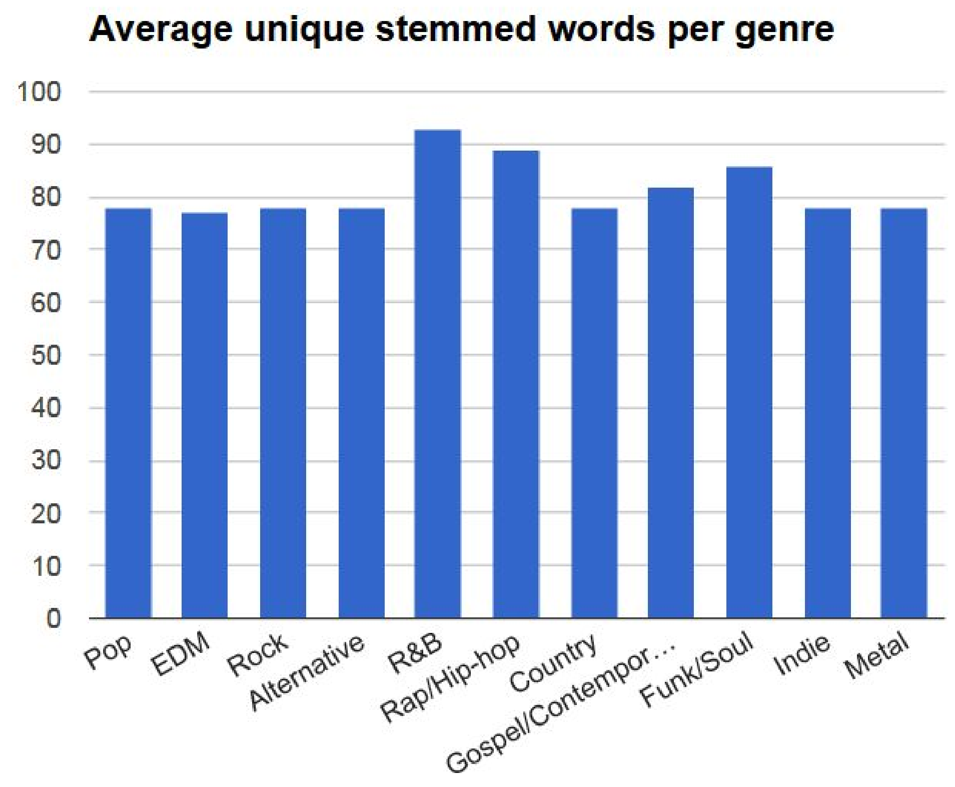
Unsurprisingly, rock and pop are the two most popular genres whereas R&B and Gospel/Contemporary have much fewer matches.

The sentiment score was calculated by using TextBlob on the bag-of-words for each song and a unique (stemmed) word count was used for lexical diversity given that each song has the same baseline 5000 word source. These fields were also added to the songs table in the Cassandra database.

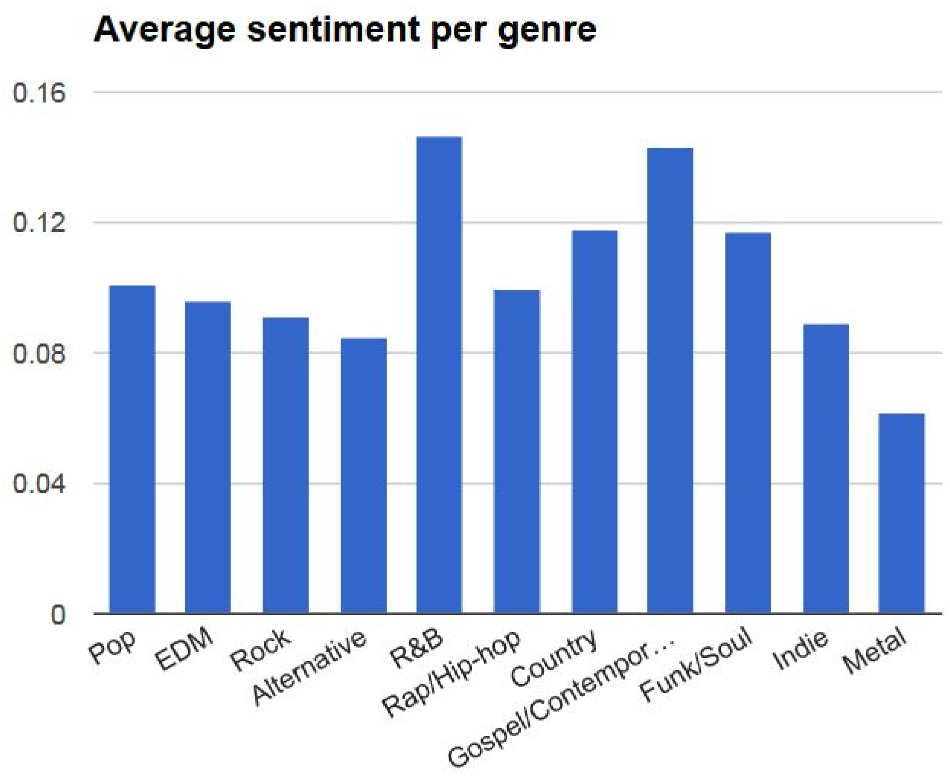
One of the hypotheses we had was that for the unique word count, a genre such as rap/hip-hop should have the highest score given the wordiness of rap songs whereas EDM should have a lower score since many of these songs have much less words. Similarly, for the sentiment score it would be expected that a genre such as gospel would have the highest score due to its religious connotation whereas metal would have a lower score given the darker themes such as death.

However, another challenge faced here is that aggregations for CQL are only available as of version 2.2 of Cassandra whereas we only had version 2.1. Therefore, calculating averages for different scores had to be done in a script that would access the database. Although this wasn't too much more effort, it would have been nicer to do direct operations.

Ultimately the results seem to have generally confirmed the idea behind the hypotheses as is seen in the charts below:



Perhaps surprisingly here, R&B beats out rap/hip-hop in terms of average unique words and several genres seem to have an almost equally low amount of unique words. The EDM umbrella genre does not seem to be as low as expected and this could be due to the fact that lyricless songs are not included in the dataset.



In terms of the sentiment, our expectations are not far off either although R&B appears as having the highest average sentiment.

4. Clustering Analysis

a. Mission

One of our main missions when exploring the MSD dataset was to design a simple yet effective music recommendation application that can identify and output similar songs a user likes based on a well-liked song that is inputted by the user. According to previous work and research, recommendation systems are built by performing a manual tagging of genres to songs (or artists) and then using a classifier like Naive Bayes to predict genres given certain song attributes. Then, similar songs falling into the same genre as the inputted song are recommended to the user. Unfortunately, manually training the data is very time-consuming and expensive. Another tried approach is collaborative filtering but there can be significant biases especially when people give wrong ratings for a song and there is a lot of missing or insufficient data. Thus, for our project we wanted to build a music recommendation system by using a clustering algorithm to group similar songs together.

b. Algorithm

In order to accomplish this task, we first had to categorize all the songs into k distinct groups, where k is the optimal level of classes that well separates one group of songs from another. Since each cluster has a unique distribution representing the song features, it should be relatively simple to predict the cluster of new songs that were not included in the MSD dataset. Once we know each song's cluster label and we store the aggregate track and cluster information in a database or file system, we can efficiently query songs that have the same cluster label as the user entered song.

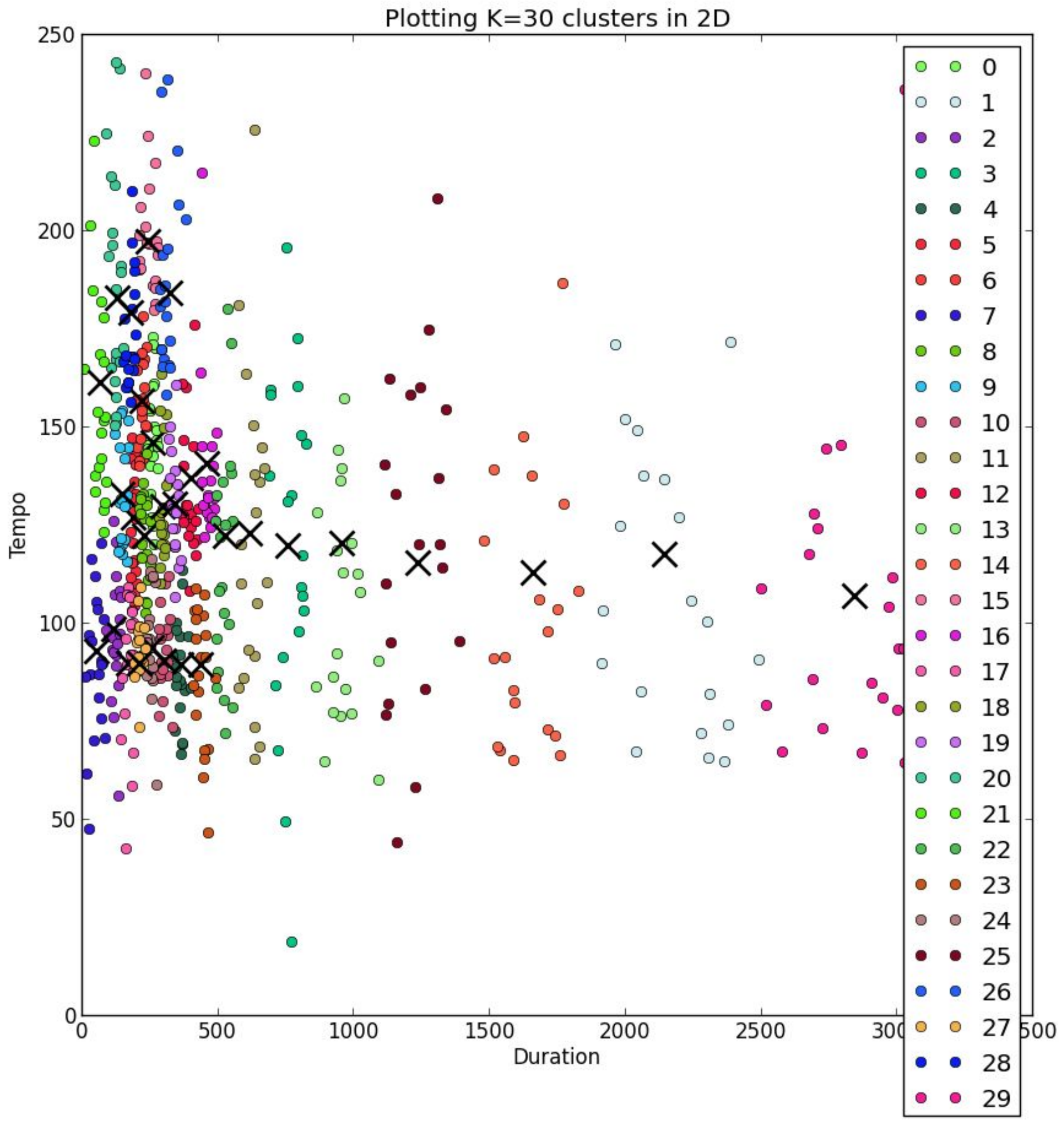
Feature Selection:
Much past research has been done on selecting song features that are very specific to a song such as timbral texture, rhythmic and pitch content. However, using these sparse features to categorize songs into genres causes overfitting of the model to the training data. This may be due to the fact that songs within a genre can have significantly different beats and rhythmic patterns. For this reason, we did not want to classify songs based solely on inherent features of a track. Instead, we decided to use a combination of subjective and objective song features. For the sake of simplicity and having a huge corpus of complete data, we chose the following 5 track attributes:

1. Loudness (deciBel) - the "dynamics of the song". The amount of volume of the overall track.
2. Tempo (beats per minute)
3. Time Signature - how long each note is played for
4. Duration (seconds)
5. Key - the tonic note and chord which gives a subjective sense of arrival and rest

The time signature in addition to tempo contextualizes the notes played in each segment of the song.

Classifiers:
First, we tried Gaussian Mixture Models with k=25 clusters to categorize the 1 million songs; however, because the model does soft classification, most songs were predicted to be in 1 cluster. Next, we tried to fit a k-means clustering model, a hard classifier, on the songs with the aim of partitioning n observations into k clusters in which each observation belongs to the cluster with the nearest mean. Before we fit our classifier, we need to calculate the Within Set Sum Square Error (WSSSE) by testing with different values of k. Ideally, you choose the number of components (k) when the WSSSE value does not significantly decrease as k increases; so for our dataset, the k value was 30.

K-means Clustering Plot on 2 Song Features

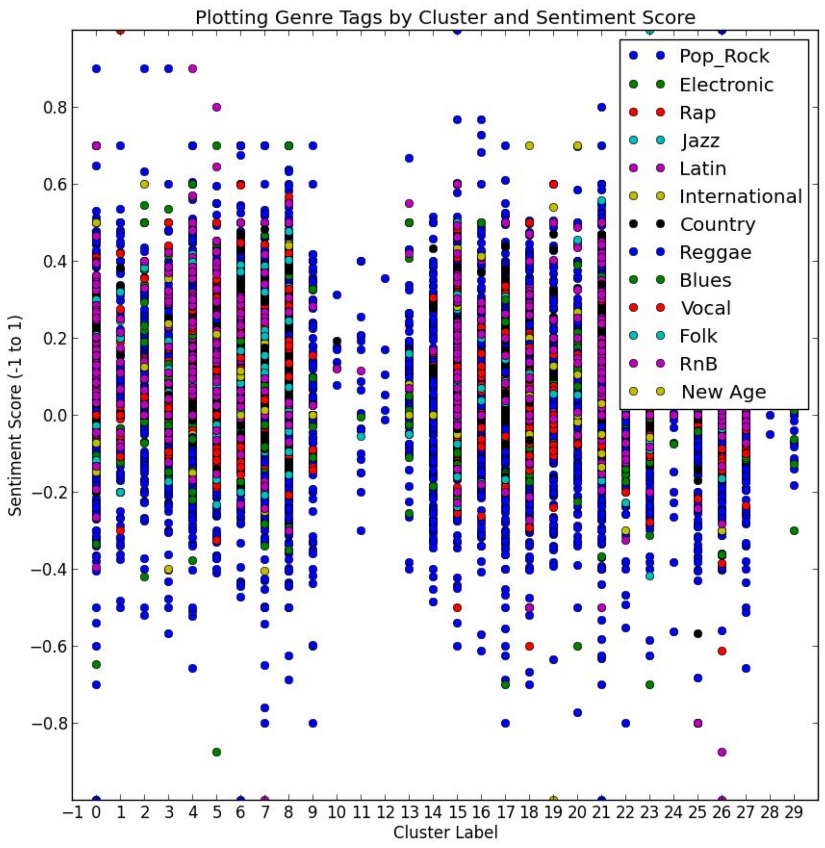


c. Prediction

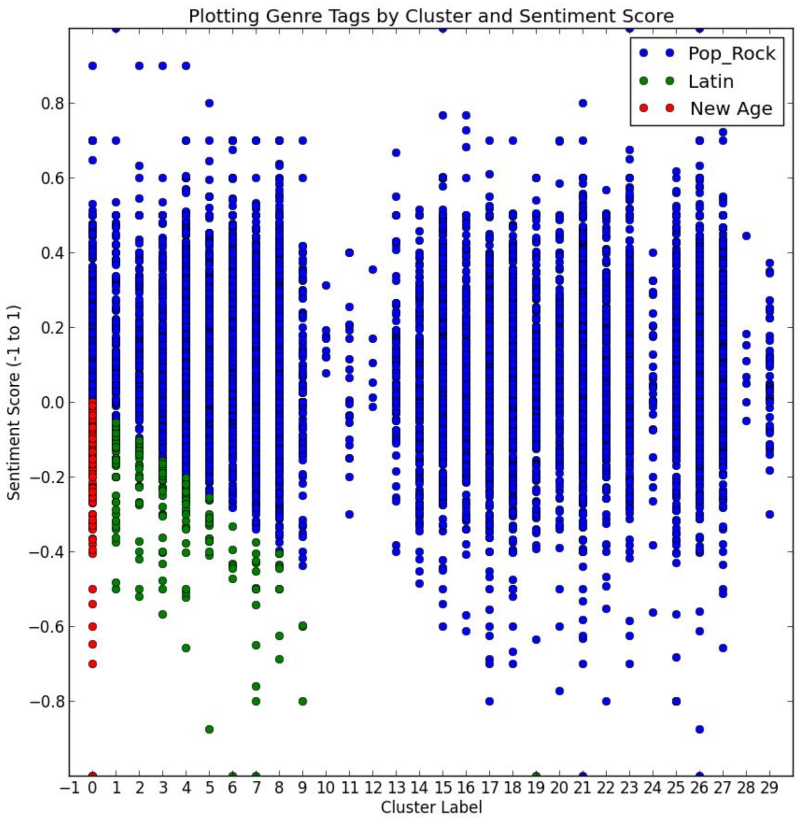
After categorizing songs into 30 clusters and performing sentiment analysis to get a score between -1 and 1 (representing the emotion/mood of the song), we wanted to predict the genre of song given the cluster label and lyric sentiment score. Each cluster can been seen as a summary of its 5 song attributes, to which the sentiment score adds another important predictor of the dependent variable (genre label). We used the AllMusic Top Genre Dataset (Top MAGD), which contains 13 categorical variable genres like Pop/Rock, Electronic, and New Age, and first converted each label to integers ranging from 0-13. Since this dataset contains only a subset of the MSD, we performed our prediction analysis only on songs that are in both the MSD and Top MAGD. We also split 90% of the available data to be training and the rest as test. Next, we used the MLlib Logistic Regression classifier to fit the training data to the model. Lastly, we collected and organized the predictions and actual genre labels.

With this approach, we achieved an accuracy of 72.26 % on the test data but the classifier was only able to class 3 of the most popular genres out of the 13 genres. This may be due to the classifier not being strong enough, unequal data for each genre label, or choosing predictors that are not strongly correlated with classifying the label.

Distribution of Actual Genre Labels by Cluster and Sentiment Score



Distribution of Predicted Genre Labels by Cluster and Sentiment Score



5. Conclusion and Future Work

Our analysis was a good (and fun) exploration of the MSD dataset using various tools that we learnt and got to use throughout the course. While most of our work was quite fulfilling with respect to getting the answers that we were looking for, we faced a number of limitations from a few fronts:

- The composition of the dataset (e.g. the HDF5 format that require initial data extraction into Cassandra; how the songs' genres are set, or the variant use of "artist terms" to classify their songs)
- Our cluster setup (e.g. limitation of Cassandra 2.1 in comparison to Cassandra 2.2)
- Available libraries (e.g. PySpark not yet supporting Twitter streaming)

The occasional frustration notwithstanding, this was a great learning experience for how we can not only work with this MSD dataset in particular, but also any other dataset (structured or otherwise) in the future. With this in mind, and with respect to the MSD dataset, we think the following future work items would add great value to our current analysis:

- Billboard discriminator: Predicting whether a certain song can make it to the Billboard based on song statistics and lyrics.
- Twitter stream integration: If a tweet has hashtags or mentions that happen to be a song title or an artist name, lookup and show info about the song along with the hashtag/mention.
- Plagiarism detection: Find songs whose lyrics or bars, beats and fade composition are quite similar to each other.
- Clustering and prediction improvements: Improve the clustering algorithm and accuracy of predictions by assimilating more secondary datasets for sentiment analysis.

6. Appendix

a. Softlayer Cluster Setup (master: muziki)

```
:..........:..........:................:............:..........:
:    id    : hostname :   primary_ip   : datacenter : root_pwd :
:..........:..........:................:............:..........:
: 13696941 :  muziki  : 169.53.141.8   :   sjc01    : HCmDs2vl :
:..........:..........:................:............:..........:
: 13696941 :  muziki1 : 169.53.141.14  :   sjc01    : FuWL5cAh :
:..........:..........:................:............:..........:
: 13696941 :  muziki2 : 169.53.141.7   :   sjc01    : Nyy9rete :
:..........:..........:................:............:..........:
: 13696941 :  muziki3 : 169.53.141.11  :   sjc01    : EXE3x5lP :
:..........:..........:................:............:..........:
: 13696941 :  muziki4 : 198.11.214.179 :   sjc01    : Yd4ncn8h :
:..........:..........:................:............:..........:
```

b. Code repository in GitHub
https://github.com/StephTruong/W251-MillionSong

c. Using Kibana
- Log in to the master node
- Make sure ElasticSearch works by submitting this query to get the count of documents:
  - `curl -X GET 'http://169.53.141.8:9200/msd/just_clusters/_count'`
  - Returns 188,541 songs
- Start Kibana
  - `cd ~`
  - `kibana-4.1.3-linux-x64/bin/kibana`
- Navigate to http://169.53.141.8:5601 (index = msd, type= just_clusters)

d. Out cluster setup notes
https://docs.google.com/document/d/1gf_70CBgR9HSy57r4P7MRJ9Km87ZhLu9WTQR5JcLeqg/edit