

Names: Cian Burke 11456822 Eoghan McMullen 11442938

Programme: CASE4

Module Code: CA4010

Assignment Title: Group 20 Data Mining Report

Submission Date: 30/11/2015

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at <http://www.library.dcu.ie/citing&refguide08.pdf> and/or recommended in the assignment guidelines.

Name: ***Cian Burke***

Name: ***Eoghan McMullen***

Page Content	Page Number
1. Project Outline - Idea & Dataset Description	2
1.1 - Dataset Description	
1.2 - Idea	
1.3 - Note on code	
1.4 - Submission folder content	
2. Dataset Preparation - From Workshop	3
3. Analysing the data	4
4. Algorithm Description	7
5. Results and Analysis of Findings.	9
6. Conclusion	10
7. References	10

1. Project Outline

1.1 Dataset Description

The dataset chosen was the Million Song Dataset. The dataset consists of nearly all of the information available through the Echo Nest API for one million tracks. The full dataset and the subset (1% of the full dataset) are publicly available for download.

1.2 Idea

The goal of this project was to use the Million Song Dataset to predict the popularity of a song defined by an attribute in the dataset (song_hottnesss) and determine how the attributes mentioned below affect this value.

1.3 Note on code

The java program written for this datamining project is very long. Not all of the code was written to perform the prediction, many of the methods were simply used to perform some analysis on sections of the data. To make it easier to correct we will include the line number that the major methods are located on when appropriate. This will save the corrector having to look through all of the code. The code is currently set to run only the prediction algorithm. This will return a list of numbers(described later) to a text file out.txt in the working folder.

1.4 Submission folder content

java program "CreateCsvsByKey.java"

CSVs

- MillionSongData.csv - original csv before applying cleaning (MillionSongData.csv no longer included in submission, submission limit too small, file is 700mb)
- CleanCombinedKeys.csv - millionsongdata.csv after cleaning. i.e. our full dataset.

Folders

- COM - contains java library for reading and writing to csvs
- Text Files mentioned in report - contains all the analysis text files mentioned in the report

Dataset Attributes (Echo Nest API):

- **track_id**: A unique value given to a track as an ID.
- **song_hottnesss(0. - 1)**: Represents the measure of popularity of a song.
- **artist_familiarity(0. - 1)**: This represents a numerical estimation of how familiar an artist is currently across the world. Can be described as the likelihood that any person selected at random will be familiar with or have heard of that artist.

- **artist_hottnesss(0. - 1):** This is a numerical estimation of how hot an artist is currently, i.e. mentions on web pages, play counts of their material.
- **key(0-11 chromatic scale):** The key of the song, following the chromatic scale. Each value here corresponds to a key.
 - ❖ 0 = C
 - ❖ 1 = C-Sharp
 - ❖ 2 = D
 - ❖ 3 = D-Sharp
 - ❖ 4 = E
 - ❖ 5 = F
 - ❖ 6 = F-Sharp
 - ❖ 7 = G
 - ❖ 8 = G-Sharp
 - ❖ 9 = A
 - ❖ 10 = A-Sharp
 - ❖ 11 = B
- **mode(minor or major):** Can be two values, 0 or 1 and determines whether the song is minor(0) or major(1) .
- **loudness(dB):** The overall loudness(how loudly the song was recorded) of the song in decibels(dB).
- **tempo(BPM):** This a measure of the number of beats per minute in a song.

2. Data Preparation

The files in the downloadable dataset were in hdf5 format. Each hdf5 file contained a huge number of attributes for each song in the dataset. The attributes mentioned above needed to be extracted for each song in the dataset. The data needed to be read and stored into a csv file, one attribute per column.

The conversion from HDF5 format to csv format was accomplished using a python script. While performing the conversion we also choose what attributes for each song would be taken. There were many attributes which needed to be discarded as they were not relevant to the prediction we would be attempting to make.

Data Cleaning:

After creating a csv file with all of the attributes mentioned above, cleaning of the data was necessary. Many of the rows in the csv "MillionSongData.csv" were unusable for our prediction.

- Rows with a **song_hottnesss** value of null or zero were removed. If the song had no rating we could not determine its popularity. Any ratings of zero were removed as they could greatly skew our data.

- Some rows had a value of “null” for **song_hottnesss**, these rows were removed.
- We found that the csv contained many duplicate rows(Sometimes as many as 6 counts of the same song). We used the **track_id** attribute to locate and delete duplicates.
- For our prediction we needed each track in the dataset to have a value for **BPM**(beats per minute) we checked for any rows not containing a value for bpm and removed those rows.
- Some **artist_familiarty** entries were invalid i.e. had a value of “null” or “nan”. The rows containing such values were removed as they would be unusable in our prediction

To more easily deal with the different keys in our dataset,(of which there are 12) while cleaning the dataset we decide to split the csv into 12 new csv's, one for each key. This made dealing with each key individually much more simple. using this strategy we could later run methods on individual keys and retrieve results in a much more efficient manner than looping through the entire **MillionSongData.csv**.

After creating the clean 12 key csv's we created a method to merge them into one csv file to give us an overall view of the clean data. This file is called **CleanCombinedKeys.csv**.

The original Million song dataset in HDf5 format was 300GB. After extracting only the attributes we were going to use it shrank to 700MB. After applying our cleaning to the 700MB csv we were left with a 60MB csv, **CleanCombinedKeys.csv**.
The clean csv contains just under 500,000 usable songs for our prediction.

The cleaning of our data was performed using the method **makeKeyCsvs(String originalCSV)**, line 69 this is passed the name of the original CSV. It performs cleaning while creating the 12 key sorted csvs.

3. Analysing the data

KEYS

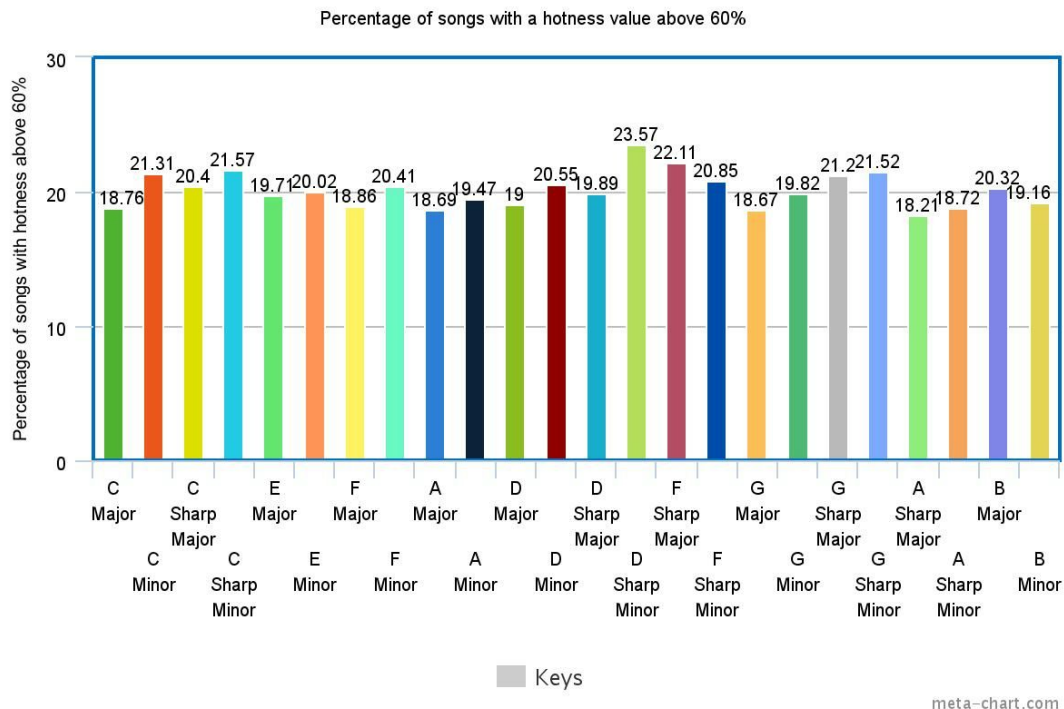
Before making any predictions with our dataset we wanted to ensure that the values we included in the prediction really made a difference to the popularity of a song. This analysis is what made up the bulk of our code(1500+ lines).

There are 24 keys in the dataset (12 with a major and minor for each). We analyzed each key separately to see the distribution of song popularity. This analysis was performed with the method **getHotnessDispersion** Line 673. The method reads in the major and minor songs for a key and returns with a table of the percentage of songs that lie within each hotness category. Below is an example of the data returned for c major. The full data can be seen in **hotnessDispersionEachKey.txt**

C Major:

0 - 20%: 0.76%
 20 - 40%: 40.44%
 40 - 60%: 40.04%
 60 - 80%: 16.93%
 80 - 100%: 1.83%

A table like this was created for each key and each mode(major or Minor). Using these tables we calculated the percentage of songs with a hotness above 60% for each key. This was done to try and determine if the key of the song had an impact on a song's popularity. The results are shown in the bar chart below:



The chart above shows us that the key of a song made little to no difference to the popularity of a song. Above the key with the highest percentage of popular songs in it is D-Sharp-Minor. The difference in the percentage of popularities in each key was so small that we deduced that the key of a song does not affect its popularity. D sharp minor was the most popular key in our dataset but given the percentage difference to the other keys we decided that with some different data the results would be inconsistent. From our findings here we decided that we could not use the key of a song when creating our prediction algorithm.

BPM

When checking to see if the BPM of a song affected its popularity we used a similar approach to checking if a key affected the popularity. Tables were created for each popularity range for each key. A sample below shows that there are more popular songs with a higher BPM.

Low Popularity:

130+ bpm: 35.46%

Low medium Popularity:

130+ bpm: 38.14%

High Medium Popularity:

130+ bpm: 41.74%

High Popularity:

130+ bpm: 44.39%

This data was consistent across nearly all keys. We determined that the BPM did affect song popularity and so could be used in our prediction algorithm. The full findings of BPM can be seen in BPM Distribution.txt. The method **getBPMcounts Line 835** performs the BPM analysis.

Loudness

The **loudness** attribute was verified to be useful to our prediction in the same way that **BPM** was. The method **loudnessCategories Line 992** performs the analysis. A sample of the findings is shown below. The full findings can be seen in **Volume Distribution.txt**

Low Popularity:

Very Loud Volume: 69.45%

Low medium Popularity:

Very Loud Volume: 71.85%

High Medium Popularity:

Very Loud Volume: 78.6%

High Popularity:

Very Loud Volume: 82.09%

It can be seen that as the loudness of the song was increasing that the percentage of popular songs was also increasing. This was consistent across all keys in the dataset. The **Loudness** of song seems to have a direct relationship to a song's popularity.

Artist_Hottness and Artist Familiarity

These are what we considered to be the most influential factors when determining how popular a song would be. We decided to verify this assumption. The verification was performed in a similar manner as the other attributes. For **Artist_Hottness** the full verification can be seen in **Artist Hottness distribution by key.txt**. A sample is shown below. It can be seen that as the popularity of songs is increasing that the **Artist_Hottness** is also increasing. showing a direct relationship between the popularity of a song and an artist's hottness. The method **artistHottnessDistribution Line 1316** was created for this analysis.

Low song Popularity:

High artistHottness: 0.48%

Low medium song Popularity:

High artistHottness: 1.07%

High Medium song Popularity:

High artistHotness: 3.89%

High song Popularity:

High artistHotness: 13.6%

Artist familiarity also had a direct impact of the popularity of a song as can be seen in a sample of data below. The method **artistFamiliarityDistribution** was created to verify that **Artist_familiarity** affected **songhottness**.

Low song Popularity:

Very high Familiarity: 1.42%

Low medium song Popularity:

Very high Familiarity: 2.06%

High Medium song Popularity:

Very high Familiarity: 8.51%

High song Popularity:

Very high Familiarity: 26.1%

It can be seen that as the familiarity of an artist was increasing so was the song's popularity.

Data analysis conclusion

Verifying each attribute that we were going to use for our prediction was a very important part of our data mining. We were able to determine which attributes we would be able to use when creating our prediction algorithm. Without this analysis phase we could have included a song's key and mode in our prediction only to discover that the prediction would very rarely be accurate. We also learnt that the other attributes i.e. bpm, loudness, artist familiarity and artist hotness were directly related to how popular a song was. This was good news as it meant that we might be able to create an accurate prediction.

4. Algorithm Description**Measuring Statistical Dispersion of Data - Workshop**

- ❖ Range
- ❖ Variance
- ❖ Interquartile Range (IQR)

While preparing for our data mining presentation we looked closely at how our data was dispersed. We concluded that only working with data within the quartile ranges would not give us the results we were looking for. We needed to include outliers.

Outliers:

method **getMedianAndQuartileValues** Line 777 was used to identify any outliers. We found that outliers in terms of song hotness were classified as being over 0.93 song_hottnesss. We decided not to remove the outliers from the dataset. We decided that the outliers could help us make our prediction more accurately rather than skew the results. The outliers would be using certain song attributes and we wanted to find out what they were and if the attributes contributed to the song's popularity.

Prediction Algorithm

After performing data analysis on our dataset we concluded that we could use four of our attributes to try and determine if a song would be popular(have a high song hottnesss value).

The prediction algorithm we decided to use was k-Nearest Neighbour. This allowed us to use the four values we determined to be useful to predict if a song would be popular.

The method in our code that performs the prediction is **predict** on line: 1473

To find the Euclidean distance we needed to normalize two of the attributes, **Loudness** and **Tempo(BPM)** so that they ran from 0 - 1. This is performed using the method **Normalize** on line: 1589

After ensuring data was normalized we took a list of what is considered popular songs(at least above 75% hotness) and got the average of each of the four attributes for those popular songs. The method **getPopularAverageofColumn** line: 1535 retrieved these averages.

Then in the predict method the euclidean distance for a song was calculated using this formulae.

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_i - q_i)^2 + \dots + (p_n - q_n)^2}.$$

A snippet of our predict code getting euclidean distance:

```
double aVal = Math.abs(artistHotness - artistHotnessAvg);
double bVal = Math.abs(artistFamiliarity - artistFamiliarityAvg);
double cVal = Math.abs(songVolume - volumeAverage);
double dVal = Math.abs(bpm - bpmAverage);

double euclidean = Math.sqrt((aVal*aVal) +(bVal*bVal) + (cVal*cVal) + (dVal*dVal));
```


Testing the algorithm:

To test the algorithm an arraylist filled with rows of a certain key(e.g. C Major) is supplied to the **predict** method. From this arraylist a second arraylist is generated with 30 random songs using the method **getRandomSongs** line: 1574

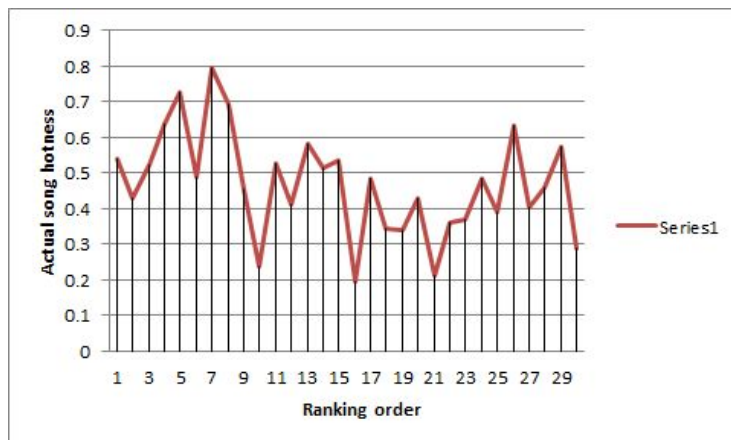
The predict method then prints the songs in order of euclidean distance from lowest to highest. The lower the euclidean value the higher the **song_Hotttness** should be.

5. Results & Findings

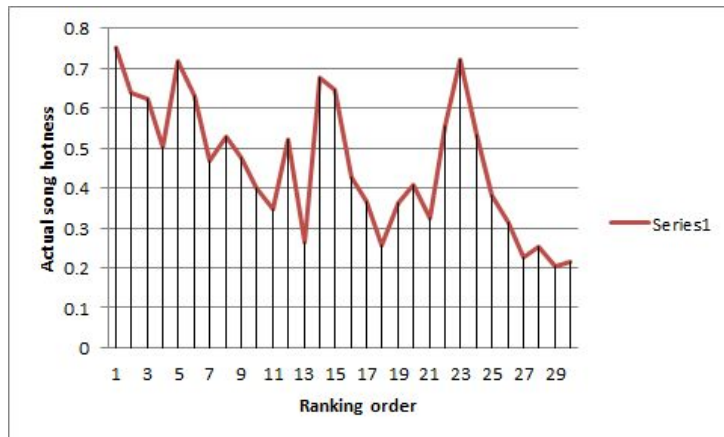
Using the **predict** method we generated text files with a list of songs in ranked order from lowest euclidean value to highest. Along with the euclidean value the actual **Song_Hotttness** value was supplied..

The y axis shows the song's actual popularity(hotttness), the x axis shows the ranking order determined by the euclidean value. According to the euclidean value the most popular song should be on the left and the least popular on the right.

In the first line chart below the popular averages were calculated using songs in the supplied key which had a hotttness rating above 0.75. This did not generate very accurate results. The ranking order seems very inaccurate:






After seeing the above inaccurate results which retrieved the popular averages using only songs with a hotttness rating above 0.75 we decided to use a higher value. In the below chart only songs with a hotttness rating above 0.9 were used to calculate the popular averages which are used in the Euclidean distance calculation. The results appear to be much more accurate.



6. Conclusion

Based off of our prediction results we have concluded that creating a very accurate prediction algorithm for determining whether or not a song will be popular is not possible. The four attributes we used “Loudness, artist_hotness, artist_familiarity, Tempo” did not give us enough information to determine if a song would be popular. For making a more accurate prediction in the future we would include more attributes like “year” “genre” “country”. With more information we believe a more accurate prediction could be made. While analysing our dataset we discovered that the key of a song made very little difference to the popularity of a song. We initially thought that the “key” would be one of the most influential attributes when trying to determine how popular a song would be. We did find that for our dataset that when calculating the euclidean distance that using popular averages with a higher “song_hottness” value made our prediction slightly more accurate.

7. References

- [1] Bertin-Mahieux, T., Ellis, D., Whitman, B. and Lamere, P. (2011). *The Million Song Dataset*. [online] Thierry Bertin-Mahieux, pp.1-6. Available at: <https://www.fbi.h-da.de/fileadmin/personal/i.schestag/Allgemein/BertEWL11-msd.pdf> [Accessed 1 Dec. 2015].
- [2]  Nest, E. (2015). *Echo Nest API Overview — The Echo Nest 4.2 documentation*. [online] Developer.echonest.com. Available at: <http://developer.echonest.com/docs/v4/index.html> [Accessed 1 Dec. 2015].
- [3]  Labrosa.ee.columbia.edu, (2015). *Million Song Dataset | scaling MIR research*. [online] Available at: <http://labrosa.ee.columbia.edu/millionsong/> [Accessed 1 Dec. 2015].
- [4]  GitHub, (2015). *rcrdclub/mm-songs-db-tools*. [online] Available at: <https://github.com/rcrdclub/mm-songs-db-tools> [Accessed 1 Dec. 2015].
- [5] Openscv.sourceforge.net, (2015). *opencv 3.6 API*. [online] Available at: <http://opencv.sourceforge.net/apidocs/> [Accessed 1 Dec. 2015].