Million Song Database Analysis

**By the *Dancing Dads* aka: John Butcher, Austin Poulton, Ian McNair and Barrington Trim**

Submitted for: Big Data Processing (ECS765P) Coursework Assignment 2

18 December 2015

# Problem Statement

The objective of this group coursework assignment was to demonstrate large data set processing and analysis techniques in a parallel computation environment using Apache Hadoop and/or Spark.

Perform detailed analysis on the Million Song Data using Spark and Scala to understand

* The nature of the data
* Key correlations
* Key insights

The analysis was performed using the following techniques

* 1. Spearman correlation key fields
* 2. Song Tempo and Song 'Hotness Heat map
* 3. Linear Regression classifier

The Million Song Dataset can be downloaded at <http://labrosa.ee.columbia.edu/millionsong/>

# Approach

The following approach was used

* 1. High-level data analysis to understand data structure and to inform further analysis
* 2. Set-up development environment
* 3. Develop and unit test Spark code to analyze the data
* 4. Review results to feed into further analysis and development

# Raw Dataset Analysis

From an initial inspection of the dataset comprised on Hadoop, the data comprised of 26 csv files containing a total of 1 million data records.

The dataset itself had no metadata, so this had to be determined before detailed analysis could commence. Meta data was constructed by comparing the data to various examples on the MSD and EchoNest sites and looking at field position.

A high-level summary analysis was performed on the fields in order to understand data quality and completeness, key observations are listed below

Understanding Track Data

* Each field per row was contained within ““ , e.g. *"1","Progressively Funky","401198","0.0","*   
  - this would require a well-thought “regex” to select the features into correct columns for RDDs
* 1 million total records exist within the data

Feature Analysis

* 50% of Artist latitude and longitude were missing
* Dance ability was meaningless – all values where 0
* Musical features such as mode, key and time signature whilst looking promising provided very little value and differentiation within the dataset. The values were so aggregated/coarse that they provided almost no musical analysis value.

# Implementation Approach

## Development Environment

The team adopted a collaborative development approach by hosting Scala source code, analysis artefacts and the report itself on the Queen Mary EECS Enterprise Github. The project repository is available here: <https://github.research.its.qmul.ac.uk/ec15541/million-song>

Whilst a Maven build was provided, the team favoured a Scala Build Tool (SBT) build process. The team developed their own SBT build, including dependencies to support the specific development and testing requirements. In particular, the Breeze library was used for matrix and vector operations to support regression based machine learning whilst ScalaTest was used for unit/integration testing. The project dependencies are listed below:

**val** *mockitoAll* = "org.mockito" % "mockito-all" % Version.*mockito***val** *scalaTest* = "org.scalatest" %% "scalatest" % Version.*scalaTest***val** *sparkCore* = "org.apache.spark" % "spark-core\_2.10" % Version.*sparkCore* //% "provided"  
**val** *breezeCore* = "org.scalanlp" %% "breeze" % Version.*breeze***val** *breezeNative* = "org.scalanlp" %% "breeze-natives" % Version.*breeze***val** *breezeViz* = "org.scalanlp" %% "breeze-viz" % Version.*breeze*

### Testing

The team believed in a principled approach to testing the Spark drivers and relied on a ScalaTest library for unit testing and local Spark cluster integration testing. This allowed the team to develop and test against a real subset of the million song dataset containing 2000 tracks hosted in RDDs on a local (single node) Spark cluster without adversely effective the moonshot cluster. This allowed driver and all supporting functionality to be developed and tested in isolation before being promoted and executed on the Moonshot cluster against the entire dataset.

The test tracks were loaded from /src/test/resources/TestTracks.csv by the BaseDriverSpec class which provided common functions to all descendent unit test suites.

## A Framework for Spark Analysis of MSD

* Discussion of Track encapsulation and companion object
* Discussion of Model trait and classification/regression models
* tracks, objects, how was the data cleaned\normalized in the framework etc
* Pattern for Spark driver objects
* Collaborative working
* Code and making into generic class
* Key design principles – e.g. raw tracks into object structure,
* Data cleansing strategy
* Unit Testing

# Data Analysis and Results

## Feature Correlation Analysis

There are a number of numeric features within the Million Song Dataset that we expected would demonstrate some correlation. To explore this hypothesis we developed a correlation calculation based on the Spearman approach which makes no assumption about the distribution of the the dataset. This is a single value between -1 and 1 where a value near either limit shows a correlation: as one feature value increases, so does the other (or decreases). A value close to 0 indicates no correlation.

### Tracks

The first track relationship we chose was between song tempo and song “hotness”.

An initial Scala object called “BasicCorrelations” was coded [without much consideration for performance] which performed the following:

1. Load RDD from Million Song Dataset mapped to named 'Track' features (see Track.scala)
2. Map an RDD of non-zero tempos as a tuple of track id and tempo (track7IDd , tempo)
3. Map an RDD of non-zero hotnesses as a tuple of track id and hotness (track7Id, songHotness)
4. Create an RDD join of tempo and hotness by track id -> (track7ID, (tempo,songHotness))
5. Persist this RDD
6. Project the RDD several times to collect numeric values for calculations (e.g. sum of tempo \* hotness)
7. Calculate the Spearman correlation and print to console *(No result data was required to be saved back to HDFS)*

This was later improved by a re-write that employed an accumulator to aggregate the column values per RDD partition and then combine each to a single summed set of values. This removed the need for the intermediate RDDs and proved to be much faster. In tests, the first correlation job completed in 319146 milliseconds from the original track mapping to completion whereas the second job using accumulators completed in just 63020 milliseconds. To gain these metrics, a simple grab of “System.currentTimeMillis” was employed.

Unfortunately, the actual result of the correlation shows almost zero correlation between song hotness and tempo:

#DancingDads : Spearman correlation, r(tempo, hotness) = **0.026070182686774128** in 63020 milliseconds

A later correlation between year and hotness was explored but this again showed nothing exciting:

#DancingDads : Spearman correlation, r(year, hotness) = **0.09286060647056213**

…but wait! We then looked at an R scatter plot of Artist familiarity against Song hotness and saw a vague-looking linear relationship so gave that one a spin around the block:

#DancingDads : Spearman correlation, r(familiarity, hotness) = **0.47956310486659387**

Thus we see almost half-way to a decent, positive correlation between how familiar and artist is and their song “hotness”.

In total, we ran correlation checks against the following:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Tempo** | **Song Hotness** | **Familiarity** | **Year** |
| **Tempo** | 1 |  |  |  |
| **Song Hotness** | 0.026 | 1 |  |  |
| **Familiarity** | 0.028 | **0.480** | 1 |  |
| **Year** | 0.016 | 0.093 | 0.023 | 1 |

The next section takes a deeper dive into the seemingly uninteresting tempo/hotness correlation to uncover some hidden patterns.

### Song Tempo and Song 'Hotness' Heat Map

The table below is a binned scatterplot between song hotness (x, columns) and song tempo (y, rows)

|  | **Song Hotness** | | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Song Tempo** |  | **0.1** | **0.2** | **0.3** | **0.4** | **0.5** | **0.6** | **0.7** | **0.8** | **0.9** | **1** |
| **15** | 0 | 0 | 6 | 1 | 7 | 2 | 3 | 1 | 1 | 0 |
| **30** | 0 | 2 | 57 | 42 | 45 | 32 | 13 | 8 | 0 | 0 |
| **45** | 0 | 14 | 435 | 322 | 353 | 283 | 145 | 65 | 22 | 1 |
| **60** | 0 | 38 | 908 | 720 | 760 | 651 | 345 | 158 | 46 | 4 |
| **75** | 0 | 106 | 3332 | 2538 | 2939 | 2336 | 1505 | 605 | 176 | 33 |
| **90** | 0 | 341 | 9938 | 8512 | 9892 | 8658 | 5472 | 2464 | 741 | 131 |
| **105** | 0 | 487 | 16438 | 13889 | 16646 | 14444 | 9414 | 4052 | 1191 | 204 |
| **120** | 0 | 521 | 14612 | 12362 | 14908 | 12752 | 8185 | 3877 | 1119 | 203 |
| **135** | 0 | 750 | 18108 | 15856 | 18340 | 15441 | 10119 | 4764 | 1417 | 233 |
| **150** | 0 | 437 | 11887 | 10541 | 12440 | 10786 | 7195 | 3423 | 978 | 146 |
| **165** | 0 | 306 | 8468 | 7549 | 9180 | 8327 | 5537 | 2580 | 815 | 144 |
| **180** | 0 | 216 | 5228 | 4596 | 5909 | 5168 | 3332 | 1558 | 407 | 59 |
| **195** | 0 | 118 | 3226 | 2994 | 3609 | 3193 | 2154 | 887 | 200 | 31 |
| **210** | 0 | 58 | 1755 | 1596 | 2014 | 1861 | 1216 | 545 | 117 | 23 |
| **225** | 0 | 32 | 863 | 807 | 1103 | 1006 | 647 | 273 | 70 | 12 |
| **240** | 0 | 16 | 411 | 394 | 477 | 491 | 272 | 119 | 35 | 3 |
| **255** | 0 | 11 | 203 | 171 | 179 | 178 | 84 | 50 | 7 | 1 |
| **270** | 0 | 0 | 6 | 5 | 9 | 5 | 6 | 1 | 1 | 0 |
| **285** | 0 | 1 | 2 | 1 | 3 | 0 | 2 | 0 | 0 | 0 |
| **300** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **315** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Table 2:** Tempo vs Hotness Heatmap

## Linear Regression

A linear regression model was developed to experiment with prediction of track and artist features from the MSD. The model training logic is encapsulated in LinearRegressor and prediction in the LinearRegressionModel. The objective of linear regression is to minimize the r*esidual sum of squares* (RSS) or sum of squared errors (SSE) between predicated and actual continuous values:

where **w** is the vector of weights or coefficients,

**x** is the vector of features values (inputs) and

y is the actual values (labels).

Model training was based on iterating over the training dataset and computing the cost or error on each iteration:

where λ acts as a configurable regulariser to control model over fitting.

A gradient descent optimization process was used for training the model weights. Costs were computed for each iteration over the training set and adjusted in the direction of the minima (where the differentiated cost function = 0) using a gradient descent approach

where α is the learning rate or gradient descent rate and *m* = the number of features

**Experimentation with the 2000 track test subset**

The scatter plot below shows artist hotness and familiarity along with an Excel trend line (linear regression model). The trend line is based on fit against the artists from the entire 2000 track subset.

We then used the linear regressor to train the model weights against a 60% split of the subset. The remaining 40% was used to evaluate the trained model by making predictions and measure the error between predictions and actual values.

Using the target model (based on Excel linear trend with coefficients, w0= 0.1854 and w1= 0.4028 the following error rates were achieved against the training and test sets.

Training error 1.7082741133758377

Test error 1.0919065052694035

The chart below shows the model error and coefficients using our linear regressor with gradient descent and regularization.

# **5. Summary and Conclusions**

This group exercise has been challenging on a number of fronts. Essentially, the collaboration has worked well with each member being able to contribute to the project. Most of the team were inexperienced in the use of GitHub and so this has been a valuable skill to acquire along with an increased awareness of Apache Spark and the Scala language. We have also used SBT and have learned of its importance in a Scala development environment.

Some lessons have been learned with regards to the performance aspects of running jobs in a Spark cluster environment. Whilst RDDs provide a useful abstraction from the underlying parallelism, the method of processing large data sets still needs to be considered. In particular, the initial correlation task where many RDDs were created and an expensive join performed, highlight the need to look for ways to derive intermediate and final data in more efficient ways (in our case through accumulators).

The Million Song Dataset has proved to be reluctant to give up its secrets. The validity of features such as “hotness” are brought into question and musical data perhaps requires more domain knowledge than any of the group possess. Regression exercises too have shown the data to be a difficult beast for predicting continuous data. This may sound a bit depressing but it must be re-stated that in general, the learning outcomes of the project outweigh the disappointments in data analysis. We all now have an appreciation of code-control, collaboration tools, IDEs and understand the potential of Scala, SBT and Apache Spark.

Key observations:

* It is easy to “slay” a Spark cluster
* Agree on a development environment (IDEs, build tools, unit-test) as early as possible and make sure everyone is up-to-speed on the technology
* Data is Data; make sure you can turn it into information in spite of gaps and inconsistencies
* Background knowledge in parallel processing is essential
* If you’re a hot artist, your songs are likely to be hot too

Directions for future efforts (given more time):

* Investigate the Geospatial aspect of the data (we started on this but ran out of time)
* More effort on filling in the gaps (for example extrapolating coordinates from artist location text)
* Classification exercises (can we predict the year or cluster by genre)

# **6. References**

Apache Foundation (2015) *Spark Programming Guide*.[online] Available at: <http://spark.apache.org/docs/latest/programming-guide.html> [Accessed 5 December 2015].

ScalaNLP (2015) *Breeze Quick start* Numeric Processing for Scala [online] Avaialble at <https://github.com/scalanlp/breeze/wiki/Quickstart> [Accessed 10 December 2015].

Murphy, K (2012) *Machine Learning – A Probabilistic Perspective* MIT Press, Cambridge, Mass. USA