DJ Tini Data

The small scale implementation of DJ Byg Data

Diego Ballesteros, Trajche Kralev, Tribhuvanesh Orekondy

October 28, 2014

1 Introduction

The general aim of this project is to develop a system able to produce information out of data extracted from songs. This is the general aim of the research field known as Music Information Retrieval (MIR), this field has been active for quite some time [6] and its applications can be seen in several commercial products such as Pandora or Spotify [4].

A great portion of the activity in MIR deals with song meta-data and audio features [5]. However, in this project we focus on lyrics as the main source of data. The main task that we pursue is to answer the following question:

"Is it possible to determine the genre of a song based solely on its lyrics?"

In order to accomplish this, we first explored how to extract genre information from the songs' lyrics and then evaluated the quality of the judgments. Lastly we made the developed methods available as a web application where arbitrary lyrics can be input in order to obtain a genre prediction.

The report is organized as follows, first we present the input datasets used for the task and their respective preprocessing and manipulation. We then proceed to briefly explain the computing and analysis techniques chosen for the task. In the third section we present details of the implementation of these techniques. Finally the obtained results and the performance of the algorithms is discussed.

2 Data

2.1 Description

In order to carry out the previously stated task, we selected the Million Songs Dataset (MSD) [1]. This dataset is a free collection of audio features and metadata for a million of tracks, it was collected by the LabROSA at Columbia University using the EchoNest API [1].

However, the main dataset contains only song metadata (e.g. artist and album) and audio features which are not of interest for this project. Fortunately, the dataset is accompanied by four companion datasets:

SecondHandSongs List of cover songs within the MSD.

Taste profile User data from EchoNest for a subset of the MSD.

musiXmatch Lyrics for the songs in the MSD [2].

Last.fm Tags and similarity information for songs in the MSD.

Of interest for the current task are the last two datasets. The musiXmatch dataset provides the lyrics needed for the analysis and the Last.fm tags can be used to identify the genre of the songs.

Summarizing, the data to be used is:

- The main MSD, only the IDs are used. There are 1M in the dataset and its uncompressed size is of about 280GB.
- The musiXmatch dataset has lyrics for almost 240k songs from the MSD. The reasons for the reduced size are: copyright, duplicates and instrumental tracks.
- The Last.fm dataset includes 94% of the songs in the MSD, and in total around 500k have at least one tag.

[1] `http://the.echonest.com/`

[2] Lyrics are available only in bag of word representation due to copyright.

Table 1: Sizes and formats for the data subsets

Dataset	Format	Size (GB)
MSD	Text files	2.6
Last.fm	SQLite	0.5
musiXmatch (Full)	SQLite	2.3

2.2 Downscaling

For this first milestone, we were tasked with selecting subset that can run in a single machine. In this case, the selection was straightforward since there is a 10k randomly sampled subset available in the MSD website. We considered this an appropriate size for the initial task.

For the Last.fm dataset it was also possible to get the matching 10k subset but for the musiXmatch we needed to download the whole dataset with the 240k songs' lyrics.

The sizes and formats of these subsets are listed in table 1.

2.3 Preprocessing

In order to have a consistent access to all the information per track, we processed the input text files and SQLite databases and merged them into a single text file where each line is a JSON object with the track ID, its associated genres and the word frequencies for its lyrics.

Here it was necessary to filter out the tracks without any genre information or less than 10 words in its lyrics, after this filtering the subset shrank to 1.4k songs and in its JSON representation it had a size of 1.1MB.

2.3.1 Discovering genres in tags

As mentioned in 2.1, the Last.fm provides us with human-made tags, these tags however contain all sorts of information and are not focused on genres only. In order, to use these as ground truth for the experiments we cleaned the tags by comparing them against a comprehensive online list of genres [3]. In order to capture possible misspellings we considered a tag equal to a genre in the list if their Damerau-Levenshtein distance does not exceed 1.

3 Design

3.1 Algorithms

For this milestone we explore two approaches, namely clustering and locality sensitive hashing.

3.2 Clustering

In order to classify a set of songs as belonging to one particular genre, we chose the unsupervised learning technique of clustering as one possible solution. Our goal is to create a set of clusters covering the set of all songs with lyrics and then using the clean genre tags determine the genres that are the majority in each cluster. The intuition here is that if a genre is present in a 50% majority of the songs in cluster, then the songs can be classified as belonging to that genre.

3.3 Locality Sensitive Hashing (LSH)

Locality Sensitive Hashing is a popular technique used to solve the Nearest Neighbour problem in extremely high dimensional spaces. Given that we too deal with finding clusters, or similar neighbours of songs in a 5000-dimensional space, we consider applying LSH to solve our problem.

The intuition behind LSH is that similar items are hashed to the same bucket. Hence by looking at the buckets, we can detect collisions, or in our case, a cluster of songs that share similar vocabulary.

In contrast to k-means clustering, LSH can be implemented by making a single pass over the data. This in turn will help us later analyze whether k-means is worth the computational overhead.

[3] `http://www.musicgenreslist.com/`

3.4 Evaluation & Metrics

Clustering validation techniques usually rely on specific problem knowledge as it is sometimes the case that the cluster metrics that the algorithms minimize do not reflect on the quality of the clusters when their information is processed by humans [3].

In this particular task we made use of the cleaned genre tags to define a metric that intuitively rates the quality of the cluster for genre prediction. This metric is defined as the ratio between the number of clusters with at least genre that is present in more than 50% of the tracks in the cluster, and the total number of clusters. This suggests the number of good clusters for identifying genres of songs, and the goal is to have a high number of good clusters. This metric is presented in equation 1.

$$Q = \frac{\#good\ clusters}{\#clusters} \tag{1}$$

4 Implementation

This section describes the details of the implementation of the algorithms outlined above. The code for this implementation was written in its majority in python, along with some parts written in Javascript, HTML and CSS, namely the web frontend.

4.1 Clustering in a single machine

In order to perform clustering in the input data, first it is necessary to define a representation for the data and a distance metric in this representation. Since we are dealing with lyrics, which are text documents representing the songs, we decided that using a TF-IDF vector representation is a sensible choice.

The term frequency (TF) calculation and smoothed inverse document frequency (IDF) are presented in equations 2 and 3, respectively. Where d represents a document and t a term.

$$tf(d,t) = \sqrt{count(d,t)} \tag{2}$$

$$idf(t) = 1 + \log \frac{\#tracks}{\#tracks\ containing\ t + 1} \tag{3}$$

Finally, the TF-IDF value is simply the product of the TF and IDF.

After the vectors were calculated and normalized, they were stored in a text file for further processing. The next step in the implementation is the clustering itself.

For the clustering we chose K-means since it is a well known algorithm to approximate the optimal solution to the clustering problem of minimizing the sum of the distances from each point to the closest cluster center. We used scikit-learn [4] implementation of K-means and tuned the parameters to obtain the best results.

4.1.1 Clustering on Hadoop

This current approach runs well in a single machine with the current subset of the data, as it will described in the results section, however as we look forward to scale up the data size, it is necessary to consider changes in the algorithm.

One possible variation of the clustering algorithm that has a straightforward implementation in a distributed computing environment, such as Hadoop's MapReduce, is the use of coresets to do a weighted sampling of the data before running a clustering algorithm on these samples [2].

4.2 LSH in a single machine

We use the library NearPy to perform Locality Sensitive Hashing on our dataset. This option proved quite convenient, since the project is well-documented, active and provides an already optimised single-node implementation.

NearPy works as shown in Figure 1. Given an input vector, hashes are used to generate bucket keys to which the input hashes to. At query time, the vector is paired up with all the candidate vectors that's contained in the respective buckets. After the candidates have been collected, a distance metric (Euclidean in our case) is used to identify the nearest neighbours.

In our implementation of LSH, we make two passes over the dataset. First pass to hash the vectors to their respective buckets. Followed by a second pass, where each track acts as a query and identifies the nearest neighbours. The query vector and the neighbours are then treated as a single cluster.

[4] `http://scikit-learn.org/`

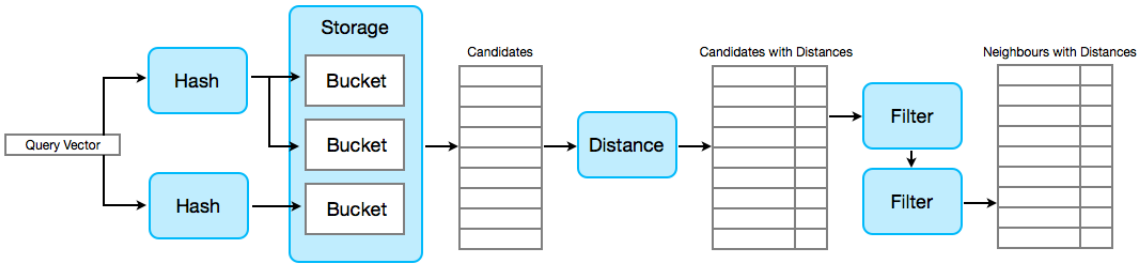

Figure 1: NearPy's LSH pipeline (Taken from `www.nearpy.io`)

4.2.1 LSH on Spark

Apache Spark's MLLib does not provide LSH out-of-the-box. This is complicated by the fact that NearPy's implementation does not scale to multiple machines, although the algorithm is inherently scalable. Hence, we intend to develop our own simplified version of LSH using Python for the next milestone.

5 Results

5.1 Quality metrics

5.1.1 K-Means

Figure 2: Fraction of majority clusters as the number of clusters increases.

Figure 2 shows the effect of the number of clusters in the quality metric established for the clustering model. As expected, increasing the number of clusters does not necessarily mean an increase in the quality of the solution for the problem although it does decrease the objective function.

Looking at the generated clusters and their majority genres, we were able to observe that the cluster system is able to distinguish between 4 major genres: Latin, World, Hip-Hop and Rock, however most of the clusters are labeled as Rock. We think this is due to two factors, firstly because of the quality of the tags, since most of them map to the Rock genre, secondly because the clustering is only distinguishing between widely varying vocabularies, namely Spanish words for Latin, other non-English words for World, slang for Hip-Hop and the rest is identified as Rock.

We expect that as the data size increases, it will be possible to identify smaller clusters belonging to other genres and have a better quality.

5.1.2 LSH

Figure 3 summarizes the accuracy of LSH to detect genres based on lyrics. The y-axis denotes our quality metric majority genre cluster quality, which expressed what fraction of the cluster adhere to the majority genre exhibited by

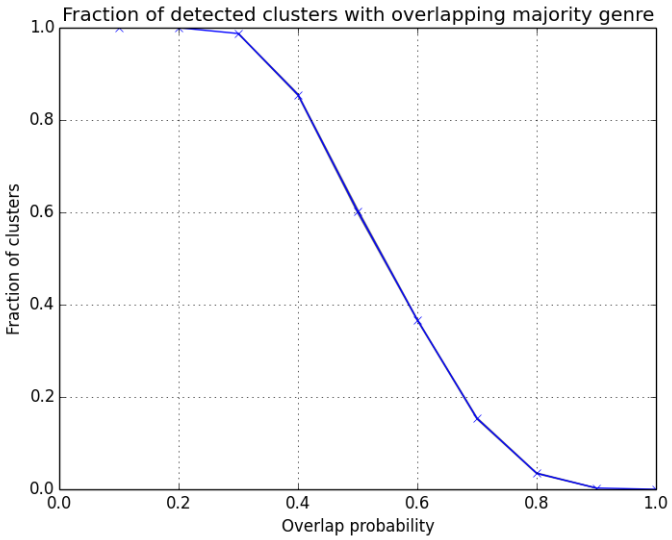

Figure 3: Majority genre cluster quality vs. fraction of cluster adhering to the majority genre. The graph was obtained over a series of 10 experiments, with varying hash functions.

that cluster. In other words, if the overlap probability is 0.5, it implies that at least half of the tracks have a genre in common.

5.2 Performance

With the size of the current dataset, the single machine algorithms described in the implementation section were sufficient. Most of the running times were in the order of seconds or minutes at the most.

Moreover, after the experimentation phase was done and the optimal parameters for clustering were selected, running a set of K-means iterations yield a set of cluster centers which can be used to answer web queries in real time since the calculation is linear on the number of clusters which does not scale with the data, making it effectively $O(1)$.

In order to scale this up to a larger dataset, the algorithm implementations must be changed according to the guidelines mentioned in sections 4.1.1 and 4.2.1. This is needed because k-means runs on quadratic time and therefore does not scale vertically. On the other hand even though LSH has a linear running time it can be easily adapted to a parallel implementation thus benefiting greatly from a distributed computing environment.

References

[1] Bertin-Mahieux, T., Ellis, D. P., Whitman, B., and Lamere, P. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)* (2011).

[2] Feldman, D., Faulkner, M., and Krause, A. Scalable training of mixture models via coresets. In *Advances in Neural Information Processing Systems* (2011), pp. 2142–2150.

[3] Halkidi, M., Batistakis, Y., and Vazirgiannis, M. On clustering validation techniques. *Journal of Intelligent Information Systems 17*, 2-3 (2001), 107–145.

[4] Lee, K., Yeo, W. S., and Lee, K. Music recommendation in the personal long tail: Using a social-based analysis of a user's long-tailed listening behavior. In *Proceedings of the Workshop on Music Recommendation and Discovery (WOMRAD)* (2010), pp. 47–54.

[5] McFee, B., Bertin-Mahieux, T., Ellis, D. P., and Lanckriet, G. R. The million song dataset challenge. In *Proceedings of the 21st International Conference Companion on World Wide Web* (New York, NY, USA, 2012), WWW '12 Companion, ACM, pp. 909–916.

[6] Tague-Sutcliffe, J., Downie, S., and Dunne, S. Name that tune! an introduction to musical information retrieval. In *Proceedings of the 21st Annual Conference of the Canadian Association for Information Science* (1993), pp. 204–216.