Finding Clusters of Similar Artists - Analysis of DBSCAN and K-means Clustering

Walter Nordström
walterno@kth.se
+46705450021
Voxnegränd 24, 12843 Bagarmossen
and
Jacob Håkansson
jacobhak@kth.se
+46767741383
Professorsslingan 39, 11417 Stockholm

DD143X, Degree Project in Computer Science, First Level
School of Computer Science and Communication
Royal Institute of Technology
Supervisor: Michael Minnock

April 2012

Abstract

We have applied k-means clustering and DBSCAN to the problem of finding sets of similar artists based on a large number of artists and their genres. For our experiments we used data from the Million Song Dataset, which is a freely available collection of a million popular music tracks' metadata created specifically for research. We ran the algorithms with varying values on their parameters and studied the effects. The resulting clusters were analyzed and for k-means we found three different types of clusters. Although the results from k-means were quite noisy, many of the clusters could be used gain some insight in the similarity between artists.This implied that using distances as a representation of similarities between artists is viable. DBSCAN did not prove to be as useful. This was because its clustering method is density-based and the density of the clusters in the input data differed by far too much for DBSCAN to handle. We found that more features in the input data, such as genre per track, would be desirable and would probably improve the results of the algorithms. Further study of other clustering algorithms applied to the same data would shed light on the actual effectiveness of the algorithms studied here.

Referat

Vi har tillämpat k-means klustring och DBSCAN på problemet att hitta grupper av liknande artister baserat på ett stort antal artister och deras genrer. Till våra experiment har vi använt data från Million Song Dataset, som är en fritt tillgänglig samling av en miljon populära sångers metadata, som skapats speciellt för forskning. Vi körde algoritmerna med varierande värden på deras parametrar och studerade effekterna. De resulterande klustren analyserades och för k-means fann vi tre olika typer av kluster. Trots att resultaten från k-means innehöll ganska mycket brus, så skulle många av klustren kunna användas för att få en viss inblick i likheten mellan artister. Detta implicerar att man kan använda avstånd som en representation för likheter mellan artister. Resultaten från DBSCAN visade sig inte vara lika användbara. Detta berodde på att dess klustringsmetod är densitetsbaserad och densiteten hos klustren i indata skilde sig alltför mycket för att DBSCAN skulle klara av hitta dem. Vi fann att fler egenskaper i indata, såsom genre per spår, skulle vara önskvärt och skulle sannolikt förbättra resultaten från algoritmerna. Ytterligare studier av andra klustringsalgoritmer som tillämpas på samma data skulle belysa den faktiska effekten av de algoritmer studerade här.

Contents

1 Statement of collaboration

The majority of the report and the implementation were written collaboratively. This means that the sections not specified below were written with equal effort from both authors. The sections 2.3 and 3.1 were mainly written by Walter Nordström. 2.2, 3.2 and 6 were mainly written by Jacob Håkansson.

2 Background

2.1 Introduction

Today, Cluster Analysis is important for many fields of study, for example biology, psychology and other social sciences[1]. More importantly for this paper it provides an effective way of extracting knowledge from large sets of data, i.e. Data Mining[2]. Specifically, Cluster Analysis is a method for partitioning data into groups where members of a group is similar in some way and dissimilar to members outside the group. These groups are referred to as clusters[1].

In this paper we will examine the nature of some clustering algorithms when applied to music classification. The classification of music is especially important for digital music distribution[3]. One goal is to make it easier for consumers to find new music that fit their taste, by presenting relations between different music[3]. Music classification can be done in many ways due to the many different properties a sample of music can have. For example one could look for similarities on an objective level by examining the tempo, chord-progressions and rhythm of a sample[3]. Another approach is to look at relationships between musicians, for example membership in a particular band[3]. A less objective approach is to look at similarities in music genres[3], which is the problem to which we will apply clustering algorithms.

Although the characteristics of a specific musical genre are somewhat arbitrary, there's clearly a greater chance that an artist which has been tagged with the same genre as an artist that a consumer already has been listening to, will cater to the consumers taste, than something as objective as a specific tempo.

The purpose of this paper is not to provide a definition of genres, however interesting that debate may be, instead we will examine the properties of a couple of algorithms when they're applied to this specific kind of data.

2.2 Data

For our experiment we have chosen to rely on data found in the Million Song Dataset[4]. The dataset obviously contains musical data from a million songs, but it also provides us with the specific properties of that data that we're interested in, artists and their genres. The Million Song Dataset was created with projects like ours in mind[4], and as such is freely available from their website[4]. The dataset is compiled from several sources and songs was chosen carefully[4], however that is not in this paper's scope.

The size of the whole dataset is quite large, with each song having 47 properties, as a solution to researchers with limited storage capacity there's also several different subsets of that data available, specifically there's a dataset which only contains artist IDs and their associated tags. The tags are mostly musical genres, but also contain geographical information, such

as the city of origin of an artists. That subset is the data on which we have based our experiment.

2.3 Algorithms

2.3.1 DBSCAN

DBSCAN is a density based clustering algorithm. That is, the clusters it discovers consist of elements that in the input data forms a dense subset of the input. In order to find these clusters parameters must be passed to the algorithm which are used to determine which areas are to be considered dense and which are not. These fundamental parameters are ϵ and $min\_samples$. ϵ defines the maximum distance between two points for them to be considered to be in the same neighborhood. $min\_samples$ defines the minimum amount of points in the neighborhood of a point for it to be considered a core point for a cluster. Elements not found in the dense areas is considered as noise and are not assigned to any cluster.

Due to its properties DBSCAN can discover clusters of arbitrary shape. It is also applicable for data in any space and with any distance function. A shortcoming however is that the different dense regions must have about the same density or DBSCAN will either include noise in the clusters or consider less dense clusters as noise. For a more detailed description of the algorithm, please refer to the original publication[5].

2.3.2 K-means Clustering

K-means clustering partitions a given dataset into k clusters, where the objects belong to the cluster with the closest center. The task can be formulated as an optimization problem: Find k center points that minimizes the mean squared distance from each data point to its nearest center point. K-means solves the problem in an iterative way, and in each iteration new center points p_{i+1} are chosen as the centroid of the data points that are nearest to the center points p_i from the previous iteration.[6]

3 Approach

3.1 Representation of the artists

To apply the algorithms to the problem the first task was to determine how the artists (the input data) should be represented in a suitable way. Since the input in this case isnít a set of physical objects in a real space there is a level of freedom in deciding which way the artists should be represented. The goal here is to make the data as clustering-friendly as possible. This means that artists which are considered similar should be close to each other, forming clusters, and these in turn should be far away from other clusters.

However, one must consider in what form the algorithms require the input and potential limitations of the dataset. One limitation is that there is no possibility to weight the artists genres, so if an artists has some genres that only apply to a small fraction of its tracks while another is found in almost all of the tracks there is no way to derive this information from the dataset.

3.1.1 Input data for k-means clustering

In order to apply k-means clustering the data must be represented as points in an euclidean vector space. A sensible way of doing this, with the properties of the dataset in mind, is to let the the genres represent dimensions. In this way an artist can be represented as an boolean vector (point) where where its components are either 1 or 0 depending on if the corresponding genre is associated with the artist in the dataset or not.

If the total number of genres was 3, the shape of the data would be a cube with side 1 in the first octant in $\mathbb{R}^3$, and the possible points representing artists would be the cornerpoints of this cube. In our case the number of genres are 2274 so the set of data is represented as cornerpoints of a hypercube in the first orthant in $\mathbb{R}^{2274}$.

This representation has some desirable properties. The euclidean distance between two artist will be zero only if they have exactly the same genres. This is a good property since it is convenient to consider them very similar, based on our data, if they are tagged in the same way. Furthermore, if two artists have about the same number of tags, the distance between them is larger the more genres they differ in.

A drawback with this representation however is when the number of tags differ a lot. There are two different cases when this occurs. The first case occurs if two artists have one or a few genres in common but one of them has lots of additional genres as well. This yields an euclidean distance which, depending on the number of genres, can be larger than between two artists with no genre in common but less genres in total. The second case occurs if two artist has no genres in common, then the distance between them will only depend on the amount of genres they have. Regardless of whether these qualities are to be considered good or bad (probably depending on artists being compared) they cannot be altered and will influence the the output.

3.1.2 Input data for DBSCAN

Unlike k-means, the DBSCAN implementation used does not require points in euclidean space. Instead one can choose an arbitrary distance function. Because of the drawback discussed regarding the euclidean distance we came up with an alternative distance function. Let a and b be two artists, then

the distance D_{ab} between them is defined as

$$D_{ab} = 1 - \frac{inCommon(a, b)}{avg(a, b)}$$

where $inCommon$ means the number of genres the two have in common and avg is the average number of genres of the two.

With this distance function each pair of artists with no genres in common the distance is 1, which is the greatest distance possible. The distance is also always nonnegative, and zero only if the two are identical, which is desirable. Also, the greater the average number of genres is compared to the number of matching genres the greater the distance will be which also is a sensible feature.

3.2 Implementation

The input data was given in an SQLite database, so we used that data in the form we got it. However, as mentioned before, the set of tags contained a subset that contained purely nationalities, which we found created irrelevant noise in our data. We therefore removed all nationality tags from the artist during processing. After this process the dataset contained 7789 artists and 2274 genres.

We found implementations of the algorithms of this study in the Python language in two separate packages, this combined with the support for numerical analysis from other Python packages left us with the impression that Python would be the favorable language of choice. The k-means algorithm used in our study is the one found in the PyCluster package which is based on The C Clustering Library[7]. The implementation of DBSCAN we have used is the one found in Scikit-learn, which is a Machine Learning package for Python[8].

3.3 Parameter estimation

Both k-means and DBSCAN requires some input parameters apart from the input data. K-means needs the number of clusters to be discovered, this will be referred to as k. DBSCAN needs two parameters which defines what should be considered as a dense region in the data.

The parameter k affects the output of k-means significantly. We studied the properties of this parameter by starting with assigning k the value 5 and successively incremented it up to 30. We observed some patterns in the output data throughout this process.

DBSCAN has two parameters referred to as ϵ and $min\_samples$. These together defines the core density of the target clusters. In both the case where we used euclidean distances and the case where we used our own distance function there were only a finite number of possible distances. This is

because the vectors used in the euclidean space are boolean and our distance function depends on a finite set of discrete variables. We ran the algorithm for all the possible values for ϵ and let $min\_samples$ vary between 10 and 500.

4 Results

4.1 Results for k-means clustering

We found three types of clusters with fundamentally different properties, we chose to refer to them as type-A, -B and -C clusters. In the type-A clusters all artists had a single genre that all members of the cluster had in common. The rest of the genres usually seemed to fit fairly well with the most common genre. The higher the value of k the more type-A clusters appear, though the clusters becomes smaller when k increases. The type-B clusters seemed to be two or three type-A clusters merged. The number of type-B clusters did not seem affected by the value of k. The type-C cluster appears only once in each output. It is significantly larger than all other clusters and always seemed to contain genres that didn't fit very well together. We studied how this type of cluster was affected by k, and found that it shrunk slightly with every increment of k.

Below follows some results for k-means with $k = 13$, which in our opinion had the most exemplary clusters of type A and B.

Cluster number	1		2		3		4		5		6	
Cluster type	B		A		A		A		C		A	
Size	240		338		251		495		2513		319	
Tag | Artists w/ tag	jazz and blues	143	pop	338	metal	251	rock	495	hip hop	109	hip hop rnb and dance hall	319
	jazz	129	rock	135	heavy metal	59	alternative rock	175	classical	88	hip-hop	47
	hard bop	12	classic pop and rock	76	rock	55	rock and indie	143	progressive rock	88	rnb	24
	swing	12	pop and chart	36	hard rock	41	indie rock	84	soul and reggae	87	hip hop	17
	bebop	10	electronic	30	death metal	27	electronic	47	world	85	rap	15
Cluster number	7		8		9		10		11		12	
Cluster type	A		A		A		A		A		A	
Size	916		208		269		720		546		553	
Tag | Artists w/ tag	classic pop and rock	916	production music	208	dance and electronica	269	rock and indie	720	folk	546	punk	553
	rock	106	classical	38	electronic	34	electronic	25	country	46	rock and indie	33
	punk	48	composer	33	electronica	21	indie rock	19	classic pop and rock	39	rock	23
	progressive rock	27	opera	16	warp	12	britannique	16	rock	23	punk rock	22
	alternative	19	soundtrack	6	downtempo	8	electronica	13	blues	11	anarcho	17
Cluster number	13											
Cluster type	B											
Size	421											
Tag | Artists w/ tag	electronic	243										
	pop and chart	189										
	electro-industrial	26										
	electronica	24										
	industrial	20										

Figure 1: The clusters are presented with their size and the 5 most common genres found among its artists.

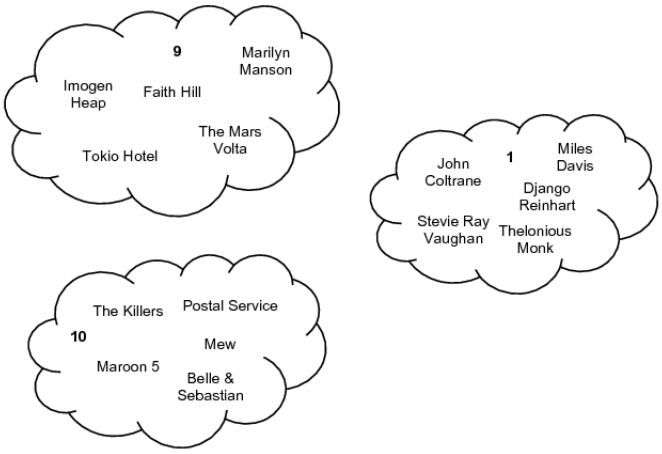

Figure 2: Sample clusters with some of the artist found in each of them. Artists was picked based on familiarity according to Echo Nest.

4.2 Results for DBSCAN

Using euclidean distance, we noticed that more than one cluster was found only when ϵ was chosen less than $\sqrt{2}$. One of them were always relatively large and the rest small, containing only artists with the exact same genres. When *min_samples* increased the amount of noise did as well.

When our own distance function were used, the results appeared slightly different for some values of the parameters. Still, one large cluster seemed to contain the majority of the artists not considered as noise, and a majority of the remaining clusters contained only artists with exactly the same genres. Some clusters however, contained differently tagged artists with, in our opinion, similar genres. This occurred when $\epsilon = 0.39$ and *min_samples* in the interval $[10, 30]$. For other values the results seemed similar to the ones found using euclidean distances. Figure 3 shows a sample output for one of the relevant clusters found.

5 Discussion

The type-C cluster of k-means can probably be viewed as noise, due to the fact that it contains many different genres that have no obvious similarity. It seems as the C-cluster is a union of many type-A clusters that were too small to be considered clusters themselves. This is supported by the fact that the type-C cluster keeps shrinking when k is increased.

The type-A clusters can at a first glance look trivial, because all artists in the cluster share one specific genre. However, this is not the same as

Cluster number	5	
Size	499	
Tag | Artists w/ tag	rock	324
	classic pop and rock	143
	rock and indie	126
	alternative rock	125
	punk	113

Figure 3: Sample cluster from DBSCAN, this run had 3512 artists considered as noise.

finding all the artists tagged with that genre and deciding they belong to that same cluster. This can be seen in our example of output data, cluster 4 and cluster 10 both contain artists with genre "rock and indie". According to k-means clustering the artists in cluster 4 is more similar to each other than to an artist in cluster 10. If this is true in reality is still up for discussion.

The type-B clusters are in our opinion the most interesting. In these clusters there is no single genre that unifies a cluster, yet the genres of the artists within are musically similar. This means that k-means can find clusters with artists that are not directly connected to each other through one genre, but instead has piecewise intersections of genres connecting them.

Considering the parameter k, its effect when increasing was not only making the type-C cluster smaller, but also all other clusters. In our opinion the most interesting clusters were found when k was chosen between 11 and 14. Otherwise the clusters just became too small, or too few to really give any insight.

One thing that probably would improve the results is more information about the artists genres. For example, with information about the percentage of tracks an artist has with its different genres, the data points would move from the corner points to inside the hypercube which would give a more precise position for the artists.

Regarding the results from DBSCAN using euclidean distance, it seems like the density in the different regions with artists were too diverse. This is due to the fact that we were unable to find any values for the parameters for which the algorithm did not find either one large cluster or one cluster per genre. Using our own distance function the algorithm produced a bit more interesting clusters, but this is still not sufficient to make it usable for finding actual similarities.

6 Conclusions

We have found k-means clustering can give some insight in the similarity of different artists. This implies that using distances as a representation of similarities between artists is viable. The actual effectiveness of k-means is questionable, as the output always contained a relatively large portion of noise. This clearly depended on our input data, which essentially had only two features. With more features available the results would certainly improve. DBSCAN was largely ineffective on this data, and only in fringe cases did it find useful clusters. Note that this was only while using a specially constructed distance function. Further exploration of other types of clustering algorithms regarding this particular input data is encouraged.

7 References

1. Tan, Pang-Ning. Steinbach, Michael. Kumar, Vipin. Introduction to Data Mining. Pearson Education, 2005. p. 487- 490.

2. ACM Special Interest Group on Knowledge Discovery and Data Mining. Curriculum Proposal. Available at http://www.sigkdd.org/curriculum.php (Accessed 4/3/2012).

3. Pachet, Francois. Westermann, Gert. Laigre, Damien. Musical Data Mining for Electronic Music Distribution. Sony Computer Science Laboratory Paris, 2007.
Available at http://www.csl.sony.fr/downloads/papers/2001/pachet01c.pdf (Accessed 4/3/2012).

4. Bertin-Mahieux, Thierry. P.W. Ellis, Daniel. Whitman, Brian. Lamere, Paul. The Million Song Dataset. In Proceedings of the 12th International Society for Music Information Retrieval Conference, 2011. The Dataset is available at http://labrosa.ee.columbia.edu/millionsong/ (Accessed 6/3/2012).

5. Ester, Martin. Kriegel, Hans-Peter. Sander, Jo‡rg. Xu, Xiaowei. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining, Portland, Oregon, 1996.

6. Kanungo, Tapas. Mount, David M. Netanyahu, Nathan S. Piatko, Christine D. Silverman, Ruth. Wu, Angela Y. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 7, 2002.

7. de Hoon, Michiel. Imoto, Seiya. Miyano, Satoru. The C Clustering Library. The University of Tokyo, Institute of Medical Science, Human Genome Center, 2010. Documentation: http://bonsai.hgc.jp/ mdehoon/software/cluster/cluster.pdf PyCluster distribution:
http://bonsai.hgc.jp/ mdehoon/software/cluster/software.htm#pycluster
(Both accessed 3/4/2012)

8. Pedregosa et al. Scikit-learn: Machine Learning in Python. The Journal of Machine Learning Research 12, pp. 2825-2830, 2011.
http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html (Accessed 3/4/2012)