A Classification System Applied to Music Reviews

Carl-Emil Lagerstedt
Department of Computer Science
Lund University
dat01cal@ludat.lth.se

Örjan Berglin
Department of Computer Science
Lund University
dat01orb@ludat.lth.se

Abstract

This paper describes a system for classification of music reviews. The system uses a clustering algorithm to build a tree out of a corpus of reviews. Reviews are clustered together based on the similarity of their contents, thereby providing a way to make suggestions of similar artists. Our results show that this approach has potential and should be further explored.

1 Introduction

Staying updated in the world of music is not an easy task. New artists and genres emerges everyday and the most obvious way of learning about new artists is by reading articles and reviews. This can be quite time consuming, and therefore we wanted a way to classify reviews to support the selection of the interesting ones.

This paper describes such a system for classification based on mutual information.

The classification algorithm computes the intersection of two documents and returns a fractional value between 0 and 1, called *document similarity*. The closer the value is to 1, the more the documents resemble each other. A tree is then built based on the document similarities.

2 Purpose

The purpose of this paper is to describe a method for automatically making recommendations, based on the content of reviews. We believe that such a system could be useful in many cases, such as in online shopping. A site could keep a large database of reviews and use them to make suggestions to customers. The benefits of this system is to reduce manual labour.

There is also the possibility that the system will be able to visualize hitherto unknown or unexpected connections between artists, that might not be visible while using traditional keyword-based indexing.

3 Background

Often there is a need to find and show information that is related to a certain topic, for instance, many shopping sites on the Internet have some way of showing recommendations to the customer based on the item currently viewed. This can make it easier for the customer to find interesting objects, and might increase sales.

The methods for producing this information are often primitive, and are often based on the shopping habits/recommendations of previous customers/visitors. Many times this approach is adequate, but the method has obvious drawbacks. Information about previous customers might not be enough or even applicable to this customer. For instance, while browsing crime fiction, a customer is probably not interested in the fact that someone else has bought a cookbook together with the crime novel.

There are also methods that are based on manual labour, where someone has to enter keywords by hand.

This is a method that works quite well, but it can demand a lot of manpower, and due to it's nature is error prone.

We will explore three websites that all have some kind of recommendation system. We do not claim to have exact knowledge about the systems that these sites use. This is just our impression of how the systems work.

3.1 The All Music Guide[1]

The All Music Guide (AMG) is an online resource of artist biographies and record reviews. They have almost 250,000 reviews in their database.

In the biography for an artist, AMG presents a list of similar artists. This list is compiled from data manually entered by AMG's editors and visitors to the site.

3.2 Amazon.com[2]

Amazon.com makes recommendations based on what other buyers of an item has bought. The system seems to be based entirely on the sales statistics, and thus the system does not make intelligent suggestions. Our research shows that this system works quite well, with two exceptions. The first is that one might get only suggestions of the same author. The second is that if an item has not been bought by anyone, no suggestion is made, since there is no sales statistics for that item.

3.3 The Internet Movie Database[3]

The Internet Movie Database describes their recommendation system like this:

> *"With over 384,000 titles on the IMDb it isn't feasible to handpick Recommendations for every film. That's why we came up with a complex formula to suggest titles that fit along with the selected film and, most importantly, let our trusted user base steer those selections. The formula uses factors such as user votes, genre, title, keywords, and, most importantly, user recommendations themselves to generate an automatic response."*

3.4 Background Summary

Even though all the described sites have many reviews, none of the systems makes use of linguistic methods for producing recommendations. In fact, we have failed to find *any* such system.

4 The Music Reviews

Based on the observation that reviews often references other artists, we draw the conclusion that this would be useful in the classification process. For example, in a review of the album *Gold* by *Ryan Adams*, the reviewer mentions that "*...the album is an impressive exploration of territory previously covered by Bob Dylan, Neil Young and other greats before him*". It does not seem farfetched that a *Ryan Adams* fan might also enjoy *Bob Dylan* and *Neil Young*.

It also seems likely that if artist *A* has received reviews that are similar to those of artist *B*, then their work might also be similar.

The corpus of reviews was collected from several online resources, as well as OCR-ed from print magazines. The corpus consists of a few hundred reviews, selected by us. The reviews cover popular music from the 1960s up until the present date. Therefore our personal preferences may have influenced the selection.

5 Classification Algorithm Overview

We use a clustering algorithm based on intersection. Each document is considered to be a set of words and the intersection of two documents represents how similar they are.

Each document is considered a node. Each node is compared to every other node and the pair with the greatest similarity is selected out. A new node is created with the two selected nodes as its children and reinserted into the set of all nodes, containing all the words from the two nodes.

This is repeated until there is only one node left. In the resulting tree, nodes that are close should have similar content.

[1] http://www.allmusic.com
[2] http://www.amazon.com
[3] http://www.imdb.com

5.1 Document Similarity

The union of two documents is the base for the *document similarity*.

A fractional number between 0 and 1 is returned by the following equation:

$$S = |D_1 \cup D_2| / (|D_1| + |D_2|),$$

where D_1 and D_2 are the documents and S is the document similarity. A high document similarity indicates that the two documents have similar content. The document similarity is used only on the lowest level of the tree. When compound nodes are compared, the set of words in each node's children are computed recursively.

5.2 Algorithm Efficiency

Since, in every iteration, each node has to be compared to every other node, the algorithm has a time complexity of $O(n^2)$, where n is the number of nodes. Some of the computation could be eliminated by using dynamic programming. Even though the algorithm is slow, it is not a major impediment, since the tree needs to be built only once. The only time the tree must be rebuilt is when additional reviews are added to the database.

6 Prototype Implementation

Our prototype is coded entirely in Java and makes heavy use of hash tables to store document content. We use a stop list to remove common words that provides little or no information. This reduces the number of comparisons in the algorithm. As a consequence, the algorithm runs faster, and also more accurate. The reviews are stored as plain text files. Output is visualized using *GraphViz* from AT&T Labs-Research[4]. GraphViz produces a visual graph that can be zoomed and panned to study the results. The running time for building a tree with 240 reviews is about 60 minutes on a 2.0 GHz Pentium 4.

7 Results

Our initial idea, that we could group similar reviews together, proved to work pretty well. Based on observations of our implementation, we estimate that about 60-70% of the reviews are clustered correctly.

We saw that one important factor was the quality of the corpus. If a review is too short, it doesn't contain very much factual information, and is therefore not very usable in the clustering effort.

One other objective would be to produce a more balanced tree. This way we could easily cut the tree at an arbitrary level, to produce clusters. With the current implementation, the tree can become very unbalanced.

Similar artists do get clustered together. Efficiency is, however, hard to measure, since different people might have different views on how the artists should be grouped. An example of a succesful (in our opinion) grouping, is that of *Ryan Adams* and *Bob Dylan* getting grouped together. An example of an obviuos grouping is that *Bob Dylan* reviews gets grouped with other *Bob Dylan* reviews.

However, some reviews get clustered in at a completly wrong place in the tree. For example, *Velvet Underground* has, in our tests, been clustered with *Abba* as its closest neighbour, which, in our opinion, feels totally wrong. There should be better matches for both these bands. We believe that this bad matching has two reasons. First, in our implementation, all documents are eventually put in the tree, regardless of whether they have any similarity to an other document. Second, we have a pretty limited corpus. A solution to the first problem is to have a threshold value for the document similarity. This would eliminate the insertion of irrelevant nodes.

8 Conclusion

We are satisfied with our results, even though they might not seem very impressive. Our proposition seems to hold, that is, an automated clustering of reviews is a useful idea.

More work needs to be done, but we are confident that, given enough time and effort, this could also turn out to be a useful application.

[4] http://www.research.att.com/sw/tools/graphviz/

9 Future Directions

We would like to implement the vector space model, to compare the efficiency and the results of the algorithms.

When a review has very little in common with other review, there is no point in inserting it in the tree. As previously mentioned, a threshold value for the document similarity could be used to remove those reviews.

The use of an inverted index (where rare words are weighted up) should be explored.

The ability to detect bold and italicised words might provide additional clues as to what words in the review are important.

A name extraction feature would further enhance the similarity values, since artist names commonly are used as references when writing a review.

To reduce the running time of the algorithm, dynamic programming should be utilized to reduce the number of comparisons that need to be made.

To speed-up the application, distributed data processing could be used. It would be interesting to test the algorithm on a distributed system, using a large corpus, such as The All Music Guide.