Large Scale Music Information Retrieval by Semantic Tags

Zhao Zhendong (HT080193Y)

Under Guidance of Dr. Wang Ye

A Graduate Research Paper Submitted
for the Degree of Master of Science
Department of Computer Science
National University of Singapore
July, 2010

Abstract

Model-driven and Data-driven methods are two widely adopted paradigms in Query by Description (QBD) music search engines. Model-driven methods attempt to learn the mapping between low-level features and high-level music semantic meaningful tags, the performance of which are generally affected by the well-known semantic gap. On the other hand, Data-driven approaches rely on the large amount of noisy social tags annotated by users. In this thesis, we focus on how to design a novel Model-driven method and combine two approaches to improve the performance of music search engines. With the increasing number of digital tracks appear on the Internet, our system is also designed for large-scale deployment, on the order of millions of objects. For processing large-scale music data sets, we design parallel algorithms based on the MapReduce framework to perform large-scale music content and social tag analysis, train a model, and compute tag similarity. We evaluate our methods on CAL-500 and a large-scale data set ($N = 77,448$ songs) generated by crawling Youtube and Last.fm. Our results indicate that our proposed method is both effective for generating relevant tags and efficient at scalable processing. Besides, we also have implemented a web-based prototype music retrieval system as a demonstration.

Acknowledgments

I thank my supervisor Dr. Wang Ye for his inspiring and constructive guidance since I started my study in School of Computing.

Dedication

To my parents.

Contents

List of Publications

Large-scale Music Tag Recommendation with Explicit Multiple Attributes.

Zhendong Zhao, Xi Xin, QiaoLiang Xiang, Andy Sarroff, Zhonghua Li and Ye Wang

ACM Multimedia (ACM MM) 2010 (Full paper, coming soon).

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The way of accessing music has been changed rapidly over the past decades. As almost all of the music items will be accessible online in the foreseeable future, the development of advanced Music Information Retrieval (MIR) techniques are clearly needed. Many kinds of music information retrieval techniques are being studied for this purpose of helping people to find their favorite songs. The ideal system should allow intuitive search and require a minimal amount of human interaction. Two distinct approaches to search large music collection coexist in literatures: 1) Query-by-example (QBE) such as Query-by-Hamming; 2) Query-by-text (metadata and semantic meaningfull description), hence it has two sub-categories: Query-by-metadata(QBM) and Query-by-Description(QBD).

QBD is challenging due to the well-known semantic gap between a human being and a computer, making it extremely difficult to find the exact results that satisfy the user. For instance, users may describe a song using the words “happy Beatles guitar”. However, it is difficult for the computer to interpret music in this way. Current state-of-the-art media retrieval systems

(e.g. music web portals, Youtube.com, etc), allow users themselves to describe the media items by their own tags. Subsequently, users in the systems can retrieve the media items via keyword matching with these tags. With this form of collaborative tagging, each music item have tags providing a wealth of semantic information related to it. By September of 2008, users on Last.fm (music social network system) has annotated 3.8 million items over 50 million times using a vocabulary of 1.2 million unique free-text tags. Due to the social tags containing rich semantic information, plenty of works have explored the usefulness of social tags on information retrieval [1–3].

However, social tagging invokes two problems that makes it hard to be incorporated for information retrieval. First, social tags are error-prone as the tags can be annotated by any user using any word. Second, there is the long tail theory – most of tags have been annotated to a few popular objects. Therefore, the tags appear useless as it is often easier to retrieve popular items via other means (also known as sparsity problem).

Currently, many works focus on the sparsity problem of social tags using automatic annotation techniques. By employing such techniques, tags can be applied to the items that are similar to the annotated items. The challenges these are multi-fold, such as whether a *model-driven* method or a *data-driven* approach is more suitable to address this problem. Model-driven means that one attempts to build a model relating query words with audio data and noisy social tags. Data-driven on the other hand seeks to relate noisy social tags with query words. In this thesis, we focus on how to design a novel Model-driven method and combine these two approaches to improve the performance of music search engines.

1.2 What We Have Done

To address social tagging problems, in this thesis, we will propose three novel methods.

1. We proposed two Model-driven methods (Method 1 and 2) to improve the performance of automatic annotation, all them will be introduced in Chapter 3.

2. We also proposed one scheme combined method (Method 3) to address large-scale tag recommendation issue, it will be introduced in Chapter 4.

1.3 Contributions

Our main contributions are summarized as follows:

1. We modify the Corr-LDA model as Method 1 that is from a family of models that have been used in text and image retrieval for the music retrieval task.
2. The proposed Method 2 – TOB-SS performs very well;
3. We propose an alternative data fusion method that combines social tags mined from the web with audio features and manual annotations.
4. We compare our method with other existing probabilistic modeling methods in the literature and show that our method outperforms the current state-of-the-art methods.
5. We also evaluate the performance of diverse music low-level features, include Mixture Gaussian Model (GMM) and Codebook techniques.
6. To the best of our knowledge, the Method 3 is the first work to consider Explicit Multiple Attributes based on content similarity and tag semantic similarity for automatic music domain tag recommendation.
7. We present a parallel framework in Method 3 for offline music content and tag similarity analysis including parallel algorithms for audio low-level feature extractor, music concept detector, and tag occurrence co-occurrence calculator. This framework is shown to outperform the current state of the art in effectiveness and efficiency.
8. We have implemented a prototype search engine for Query-by-description to demonstrate a novel way for music exploration.

1.4 Organization of the Thesis

From what has been discussed above, several challenges are invoked in this domain. This thesis will address such challenges in the following chapters: a comprehensive survey of the existing literatures will be presented in Chapter 2, two proposed Model-driven methods will be introduced in Chapter 3 and one combined method will be presented in Chapter 4. A prototype QBD system for demonstrating the idea of search engine will be shown in Chapter 5. In Chapter 6, we will draw a conclusion of whole thesis. The details of mathematic proof on proposed Method 1 will be listed in Chapter 6.

Chapter 2

Existing Work

Query-by-text, in particular Query-by-description(QBD) is popular in academic society. Several years ago, because the number of songs is pretty small, thus can be managed by human being. As long as increasing number of music is avaliable online, to manually annotate the music pieces is extremely difficult. As discussed above, we have known that the key of QBD system is to compute the score matrix of each song given by the query. There distinct methods in the literature aim to address this problem.

1. Model-driven Method
2. Data-driven Method
3. Combined Method

2.1 Model-Driven Method

In Model-Driven method, the relationship between semantic meaningful words(e.g. social tags and annotation) and music low-level features will be learnt by adopting some powerful machine

learning algorithms, such as GMM model and SVM, which contains the following important issues:

1. What to be used for representing music items?

2. How to map the music items to semantic space?

2.1.1 What to be used for representing music items?

Pandora [1] employs professional or musicians to annotate the aspects of music items, such as the genre, instrument, etc. However, this approach is labor intensive and slow. With the increasing amount of music appearing every month, it is almost impossible to annotate all the music items in time. Fortunately, with the popular of Web 2.0, people are getting more and more interested in tagging web resources including music pieces for further search in social networks system. Thus the Internet becomes an important source for collecting tags of music items:

Web pages - With the advancement of search techniques, some search engine such as Google can return more relevant documents when issued with a user query, which can be used to represent a music item. Peter Knees et al. [4] use the terms from content of top 100 Web pages returned by Google for representing music items.

Blogs - With the popular of Blogs, some web users write some music review on their Blogs, which makes them another resource for representing music items. Malcolm Slaney et al. [5] collected a few Blog pages to represent the related songs. [2]

Social Tags - With the rising of music social networks, such as Last.fm and Youtube, users tend to use a few short words to annotate music items. Therefore, a music item can be represented with those tags associated with it. By September 2008, over 50 million free-text tags of

[1] http://www.pandora.com
[2] http://hypem.com

which 1.2 million tags are unique have been used for annotating 3.8 million items [6].

2.1.2 How to learn the mapping between music items and music semantic meanings?

The semantic gap generally affects the domain of multimedia search and researchers have been trying to find out effective ways to bridge the semantic gap. Consequently, we need to construct a semantic space and learn a mapping between the low-level feature space and the semantic space.

Construction of the semantic space

The semantic space is a set of terms, which has different semantic meanings. All the research works have constructed a semantic space to represent the music items. The only difference is that how to choose the words as the basis of semantic space. The semantic space can be constructed manually, which can be very useful but cannot be extended easily. Bingjun et al. [7] construct such space with limited dimensions, such as genre, mood, instrument, etc. Therefore, automatically constructing a music semantic space is very attractive by using the online web resources such as Web documents [4, 8], Blogs, social tags [3] and so on. However, it contains more noise than manually constructed semantic space, which calls for more efficient algorithms to construct such space from the raw document and/or social tags.

Representing the music items by using constructed semantic space

Machine learning methods such as graphic model and classification-based methods are widely employed to learn the mapping. Blei et al. proposed a generative model to modeling the annotation data [9], which is further extended to learn the mapping between tags and media

items such as images and songs. In [10, 11], Muswords, similar to bag-of-word in text domain, was created by content analysis of songs. They also constructed a bag-of-word of tags, and *Probability Latent Semantic Analysis*(PLSA) was used to model the relationship between music content and tags. In [12], the authors constructed a tag graph based on TF-IDF similarity of tags. The semantic similarity between music items can be obtained by computing the joint probability distribution of content-based and tag-based similarity. Carnario et al. [13] proposed a novel method – *supervised multi-class labeling* (SML) to learn the mapping function between images and tags. Douglas et al. [8,14] applied the method used in [13] to represent music items by a predefined tag vocabulary.

The work presented in [3] is an example of classification-based methods, a bank of classifiers (Filterboost) are trained to predict tags for music items. The mapping between low-level features and semantic items (e.g. tags) can be determined by using SVM classifiers [7, 15] to map the low-level features into different categories in semantic space.

Slaney et al. used a different approach to learn the mapping. They tried to learn a metric for measuring the semantic similarity between two songs. The forms and parameters of a metric are adjusted so that two semantic close songs get high value of similarity [5].

Paper Index	Learning Methods	Semantic Space	Application
[3]	Filterboost	Top tag from last.fm	Automatic tagging
[12]	MRF	All tags from dataset	Classification
[10, 11]	PLSA	Social tags	Retrieval
[8, 14]	SML	Social tags, web pages	Retrieval
[7, 15]	SVM	Predefined categories	Retrieval
[4]	PLSA	Terms from related Web pages	Retrieval

Table 2.1: Summary of the related works

2.2 Data-driven Method

As an emergent feature in Web 2.0, social tags, is allowed by many websites to markup and describe the web items (Web pages, images or songs). Such social tags, in some senses, has tremendous semantic meaning. For instance, Youtube accepts customers to upload video clips and advocates them to attach relevant meaningful descriptions (social tags). Data-driven method assume that as long as increasing number of human being attach a certain item with similar tags, the tags could be correct to describe the item. Such kind of knowledge from plenty of folks, also be known as folksonomy, directly contributes to many commercial system, such as Youtube, Flicker and Last.fm. The retrieval engines in such commercial product directly index the tags using maturely text retrieval techniques. It is valuable to highlight that such method does not involve any content-based techniques, it could be efficient enough and easy to be deputed as a stable system to handle millions even billions of images or songs. Unfortunately, such method only performs well when the items in such system has large mount of tags, in turn with few tags, the performance of it is pretty poor.

2.3 Existed Works in Image Community

In order to improve the quality of online tagging, there has been extensive work dedicated to automatically annotating images [16–19] and songs [3, 20–22]. Normally, these approaches learn a model using objects labeled by their most popular tags accompanied by the objects' low-level features. The model can then be used to predict tags for unlabeled items. Although these model-driven methods have obtained encouraging results, their performance limits their applicability to real-world scenarios. Alternatively, Search-Based Image Annotation (SBIA) [23, 24], in which the surrounding text of an image is mined, has shown encouraging results for automatic image tag generation. Such data-driven approaches are faster and more scalable than model-driven approaches, thus finding higher suitability to real-world applications. Both the model-

driven and data-driven methods are susceptible, however, to similar problems as social tagging. They may generate irrelevant tags, or they may not exhibit diversity of attribute representation.

Tag recommendation for images, in which tags are automatically recommended to users when they are browsing, uploading an image, or already attaching a tag to an unlabeled image, is growing in popularity. The user chooses the most relevant tags from an automatically recommended list of tags. In this way, computer recommendation and manual filtering are combined with the aim of annotating images by more meaningful tags. Sigurbjörnsson *et al.* proposed such a tag recommendation approach based on tag co-occurrence [25]. Although their approach mines a large-scale collection of social tags, Sigurbjörnsson *et al.*do not take into account image content analysis, choosing to rely solely on the text-based tags. Several others [26,27] combine both co-occurrence and image content analysis. In this thesis, we propose a method (Method 3) that considers both content and tag co-occurrence for the music domain, while improving upon diversity of attribute representation and refining computational performance.

Chen *et al.* [28] pre-define and train a concept detector to predict concept probabilities given a new image. In their work, 62 photo tags are hand-selected from Flickr and designated as concepts. After prediction, a vector of probabilities on all 62 concepts is generated and the top-n are chosen by ranking as the most relevant. For each of the n concepts, their system retrieves the top-p groups in Flickr (executed as a simple group search in Flickr's interface). The most popular tags from each of the p groups is subsequently propagated as the recommended tags for the image.

There are several key differences between [28]'s approach and our method 3. First, we enforce Explicit Multiple Attributes, which guarantees that our recommended tags will be distributed across several song attributes. Additionally, we design a parallel multi-class classification system for efficiently training a set of concept detectors on a large number of concepts across the Explicit Multiple Attributes. Whereas [28] directly uses the top n concepts to retrieve relevant groups and tags, we first utilize a concept vector to find similar music items.

Then we use the items' entire collection of tags in conjunction with a unique tag distance metric and a predefined attribute space. The nearest tags are aggregated across similar music items as a a single tag recommendation list. Thus, where others do not consider attribute diversity, multi-class classification, tag distance, and parallel computing for scalability, we do.

Chapter 3

Model-driven Methods

In this chapter, we mainly focus on Model-driven method, and there are two fundamental problems we have to face are:

1. What kind of music representation (low-level content features) is more suitable for such task ?
2. What kind of model is more suitable for music automatic annotation task ?

We propose employing a novel method to improve the performance of previous work as well as evaluating diverse low-level features on such model. We plan to investigate the *problem 1* that discussed above, to evaluate what kind of music representation is more suitable for music automatic annotation under the discriminative model, such as SVM classifier. To this end, we study diverse state-of-the-art probabilistic models, such as: SML [20], CBA [21], and we propose employing a revised Corr-LDA [9], Corr-LDA for short, and Tag-level One-against-all Binary approach, named TOB-SS, to improve the performance of previous work. Our main contributions in this chapter are as follows:

1. We modify the Corr-LDA model that is from a family of models that have been used in

text and image retrieval for the music retrieval task.

2. The proposed method 2 – TOB-SS outperforms all the state-of-the-art methods on CAL500 dataset;
3. We propose an alternative data fusion method that combines social tags mined from the web with audio features and manual annotations.
4. We compare our method with other existing probabilistic modeling methods in the literature and show that our method outperforms the current state-of-the-art methods.
5. We have implemented a prototype search engine for Query-by-description to demonstrate a novel way for music exploration.
6. We also evaluate the performance of diverse music low-level features, include Mixture Gaussian Model (GMM) and Codebook techniques.

In this chapter, Section 3.1 presents our music retrieval framework, and Section 3.2 explains our features used. Section 3.3 present the modified Corr-LDA model as well as the other models we explore. Section 3.4 illustrates our evaluation measures, experiment results, analysis, and introduces our prototype system.

3.1 Framework

In this section we present an overview of the music retrieval system. Figure 3.1 illustrates the framework of this system. Users search music by typing keyword queries[1] such as "classical music piano" to obtain a ranked list of songs. This ranking is computed from the scores of each song given the keyword, and is in turn computed from an annotation model.

Initially, the system is presented with a labeled data set that consists of manually annotated songs (audio data). First, *feature extraction* is performed on the audio data to extract low level

[1]We assume the keyword queries is from a fixed vocabulary of annotations provided.

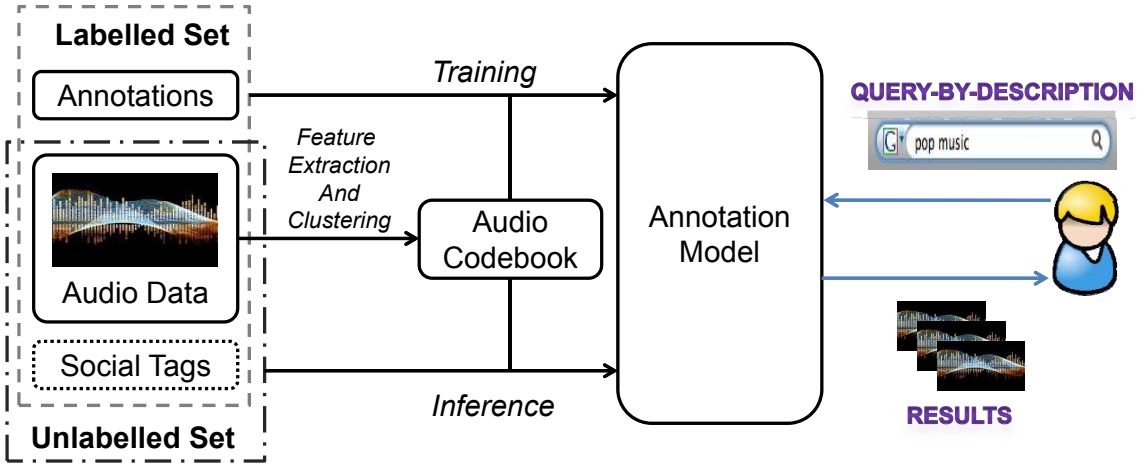


Figure 3.1: Basic Framework of an Music Text Retrieval System

(a) Model level

(b) Data level

Figure 3.2: Two different methods of fusing multiple data sources for annotation model learning

audio features. Then, a codebook is created via *clustering*. Each song is now represented by a bag of codewords. Next, an annotation model is *trained* using the new representation and annotations. Finally, the remainder of the unlabeled (without annotations) songs are annotated via *inference* with the model. New songs can be introduced to the system by representing them as a bag of codewords using the codebook and annotating them using the model. For retrieval, scores for each song given a keyword is computed using the annotation model and the top results presented to the user.

For this preliminary work, we further investigate the fusion of multiple sources of information such as “social tags” that are obtained from a real-world collaborative tagging web site. This is a source of additional information to the framework and is marked with a dotted box

in Figure 3.1. There are two ways in which social tags can be incorporated into the annotation model. First is the model level fusion method illustrates in Figure 3.2(a). A separate model is built for audio-annotation and for social-annotation. Then an ensemble method is used to combine the models. This was explored in [8]. Second, is the data level fusion method where the social tags are directly used to augment the song representation. The social tags are treated as new codewords and the same method is used to train the annotation model. We take the second approach in this report using the Correspondence LDA model [9] as we believe ensemble methods introduce too many additional parameters with added complexity to the model.

3.2 Features

The music data we use is the publicly available data set, Computer Audition Lab 500 (CAL500) [29]. It consists of a set of 500 "Western popular" songs from 500 unique artists. Each music track has been manually annotated by at least three people. These annotations construct a vocabulary of 174 "musically-relevant" semantic words.

3.2.1 Audio Codebook

In this chapter, we use Mel-Frequency Cepstral Coefficient (MFCC) as the music audio low-level feature. Each song is represented as a bag-of-feature-vectors [29]: a set of feature vectors that are calculated by analyzing a short-time segment of the audio signal. In particular, the audio is represented with a series of Delta-MFCC feature vectors. A time series of MFCC vectors is extracted by sliding a half-overlapping, short-time window (23 msec) over the songs digital audio file. A Delta-MFCC vector is calculated by appending the first and second instantaneous derivatives of each MFCC to the vector of MFCCs. The CAL500 data set provides MFCCs from three time windows, a total of 10,000 39-dimension feature vectors per song. Such huge number of features is tedious for training a model as there may be up to 5 million audio samples

for 500 songs. Hence codebook methods are required.

To create a codebook representation of MFCC data, we perform clustering on all MFCC feature vectors. We use standard K-means clustering with 500 clusters. Each cluster is a codeword in the codebook. Then, we represent the audio data of each song as a bag of codewords. Specifically, each song has 500 audio codeword features. The values of these features is the count of MFCCs of the song that belongs to the codeword (cluster). This is similar to the codebook approach in [21].

Gaussian Mixture Model (GMM)

Gaussian Mixture Model is very popular in multimedia clustering and classification. We employ this method to cluster the samples of each song. GMM model is relatively similar to K-means, the most different point here is that rather than perform clustering on whole data set, GMM just performs clustering on samples of one song. In this chapter, we set the number of cluster to 8.

Simple Segmentation (SS)

Another intuitive approach of dimension reduction is based on the direct segmentation of the music clip. Each music clip can be divided into K sub-clips, and the feature of each sub-clip can be represented as the mean and the standard deviation of the MFCC features within it. The number of segments in each music is closely associated with the representation accuracy. Therefore, different K values are studied and compared in our work.

3.2.2 Social Tags

In this section, we describe how to extract meaningful representations from social tags. For each pair of a song s and a tag t, we derive a relevance score $r(s,t)$ evaluating the relevance of the song and the tag. However the song-tag relevance scores resulting from social tags are considered sparse since the strength of association between songs and tags is unknown due to the nature of social context.

We can summarize each song with an annotation vector over a vocabulary of tags. Each element of this vector indicates the relevant strength of association between the song and a tag. The annotation vector is generally sparse in that most songs are annotated with only a few tags. A song-tag pair can be missing because either the song and the tag don't match or the tag is relevant but nobody has ever annotated the song with it.

As a music discovery web site, Last.fm, allows users to add tags to tracks, artists, albums, etc. via a text box in their various audio player interfaces. By September of 2008, the 20 million monthly users had annotated 3.8 million items over 50 million times by using a vocabulary of 1.2 million unique free-text tags.

For each song s in the CAL500 corpus, we collect two lists of social tags from Last.fm by using the API provided. One list relates the song to a set of tags where each tag has a tag score ranging from 0 to 100. The score is computed by integrating the number and diversity of users who have annotated the song with the tags, which is the trade secret of Last.fm. The other list associates the artist with tags and aggregates the tag score for all the songs by that artist. We gather the top 100 tags for each song and each artist, and combine the scores of the song-tag pairs and the artist-tag pairs to generate a final score $r(s,t)$ for each song-tag pair. That is, the relevance score $r(s,t)$ for song s and tag t is the sum of the same tag scores on the artist list and song list. For instance, if the song-tag pair $<$ *as long as you love me, pop* $>$ has a score of 60 and the artist-tag pair $<$ *backstreet boys, pop* $>$ has a score of 35, the final relevance

score $r(\textit{as long as you love me, pop})$ is 95. Social tag data for each song is represented by a set of song-tags with their relevance score. For the CAL500 corpus this results in a song-tag vocabulary size of slightly more than 16,000.

3.3 Modeling Techniques Investigated

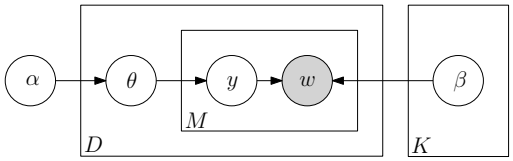
In this section, we briefly review the main models of interest as well as two other models for comparison. All four kinds of models are probabilistic in that they encode a joint probability distribution over the annotation terms (words), and the audio features (codewords). From there, the probabilities of a each word given the codewords of a particular song, i.e. $P(word|codewords)$, is used as the score to rank retrieval results for a given query word.

3.3.1 Proposed Method 1 – Correspondence Latent Dirichlet Allocation (Corr-LDA)

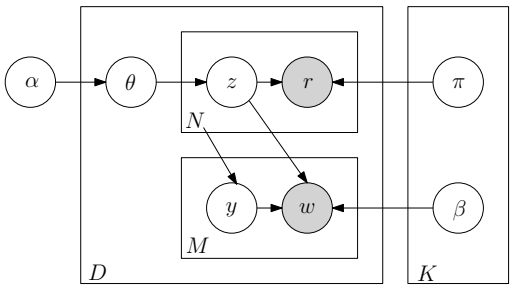
Latent Dirichlet Allocation (LDA) is a generative model originally used to model text documents [30] and is illustrated in Figure 3.3(a). Briefly, each of the D documents in the corpus has a distribution over topics, θ, drawn from a Dirichlet distribution parameterized by α[2]. For each word, w, in the document, a particular topic, y, is first drawn from θ. The particular topic is one of the K possible topics represented by β variables that are distributions over words. Then, the word is drawn from the particular β. The key point is that every word can come from a different topic and every document has a different mix of topics given by θ. The Dirichlet distribution serves as a smooth (continuous) distribution such that a particular point sampled from it will give the parameters of a multinomial distribution – in this case the distribution over

[2]For simplicity we use the same α for all Dirichlet parameters of a K dimension distribution instead of individual $\alpha_1, ..., \alpha_K$. This means that a higher value of α concentrates the probability mass more at the centre of the K topics.

topics, θ. As there are multiple levels of latent variables that are conditional on other latent variables, this is an example of a Hierarchical Bayesian Network (HBN).



(a) Latent Dirichlet Allocation



(b) Correspondence Latent Dirichlet Allocation

symbol	definition
D	Number of documents
M	Number of words
M'	Number of unique words
K	Number of topics
N	Number of codewords
N'	Number of unique codewords
α	Dirichlet distribution for θ
θ	Distribution over topics
y	Particular word topic (LDA) / Codeword identifier (Corr-LDA)
w	Particular word
β	Word topic
z	Particular codeword topic
r	Particular codeword
π	Codeword topic
R	codeword vocabulary size
W	word vocabulary size

(c) Legend of symbols

Figure 3.3: Graphical LDA Models, plate notation indicates that a random variable is repeated

The Corr-LDA model is an extension of LDA that is used to model annotated data. These are text annotations associated with some other elements in a mixed document. It has been primarily used in the image retrieval domain where the other elements are image regions [9, 31][3]. However, we observe that the model may be more generally applied to codewords instead of just image regions. These codewords can be our audio codewords from the clustered codebook, or summaries of any other types of data that have accompanying annotations, such as web sites, essays, videos, etc. This generalization allows us to treat social tags from collaborative tagging web sites as additional codewords of a particular song naturally leading to the data level fusion shown in Figure 3.2(b)[4]. The counts of the social tag codewords are represented by the relevance score (Section 3.2.2). More formally, Corr-LDA is shown in Figure 3.3(b) and has the following generative process for *each* document in the corpus D:

[3]The version of Corr-LDA we use is in-between the presented version in [9] and the supervised version in [31]. The main difference is that we do not have a class variable unlike in [31] but we use a Multinomial distribution over codewords instead of the Gaussian distribution over image regions in [9].

[4]We have assumed the audio codewords and social tags to be independent given the latent variables.

1. Sample a distribution of codeword topics from a Dirichlet distribution, $\theta \sim Dirichlet(\alpha)$

2. For each codeword, $r_n, n \in [1, N]$, in document:
   (a) Sample a particular codeword topic ($z_n \in [1, K]$), $z_n|\theta \sim Multinomial(\theta)$
   (b) Sample a particular codeword, $r_n|z_n \sim Multinomial(\pi_{z_n})$
3. For each word (annotation), $w_m, m \in [1, M]$, in document:
   (a) Sample a particular codeword identifier ($y_m \in [1, K]$), $y_m \sim Uniform(N)$
   (b) Sample a particular word $w_m|z_{y_m} \sim Multinomial(\beta_{z_{y_m}})$

Steps 1 and 2 of the generative process is exactly LDA if we rename the codeword as words. The extension for annotations is in Step 3. For each annotation the codeword identifier y_m is conditional on the number of codewords as shown in Figure 3.3(b) with an arrow from N to y. This means that we pick a word topic that corresponds to one of the codewords present in the document before proceeding to sample from the topic to get the word. The more a codeword appears in a document, the more we are likely to pick a word topic associated with it due to the Uniform distribution used in Step 3(a). This is the link between the codewords and annotations. In other words, the values for variables z_n and y_m are indexes to Multinomial distributions for codewords (π) and words (β). Learning these distributions and the value of α that controls the distribution that documents come from is the objective of training the annotation model. As π and β are Multinomial distributions we write π_{i,r_n} to be $p(r_n|z_n = i, \pi)$ and β_{i,w_m} to be $p(w_m|y_m = n, z_n = i, \beta)$.

The joint probability distribution given the parameters of a single document encoded by the Corr-LDA model is,

$$p(\mathbf{r}, \mathbf{w}, \theta, \mathbf{z}, \mathbf{y}|\Theta) = p(\theta|\alpha) \left(\prod_{n=1}^{N} p(z_n|\theta) p(r_n|z_n, \pi) \right) \left(\prod_{m=1}^{M} p(y_m|N) p(w_m|y_m, \mathbf{z}, \beta) \right) \quad (3.1)$$

where bold font indicates the sets of variables in the document and $\Theta = \{\alpha, \pi, \beta\}$ are parameters of the model. The joint probability distribution of the whole corpus is the product of the

per document distribution of all documents. The first posterior distribution of interest that can be used as a score for a word for each song is $p(w|\mathbf{r}, \Theta)$. The second is the posterior probability of a document, $p(\mathbf{w}, \mathbf{r}|\Theta)$, that is essential for estimating the parameters of the model and to compute $p(w|\mathbf{r}, \Theta)$. However computing the second value is intractable due to the coupling between the integration over θ, and summation over variables π and β during marginalization. Hence approximate inference methods must be used.

We use the same approximate inference method used in [9, 31], namely, Variational Inference. This method uses a simpler distribution,

$$q(\theta, \mathbf{z}, \mathbf{y}|\gamma, \phi, \lambda) = q(\theta|\gamma) \left(\prod_{n=1}^{N} q(z_n|\phi_n) \right) \left(\prod_{m=1}^{M} q(y_m|\lambda_m) \right), \tag{3.2}$$

where γ, ϕ, λ are free variational parameters to be estimated, to approximate the posterior distribution of the latent variables, i.e. $p(\theta, \mathbf{z}, \mathbf{y}|\mathbf{r}, \mathbf{w}, \Theta)$. From here, the lower bound of the log likelihood of a document is given by,

$$\begin{aligned} \log p(\mathbf{w}, \mathbf{r}|\Theta) &\geq \mathbb{E}_q[\log p(\theta, \mathbf{r}, \mathbf{w}, \mathbf{z}, \mathbf{y}|\Theta)] - \mathbb{E}_q[\log q(\theta, \mathbf{z}, \mathbf{y}|\gamma, \phi, \lambda)] & (3.3) \\ &= \mathcal{L}(\gamma, \phi, \lambda; \Theta) & (3.4) \end{aligned}$$

Section .1.1 presents the detailed components of Equation 3.4 and Section .1.2 shows an equivalent simplification that is used in actual computation. Maximizing Equation 3.4 is equivalent to minimizing the Kullback-Leibler (KL) divergence between $q(\theta, \mathbf{z}, \mathbf{y}|\gamma, \phi, \lambda)$ and $p(\theta, \mathbf{z}, \mathbf{y}|\mathbf{r}, \mathbf{w}, \Theta)$. Hence by directly optimizing Equation 3.4, we can obtain the lower bound log likelihood as an approximation to the true likelihood.

For optimizing the $\mathcal{L}$ of one document, we use standard numeric gradient ascent on the parameters γ, ϕ, λ to give the update equations:

$$\phi_{ni} \propto \pi_{i,r_n} \exp\left(\left(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{K} \gamma_j) \right) + \sum_{m=1}^{M} \lambda_{mn} \log \beta_{i,w_m} \right) \tag{3.5}$$

$$\gamma_i = \alpha_i + \sum_{n=1}^{N} \phi_{ni} \qquad (3.6)$$

$$\lambda_{mn} \propto \exp(\sum_{i=1}^{K} \phi_{ni} \log \beta_{i,w_m}) \qquad (3.7)$$

where $n \in [1, N], i \in [1, K], m \in [1, M]$. These updates are iterated until $\mathcal{L}$ converges or the maximum specified number of iterations is reached. The full derivation of the gradient is given in Section .1.3.

To learn the parameters, Θ, of the Corr-LDA model, Variational Expectation Maximisation (VEM) algorithm can be used. This is the same as the standard EM algorithm but with variational inference for the inference step. VEM is achieved by iterating the two steps below until the lower bound log likelihood of the entire corpus converges or the maximum number of iterations has been reached.

E-Step For each document, perform Variational Inference until $\mathcal{L}$ converges, i.e. we optimize the set of $\{\gamma_d, \phi_d, \lambda_d\}$ for one document. The lower bound log likelihood for the corpus is the sum of each document's $\mathcal{L}$ value.

M-Step Maximize the model parameters, $\Theta = \{\alpha, \pi, \beta\}$ to get the Maximum Likelihood Estimate (MLE)

1. α is maximised using the Newton-Raphson method described in [30].
2. Set $\pi_{if} \propto \sum_{d=1}^{D} \sum_{n=1}^{N_d} 1[r_n = f]\phi_{dni}$
3. Set $\beta_{iw} \propto \sum_{d=1}^{D} \sum_{m=1}^{M} 1[w_m = w] \sum_n \phi_{dni}\lambda_{dmn}$

Where $1[a = b]$ returns 1 if $a = b$ and 0 otherwise.

The details of the gradient updates in the M-Step is given in Section .2. Note that in the actual implementation, in the E-Step, we accumulate the sufficient statistics after variational inference is performed for each document. This is the accumulation of π_{if} and β_{iw} updates shown. Consequently, in the M-Step we only calculate the α update and normalize π and β.

Hence we only iterate through D once per VEM iteration. The time complexity of the VEM is $O(a \cdot (b \cdot DKN'M' + K(R+W)))$ where: a is the maximum number of EM iterations, b is the maximum number of variational inference iterations, N' is the number of unique codewords in a document, M' is the number of unique words in a document, R is the number of unique codewords in the corpus, and W is the number of unique words in the corpus. The derivation of the $b \cdot DKN'M'$ term is due to the dominance of Equation 13 in Section .1.2 and being able to multiply the appropriate probabilities for each unique codeword/word by their occurrences in all given equations of $\mathcal{L}$. The $K(R+W)$ term is the normalizing of the topic variables, π and β, using the sufficient statistics. The space complexity is $O(K(R+W))$ due to storing the Multinomial distribution parameters for the topic variables.

Finally, our posterior probability of interest that represents the score of each query word for each song is approximated by,

$$p(w|\mathbf{r}) \approx \sum_{n=1}^{N}\sum_{i=1}^{K} q(z_n = i|\phi_n)p(w|z_n = i, \beta) \tag{3.8}$$

$$= \sum_{n=1}^{N}\sum_{i=1}^{K} \phi_{ni}\beta_{iw} \tag{3.9}$$

This score is used to annotate the unlabeled songs in the data set for ranking during retrieval.

3.3.2 Proposed Method 2 – Tag-level One-against-all Binary Classifier with Simple Segmentation (TOB-SS)

An intuitive idea is to convert this problem to multi-class problem, we divided the multi-label problem into multiple classes (tags) binary classification problem, named Tag-level One-against-all *Binary* approach, TOB for short. By using TOB, we can estimate the probability to determine how good the songs can be annotated by each tag based on previous trained SVM model on this tag. After that, we can get a probability matrix, whose row denotes songs and

column means tags.

The differences between TOB approach and Audio SVM method [22] are following. Firstly, we use the different low-level features, which has been discussed in section 3.2.1. Secondly, although both methods use SVM as its own classifier, TOB is Tag-level One-against-all *binary* classifier which is differ from Audio SVM's multi-class classifier.

In this section, we are proposing a novel method TOB-SS, which combining of TOB and Simple Segmentation scheme due to that it is simple and easily be extended to a parallel algorithm, which is s crucial component in large-scale real world QBR-MIR system. The process could be divided into 3 steps:

1. For each song, we extract the short-time window MFCCs samples, then 10,000 MFCCs samples could be extracted out. By using Simple Segmentation Scheme, we then obtain N samples for each samples;

2. For each tag, we collect samples of all related songs and then set the label of these samples to +1 as well as set the irrelated samples to -1. Half songs then used as training set and the remain part used as testing set;

3. After training and testing process, we obtain the probability of this tag over all songs, then we repeat the process on each tag and can obtain the probability matrix.

Firstly, we investigate this model on diverse music representation. After we obtain the best combination, we compare such combination with state-of-the-art model, in particular, the CBA model [21], SML model [20] and Audio SVM [22].

3.3.3 Codeword Bernoulli Average (CBA)

Codeword Bernoulli Average (CBA) is an simple probabilistic model to predict what words (annotations) will apply to a song and what songs are characterized by a word. CBA models the conditional probability of a word, w ,appearing in a song, j, conditioned on the empirical distribution $\boldsymbol{n_j}$ of codewords extracted from that song.

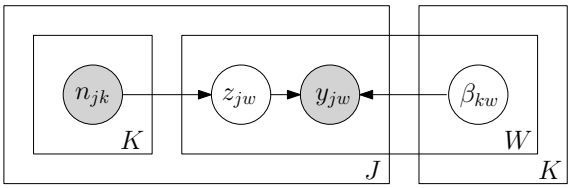


Figure 3.4: Graphical CBA Model

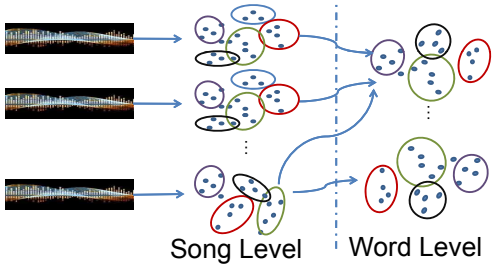


Figure 3.5: SML Model

CBA (Figure 3.4) assumes a collection of binary random variables $\boldsymbol{y}$, with $y_{jw} \in \{0, 1\}$ determining whether word, w, applies to song j. A value for y_{jw} is chosen from a Bernoulli distribution with parameter β_{kw}:

$$p(y_{jw} = 1|z_{jw}, \boldsymbol{\beta}) = \beta_{z_{jw}w} \tag{3.10}$$

$$p(y_{jw} = 0|z_{jw}, \boldsymbol{\beta}) = 1 - \beta_{z_{jw}w} \tag{3.11}$$

where z_{jw} is a codeword selected with probability proportional to the number of times, n_{jk}, that the codeword appears in song j's feature data.

We fit CBA with Maximum Likelihood Estimation (MLE) and our goal is to estimate a set of values for our Bernoulli parameters $\boldsymbol{\beta}$ that will maximize $p(\boldsymbol{y}|\boldsymbol{n}, \boldsymbol{\beta})$ of the observed words $\boldsymbol{y}$ conditioned on the codeword counts $\boldsymbol{n}$ and the parameter $\boldsymbol{\beta}$. We use the Expectation-Maximization (EM) approach because analytical MLEs for $\boldsymbol{\beta}$ are not available due to the latent variables $\boldsymbol{z}$. In the expectation step, we compute the posterior of the latent variables $\boldsymbol{z}$ given

the current estimates for the parameters $\boldsymbol{\beta}$:

$$h_{jwk} = p(z_{jw} = k|\boldsymbol{n}, \boldsymbol{y}, \boldsymbol{\beta}) = \frac{p(y_{jw}|z_{jw} = k, \boldsymbol{\beta})p(z_{jw} = k|\boldsymbol{n})}{p(y_{jw}|\boldsymbol{n}, \boldsymbol{\beta})} = \begin{cases} \frac{n_{jk}\beta_{kw}}{\sum_{i=1}^{K} n_{ji}\beta_{iw}} & \text{if } y_{jw} = 1 \\ \frac{n_{jk}(1-\beta_{kw})}{\sum_{i=1}^{K} n_{ji}(1-\beta_{iw})} & \text{if } y_{jw} = 0 \end{cases} \tag{3.12}$$

In the maximization step, we find maximum likelihood estimates of $\boldsymbol{\beta}$ given the expected posterior sufficient statistics:

$$\beta_{kw} \leftarrow \mathbb{E}[y_{jw}|z_{jw} = k, \boldsymbol{h}] = \frac{\sum_j p(z_{jw} = k|\boldsymbol{h})y_{jw}}{\sum_j p(z_{jw} = k|\boldsymbol{h})} = \frac{\sum_j h_{jwk}y_{jw}}{\sum_j h_{jwk}} \tag{3.13}$$

Iterating between the two steps until the likelihood converge or satisfy a user threshold, we find a set of values for $\boldsymbol{\beta}$ under which the training data become more likely. Next, we can use them to infer the probability that a word, w, applies to a previously unseen song j based on the counts $\boldsymbol{n_j}$ of codewords for that song:

$$p(y_{jw} = 1|\boldsymbol{n_j}, \boldsymbol{\beta}) = \frac{1}{N_j}\sum_k n_{jk}\beta_{kw} \tag{3.14}$$

3.3.4 Supervised Multi-class Labelling (SML)

The other approach is to use probabilistic models such as a Gaussian Mixture Model (GMM) for each word (annotation) based on music low-level features. This is based on a class of Supervised Multi-class Labeling (SML) models [20]. However, this method also learns many models (one for each word) that have to be combined using a variety of ensemble methods. Hence it can be viewed as being more similar to methods that use discriminative models. Figure 3.5 depicts the SML model as a Hierarchical Gaussian Mixture Model that has two steps: 1) song level GMM; 2) word level GMM. For each word, the SML model learns the probability of each song given a word $P(song|word)$. Under a uniform word prior assumption [20], the score matrix that consists of the probabilities of $P(word|song)$ is used for retrieval.

3.4 Experiments

Our primary model of interest is the Corr-LDA model. As such, we conduct the most experiments on it. The Corr-LDA model was implemented in C++ based off the freely available C code for the LDA model. The SML and CBA models were implemented in Matlab. All models were run using ten-fold cross validation – the data set was partitioned into ten parts and each part is used as the unlabeled data set with the other nine parts as the labeled data. The output for each model is a probability distribution over all the annotation terms for each song. This probability matrix is then used as a ranking score for computing the evaluation measures mentioned in the previous section. In all, we have evaluated some 540 Corr-LDA models with different parameter settings and with social tags. We did not compare the time performance of the various methods due to incompatible platforms. On average Corr-LDA without social tags mostly requires a few minutes when used with 500 codewords. However with the additional 16,000 social tags, Corr-LDA may require a few hours. Last, we implement a simple web-based prototype music retrieval system to demonstrate the results.

3.4.1 Evaluation Method

3.4.2 Evaluation

We evaluated our models performance on an annotation task and a retrieval task using the CAL500 data set. We compare our results on these tasks with two sets of published results on this corpus: those obtained by Turnbull *et al.*. using mixture hierarchies estimation to learn the parameters to a set of mixture-of-Gaussians models [20], and CBA model [21]. In the 2008 MIREX audio tag classification task, the approach in [20] was ranked either first or second according to all metrics measuring annotation or retrieval performance and the CBA model just

won the Best paper Award of ISMIR 2009 [5].

Annotation Task

To evaluate our systems annotation performance, we computed the average per-word precision, recall, and F-score. Per-word recall is defined as the average fraction of songs actually labeled w that our model annotates with label w. Per-word precision is defined as the average fraction of songs that our model annotates with label w that are actually labeled w. F-score is the harmonic mean of precision and recall, and is one metric of overall annotation performance. Following [20], when our model does not annotate any songs with a label w we set the precision for that word to be the empirical probability that a word in the dataset is labeled w. This is the expected per-word precision for w if we annotate all songs randomly. If no songs in a test set are labeled w, then per-word precision and recall for w are undefined, so we ignore these words in our evaluation.

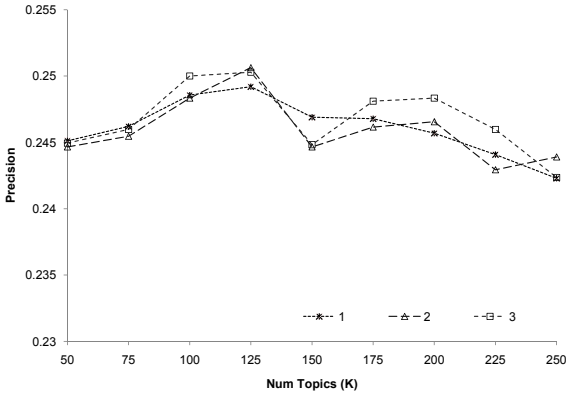
Retrieval Task

To evaluate our system retrieval performance, for each word w we ranked each song j in the test set by the score (probability) provided by the different models. We evaluated the mean average precision (MAP). MAP is defined as the average of the precisions at each possible level of recall. As in the annotation task, if no songs in a test set are labeled w then MAP is undefined for that label, and we exclude it from our evaluation for that fold of cross-validation.

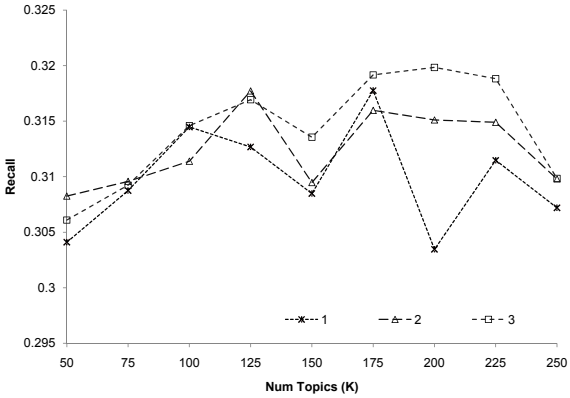
[5] http://ismir2009.ismir.net

3.5 Results & Analysis

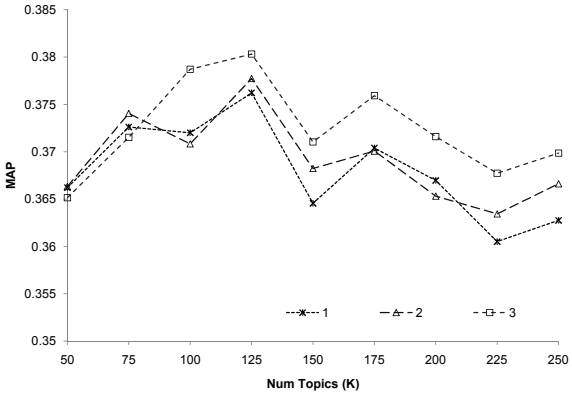
3.5.1 Corr-LDA Method



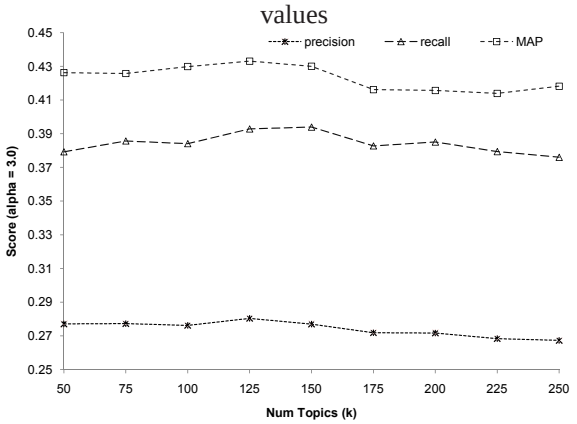
(a) Precision vs. Number of Topics for varying initial α values



(b) Recall vs. Number of Topics for varying initial α values



(c) MAP vs. Number of Topics for varying initial α values



(d) Scores vs. Number of Topics using social tags & initial $\alpha = 3.0$

Figure 3.6: Results for Corr-LDA model without social tags (a-b) and with (d)

Figure 3.6(a-c) depicts the results for the Corr-LDA model under different parameter settings. We vary the number of latent topics (i.e. K) and the initial Dirichlet parameter α for values 1, 2, and 3. For the Precision and MAP measures (Figure 3.6(a,c)), Corr-LDA is affected by the number of topics. Across all initial α settings, the scores for both Precision and MAP peaks at 125 topics. This shows that both measures are sensitive to the number of topics and Corr-LDA's performance will decrease if there are too few or too many topics. Recall (Figure 3.6(b)), on the other hand, steadily increases until 125 topics and any further increase in the number of topics has mixed results depending on the value of the initial α. For all three mea-

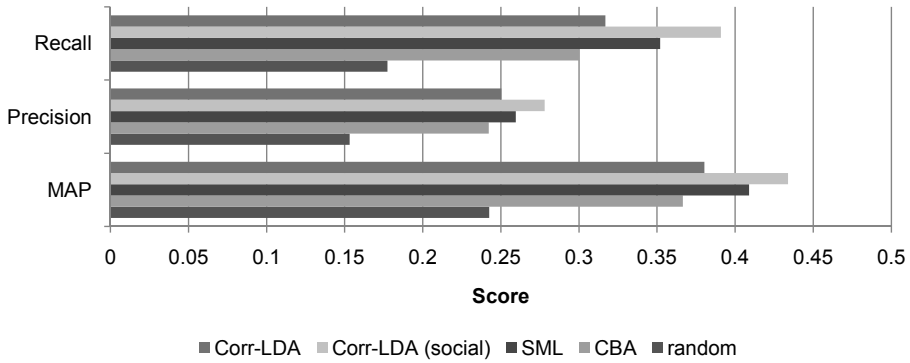


Figure 3.7: Comparison of the various annotation models. Corr-LDA has initial $\alpha = 2$ and Corr-LDA (social) has initial $\alpha = 3$. Both used 125 topics.

sures, the parameter setting of initial $\alpha = 3$ performs marginally better. Furthermore, although there is variation between the plots of α, most are slight and less than 2% different. This shows that Corr-LDA is not highly sensitive to α parameter settings – something we expect as α is learnt during training.

Figure 3.6(d) illustrates the plot for Precision, Recall, and MAP for the Corr-LDA model with social tags included as part of the data level fusion method. All three measures have improved over basic Corr-LDA. Similarly, their scores peak when 125 topics are used. This may mean that the effect of the number of topics is similar even if the codewords vocabulary size is greatly increased. The benefits of the social tags over plain Corr-LDA is seen in Figure 3.7 that compares the two variants of Corr-LDA with SML, CBA, and a random model that annotates songs randomly. Here we observe that incorporating social tags improves Corr-LDA by 5.35%, 2.78% and 7.40% for MAP, Precision, and Recall respectively. Furthermore, plain Corr-LDA performs better than CBA that uses a simpler probabilistic model. This shows that there is potential in the Corr-LDA model especially if approximate inference can be improved. Conversely, the SML model is better than the plain Corr-LDA model. This may mean that the assumption that audio codewords are independent given the words is valid. However, the Corr-LDA model with combined social tags out-performs the SML model. The results confirm that fusing multiple sources of information at the data level is an effective method to improve performance for music retrieval with Corr-LDA.

We did not evaluate the SML model with combined social tags in this report. However, the SML model requires storage of the probability of the codeword given word. This results in a storage requirement of the size of the codeword vocabulary multiplied by the size of the word vocabulary. Corr-LDA instead requires the space of the number of topic multiplied by the sum of the vocabulary sizes – a much smaller representation due to the use of latent topics.

3.5.2 TOB-SS Method

Evaluation on Diverse Music Representations

At the beginning of experiment, we first investigate the combination of three totally different representation of music and TOB-SS algorithm. Table 3.1 shows that with TOB model, almost all the Simple Segmentation Scheme (with different N) outperform other representations, in particular GMM and Codebook. In this table, we also can see that the training time will soar if we increase the N of each songs. The rational behind this is that the size of training samples depends on N.

Obviously, the TOB model obtains the best result while N being set to 12. Our goal of this combination is to find the tradeoff between N and MAP.

	Accu.	Prec.	Recall	F-measure	MAP	Train. Time(s)	Feat. Extraction Time (s)
GMM	87.11	0.228	0.101	0.140	0.491	1200.26	$>$ 24 hours
Codebook	87.48	0.283	0.097	0.144	0.339	147.57	$>$ 24 hours
$N = 1$	87.17	0.224	0.101	0.139	0.335	30.80	1095.28
$N = 4$	88.29	0.435	0.112	0.178	0.682	346.33	1083.19
$N = 8$	89.15	0.673	0.14	0.232	0.777	1096.78	1102.2
$N = 12$	89.65	**0.674**	0.185	0.29	**0.801**	**2340.52**	1083.92
$N = 16$	**89.9**	0.672	**0.221**	**0.332**	0.787	4474.91	1089.96

Table 3.1: The results

Evaluation on Different Models

So far, we have known that the combination, say TOB-SS, can obtain the highest performance. In this section, we will combine this method with other state-of-the-art models. We have re-implemented two state-of-the-art models: SML [20] and CBA [21], the former is the best performing system in MIREX 2008 and the latter one is the best paper of ISMIR 2009. We also compare this model with one work, which just posted on ACM Multimedia 2009 last month [22]. Since we do not have enough time to re-implement their method (Audio SVM and Affinity SVM), we just directly fetch the results from their paper. As shown in Table 3.2, our proposed algorithm outperform all of them except Affinity SVM in F-measure. Because the paper does not provide Precision, Recall and MAP, we cannot compare this with their model. Since MAP is differ from F-measure, we cannot estimate whether the model also enjoy better MAP. One thing for sure, the Affinity SVM is Two-Step algorithm framework, its first step is Audio SVM whose F-measure is little bit lower than TOB-SS model has, we can easily replace the first step with our model in the near future.

	Prec.	Recall	F-measure	MAP	Training Time
CBA	0.275	0.16	0.202	0.385	$>$ 24 hours
SML	0.284	0.162	0.206	0.409	$>$ 48 hours
TOB-SS (N=16)	**0.672**	**0.221**	0.332	**0.787**	**1.2 hours**
Affinity SVM			**0.498**		

Table 3.2: Comparison Between Different Models

3.5.3 Computational Cost

In this section, we will study the effect on the parameter N in SS. Firgure 3.8 shows the relationship between MAP, Training Time and N. The training time soars while increasing N. However, at same time, the MAP seems converged after $N = 8$. The figure illustrates that $N = 8$ can be the best choice.

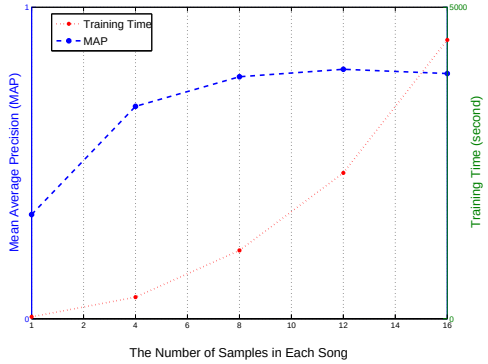


Figure 3.8: MAP vs. Training Time Curve

Chapter 4

Combined Method - Method 3

The previous chapter, we introduced the Model Driven methods. Along with the soaring of mount of data, especially the case using data crawling from Internet, the model based methods are extremely difficult to handle the issue with large scale data, both from the computation and accuracy aspects.

In this chapter, we will combine the Model Driven method and Data Driven method to address the large scale data. Content of this chapter is mainly based on our ACM Multimedia 2010 regular paper listed in **List of Publications**.

4.1 Large-scale Music Tag Recommendation with Explicit Multiple Attributes

In just over a decade, online music distribution services have proliferated, giving music a ubiquitous presence on the Internet. As the availability of online music continues to expand, it becomes imperative to have effective methods that allow humans to satisfactorily explore a

large-scale space of mixed content. This is a significant challenge, as there is no predefined universal organization of online multimedia content and because of the well-known semantic gap between human beings and computers, in which computers cannot interpret human meaning with high accuracy. For example, a human may search for a song with the primary keywords, "happy," "Beatles," and "guitar." A human intuitively understands that 'happy" is a common human emotion, "Beatles" is a popular rock band from the 1960's, and "guitar" is a 6-stringed instrument. Yet it is difficult to computationally interpret these words with high semantic accuracy.

Social tagging has gained recent popularity for labeling photos, songs and video clips. Internet users leverage tags found on social websites such as Flickr, Last.fm, and Youtube to help bridge the semantic gap. Because tags are usually generated by humans, they may be semantically robust for describing multimedia items and therefore helpful for discovering new content. However, because they are often generated without constraint, tags can also exhibit significant redundancy, irrelevancy, and noise.

In order to address the deficiencies of socially collaborative tagging, computer based tag recommendation has recently emerged as a significant research topic. Current recommendation systems rely on term frequency metrics to calculate tag importance. However, some attributes of online content are tagged less frequently, leading to attribute sparsity. For instance, music encompasses a high-dimensional space of perceived dimensions, including attributes such as vocalness, genre, and instrumentation. Yet many of these are relatively underrepresented by social tagging. For example, the four most popular tags associated with the musician Kenny G on Last.fm are "saxophone," "smooth jazz," "instrumental jazz," and "easy listening," which are *Instrument* and multiple *Genre* attributes. Thus, three out of the four most popular Kenny G attributes are related to genre. According to [3], *Genre* tags represent 68% of all tags found on Last.fm. Most of the remaining attributes are related to *Location* (12%), *Mood & Opinion* (9%), and *Instrument* (4%).

Because attribute representation is so highly skewed, the term frequency metric which most recommendation systems use may ignore important but less frequently tagged attributes, such as era, vocalness, and mood. In this chapter, we build upon the current image domain tag recommendation frameworks by considering Explicit Multiple Attributes and apply them to the music domain. The result is a recommendation system which enforces attribute diversity for music discovery, ensuring higher semantic clarity.

There were several novel challenges undertaken in our work. First, we constructed a set of music-domain Explicit Multiple Attributes. Second, scalable content analysis and tag similarity analysis algorithms for addressing millions of song-tag pairs were considered. Last, a fast tag recommendation engine was designed to provide efficient and effective online service. Our main contributions are summarized as follows:

1. To the best of our knowledge, ours is the first work to consider Explicit Multiple Attributes based on content similarity and tag semantic similarity for automatic music domain tag recommendation.

2. We present a parallel framework for offline music content and tag similarity analysis including parallel algorithms for audio low-level feature extractor, music concept detector, and tag occurrence co-occurrence calculator. This framework is shown to outperform the current state of the art in effectiveness and efficiency.

The structure of this chapter is as follows. In Section 4.2 we present the system architecture. We perform several evaluations of our system using two data sets in Section 4.3 and discuss our results in Section 4.4.

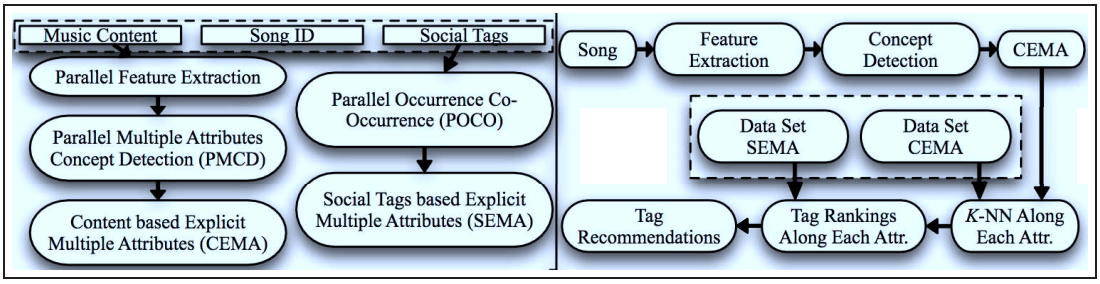


Figure 4.1: Flowchart of the system architecture. The left figure shows offline processing. In offline processing, the music content and social tags of input songs are used to build CEMA and SEMA. The right figure shows online processing. In online processing, an input song is given, and it K-Nearest Neighbor songs along each attribute are retrieved according to music content similarity. Then, the corresponding attribute tags of all neighbors are collected and ranked to form a final list of recommended tags.

4.2 System Architecture

Our system architecture, which is designed for scalability, is graphically depicted in Figure 4.1. We use a framework built on MapReduce to handle parallel processes. The system is functionally divided into two parts: offline processing and online processing, and comprised of two modules, Content based Explicit Multiple Attributes (CEMA) and Social tags based Explicit Multiple Attributes (SEMA). The CEMA and SEMA modules consequently maintain indexed lists of Multiple Attribute Fuzzy Music Semantic Vectors (MA-FMSVs) and Multiple Attribute Tag Distance Vectors (MA-TDVs). During offline processing, a large database of songs is analyzed. For each song, MA-FMSVs and MA-TDVs are generated by the Parallel Multiple Attributes Concept Detector (PMCD) and Parallel Occurrence Co-Occurrence (POCO) algorithms respectively. During online processing, the system quickly recommends attribute-diverse tags for a user presented song. The song's MA-FMSV is predicted by the Concept Detector and consequently used to index into CEMA and find its nearest neighbors. The nearest neighbors are in turn indexed into SEMA, resulting in a rank-sorted list of tags for each attribute. Each of the architectural components are discussed in detail below.

4.2.1 Framework

As the volume of multimedia data to be processed is potentially huge, multimedia information retrieval systems need to efficiently handle large-scale data-intensive computations. Therefore, the scalability of these systems is a major concern. Our framework attends to this issue directly.

A practical solution for addressing scalability is to distribute computations across multiple machines [32]. With traditional parallel programming models such as the Message Passing Interface, developers maintain the burden of explicitly managing concurrency. Thus, significant energy must be devoted to managing system-level details. In contrast, the MapReduce programming paradigm presents an attractive alternative [33]. MapReduce is based on the simple observation that many tasks share the same basic structure. With MapReduce, computation is applied over a large number of nodes to generate partial results and then the results are aggregated in some fashion [32]. MapReduce provides an abstraction for programmer defined "mappers" $(k_1, v_1) \rightarrow [(k_2, v_2)]$ and "reducers" $(k_2, [v_2]) \rightarrow [v_3]$, and keeps most of the system-level details hidden, such as scheduling, coordination, and fault tolerance. As shown in Figure 4.2, the "mappers" receive every $(key, value)$ pair from the input partition and emit an arbitrary number of intermediate $(key, value)$ pairs. A barrier then shuffles and sorts the intermediate pairs. "Reducers" are applied to all pairs with the same key to emit an output $(key, value)$ pair.

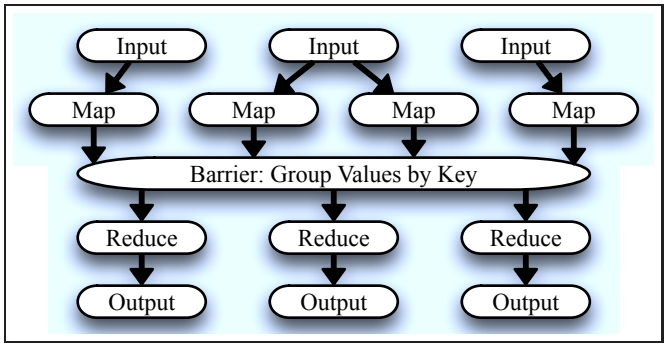


Figure 4.2: MapReduce Framework. Each input partition sends a $(key, value)$ pair to the mappers. An arbitrary number of intermediate $(key, value)$ pairs are emitted by the mappers, sorted by the barrier, and received by the reducers.

In our work, we use Hadoop[1] for back-end parallel processing, which is an open-source implementation of MapReduce. In Hadoop, a mapper is a JAVA class that contains three functions: setup, map, and cleanup. The setup function is called once when a mapper is started, the map function is called several times for each input key-value pair, and the cleanup function is called once when a mapper is going to be destroyed.

4.2.2 Explicit Multiple Attributes

Our work uses Explicit Multiple Attributes to enforce controlled attribute diversity for music content analysis and social tag recommendation, respectively. At the outset, we define a constrained set of A attributes and 2 attribute spaces. Each attribute in an attribute space may hold any number of elements, as long as more than one. We give both the CEMA and SEMA modules their own Explicit Multiple Attribute space with the same A attributes. However, their attribute spaces may differ in the elements they contain. The CEMA attribute space is used to to define the Multiple Attribute Fuzzy Music Semantic Vectors (discussed below). That is, every input song to the system will be classified by its representation within the CEMA Attribute space. The SEMA attribute space is used as an anchor point for the corpus of social tags. Since the global social tag space is noisy and contains many redundant and irrelevant terms, the elements in the SEMA attribute space are used as centroids to the entire tag corpus. As will be discussed below, any tag in the corpus is described in terms of its distances to the SEMA attribute space. These distances are stored in Multiple Attribute Tag Distance Vectors. By predefining these two attribute spaces, we can ensure attribute diversity and semantic clarity for tag recommendations.

[1] `http://hadoop.apache.org/`

4.2.3 Parallel Multiple Attributes Concept Detector (PMCD)

The Parallel Multiple Attributes Concept Detector (PMCD) is responsible for predicting the MA-FMSVs in offline and online processing. First, we train it on a database of labeled songs. Afterwards, we can use it to predict (offline) the MA-FMSVs of additional songs, giving us great flexibility for expanding the system's song tag representation without any additional training. Finally, the Concept Detector is used during online processing for recommending tags. Below, we discuss MA-FMSVs, the input to the Concept Detector (which is a vector of low-level music features), and the training process.

Multiple Attribute Fuzzy Music Semantic Vectors (MA-FMSVs)

For music content analysis, each song is represented by a Multiple Attribute Fuzzy Music Semantic Vector (MA-FMSV) which indicates, for each attribute, which element the song belongs to. FMSVs were proposed by [7] for use on music similarity measures, and are easily computed by a SVM classifier. The FMSV for one piece of song is a probability vector, in which each dimension denotes how similar to a certain aspect of music. For instance, the number of dimension in vector is 4(Pop, Jazz, Classical, Blues), the vector for a song represents the probability belongs to Pop, Jazz, Classical or Blues. For convenience, we concatenate the FMSV elements from each attribute to form a single vector, the MA-FMSV. Every song in our system is represented by its MA-FMSV. We first use a set of songs described by their low-level audio features and manually labeled with their MA-FMSVs for training the Concept Detector. Afterwards, any unlabeled song can be automatically assigned its MA-FSV by the Concept Detector.

MA-FMSVs are easily indexed using Locality Sensitive Hashing (LSH) [34]. As evaluated in [7], FMSV representations and LSH techniques accelerate the searching process among a large-scale data set ($\approx$ 0.5 seconds on a data set with 3000 samples and $\approx$ 1.7 seconds on a data set with 1 million samples). With LSH, we are able to efficiently find the K-Nearest

Neighbors of a predicted MA-FMSV. This is significant to saving time in our online processing for tag recommendation.

Low-level Music Feature Extraction

Low-level feature extraction is performed on all songs. Because the individual song feature extractions are independent of each other, it is easy for us to leverage the MapReduce framework and design a parallel algorithm for feature extraction. In this case, we only use the MapReduce mappers (Figure 4.2). Each song is stored in the cluster as a single line and is fed into a mapper. In the mapper, we use Marsyas [35][2] to extract low-level audio features, such as *Spectral Centroid*, *Rolloff*, *Flux*, and *Mel-Frequency Cepstral Coefficients (MFCCs)* for each short time frame. Finally, the averages and standard deviations across frames are used to summarize each song, resulting in a 64-dimensional feature space.

Training

Our concept detector uses a multi-class SVM predictor. Because our system does not set any constraints on the size of the number of elements in the CEMA attribute space, parallel processing is critical to ensuring scalability. Yet, it is difficult to design a SVM classifier with parallel processing. If using the MapReduce framework, one can allocate a mapper and a reducer for each iteration in the training stage [36]. However, the process can become cumbersome with large iteration sizes, so we seek an alternative algorithm for parallel computing.

A multi-class SVM classifier is usually decomposed into a set of independent binary SVM classifiers. Using this approach, we can take advantage of the MapReduce framework. There are several methods for decomposing a multi-class SVM classifier into multiple binary classifiers. We use the "one-versus-one," method because it performed the best on our data set during

[2] `http://marsyas.info`

informal evaluations. In "one-versus-one" binary classification, a set of classifiers is built for every pair of classes and the class that is selected by the most classifiers is voted as best.

In our work, we use a novel algorithm, which couples the Pegasos SVM solver [37], which is a very fast linear SVM solver, with a "Random Emitter" approach to Multi-Class SVM with MapReduce, as opposed to a "Normal Emitter" approach. In a "Normal Emitter" mode, the mapper acts as an emit controller. Each sample is emitted $N_C - 1$ times with a different classifier, where N_C is the number of classes in the data set. The two class labels (one-versus-one) are emitted as the key of the mappers' output. After sorting, all the samples with same key are sunk into the same reducer. Each sample in a reducer has a "$+1$" or "-1" label, where "$+1$" denotes that it belongs to the first class, and "-1" that it belongs to the other. The reducer then calls the Pegasos SVM solver to train a model for this category pair and dumps the model as the reducer's output.

The Pegasos implementation of binary SVM classification selects at random only a subset of samples to train a model, and the size of the subset is a function of the maximum iteration size specified by the user. Because of this, it is unnecessary for the mapper to emit all samples. A more sophisticated method of using MapReduce is "Random Emitter" (Algorithm 1), which randomly outputs samples and limits the size of the output to guarantee the number of samples is larger but not too much larger than the binary classifier's needs. Intuitively, the "Random Emitter" acts as the "Random Sampling" process within Pegasos. Note that "Random Emitter" is more efficient only when the size of the training data set is larger than the maximum iteration size of the binary SVM classifiers. The appropriate threshold can be calculated using this equation:

$$P_+ = P_- = \alpha \times \frac{N_C \times I}{2 \times N} \tag{4.1}$$

where P_+ is the threshold of emitting the sample as "$+1$," P_- is the threshold of emitting the sample as "-1," I is the maximum iteration size of the binary SVM classifier, N_C denotes the total number of classes, N represents the size of data set (the number of samples), and $\alpha > 1$

is a scalar to guarantee the number of emitted samples is larger than maximum iteration.

Algorithm 1 Random Emitter

Procedure: $RandomEmitter$
Input: S, N_C, I and N
Output: Sample string
1: Initialize P_+ and P_- by Equation 4.1
2: Get label $Label$ of input S (Sample string)
3: **for all** $i < Label$ **do**
4: Get random variable $r \in [0,\ 1]$
5: **if** $r < P_-$ **then**
6: Keys = i + "$-$" + $Label$
7: Values = "-1" + sample value
8: **end if**
9: **end for**
10: **for all** $j > Label\ and\ j < N_C$ **do**
11: Get random variable $r \in [0,\ 1]$
12: **if** $r < P_+$ **then**
13: Keys = $Label$ + "$-$" + j
14: Values = "$+1$" + sample value
15: **end if**
16: **end for**
17: **for all** $Key \in Keys$ **do**
18: Emit (Key, Value)
19: **end for**

Intuitively, if the number of training samples in the data set is larger than number of samples that the binary SVM classifier requires, then "Random Emitter" should be performed to limit the mappers' output. The expected output can be computed using the following equations:

$$I_E = I_{E+} + I_{E-} \tag{4.2}$$

$$I_{E+} = \frac{N}{N_C} \times P_+, \quad I_{E-} = \frac{N}{N_C} \times P_- \tag{4.3}$$

where I_E is the expected number of output samples, I_{E+} denotes the number of output samples with a "$+1$" label, I_{E-} denotes the number of output samples with a "-1" label, and P_+ represents the fraction of the number of emitted positive samples over the number of input

samples in a particular category. Consequently, we may easily infer the value of P_+:

$$I_E = 2 \times \frac{N}{N_C} \times P_+ \tag{4.4}$$

$$P_+ = \frac{N_C \times I_E}{2 \times N} \tag{4.5}$$

Obviously, if $r \sim U(0,1)$ (as described in Algorithm 1), then the size of the generated numbers in the range of $0 \sim P_+$ should be equal to the amount of samples that the Pegasos binary SVM training procedure needs. To guarantee the size of emitted samples is larger than required, a scalar α is used in Equation 4.1.

4.2.4 Parallel Occurrence Co-Occurrence (POCO)

The number of unique tags increases as more songs are collected, making it more challenging and time consuming to compute the co-occurrences between all tags. To tackle the scalability issue, a Parallel Occurrence Co-Occurrence (POCO) algorithm is proposed to generate the Multiple Attribute Tag Distance Vectors (MA-TDVs), which enable the online tag recommender to quickly retrieve appropriate attribute-diverse tags from the entire corpus of tags. Below, we describe MA-TDVs in more detail, including the tag distance metric used, and our POCO algorithms.

Multiple Attribute Tag Distance Vectors (MA-TDVs)

Multiple Attribute Tag Distance Vectors (MA-TDVs) are designed so that we can relate any tag in a tag corpus to a simplified diverse attribute space. Specifically, the vectors describe a song's tag distances between its socially ascribed tags and the SEMA attribute space chosen at

the outset of system implementation.

As there is no existing social web site which ascribes the distance between music tags, we must define our own tag distance metric for building our MA-TDVs. We use Google's word distance metric [38] for measuring tag distance:

$$d(t_i, t_j) = \frac{\max(\log f(t_i), \log f(t_j)) - \log f(t_i, t_j)}{\log N - \min(\log f(t_i), \log f(t_j))} \tag{4.6}$$

where $f(t_i)$ and $f(t_j)$ are the counts of songs containing tag t_i and t_j (occurrence), and $f(t_i, t_j)$ represents the number of songs having both t_i and t_j (co-occurrence). N stands for the total number of songs in the corpus.

The TDV for each tag is then calculated as the distance between itself and each of the terms in the SEMA attribute space. The terms in the SEMA attribute space act as a "codebook" for the music social tags space, and any social tag can be represented using a distance vector and the codebook. In this way, the TDVs of all music attributes can be calculated. For convenience, we concatenate the TDVs from each attribute to form the MA-TDV.

Design of a Scalable POCO algorithm: POCO-AIM

Efficient parallel word co-occurrence algorithms have been presented by [39], in which two methods using the MapReduce framework, "Stripes" and "Pairs," are evaluated. For our system, we begin by modifying the "Stripes" algorithm, which has been shown to be more efficient than "Pairs" if all words can be loaded into memory. In our case, the "words" are song tags, and we are calculating occurrence and co-occurrence between the terms in the SEMA attribute space and the tags associated with each song. Because tag occurrence is needed in our implementation for measuring tag distance (Equation 4.6), we must adapt the algorithm to also calculate word occurrence. Because only the distances between social tags and the terms in the SEMA attribute space are required in our work, we can reduce the space requirement of a

tag co-occurrence matrix from $O(N_T^2)$ to $O(N_T \times m)$, where N_T is the number of tags in the corpus and m is the number of terms in the SEMA attribute space.

In the modified "Stripes" mapper function, a key is one term in the SEMA attribute space. Its output is an associate array, which contains all tags not in the attribute space and their co-occurrences with the key. The mapper function thus generates a large number of intermediate results. We observe that a more sophisticated method is to aggregate the results in the mapper, rather than using a combiner or emitting them line by line [40]. We introduce this conservational upgrade into the algorithm's design and name the new method as POCO Aggregating in Mapper (POCO-AIM). Its implementation is given in Algorithm 2.

Algorithm 2 POCO-AIM

Class: $Mapper(Key, Tags \in Song)$
Input: $< Key, Tags \in Song >$
Output: $< tag, H >$
Procedure: $setup()$
1: INITIALIZE(H)
2: Load SEMA attribute set SA
Procedure: $map(Key, Tags)$
3: $I = Tags \bigcap SA$ // Intersection of Tags and SA sets
4: $D = (Tags - SA)$ // Difference of Tags and SA sets
5: **for all** t1 $\in$ I **do**
6: **for all** t2 $\in$ D **do**
7: H{t1}{t2} ++
8: **end for**
9: **end forProcedure:** $cleanup()$
10: **for all** $t \in H$ **do**
11: EMIT(tag,H(tag))
12: **end for**
Procedure: $Reduce(tag, [H_1, H_2, H_3, ...])$
Input: $< tag, [H_1, H_2, H_3, ...] >$
Output: $< tag, H >$
1: INITIALIZE(H)
2: **for all** $h \in [H_1, H_2, H_3, ...]$ **do**
3: MERGE(h,H)
4: **end for**
5: EMIT(tag,H)

In the setup function, the tags in the SEMA attribute space are loaded, and an associate array H is initialized. The input to the map function is the song ID and an array of its tags. In the map

function, the tags are processed and then classified into two groups. The first group I contains all the tags that occur in the SEMA attribute space, and the second group D contains the rest of the tags. Then, the co-occurrence between tags in I and D are computed and the associate array H is updated. Finally, in the cleanup function, the keys stored in H and their values are emitted. Compared with the modified "Stripes" method, the number of intermediate results and time taken to shuffle them is greatly reduced, leading to less overall computational time.

4.2.5 Online Tag Recommendation

In offline processing, our system constructs the CEMA MA-FMSVs and the SEMA MA-TDVs for all songs. In online processing, given a song without any tags, the system recommends the most appropriate tags within each attribute. Upon receiving an untagged song from a user, the online system extracts its audio low-level features. Then the online process predicts its MA-FMSV. The system looks for the K nearest songs by using the LSH index. In turn, the MA-TDVs are collected from the K nearest songs. The recommender sums and ranks the K MA-TDVs along each attribute to find the Top$N$ most relevant tags. The values for K can and N can be changed as parameters.

It is informative to take a closer look at tag ranking time, since the worst-case complexity of sorting is $O(n \log n)$. In our system, online tag ranking happens in two stages. In the first stage, n denotes the m elements in the SEMA attribute space. In the second stage, n is the total number of social tags in K-Nearest Neighbors that have been retrieved. Therefore, tag ranking time is expected to be much smaller than retrieval time.

4.3 Materials and Methods

We evaluated the quality of our system in several experiments using multiple data sets and evaluation criteria. In this section, we describe materials and methods for the experiments.

4.3.1 Data Sets

We gathered several data sets, summarized in Table 4.1, to train the concept detector and test the effectiveness of the tag recommendation system.

Name	Classes (Attr.)	Size (Train / Test)	Feat.
CAL-500	174 (6)	500	64
WebCrawl	20 (4)	77,448	64
HandTag	20 (4)	17,000	64

Table 4.1: Data sets used for training and testing.

CAL-500

CAL-500 [29] is a smaller-scale database that has been made publicly available for tag annotation and recommendation tasks. It includes a 39-dimensional feature set based upon differential MFCCs and has been used as a benchmark data set for several recent automatic tagging tasks, such as [3, 20, 21]. It consists of 500 songs and 174 classes distributed across 6 attributes: *Mood, Genre, Instrument, Song, Usage,* and *Vocal.* All tags were manually generated under controlled experimental conditions and are therefore believed to be of high quality.

WebCrawl

Our system is designed to efficiently operate on large-scale music data sets. Therefore, we needed an appropriately large data set to evaluate for testing. We generated WebCrawl by

crawling $488,407$ music items with metadata (*e.g.*title, album name, and artist) and social tags from Last.fm. We then used the title and artists' names to search for and download more than 200,000 songs from Youtube. After collecting all music items, we removed misspelled and stop words from the social tags using Wordnet [41] [3] and filtered out any songs without tags. We were left with 77,448 songs.

HandCrawl

The HandCrawl data set is another high quality manually tagged data set that has recently been used in [15]. The 17,000 songs were selected as the most popular in Last.fm's data base using its track popularity API. The tracks and metadata were retrieved by crawling YouTube. Socially tagged ground truth data was collected in controlled experimental conditions and cross checked by amateur musicians with reference to Last.fm. The ground truth data was associated with 4 attributes and 20 associated elements, as shown in Table 4.2.

Genre (14,713)		**Mood** (597)	**Vocalness** (2,131)	**Instrument** (1,588)
Classical	Jazz	Pleasure	Male	Brass
Country	Rock	Joyful	Female	WoodWinds
Electronic	Pop	Sad	Mixed	Strings
HipHop	Metal	Angry	NonVocal	Percussion

Table 4.2: The Explicit Multiple Attributes and elements in the HandTag data set. The number of songs represented by each attribute are shown in parentheses.

4.3.2 Evaluation Criteria

Our system is designed to recommend attribute-diverse and relevant tags given an input song. Additionally, we have proposed several methods for increasing computational efficiency when processing large-scale data spaces. In this subsection we set forth the main criteria by which we experimentally evaluated the system.

[3] `http://wordnet.princeton.edu/`

Precision and Accuracy

To evaluate our system's recommendation effectiveness, we follow the examples set in [20] and compute the average per-tag precision, recall, and F_1 score. Per-tag precision is the percentage of songs that our model recommends with tag t that are actually labeled with t in the song's ground truth tag vector. Recall is the percentage of songs labeled with t in the ground truth vector for which our model also recommends tag t. The F_1 score is the harmonic mean of precision and recall, and is a good metric for overall recommendation performance.

For each song, the tag recommenders provide a ranked list in order of predicted relevancy. In order to evaluate the quality of the recommender's ranking system for suggesting relative tags, we use Mean Average Precision (MAP@n), defined as the average of the precisions at each possible level of recall, where n is the recall depth (n is also termed the Top$N$ value). Therefore, MAP@n summarizes effectiveness of precision, recall, and ranking in a single metric. Again following [20], if our system doesn't recommend a tag t that is in the ground truth vector, then per-tag precision and recall for t are undefined, and we ignore these words in our evaluations.

Diversity

Our system aims to enforce attribute diversity in its tag recommendations. To quantify the diversity of a set of recommended tags, we define Diversity@n, which computes the proportion of attributes automatically generated in the top n tags:

$$\text{Diversity@}n \equiv \frac{\sum_i^n A(t_i)}{N_A} \tag{4.7}$$

where N_A is the total number of attributes, A is a vector and elements $\in \{0, 1\}$. $A(t_i)$ denotes which attributes t_i is a member of.

Computational Scalability

We have proposed several methods for improving the efficiency of parallel processes for large-scale tag recommendation. The main criteria that we investigate in our evaluations are computational time and data throughput.

4.3.3 Experiments

We executed two experiments designed to evaluate the two basic contributions of our work. The first evaluates effectiveness of tag recommendations on varying sized data sets. The second investigates the computational efficiency of the system architecture.

Tag Recommendation Effectiveness

We conducted two independent evaluations of tag recommendation effectiveness using two datasets: CAL-500 and WebCrawl. The CAL-500 data set is a popular benchmark for tag recommendation tasks. Thus, we are able to evaluate our work against others'. Hoffman *et al.*nicely summarized recent tag recommendation algorithms along with their own in [21]. We borrow their review and compare those results against several other implementations. In particular, we report evaluations on tag recommendation for seven methods, including our own:

1. MixHier: Based on a Gaussian Mixture of Models, uses the features included with CAL-500 [20].
2. Autotag: An AdaBoost based system using additional training data and features, along with those included with CAL-500 [3].
3. CBA: **C**odeword **B**ernoulli **A**verage is a probabilistic model based on using a codebook of size K [21]. For purposes of comparison, we chose to only report results with $K =$

500. Uses the standard feature set in CAL-500.

4. MD: A SVM method without tag propagation and ranking. This is similar to **M**odel-**D**riven methods with limited labels.

5. SB: Similar to the **S**earch-**B**ased Image Annotation [23, 24], a method that uses low-level features, rather than MA-FMSVs to find the K-NN songs.

6. FMSV: A method that uses **F**uzzy **M**usic **S**emantic **V**ectors, but doesn't consider Explicit Multiple Attributes [7].

7. MA-FMSV: Our system—tag recommendation with **M**ultiple **A**ttribute Explicit **F**uzzy **M**usic **S**emantic **V**ectors.

We note that we excluded results by Ness *et al.* [22] for two reasons: First, they do not use the full tag space available in CAL-500. Second, our concept detector is similar to the first stage of their two-stage framework; it can easily be extended to include the second stage. Procedures 4–7 were directly implemented by us. We used the feature extraction space discussed in this chapter, rather than CAL-500's feature set. Training and testing was done on the same data set, using 2-fold cross validation. For procedures 5–7, parameters K and N were set at 15 and 12, respectively.

For our second evaluation, we trained our system on the HandTag data set and tested on the WebCrawl data set. This evaluation was designed to test the system on a data space of much larger scale than the CAL-500 experiments. As such, we only report tag recommendation performance using procedures 4–7 above. For procedures 5–7, parameters K and N were set at 15 and 8, respectively.

In addition to the above evaluations, we also study the impact of K in K-NN and N in Top$N$ on the recommendation effectiveness of procedures 5–7.

Tag Recommendation Efficiency

We test the efficiency of our our system at two points: the PMCD algorithm, and the POCO-AIM algorithm. We evaluate the improvement of POCO-AIM's computational efficiency over a modified "Stripes" implementation, comparing the size of the mappers' intermediate output and the computing times. We used the Last.fm data set, as its size is considered to be appropriately large to model real-world tasks.

4.3.4 Computing

Our system runs on a cluster of 77 nodes (1 master, 76 slaves) comprising 22 TB storage capacity. A server is used as the master node, which has 2 X 4 core CPU (2.5 GHz) and 32GB memory. 28 machines with 2 core CPU (SUN V20Z, 2.18 GHz) and 2GB memory serve as slave nodes. The remaining 48 slave nodes come from 6 servers, and each server is divided into 8 virtual machines. Each server has 2nd Intel Quad Core E5506 Xeon CPU (2.13GHz, 4M Cache, 4.86 GT/s GPI) and 32GB memory. The expandable nature of the system guarantees that it can be easily extended to handle millions or even billions of songs.

4.4 Results

4.4.1 Tag Recommendation Effectiveness

CAL-500

Table 4.3 compares the results of evaluating multiple procedures on the CAL-500 data set. As reported in [20], the top two rows show the upper bound and a random baseline, respectively.

The SVM-based methods (MD, SB, FMSV, & MA-FMSV) performed better than any of the others; this has also been supported by [22]. The best recall and F_1 score results were obtained by the simplistic model-driven (MD) method, while precision was similarly high for MD and FMSV methods. Our system performed approximately 85% better than the next highest method (FMSV) in enforcing attribute diversity. Additionally, MA-FMSV was the best system for appropriately ranking its recommendations.

Method	Prec.	Recall	F_1 Score	MAP	Diver.
UpperBnd	0.712	0.375	0.491	1	1
Random	0.144	0.064	0.089	-	-
MixHier	0.265	0.158	0.198	-	-
Autotag	0.312	0.153	0.205	-	-
CBA	0.286	0.162	0.207	-	-
MD	0.606	**0.212**	**0.314**	0.511	0.272
SB	0.412	0.082	0.137	0.644	0.524
FMSV	**0.637**	0.121	0.203	0.7204	0.539
MA-FMSV	0.588	0.206	0.307	**0.739**	**0.997**

Table 4.3: Comparison between tag recommendation procedures on the CAL-500 data set.

We wanted to evaluate the effect of the K parameter for nearest neighbors on recommender effectiveness. In theory, by using nearest neighbors, a system should be able to recommend a richer set of tags. As opposed to the SB method, the FMSV methods consider music content in their nearest neighbor search, while MA-FMSV enforces attribute diversity. We therefore tested the relationship between number of neighbors and the effectiveness of the three recommendation systems. Figure 4.3 illustrates that FMSV exhibited the best precision over all values for K. All three SVM methods were quite sensitive to the K value, gaining considerable performance as K increased. This is understandable, as the data set's tag space was a relatively clean one. Therefore, increasing the number of nearest neighbors will increase the number of high quality tags aggregated in SEMA, thereby reducing informational signal to noise ratio. The recall, F_1 score (not shown), and MAP measurements were less sensitive to K value for all three methods, yet MA-FMSV performed better across the board (except for MAP when $K > 55$). K did not have a significant effect on Diversity measurements.

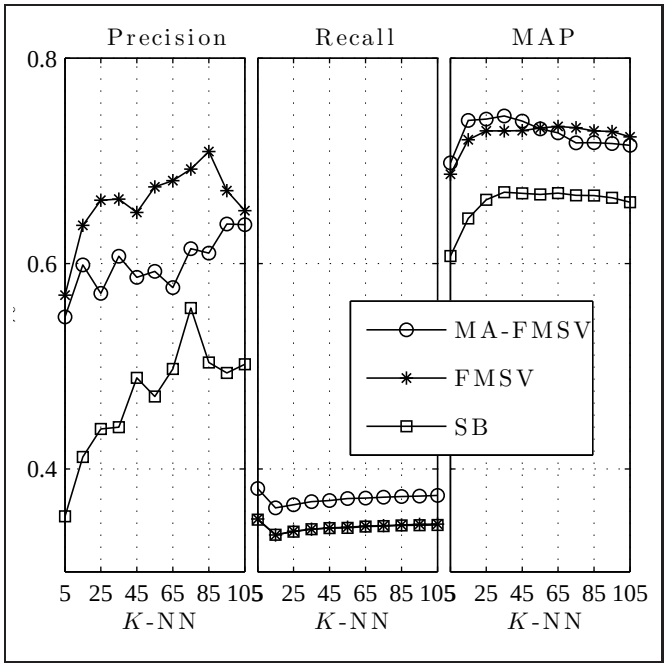


Figure 4.3: K variable versus recommendation effectiveness for the CAL-500 data set ($N = 12$).

Figure 4.4 illustrates the effect of parameter N for tag recommendation MAP and Diversity. All methods suffer in MAP performance as N is increased. The two non attribute-diverse methods, SB and FMSV, show considerable gain in Diversity performance when N is increased. However, they are only able to achieve approximately 65% of the performance that MA-FMSV does. Therefore, MA-FMSV can recommend a highly attribute-diverse set of tag while maintaining relatively good MAP performance.

WebCrawl

When presented with a much larger-scale training and testing data set, all SVM methods perform noticeably worse. This underscores the necessity of evaluating tag recommendation systems on data sets that realistically approximate real-world scenarios. Table 4.4 shows that the pure model-driven method no longer obtains the best results in a large-scale data set such as WebCrawl. Therefore, we suggest that MD's optimal performance on a small, clean data set

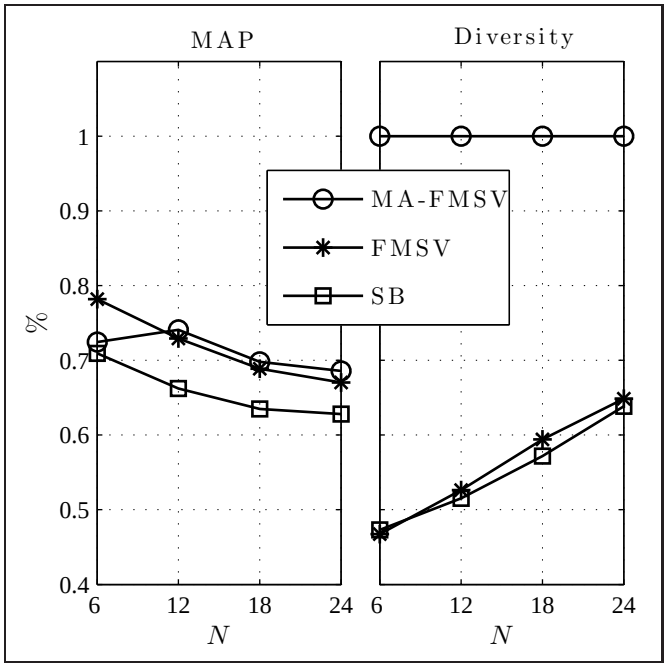


Figure 4.4: N variable versus recommendation effectiveness for the CAL-500 data set ($K = 15$).

does not generalize to larger data sets. Despite overall decreased performance, the MA-FMSV outperforms all other SVM methods (except on recall).

Method	Prec.	Recall	F_1 Score	MAP	Diver.
MD	0.133	0.388	0.198	0.218	0.723
SB	0.164	0.456	0.242	0.336	0.678
FMSV	0.166	**0.458**	0.244	0.335	0.680
MA-FMSV	**0.210**	0.417	**0.279**	**0.362**	**0.958**

Table 4.4: Comparison between tag recommendation procedures on the WebCrawl data set.

Again, we examine the impact of the tunable parameters K and N on the effectiveness of SVM systems, but with a large-scale data set. In Figure 4.5, FMSV and SB obtain nearly exactly the same results and a slight increase in performance over increasing K. MA-FMSV shows better performance across all K, except for the recall measurement when $K < 25$.

With regard to Top$N$ values and the WebCrawl data set, we find trends similar to Figure 4.4 in Figure 4.6. In this case, however, at a high enough N value, all SVM methods perform at near unitary Diversity. Yet, Figure 4.4 shows that cost in MAP performance may be avoided if

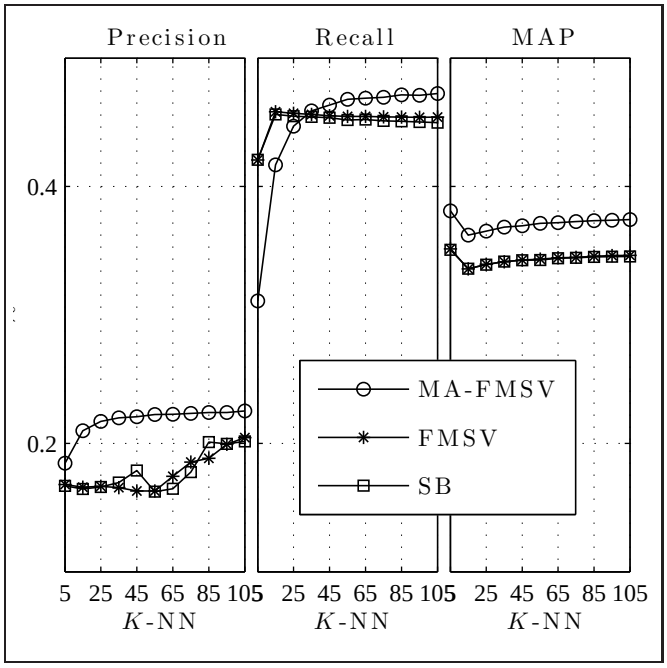


Figure 4.5: K variable versus recommendation effectiveness for the WebCrawl data set ($N = 8$).

MA-FMSV is used for tag recommendation.

4.4.2 Tag Recommendation Efficiency

PMCD

In our system, the Pegasos based PMCD algorithm was modified with a "Random Emitter" method to reduce MapReduce payload when given a large number of input samples. In order to check that our decomposed and modified version of Pegasos performs correctly, we tested it on a generic multi-class problem set of 1,000,000 samples and 20 classes. In all cases, PMCD performed similarly to or better than LibSVM. We are therefore confident that our modifications do not come with loss in classifier accuracy.

To show the efficiency of the revised "Random Emitter" method over standard methods, we plot the number of samples output from the mapper as a function of sample size input. The left

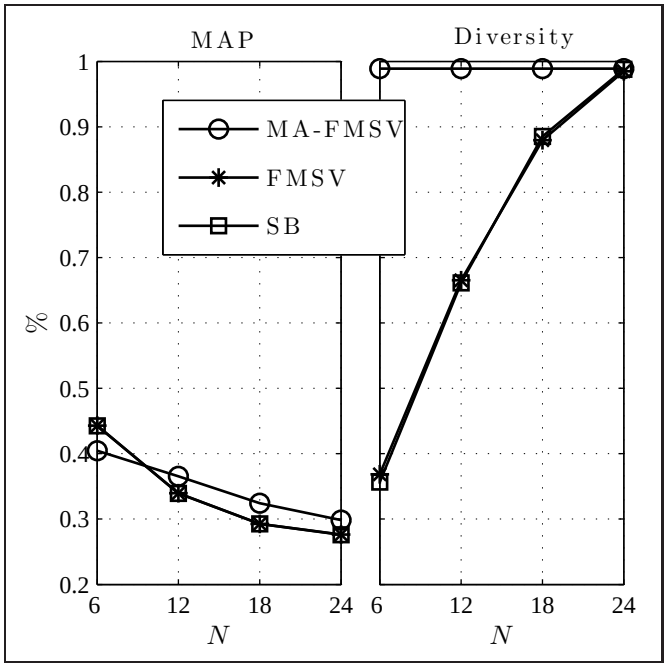


Figure 4.6: N variable versus recommendation effectiveness for the CAL-500 data set ($K = 15$).

graph in Figure 4.7 shows that the "Normal Emitter" and "Random Emitter" have exactly the same number of emitted samples when the size of data set is small. However, as the size of data set increases, the "Random Emitter" pre-samples the data and limits the output. In this case, if we set $C = 20$ and $T = 100,000$, then the total required sample size is $\frac{C \times (C-1)}{2} * T$. As can be seen, when the size of the dataset is larger than the number of samples required by Pegasos, the "Random Emitter" limits the system's output, while the output of a "Normal Emitter" increases linearly.

POCO-AIM

In our work, we have proposed the POCO-AIM algorithm for calculating the occurrence and co-occurrence between social tags and elements in SEMA. In doing so, we first modified the "Stripes" method proposed by Lin *et al.* [39] by adding functionality for counting term occurrence. We have designated the modified algorithm as POCO-Revised Stripes (POCO-RS).

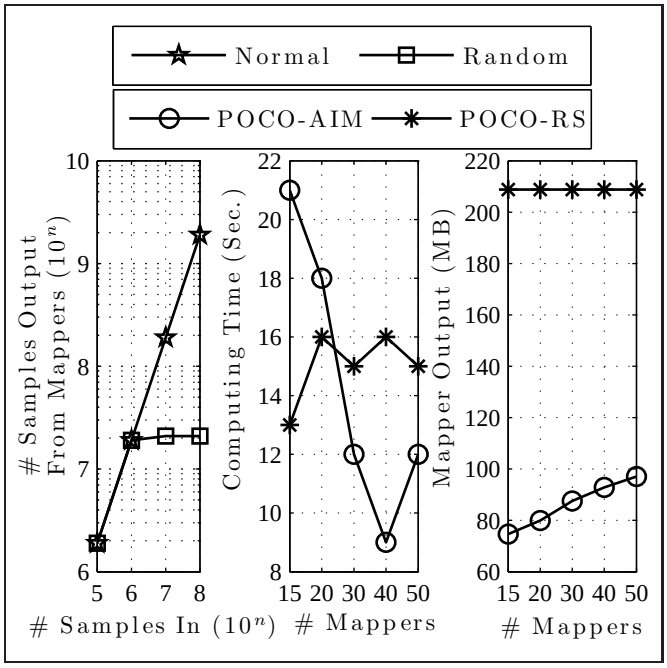


Figure 4.7: System efficiency measurements. The left plot shows the number of mappers required, as a function of the number of input samples, for the "Normal" and "Random" methods of concept detection with MapReduce. The middle graph shows differences in computing time, as more mappers are used with two different implementations of a parallel occurrence co-occurrence algorithm. The right graph shows reduced mapper output per mapper for the POCO-AIM algorithm.

Then, we introduced additional modifications for improving the computational efficiency of POCO-RS as POCO-AIM.

In order to model real-world computational requirements, we crawled much of the Last.fm data set, which has 8,338,431 unsorted tags over 440,407 songs to test the computational efficiency of our parallel processing algorithm, POCO-AIM. In the middle graph of Figure 4.7, we show that the running time of POCO-AIM decreases as the number of mappers increases by a significant amount until the system's memory resource are depleted (when the number of mappers exceeds 40). As can be seen, POCO-AIM requires approximately 33% of the computational time that POCO-RS does when 40 mappers are in use. Therefore, POCO-AIM outperforms the modified "Stripes" as long as the vocabulary of all tags in use is small enough to be stored directly in memory. The corpus of tags used to describe music is relatively small compared to that of text, image and video, so POCO-AIM is an appropriate method for tag

recommendation. POCO-AIM accomplishes computational efficiency by aggregating results in the mapper, therefore reducing the number of intermediate results emitted from all mappers. The right side of Figure 4.7 shows that the size of the intermediate results emitted from all the mappers in POCO-AIM is much less (approximately 50% when the number of mappers = 40) compared to the modified “Stripes” algorithm.

Chapter 5

Query-by-Description Music Information Retrieval(QBD-MIR) Prototype

5.1 QBD-MIR Framework

5.1.1 QBD-MIR Demo System

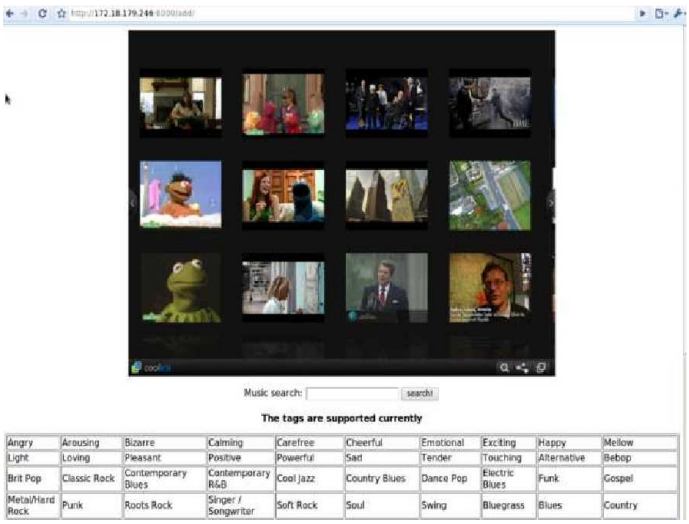


Figure 5.1: The homepage of QBD-MIR system

Figure 5.1 is the home page of our toy QBD-MIR system, the bottom table in this figure indicates that which kind of tags (description) are supported currently. The tags here are certain descriptions on music content not the Meta data, it means that all the commercial systems are difficult to explore music in this way. By typing a tag in the search form, the system will return a set of relevant songs regarding to the tag. One thing valuable to be noticed is that the query process could be very fast due to it just needs to rank the relevant scores and fetches the top 10 songs. Figure 5.2 demonstrates whether the retrieved top 10 songs are truly related to such query or not. The first column is a list of music video clips fetched from Youtube, and the second column is the *Songs names* and *tags from ground truth data set*, which annotated by three persons separately. In this figure, the correct tags have been highlighted.

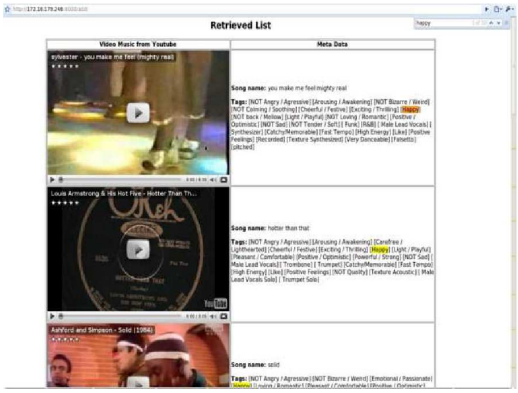


Figure 5.2: The top 10 retrieval video list

Chapter 6

Conclusion

In conclusion, we have proposed three methods to address social tagging issues: sparsity and noise.

We have investigated the use of various probabilistic models for text-based QBD retrieval of music. In particular, we have focused on applying our modification of the Corr-LDA model(Method 1), previously used in image retrieval, to a new domain. Also, we presented an alternative method for fusing multiple information sources. This data level fusion involves clustering to obtain an codeword representation of raw audio features and combining them with social tags mined from the WWW. Our experiment results indicate that Corr-LDA is competitive in the music retrieval domain when compared against other existing probabilistic models. Furthermore, our method of data level fusion results in the best performance. Last, we have implemented a prototype retrieval system that retrieves music based on text-based query. Moreover, a novel approach called TOB-SS(Method 2) is also proposed to improve the performance of previous models. The experimental results have demonstrated that our approach outperforms other methods on the benchmark data set. Another contribution in this project is that we set up a real system to help people explore the music in a new way, where users can find music by semantic meaningful description.

Futhermore, we also have presented a framework for large-scale music tag recommendation with Explicit Multiple Attributes(Method 3). The system guarantees that recommended tags will be attribute-diverse. Additionally, we have detailed parallel music content analysis, concept detection and parallel social tags mining algorithms based on the MapReduce framework to support large-scale offline processing and fast online tag recommendation in each pre-defined attribute.

Our experiments have shown that our system's tag recommendation is more effective than many existing recommenders and at least as effective as other SVM-based methods. In all cases, recommended tags are more attribute-diverse and the recommender's ranking system has been shown to be more effective. Additionally, we have proven that our tag recommender is scalable to very large data sets and real world scenarios. Due the generality of our proposed framework and three parallel algorithms, we believe that it may be used in other multimedia content analysis and tag recommendation tasks, as well.

Our future tasks include evaluating the performance of our framework using mismatched and larger sized CEMA and SEMA attribute spaces. We also aim to compare our POCO method with purely co-occurrence based schemes. During testing, we found that speedup was not as optimal as desired when we approached the limits of our computational resources. We therefore plan to investigate how speedup may be further optimized. Finally, we are working to design a human-friendly interface for our recommendation system so that we may distribute it to the public domain.

Bibliography

[1] M. Mandel and D. P. Ellis. Labrosa's audio music similarity and classification submissions. In *Proc. ISMIR 2007 - Mirex (2007)*, 2007.

[2] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. Vlahavas. Multilabel classification of music into emotions. In *Proc. 9th International Conference on Music Information Retrieval (ISMIR 2008), Philadelphia, PA, USA, 2008*, 2008.

[3] Thierry Bertin-Mahieux, Douglas Eck, François Maillet, and Paul Lamere. Autotagger: A model for predicting social tags from acoustic features on large music databases. *Journal of New Music Research*, 37(2):115–135, 2008.

[4] Peter Knees, Tim Pohle, Markus Schedl, and Gerhard Widmer. A music search engine built upon audio-based and web-based similarity measures. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 447–454, New York, NY, USA, 2007. ACM.

[5] M. Slaney, K. Weinberger, and W. White. Learning a metric for music similarity. In *ISMIR*, pages 313–318, 2008.

[6] Paul Lamere. Social tagging and music information retrieval. *Journal of New Music Research*, 37(2):101–114, 2008.

[7] Bingjun Zhang, Jialie Shen, Qiaoliang Xiang, and Ye Wang. Compositemap: a novel framework for music similarity measure. In *SIGIR '09: Proceedings of the 32nd inter-*

national ACM SIGIR conference on Research and development in information retrieval, pages 403–410, New York, NY, USA, 2009. ACM.

[8] Douglas R. Turnbull, Luke Barrington, Gert Lanckriet, and Mehrdad Yazdani. Combining audio content and social context for semantic music discovery. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 387–394, New York, NY, USA, 2009. ACM.

[9] David M. Blei and Michael I. Jordan. Modeling annotated data. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 127–134, New York, NY, USA, 2003. ACM.

[10] M. Levy and M. Sandler. Music information retrieval using social tags and audio. *Multimedia, IEEE Transactions on*, 11(3):383–395, 2009.

[11] Mark Levy and Mark Sandler. Learning latent semantic models for music from social tags. *Journal of New Music Research*, 37(2):137–150, 2008.

[12] Ling Chen, Phillip Wright, and Wolfgang Nejdl. Improving music genre classification using collaborative tagging data. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 84–93, New York, NY, USA, 2009. ACM.

[13] G. Carneiro, A. B. Chan, P. J. Moreno, and N. Vasconcelos. Supervised learning of semantic classes for image annotation and retrieval. *IEEE Trans Pattern Anal Mach Intell*, 29(3):394–410, March 2007.

[14] Luke Barrington, Douglas Turnbull, David Torres, and Gert Lanckriet. Semantic similarity for music retrieval. In *Proceedings of the International Symposium on Music Information Retrieval*, 2007.

[15] Bingjun Zhang, Qiaoliang Xiang, Huanhuan Lu, Jialie Shen, and Ye Wang. Comprehensive query-dependent fusion using regression-on-folksonomies: a case study of mul-

timodal music search. In *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, pages 213–222, New York, NY, USA, 2009. ACM.

[16] Jia Li and James Z. Wang. Real-time computerized annotation of pictures. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*, pages 911–920, New York, NY, USA, 2006. ACM.

[17] Rui Shi, Chin-Hui Lee, and Tat-Seng Chua. Enhancing image annotation by integrating concept ontology and text-based bayesian learning model. In *MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia*, pages 341–344, New York, NY, USA, 2007. ACM.

[18] G. Sychay, E. Chang, and K. Goh. Effective image annotation via active learning. In *2002 IEEE International Conference on Multimedia and Expo, 2002. ICME'02. Proceedings*, volume 1, 2002.

[19] Florent Monay and Daniel Gatica-Perez. On image auto-annotation with latent space models. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 275–278, New York, NY, USA, 2003. ACM.

[20] D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet. Semantic annotation and retrieval of music and sound effects. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(2):467–476, 2008.

[21] Matthew Hoffman, David Blei, and Perry Cook. Easy as cba: A simple probabilistic model for tagging music. In *Proc. International Symposium on Music Information Retrieval*, 2009.

[22] Steven R. Ness, Anthony Theocharis, George Tzanetakis, and Luis Gustavo Martins. Improving automatic music tag annotation using stacked generalization of probabilistic svm outputs. In *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, pages 705–708, New York, NY, USA, 2009. ACM.

[23] Xin J. Wang, Lei Zhang, Xirong Li, and Wei Y. Ma. Annotating images by mining image search results. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1919–1932, 2008.

[24] Changhu Wang, Lei Zhang, and Hong-Jiang Zhang. Learning to reduce the semantic gap in web image retrieval and annotation. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 355–362, New York, NY, USA, 2008. ACM.

[25] Börkur Sigurbjörnsson and Roelof van Zwol. Flickr tag recommendation based on collective knowledge. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 327–336, New York, NY, USA, 2008. ACM.

[26] Lei Wu, Linjun Yang, Nenghai Yu, and Xian S. Hua. Learning to tag. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 361–370, New York, NY, USA, 2009. ACM.

[27] Dong Liu, Xian-Sheng Hua, Linjun Yang, Meng Wang, and Hong-Jiang Zhang. Tag ranking. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 351–360, New York, NY, USA, 2009. ACM.

[28] Hong-Ming Chen, Ming-Hsiu Chang, Ping-Chieh Chang, Ming-Chun Tien, Winston H. Hsu, and Ja-Ling Wu. Sheepdog: group and tag recommendation for flickr photos by automatic search-based learning. In *MM '08: Proceeding of the 16th ACM international conference on Multimedia*, pages 737–740, New York, NY, USA, 2008. ACM.

[29] Douglas Turnbull, Luke Barrington, David Torres, and Gert Lanckriet. Towards musical query-by-semantic-description using the cal500 data set. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 439–446, New York, NY, USA, 2007. ACM.

[30] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[31] Chong Wang, David Blei, and Li F. Fei. Simultaneous image classification and annotation. In *Proceedings of CVPR*, 2009.

[32] Jimmy Lin. Brute force and indexed approaches to pairwise document similarity comparisons with mapreduce. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 155–162, New York, NY, USA, 2009. ACM.

[33] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *Usenix SDI*, pages 137–150, 2004.

[34] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

[35] George Tzanetakis and Perry Cook. Marsyas: a framework for audio analysis. *Org. Sound*, 4(3):169–175, 1999.

[36] Gary Bradski, Cheng-Tao Chu, Andrew Ng, Kunle Olukotun, Sang Kyun Kim, Yi-An Lin, and YuanYuan Yu. Map-reduce for machine learning on multicore. In *NIPS*, 12/2006 2006.

[37] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated subgradient solver for svm. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 807–814, New York, NY, USA, 2007. ACM.

[38] Rudi L. Cilibrasi and Paul M. B. Vitanyi. The google similarity distance. *IEEE Trans. on Knowl. and Data Eng.*, 19(3):370–383, 2007.

[39] Jimmy Lin. Scalable language processing algorithms for the masses: a case study in computing word co-occurrence matrices with MapReduce. In *EMNLP '08: Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 419–428, Morristown, NJ, USA, 2008. Association for Computational Linguistics.

[40] Richard McCreadie, Craig Mcdonald, and Iadh Ounis. Comparing distributed indexing: To mapreduce or not? In *Proceedings of the 7th Workshop on Large-Scale Distributed Systems for Information Retrieval (LSDS-IR'09) at SIGIR 2009*, July 2009.

[41] Christiane Fellbaum, editor. *WordNet: an electronic lexical database*. MIT Press, 1998.

Appendix

.1 Corr-LDA Variational Inference

This section presents the details of the components of $\mathcal{L}(\gamma, \phi, \lambda)$ (Equation 3.4), used in Variational Inference (Method 1 - Corr-LDA). Where obvious, the parameters of functions are omitted, e.g. $\Theta = \{\alpha, \pi, \beta\}$ from $\mathcal{L}(\gamma, \phi, \lambda)$ and γ, ϕ, λ from $q(\theta, \mathbf{z}, \mathbf{y})$.

.1.1 Lower Bound of log likelihood

$$\begin{aligned}
\mathcal{L}(\gamma, \phi, \lambda) &= \mathbb{E}_q[\log p(\theta, \mathbf{r}, \mathbf{w}, \mathbf{z}, \mathbf{y})] - \mathbb{E}_q[\log q(\theta, \mathbf{z}, \mathbf{y})] && (1)\\
&= \mathbb{E}_q[\log p(\theta|\alpha)] + \mathbb{E}_q[\log p(\mathbf{z}|\theta)] + \mathbb{E}_q[\log p(\mathbf{r}|\mathbf{z}, \pi)] + \\
&\quad \mathbb{E}_q[\log p(\mathbf{y}|N)] + \mathbb{E}_q[\log p(\mathbf{w}|\mathbf{y}, \mathbf{z}, \beta)] - \\
&\quad \mathbb{E}_q[\log q(\theta)] - \mathbb{E}_q[\log q(\mathbf{z})] - \mathbb{E}_q[\log q(\mathbf{y})] && (2)
\end{aligned}$$

$$\mathbb{E}_q[\log p(\theta|\alpha)] = \log \Gamma(\sum_{j=1}^{K} \alpha_j) - \sum_{i=1}^{K} \log \Gamma(\alpha_i) + \sum_{i=1}^{K} (\alpha_i - 1) \left(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{K} \gamma_j) \right) \quad (3)$$

$$\mathbb{E}_q[\log p(\mathbf{z}|\theta)] = \sum_{n=1}^{N}\sum_{i=1}^{K}\phi_{ni}\left(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{K}\gamma_j)\right) \tag{4}$$

$$\mathbb{E}_q[\log p(\mathbf{r}|\mathbf{z},\pi)] = \sum_{n=1}^{N}\sum_{i=1}^{K}\phi_{ni}\log\pi_{ir_n} \tag{5}$$

$$\mathbb{E}_q[\log p(\mathbf{y}|N)] = \sum_{n=1}^{N}\sum_{m=1}^{M}\lambda_{mn}\log\frac{1}{N} = \log\frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M}\lambda_{mn} \tag{6}$$

$$\mathbb{E}_q[\log p(\mathbf{w}|\mathbf{y},\mathbf{z},\beta)] = \sum_{n=1}^{N}\sum_{i=1}^{K}\phi_{ni}\sum_{m=1}^{M}\lambda_{mn}\log\beta_{iw_m} \tag{7}$$

$$\mathbb{E}_q[\log q(\theta)] = \log\Gamma(\sum_{j=1}^{K}\gamma_j) - \sum_{i=1}^{K}\log\Gamma(\gamma_i) + \sum_{i=1}^{K}(\gamma_i - 1)\left(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{K}\gamma_j)\right) \tag{8}$$

$$\mathbb{E}_q[\log q(\mathbf{z})] = \sum_{n=1}^{N}\sum_{i=1}^{K}\phi_{ni}\log\phi_{ni} \tag{9}$$

$$\mathbb{E}_q[\log q(\mathbf{y})] = \sum_{n=1}^{N}\sum_{m=1}^{M}\lambda_{mn}\log\lambda_{mn} \tag{10}$$

.1.2 Computation Formulation

For computation when α_i is same for all i:

$$\begin{aligned}
\mathcal{L}(\gamma,\phi,\lambda) &= \log\Gamma(\sum_{j=1}^{K}\alpha_j) - \sum_{i=1}^{K}\log\Gamma(\alpha_i) - \log\Gamma(\sum_{j=1}^{K}\gamma_j) \qquad \text{(non-K dependent terms)} \quad (11)\\
&+ \sum_{i=1}^{K}\left(\log\gamma_i + \left(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{K}\gamma_j)\right)(\alpha_i - \gamma_i)\right) \quad (12)\\
&+ \sum_{n=1}^{N}\sum_{i=1}^{K}\phi_{ni}\left(\left(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{K}\gamma_j)\right) + \log\pi_{ir_n} - \log\phi_{ni} + \sum_{m=1}^{M}\lambda_{mn}\log\beta_{iw_m}\right) \quad (13)\\
&- \sum_{n=1}^{N}\sum_{m=1}^{M}\lambda_{mn}\log(N\lambda_{mn}) \quad (14)
\end{aligned}$$

.1.3 Variational Multinomial Updates

Parameter ϕ_{ni}

$$\begin{aligned}
\mathcal{L}_{[\phi_n]} &= \sum_{i=1}^{K}\phi_{ni}\left(\left(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{K}\gamma_j)\right) + \log\pi_{i,r_n} + \sum_{m=1}^{M}\lambda_{mn}\log\beta_{i,w_m} - \log\phi_{ni}\right)\\
&\quad +\lambda_n(\sum_{j=1}^{K}\phi_{ni} - 1)\\
\frac{\partial\mathcal{L}}{\partial\phi_{ni}} &= \left(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{K}\gamma_j)\right) + \log\pi_{i,r_n} + \sum_{m=1}^{M}\lambda_{mn}\log\beta_{i,w_m} - \log\phi_{ni} - 1 + \lambda\\
&= 0\\
\phi_{ni} &\propto \pi_{i,r_n}\exp\left(\left(\Psi(\gamma_i) - \Psi(\sum_{j=1}^{K}\gamma_j)\right) + \sum_{m=1}^{M}\lambda_{mn}\log\beta_{i,w_m}\right) \quad (15)
\end{aligned}$$

Term $-\Psi(\sum_{j=1}^{K}\gamma_j)$ can be ignored as it cancels out after normalisation.

Parameter γ_i

$$\gamma_i = \alpha_i + \sum_{n=1}^{N} \phi_{ni} \tag{16}$$

New γ^{t+1} can be updated using old γ^t and ϕ^t using:

$$\gamma_i^0 \leftarrow \alpha_i \tag{17}$$

$$\gamma_i^{t+1} \leftarrow \gamma_i^t + \sum_{n=1}^{N} (\phi_{ni}^{t+1} - \phi_{ni}^t) \tag{18}$$

Parameter λ_{mn}

$$\begin{aligned} \mathcal{L}_{[\lambda_{mn}]} &= \sum_{i=1}^{K} \phi_{ni} \lambda_{mn} \log \beta_{i,w_m} - \lambda_{mn} \log \lambda_{mn} + \log \frac{1}{N} \lambda_{mn} \\ \frac{\partial \mathcal{L}}{\partial \lambda_{mn}} &= \sum_{i=1}^{K} \phi_{ni} \log \beta_{i,w_m} - (\log \lambda_{mn} + 1) + \log \frac{1}{N} \\ &= 0 \\ \lambda_{mn} &\propto \exp(\sum_{i=1}^{K} \phi_{ni} \log \beta_{i,w_m}) \end{aligned} \tag{19}$$

.2 Corr-LDA Parameter estimation

In this section we derive the gradient ascent updates in the maximisation step of the Variational Expectation Maximisation algorithm. A corpus D is represented by a bag of codewords and annotations (words), i.e.

$$D = \{(r_d, w_d)\}_{d=1}^{D}$$

.2.1 Parameter π_{if}

$$
\begin{aligned}
\mathcal{L} &= \sum_{d=1}^{D} \log P(r_d, w_d|\pi, \beta) \\
\mathcal{L}_{[\pi_{1:k}]}(D) &= \sum_{d=1}^{D}\sum_{n=1}^{N_d}\sum_{i=1}^{K} \phi_{dni} \log \pi_{i,r_n} + \sum_{i=1}^{K} \mu_i (\sum_{f=1}^{V_r} \pi_{if} - 1) \\
\frac{\partial \mathcal{L}_{[\pi_{1:k}]}}{\partial \pi_{if}} &= \sum_{d=1}^{D}\sum_{n=1}^{N_d}\sum_{i=1}^{K} \frac{\phi_{dni}}{\pi_{i,r_n}} + \sum_{i=1}^{K} \mu_i \sum_{f=1}^{V_r} 1 \\
&= \sum_{d=1}^{D}\sum_{n=1}^{N_d}\sum_{i=1}^{K} \frac{\phi_{dni}}{\pi_{i,r_n}} + \sum_{i=1}^{K} \mu_i \frac{(V_r+1)V_r}{2} \\
&= 0 \\
\pi_{if} &\propto \sum_{d=1}^{D}\sum_{n=1}^{N_d} 1[r_n = f]\phi_{dni}
\end{aligned} \tag{20}
$$

.2.2 Parameter β_{iw}

$$
\begin{aligned}
\mathcal{L}_{[\beta_{1:K}]}(\mathcal{D}) &= \sum_{m=1}^{M}\sum_{n=1}^{N}\sum_{i=1}^{K} \lambda_{mn}\phi_{in} \log \beta_{i,w_m} + \sum_{i=1}^{K} \nu_i (\sum_{w=1}^{V_w} \beta_{iw} - 1) \\
\frac{\partial \mathcal{L}_{[\beta_{1:K}]}}{\partial \beta_{iw}} &= \sum_{m=1}^{M}\sum_{n=1}^{N}\sum_{i=1}^{K} \lambda_{mn}\phi_{in} \log \beta_{i,w_m} + \sum_{i=1}^{K} \nu_i \frac{(V_w+1)V_w}{2} = 0 \\
\beta_{iw} &\propto \sum_{d=1}^{D}\sum_{m=1}^{M} 1[w_m = w] \sum_{n} \phi_{dni}\lambda_{dmn}
\end{aligned} \tag{21}
$$

.3 QBD Music Retrieval Prototype

Here are the example query and sample screenshots of the prototype.

SML	Corr-LDA (social)
Song: Crosby Nash BBC – Guinnevere **Original Annotations:** NOT Angry/Aggressive, NOT Arousing/Awakening, NOT Bizarre/Weird, Calming/Soothing, NOT Cheerful/Festive, NOT Exciting/Thrilling, NOT Happy, back/Mellow, NOT Light/Playful, NOT Loving/Romantic, Pleasant/Comfortable, NOT Powerful/Strong, Tender/Soft, Bluegrass, Folk, Acoustic Guitar, Backing vocals, Male Lead Vocals, NOT Catchy/Memorable, NOT Changing Energy Level, NOT Fast Tempo, NOT Heavy Beat, NOT High Energy, Quality, NOT Recommend, Recorded, Texture Acoustic, NOT Very Danceable, Folk	**Song:** Evanescence – My Immortal **Original Annotations:** NOT Angry/Aggressive, NOT Bizarre/Weird, NOT Carefree/Lighthearted, NOT Cheerful/Festive, Emotional/Passionate, NOT Happy, NOT Light/Playful, Loving/Romantic, Pleasant/Comfortable, NOT Positive/Optimistic, **Sad**, Tender/Soft, Touching/Loving, Soft Rock, Female Lead Vocals, Piano, NOT Changing Energy Level, NOT Fast Tempo, NOT Heavy Beat, NOT High Energy, NOT Positive Feelings, Quality, Recorded, Texture Acoustic, NOT Very Danceable, Emotional
Song: Miles Davis – Blue in Green **Original Annotations:** NOT Angry/Aggressive, NOT Bizarre/Weird, Calming/Soothing, NOT Carefree/Lighthearted, back/Mellow, NOT Light/Playful, **Sad**, Tender/Soft, Touching/Loving, Cool Jazz, Jazz, Piano, Catchy/Memorable, NOT Fast Tempo, NOT Heavy Beat, NOT High Energy, Like, Quality, Texture Acoustic, Going to sleep, Romancing, Jazz	**Song:** Fiona Apple – Love Ridden **Original Annotations:** NOT Angry/Aggressive, NOT Arousing/Awakening, NOT Bizarre/Weird, Calming/Soothing, NOT Carefree/Lighthearted, NOT Cheerful/Festive, Emotional/Passionate, NOT Exciting/Thrilling, NOT Happy, NOT Light/Playful, Loving/Romantic, Pleasant/Comfortable, Powerful/Strong, **Sad**, Tender/Soft, Touching/Loving, Alternative Folk, Singer/Songwriter, Soul, Folk, Female Lead Vocals, Piano, String Ensemble, Catchy/Memorable, NOT Heavy Beat, Like, NOT Positive Feelings, Quality, Recorded, Texture Acoustic, Romancing, Emotional, Female Lead Vocals Solo
Song: Sheryl Crow – I Shall Believe **Original Annotations:** NOT Angry/Aggressive, NOT Arousing/Awakening, NOT Bizarre/Weird, Calming/Soothing, NOT Carefree/Lighthearted, NOT Cheerful/Festive, Emotional/Passionate, NOT Exciting/Thrilling, NOT Light/Playful, Pleasant/Comfortable, Powerful/Strong, Tender/Soft, Country, Backing vocals, Bass, Female Lead Vocals, Synthesizer, Tambourine, Catchy/Memorable, NOT Changing Energy Level, NOT Fast Tempo, NOT Heavy Beat, NOT High Energy, Positive Feelings, Quality, Recorded, Texture Acoustic, Tonality, Breathy, Emotional, Vocal Harmonies	**Song:** The Carpenters – Rainy Days and Mondays **Original Annotations:** NOT Angry/Aggressive, NOT Arousing/Awakening, NOT Bizarre/Weird, Calming/-Soothing, NOT Cheerful/Festive, Emotional/Passionate, NOT Exciting/Thrilling, NOT Happy, NOT Light/Playful, NOT Positive/Optimistic, **Sad**, Tender/Soft, Touching/Loving, Blues, Folk, Backing vocals, Female Lead Vocals, Harmonica, Piano, Saxophone, String Ensemble, NOT Fast Tempo, NOT Heavy Beat, NOT High Energy, Quality, Recorded, Texture Acoustic, Texture Electric, Intensely Listening, Emotional, Saxophone Solo

Table 1: Top 3 results for query “sad” for SML and Corr-LDA(social) models