Large-scale Music Tag Recommendation with Explicit Multiple Attributes

Zhendong Zhao, Xinxi Wang, Qiaoliang Xiang,
Andy M Sarroff, Zhonghua Li, Ye Wang
School of Computing
National University of Singapore
{zhaozhendong, andy660, qiaoliangxiang}@gmail.com
andy.sarroff@nyu.edu, {lizhongh, wangye}@comp.nus.edu.sg

ABSTRACT

Social tagging can provide rich semantic information for large-scale retrieval in music discovery. Such collaborative intelligence, however, also generates a high degree of tags unhelpful to discovery, some of which obfuscate critical information. Towards addressing these shortcomings, tag recommendation for more robust music discovery is an emerging topic of significance for researchers. However, current methods do not consider diversity of music attributes, often using simple heuristics such as tag frequency for filtering out irrelevant tags. Music attributes encompass any number of perceived dimensions, for instance vocalness, genre, and instrumentation. Many of these are underrepresented by current tag recommenders. We propose a scheme for tag recommendation using Explicit Multiple Attributes based on tag semantic similarity and music content. In our approach, the attribute space is explicitly constrained at the outset to a set that minimizes semantic loss and tag noise, while ensuring attribute diversity. Once the user uploads or browses a song, the system recommends a list of relevant tags in each attribute independently. To the best of our knowledge, this is the first method to consider Explicit Multiple Attributes for tag recommendation. Our system is 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 tag recommendation system 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 recommending attribute-diverse relevant tags and efficient at scalable processing.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; H.5.5 [**Sound and Music Computing**]: Systems

General Terms

Algorithms, Experimentation, Performance

Keywords

Music, tag recommendation, explicit multiple attributes, search

1. INTRODUCTION

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 [2], *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 paper, 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 paper is as follows. First, we discuss related work and how ours compares (Section 2). In Section 3 we present the system architecture. We perform several evaluations of our system using two data sets in Section 4 and discuss our results in Section 5. Finally, we give our concluding remarks in Section 6.

2. RELATED WORK

In order to improve the quality of online tagging, there has been extensive work dedicated to automatically annotating images [9,14, 17,19] and songs [2,8,15,21]. 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 [18]. 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 [12,25] combine both co-occurrence and image content analysis. In this paper, we propose a method 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.* [4] 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 [4]'s approach and ours. 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 [4] 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.

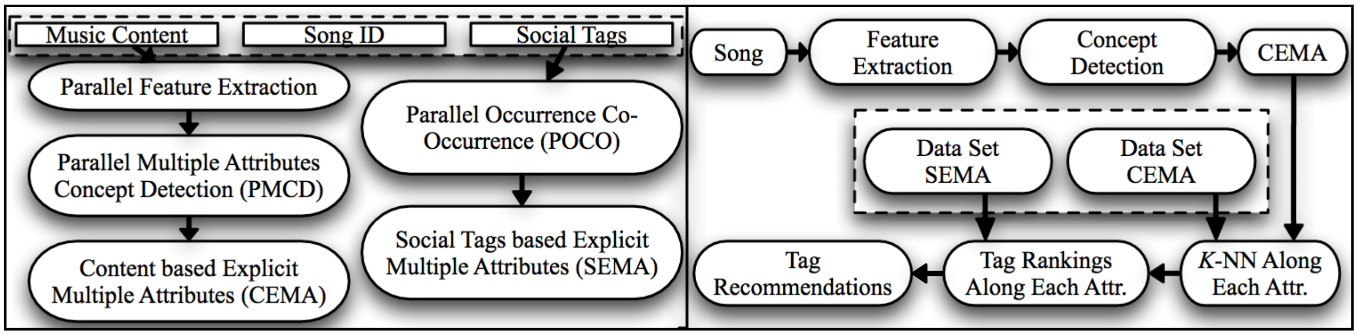
3. SYSTEM ARCHITECTURE

Our system architecture, which is designed for scalability, is graphically depicted in Figure 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. In this paper, we simply adopt same weight for each attribute, it can be easily switched to unbalanced scheme by using different lengths of rank list in our future work. Each of the architectural components are discussed in detail below.

3.1 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 [11]. With traditional parallel programming models such as the Message Passing Interface,



Music Content
Song ID
Social Tags
Parallel Feature Extraction
Parallel Multiple Attributes Concept Detection (PMCD)
Content based Explicit Multiple Attributes (CEMA)
Parallel Occurrence Co-Occurrence (POCO)
Social Tags based Explicit Multiple Attributes (SEMA)
Song
Feature Extraction
Concept Detection
CEMA
Data Set SEMA
Data Set CEMA
Tag Recommendations
Tag Rankings Along Each Attr.
K-NN Along Each Attr.

Figure 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.

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 [6]. 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 [11]. 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 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.

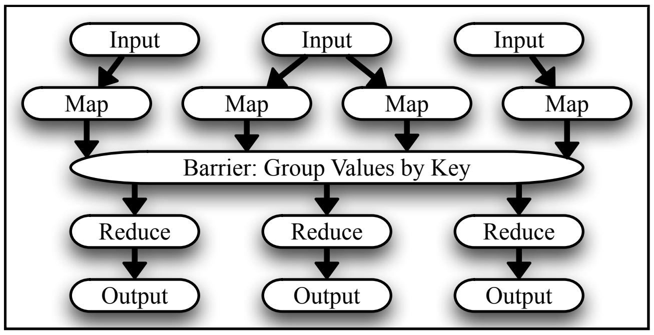


Figure 2: MapReduce Framework. Each input partition sends a $(key, value)$ pair to the mappers. An arbitrary number of intermediate $(key, value)$ pairs are emitted by the mappers, sorted by the barrier, and received by the reducers.

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.

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

3.2 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.

3.3 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.

3.3.1 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 [26] for use on music similarity measures, and are easily computed by a SVM classifier. 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) [1]. As evaluated in [26], 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.

3.3.2 *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 2). Each song is stored in the cluster as a single line and is fed into a mapper. In the mapper, we use Marsyas [22][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.

3.3.3 *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 [3]. 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 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. This scheme will be more efficient with larger size of concepts and it can also be applied to other domains such as image, video and text.

In our work, we use a novel algorithm, which couples the Pegasos SVM solver [16] 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.

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

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:

$$P_+ = P_- = \alpha \times \frac{N_C \times I}{2 \times N} \tag{1}$$

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 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:

$$I_E = I_{E+} + I_{E-} \tag{2}$$

$$I_{E+} = \frac{N}{N_C} \times P_+, \quad I_{E-} = \frac{N}{N_C} \times P_- \tag{3}$$

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_+:

$$I_E = 2 \times \frac{N}{N_C} \times P_+ \tag{4}$$

$$P_+ = \frac{N_C \times I_E}{2 \times N} \tag{5}$$

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 1.

3.4 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.

3.4.1 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 [5] for measuring tag distance:

$$d(t_i, t_j) = \frac{\max(\log f(t_i), \log f(t_j)) - \log f(t_i, t_j)}{\log N - \min(\log f(t_i), \log f(t_j))} \tag{6}$$

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.

3.4.2 Design of a Scalable POCO Algorithm: POCO-AIM

Efficient parallel word co-occurrence algorithms have been presented by [10], 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 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 [13]. 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.

3.5 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. 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.

4.1 Data Sets

We gathered several data sets, summarized in Table 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 1: Data sets used for training and testing.

4.1.1 CAL-500

CAL-500 [20] 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 [2, 8, 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.

4.1.2 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 [7] [3] and filtered out any songs without tags. We were left with 77,448 songs.

4.1.3 HandCrawl

The HandCrawl data set is another high quality manually tagged data set that has recently been used and evaluated in [27]. 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 2.

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

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

4.2 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.

4.2.1 Precision and Accuracy

To evaluate our system's recommendation effectiveness, we follow the examples set in [21] 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 [21], 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.

4.2.2 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:

$$\text{Diversity@}n \equiv \frac{\sum_i^n A(t_i)}{N_A} \tag{7}$$

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.

4.2.3 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 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.

4.3.1 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 [8]. 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 [21].
2. Autotag: An AdaBoost based system using additional training data and features, along with those included with CAL-500 [2].
3. CBA: **C**odeword **B**ernoulli **A**verage is a probabilistic model based on using a codebook of size K [8]. 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 [26].
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.* [15] 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 paper, 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.

4.3.2 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.

4.4 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.

5. RESULTS

5.1 Tag Recommendation Effectiveness

5.1.1 CAL-500

Table 3 compares the results of evaluating multiple procedures on the CAL-500 data set. As reported in [21], 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 [15]. 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 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 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.

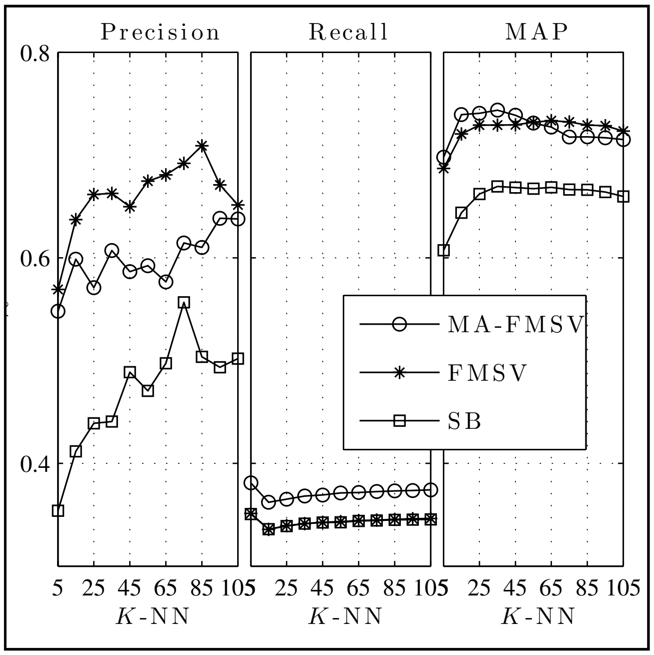


Figure 3: K **variable versus recommendation effectiveness for the CAL-500 data set** ($N = 12$)**.**

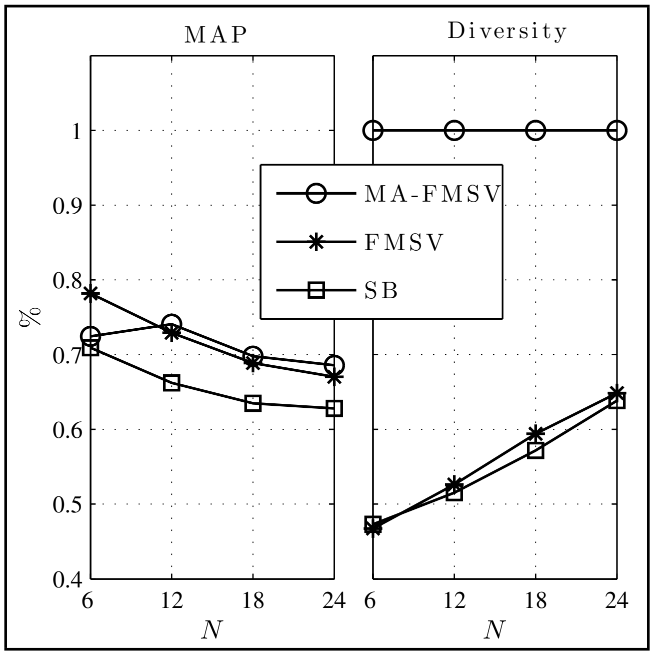


Figure 4: N **variable versus recommendation effectiveness for the CAL-500 data set** ($K = 15$)**.**

Figure 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.

5.1.2 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 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 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: 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 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 in Figure 6. In this case, however, at a high enough N value, all SVM methods perform at near unitary Diversity. Yet, Figure 4 shows that cost in MAP performance may be avoided if MA-FMSV is used for tag recommendation.

5.2 Tag Recommendation Efficiency

5.2.1 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 graph in Figure 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.

5.2.2 POCO-AIM

In our work, we have proposed the POCO-AIM algorithm for

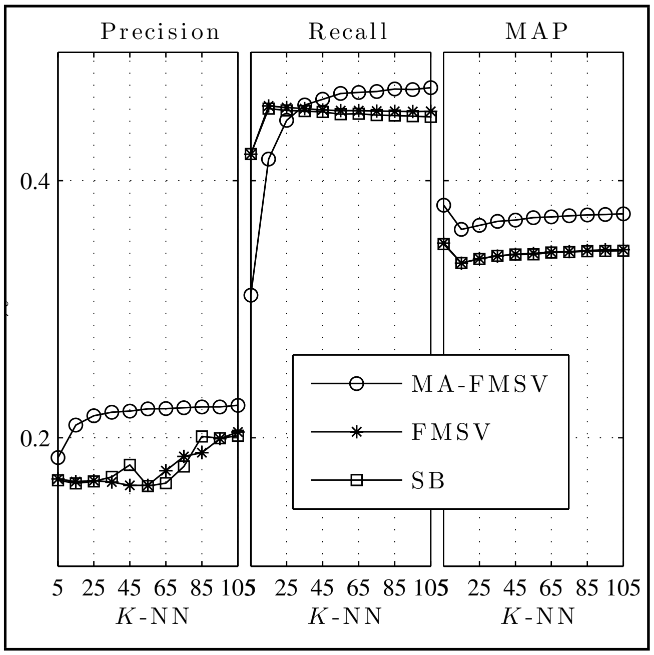


Figure 5: K **variable versus recommendation effectiveness for the WebCrawl data set (**$N = 8$**).**

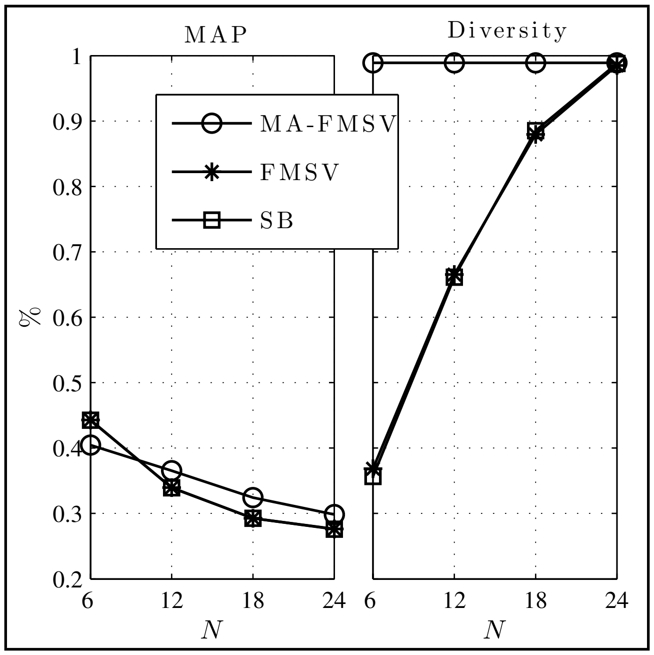


Figure 6: N **variable versus recommendation effectiveness for the WebCrawl data set (**$K = 15$**).**

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.* [10] by adding functionality for counting term occurrence. We have designated the modified algorithm as POCO-Revised Stripes (POCO-RS). 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 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 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.

6. CONCLUSIONS

We have presented a framework fo large-scale music tag recommendation with Explicit Multiple Attributes. 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 and comprehensive evaluation on efficiency will be conducted. Finally, we are working to design a human-friendly interface for our recommendation system so that we may distribute it to the public domain.

7. ACKNOWLEDGEMENT

The work was supported by Singaporean MOE grant R-252-000-381-112. We appreciate Bingjun Zhang for sharing the data sets.

8. REFERENCES

[1] 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.

[2] T. Bertin-Mahieux, D. Eck, F. Maillet, and P. 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.

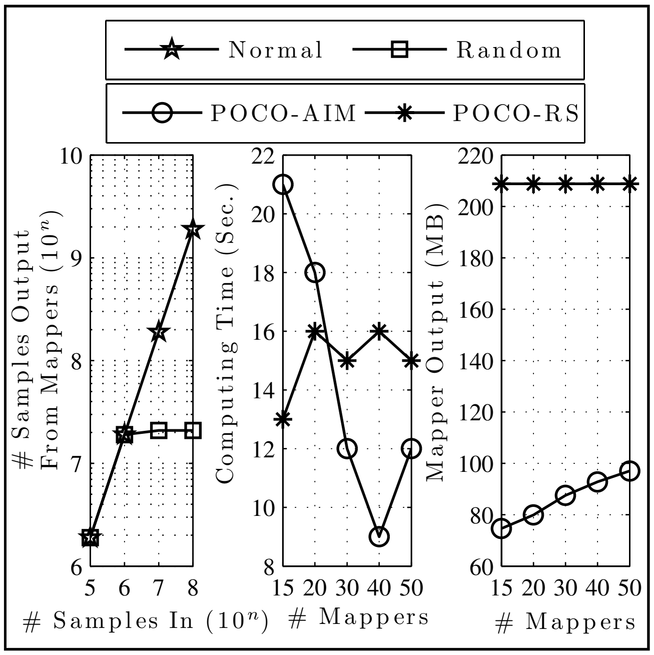


Figure 7: System efficiency measurements. The left plot shows the number of mappers required, as a function of the number of input samples, for the "Normal" and "Random" methods of concept detection with MapReduce. The middle graph shows differences in computing time, as more mappers are used with two different implementations of a parallel occurrence co-occurrence algorithm. The right graph shows reduced mapper output per mapper for the POCO-AIM algorithm.

[3] G. Bradski, C.-T. Chu, A. Ng, K. Olukotun, S. K. Kim, Y.-A. Lin, and Y. Yu. Map-reduce for machine learning on multicore. In *NIPS*, 12/2006 2006.

[4] H.-M. Chen, M.-H. Chang, P.-C. Chang, M.-C. Tien, W. H. Hsu, and J.-L. 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.

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

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

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

[8] M. Hoffman, D. Blei, and P. Cook. Easy as cba: A simple probabilistic model for tagging music. In *Proc. International Symposium on Music Information Retrieval*, 2009.

[9] J. Li and J. 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.

[10] J. 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.

[11] J. 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.

[12] D. Liu, X.-S. Hua, L. Yang, M. Wang, and H.-J. 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.

[13] R. McCreadie, C. Mcdonald, and I. 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.

[14] F. Monay and D. 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.

[15] S. R. Ness, A. Theocharis, G. Tzanetakis, and L. G. 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.

[16] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 807–814, New York, NY, USA, 2007. ACM.

[17] R. Shi, C.-H. Lee, and T.-S. 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] B. Sigurbjörnsson and R. 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.

[19] 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.

[20] D. Turnbull, L. Barrington, D. Torres, and G. 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.

[21] 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.

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

[23] C. Wang, L. Zhang, and H.-J. 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.

[24] X. J. Wang, L. Zhang, X. Li, and W. Y. Ma. Annotating images by mining image search results. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1919–1932, 2008.

[25] L. Wu, L. Yang, N. Yu, and X. 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.

[26] B. Zhang, J. Shen, Q. Xiang, and Y. Wang. Compositemap: a novel framework for music similarity measure. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 403–410, New York, NY, USA, 2009. ACM.

[27] B. Zhang, Q. Xiang, H. Lu, J. Shen, and Y. Wang. Comprehensive query-dependent fusion using regression-on-folksonomies: a case study of multimodal music search. In *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, pages 213–222, New York, NY, USA, 2009. ACM.