Codebook based Audio Feature Representation for Music Information Retrieval

Yonatan Vaizman, Brian McFee, *member, IEEE*, and Gert Lanckriet, *senior member, IEEE*

***Abstract*—Digital music has become prolific in the web in recent decades. Automated recommendation systems are essential for users to discover music they love and for artists to reach appropriate audience. When manual annotations and user preference data is lacking (*e.g.* for new artists) these systems must rely on *content based* methods. Besides powerful machine learning tools for classification and retrieval, a key component for successful recommendation is the *audio content representation*.**

Good representations should capture informative musical patterns in the audio signal of songs. These representations should be concise, to enable efficient (low storage, easy indexing, fast search) management of huge music repositories, and should also be easy and fast to compute, to enable real-time interaction with a user supplying new songs to the system.

Before designing new audio features, we explore the usage of traditional local features, while adding a stage of encoding with a pre-computed *codebook* and a stage of pooling to get compact vectorial representations. We experiment with different encoding methods, namely the LASSO, vector quantization (VQ) and cosine similarity (CS). We evaluate the representations' quality in two music information retrieval applications: query-by-tag and query-by-example. Our results show that concise representations can be used for successful performance in both applications. We recommend using top-τ VQ encoding, which consistently performs well in both applications, and requires much less computation time than the LASSO.

***Index Terms*—Music recommendation, audio content representations, vector quantization, sparse coding, music information retrieval.**

I. Introduction

In the recent decades digital music has become more accessible and abundant on the web and large scale systems for recommendation and exploration have become more popular. Since the availability of manual annotations and user preference data is limited (*e.g.* for new, unfamiliar, artists) industrial recommendation systems must incorporate *content based* methods, which interpret the actual audio content of music items and extract meaningful information from it. In the past decade much research was dedicated to constructing content based systems for music information retrieval (MIR) tasks such as music classification (to artist, genre, *etc.* [1]–[13]), semantic annotation (auto-tagging) and retrieval (query-by-tag [14]–[22]) and music similarity for song-to-song recommendation ([23]–[29]). The focus was mostly on machine learning algorithms that utilize basic audio features to perform the task.

Y. Vaizman and G. Lanckriet are with the Department of Electrical and Computer Engineering, University of California, San Diego.

B. McFee is with the Center for Jazz Studies and LabROSA, Columbia University, New York.

In this work we focus on the audio feature extraction and representation. We look for efficient methods to represent whole songs (not single frames, not short clips) in a compact way that facilitates efficient storage and communication for large music repositories, and convenient processing for fast search and recommendation. We examine whether a single representation can be rich enough to be useful for multiple MIR applications. Before we develop new low-level audio features, we try to make the most of traditional features, based on mel scaled spectra of short time frames. We use a stage of *encoding* these frame feature vectors with a pre-computed codebook, and a stage of *pooling* the coded frames (temporal integration) to get a summarized fixed-dimension representation of a whole song. The encoding detects informative local patterns and represents the frames at a higher level. The pooling stage makes the representation of a whole song compact and easy to work with (low storage, fast computation and communication), and it creates a representation that has the same dimension for all songs, regardless of their durations.

A. Related work

Many MIR research works used mel frequency cepstral coefficients (MFCC) as audio features ([1]–[4], [8], [12], [14], [16], [17], [19], [23]–[25], [27]–[30]). Other types of popular low-level audio features, based on short time Fourier transform are the constant-Q transform (CQT), describing a short time spectrum with logarithmically scaled frequency bins ([10]–[12], [16], [17]), and chroma features, which summarize energy from all octaves to a single 12-dimensional (per frame) representation of the chromatic scale ([4], [18], [31]). While MFCC is considered as capturing timbral qualities of the sound, the CQT and chroma features are designed for harmonic properties of the music (or melodic, if using patches of multiple frames). Hamel *et al.* suggested using principal component analysis (PCA) whitening of mel scaled spectral features as alternative to MFCC [32]. Some works combine heterogeneous acoustic analysis features, such as zero crossing rate, spectral flatness, estimated tempo, amplitude modulation features *etc.* ([1], [8], [26], [33], [34]).

Low-level audio features are typically extracted from short time frames of the musical clip and then some temporal integration is done. Sometimes an early integration is performed, by taking statistics (mean, variance, covariance, *etc.*) of the feature vector over longer segments, or over the entire song (*e.g.* [7], [16]). Sometimes late integration is performed, for instance: each short segment is classified and for the entire musical clip a majority vote is taken over the multiple segments' declared labels (*e.g.* [9]). Such late integration

systems require more computing time, since the classification operation should be done to every frame, instead of to a single vector representation per song.

Another approach for temporal integration is getting a compact representation of a song by generative modeling. In this approach the whole song is described using a parametric structure that models how the song's feature vector time series was generated. Various generative models were used: GMM ([1], [8], [14], [15], [18], [19], [25], [27], [28], [34], [35]) , DTM ([20]), MAR ([2], [8]), ARM ([36]), HMM ([3], [8], [37]), HDP ([27]). Although these models have been shown very useful and some of them are also time-efficient to work with, the representation of a song using a statistical model is less convenient than a vectorial representation. The former requires retrieval systems that fit specifically to the generative model while the later can be processed by many generic machine learning tools. Computing similarity between two songs is not straight forward when using a generative model (although there are some ways to handle it, like the probability product kernel ([18], [36], [38])), whereas for vectorial representation there are many efficient generic ways to compute similarity between two vectors of the same dimension. In [36] the song level generative model (multivariate autoregressive mixture) was actually used to create a kind of vectorial representation for a song by describing the sampled frequency response of the generative model's dynamic system. However, because the model was a mixture model, the resulted representation was a *bag* of four vectors, and not a single vectorial representation.

Encoding of low-level features using a pre-calculated codebook was examined for audio and music. Quantization tree ([23]), vector quantization (VQ) ([3], [29], [39]), sparse coding with the LASSO ([5]) and other variations ([10], [11]) were used to represent the features at a higher level. Sparse representations were also applied directly to time domain audio signals, with either predetermined kernel functions (Gammatone) or with a trained codebook ([6], [40]). As alternative to the heavy computational cost of solving optimization criteria (like the LASSO) greedy algorithms like matching pursuit have also been applied ([6], [39], [40]).

Heterogeneous and multi-layer systems have been proposed. The bag of systems approach combined various generative models as codewords ([22]). Multi-modal signals (audio and image) were combined in a single framework ([41]). Even the codebook training scheme, which was usually unsupervised, was combined with supervision to get a boosted representation for a specific application ([12], [41]). Deep belief networks were used in [9], also combining unsupervised network weights training with supervised fine tuning. In [13] audio features were processed in two layers of encoding with codebooks.

Several works invested in comparing different encoding schemes for audio, music and image. Nam *et al.* examined different variations of low-level audio processing, and compared different encoding methods (VQ, the LASSO and sparse restricted Boltzmann machine) for music annotation and retrieval with the CAL500 dataset [21]. Yeh *et al.* reported finding superiority of sparsity-enforced dictionary learning and $L1$-regularized encoding over regular VQ for genre classification. In [42] Coates and Ng examined the usage of different combinations of dictionary training algorithms and encoding algorithms to better explain the successful performance of sparse coding in previous works. They concluded that the dictionary training stage has less of an impact on the final performance than the encoding stage and that the main merit of sparse coding may be due to its nonlinearity, which can be achieved also with simpler encoders that apply some nonlinear soft thresholding. In [43] Coates *et al.* examined various parameters of early feature extraction for images (such as the density of the extracted patches) and showed that when properly extracting features, one can use simple and efficient algorithms (k-means clustering and single layer neural network) and achieve image classification performance as high as other, more complex systems.

B. Our contribution

In this work we look for *compact* audio content representations for full-length songs that will be powerful for two different MIR applications: query-by-tag and query-by-example. We perform a *large scale* evaluation, using the CAL10k and Last.FM datasets. We assess the effect of various design choices in the "low-level-feature, encoding, pooling" scheme, and eventually recommend a representation "recipe" (based on vector quantization) that is efficient to compute, and has consistent high performance in both MIR applications.

The remainder of the paper is organized as follows: in Section II we describe the audio representations that we compare, including the low-level audio features, the encoding methods and pooling. In Section III we describe the MIR tasks that we evaluate: query-by-tag and query-by-example retrieval. In Section IV we specify the datasets use, the data processing stages and the experiments performed. In Section V we describe our results, followed by conclusions in Section VI.

II. Song representation

We examine the encoding-pooling scheme to get a compact representation for each song (or musical piece). The scheme is comprised of three stages:

1) **Short time frame features:** each song is processed to a time series of low-level feature vectors, $X \in \mathbb{R}^{d \times T}$ (T time frames, with a d dimensional feature vector from each frame).
2) **Encoding:** each feature vector $x_t \in \mathbb{R}^d$ is then encoded to a code vector $c_t \in \mathbb{R}^k$, using a pre-calculated dictionary $D \in \mathbb{R}^{d \times k}$, a codebook of k "basis vectors" of dimension d. We get the encoded song $C \in \mathbb{R}^{k \times T}$.
3) **Pooling:** the coded frame vectors are pooled together to a single compact vector $v \in \mathbb{R}^k$.

This approach is also known as the bag of features (BoF) approach: where features are collected from different patches of an object (small two-dimensional patches of an image, short time frames of a song, *etc.*) to form a variable-size set of detected features. The pooling stage enables us to have a unified dimension to the representations of all songs, regardless of the songs' durations. A common way to pool the low-level frame vectors together is to take some statistic

of them, typically their mean. For a monotonic, short song, such a statistic may be a good representative of the properties of the song.

However, a typical song is prone to changes in the spectral content, and a simple statistic pooling function over the low-level feature frames might not represent it well. For that reason the second stage (encoding) was introduced. In a coded vector, each entry encodes the presence/absence/prominence of a specific pattern in that frame. The pre-trained codebook holds codewords (patterns) that are supposed to roughly represent the variety of prominent patterns in songs (the columns of the codebook: $D_j \in \mathbb{R}^d,\ j \in \{1, 2, \ldots, k\}$). The use of sparsity in the encoding (having only *few* basis vectors active/effective in each frame), promotes selecting codewords that represent typical whole sound patterns (comprised of possibly many frequency bands). The pooling of these coded vectors is meaningful: using mean pooling ($v = \frac{1}{T}\sum_{t=1}^{T} c_t$) results in a histogram representation, stating the frequency of occurrence of each sound pattern, while using max-abs (maximum absolute value — $v(j) = \max_{t=1}^{T} |c_t(j)|$) pooling results in an indication representation — for each sound pattern, did it appear anytime in the song, and in what strength. For some encoding methods it is appropriate to take absolute value and treat negative values far from zero as strong values. Since the songs have typically many frames, the resulted pooled representation doesn't have to be sparse, even if the frames' codes were sparse. In our experiments we use three encoding systems, the LASSO ([44]), vector quantization (VQ), and cosine similarity (CS) (all explained later), and apply both mean and max-abs pooling functions to the coded vectors.

A. Low-level audio features

In this work we use spectral features that are commonly assumed to capture timbral qualities. Since we are not interested in melodic or harmonic information, but rather general sound similarity, or semantic representation, we assume timbral features to be appropriate here (an assumption that is worth examination). Our low-level features are based on mel frequency spectra (MFS), which are calculated by computing the short time Fourier transform (STFT), summarizing the spread of energy along mel scaled frequency bins, and compressing the values with logarithm. Mel frequency cepstral coefficients (MFCCs [30]) are the result of further processing MFS, using discrete cosine transform (DCT), in order to both create uncorrelated features from the correlated frequency bins, and reduce the feature dimension. In addition to the traditional DCT we alternatively process the MFS with another method for decorrelating, based on principal component analysis (PCA). Processing details are specified in Section IV-B.

B. Encoding with the LASSO

The least absolute shrinkage and selection operator (the LASSO) was suggested as an optimization criterion for linear regression that selects only few of the regression coefficients to have effective magnitude, while the rest of the coefficients are either shrunk or even nullified [44]. The LASSO does that by balancing between the regression error (squared error) and an $L1$ norm penalty over the regression coefficients, which typically generates sparse coefficients. Usage of the LASSO's regression coefficients as a representation of the input is often referred to as "sparse coding". In our formulation, the encoding of a feature vector x_t using the LASSO criterion is:

$$c_t = \underset{c \in \mathbb{R}^k}{\operatorname{argmin}} \frac{1}{2} \parallel x_t - Dc \parallel_2^2 + \lambda \parallel c \parallel_1 .$$

Intuitively it seems that such a sparse linear combination might represent separation of the music signal to meaningful components (*e.g.* separate instruments). However, this is not necessarily the case since the LASSO allows coefficients to be negative and the subtraction of codewords from the linear combination has little physical interpretability when describing how musical sounds are generated. To solve the LASSO optimization problem we use the alternating direction method of multipliers (ADMM) algorithm. The general algorithm and a specific version for the LASSO are detailed in [45]. The λ parameter can be interpreted as a sparsity-promoting parameter: the larger it is, the more weight will be dedicated to the $L1$ penalty, and the resulted code will typically have fewer entries with effective magnitude.

C. Encoding with vector quantization (VQ)

In vector quantization (VQ) a continuous multi-dimensional vector space is partitioned to a discrete finite set of bins, each having its own representative vector. The training of a VQ codebook is essentially a clustering that describes the distribution of vectors in the space. During encoding, each frame's feature vector x_t is quantized to the closest codeword in the codebook, meaning it is encoded as c_t, a sparse binary vector with just a single "on" value, in the index of the codeword that has smallest distance to it (we use Euclidean distance). It is also possible to use a softer version of VQ, selecting for each feature vector x_t the τ nearest neighbors among the k codewords, creating a code vector c_t with τ "on" values and $k - \tau$ "off" values:

$$c_t(j) = \frac{1}{\tau} \mathbb{1} \left[D_j \in \tau\text{-nearest neighbors of } x_t \right], \quad j \in \{1, 2, \ldots, k\}.$$

Such a soft version can be more stable: whenever a feature vector has multiple codewords in similar vicinity (quantization ambiguity), the hard threshold of selecting just one codeword will result in distorted, noise-sensitive code, while using top-τ quantization will be more robust. This version also adds flexibility and richness to the representation: instead of having k possible codes for every frame, we get $\binom{k}{\tau}$ possible codes. Of course, if τ is too large, we may end up with codes that are trivial — all the songs will have similar representations and all the distinguishing information will be lost. The encoding parameter τ is a density parameter, with larger values causing denser codes. By adjusting τ we can directly control the level of sparsity of the code, unlike in the LASSO, where the effect

of adjusting the parameter λ is indirect, and depends on the data. The values in the VQ code vectors are binary (either 0 or $\frac{1}{\tau}$). Using max-abs pooling on these code vectors will result in binary final representations. Using mean pooling results in a codeword histogram representation with richer values. We only use mean pooling for VQ in our experiments.

In [29] it was shown that for codeword histogram representations (VQ encoding and mean pooling), it was beneficial to take the square root of every entry, consequently transforming the song representation vectors from points on a simplex ($\sum_{j=1}^{k} |v_j| = 1$) to points on the positive orthant of a sphere ($\sum_{j=1}^{k} |v_j|^2 = 1$). The authors called it PPK transformation, since a dot product between two transformed vectors is equivalent to the probability product kernel (PPK) with power 0.5 on the original codeword histograms [38]. We also experiment with the PPK-transformed versions of the codeword histogram representations.

D. Encoding with cosine similarity (CS)

VQ encoding is simple and fast to compute (unlike the LASSO, whose solving algorithms, like ADMM, are iterative and slow). However, it involves a hard threshold (even when $\tau > 1$) that possibly distorts the data and misses important information. When VQ is used for communication and reconstruction of signal it is necessary to use this thresholding in order to have a low bit rate (transmitting just the index of the closest codeword).

However, in our case of encoding songs for retrieval we have other requirements. As an alternative to VQ we experiment with another form of encoding, where each dictionary codeword is being used as a linear filter over the feature vectors: instead of calculating the *distance* between each feature vector and each codeword (as done in VQ), we calculate a *similarity* between them — the (normalized) dot product between the feature vector and the codeword: $\frac{\langle x_t, D_j \rangle}{\|x_t\|_2}$. Since the codewords we train are forced to have unit $L2$ norm, this is equivalent to the cosine similarity (CS). The codewords act as pattern matching filters, where frames with close patterns get higher response.

For the CS encoding we use the same codebooks that are used for VQ. For each frame, selecting the closest (by Euclidean distance) codeword is equivalent to selecting the codeword with largest CS with the frame. So CS can serve as a softer version of VQ. The $L2$ normalization of each frame (to get CS instead of unnormalized dot product) is important to avoid having a bias towards frames that have large magnitudes, and can dominate over all other frames in the pooling stage. In our preliminary experiments we verified that this normalization is indeed significantly beneficial to the performance. The CS regards only to the "shape" of the pattern but not to its magnitude and gives a fair "vote" also to frames with low power. Unlike the unnormalized dot product the response values of CS are limited to the range $[-1, 1]$, and are easier to interpret and to further process.

In the last stage of the encoding we introduce non-linearity in the form of the shrinkage function $y(x) = \text{sign}(x) * \max(|x| - \theta, 0)$ (values with magnitude less than θ are nullified and larger magnitude values remain with linear, but shrinked, response). Using $\theta = 0$ maintains the linear responses of the filters, while $\theta > 0$ introduces sparsity, leaving only the stronger responses. Such a nonlinear function is sometimes called "soft thresholding" and was used in various works before to compete with the successful "sparse coding" (the LASSO) in a fast feed-forward way ([42]).

E. Dictionary training

The training of the dictionaries (codebooks) is performed with the online learning algorithm for sparse coding presented by Mairal *et al.* ([46]). For sparse coding methods like the LASSO, the problem of finding the optimal dictionary codewords and code coefficients is a smooth but jointly non-convex problem. The online algorithm is an iterative procedure that alternates between encoding a small batch of new instances using the current dictionary, and updating the dictionary using the newly encoded instances. This algorithm converges to a stationary point and scales well to large training sets. As an initialization stage we apply online k-means to a stream of training d-dimensional feature vectors, to cluster them to an initial codebook of k codewords. This initial dictionary is then given to the online algorithm. In each iteration the updated codewords are normalized to have unit $L2$ norm.

III. MIR TASKS

We examine the usage of the various song representations for two basic MIR applications, with the hope to find stable representations that are consistently successful in both tasks. We use simple, linear machine learning methods, seeing as our goal here is finding useful song representations, rather than finding sophisticated new learning algorithms. In both applications the goal of the system is to retrieve songs from the repository and rank them in order of relevance to the query. In query-by-tag (or "semantic retrieval") the query is a tag word (describing genre, instrument, emotional content *etc.*), ultimately allowing for free-text search. In query-by-example (or "song-song recommendation") the query is a song by itself, enabling an online radio or other interfaces. Efficient content analysis methods could allow for a real-time query-by-example interface, where the user may upload an unfamiliar song to the system, and get similar/relevant songs in return.

A. Query-by-tag (QbT)

We use $L2$-regularized logistic regression as a tag model. For each semantic tag we use the positively and negatively labeled training instances (k-dimensional song vectors) to train a tag model. Then for each song in the test set and for each tag we use the trained tag model to estimate the probability of the tag being relevant to the song (the posterior probability of "positive" for the song-vector given the tag model). For each song, the vector of tag-probabilities is then normalized to be a categorical probability over the tags, also known as the semantic multinomial (SMN) representation of a song [20].

Retrieval: For each tag the songs in the test set are ranked according to their SMN value relevant to the tag. Per-tag scores

are calculated as done in [15], [20]: area under curve (AUC) is the area under the ROC curve (the curve of tradeoff between false positive rate and true positive rate, where each point is achieved by a different cutoff threshold over the ranking), precision at top-10 (P@10) is the fraction of ground truth positive items out of the top 10 ranked items and average precision (AP) is the precision averaged over all the positions in the ranking where a ground truth positive item is found. These per-tag scores are averages over the tags to get a general score (mean (over tags) AP is abbreviated MAP).

B. Query-by-example (QbE)

Given a query song, whose audio content is represented as vector $q \in \mathbb{R}^k$, our query-by-example system calculates its distance $dist(q, r)$ from each repository song $r \in \mathbb{R}^k$ and the recommendation retrieval result is the repository songs ranked in increasing order of distance from the query song. The Euclidean distance is a possible simple distance measure between songs' representations. However, it grants equal weight to each of the vectors' dimensions, and it is possible that there are dimensions that carry most of the relevant information, while other dimensions carry just noise. For that reason, we use a more general metric as a distance measure, the Mahalanobis distance: $dist(q, r) = \sqrt{(q-r)^T W (q-r)}$, when $W \in \mathbb{R}^{k \times k}$ is the parameter matrix for the metric (W has to be a positive semidefinite matrix for a valid metric).

In [47] McFee *et al.* presented a framework for using a metric for query-by-example recommendation systems, and a learning algorithm — metric learning to rank (MLR) — for training the metric parameter matrix W to optimize various ranking performance measures. In [29] the authors further demonstrated the usage of MLR for music recommendation, and the usage of collaborative filtering data to train the metric, and to test the ranking quality. Here we follow the same scheme: collaborative filtering data are used to define artist-artist similarity (or relevance), and song-song binary relevance labels. MLR is then applied to training data to learn a metric W. The learnt metric is tested on a test set. Further details are provided in Section IV-B. Same as for query-by-tag, we apply the same scheme to different audio content representations and compare the performance of query-by-example.

IV. Experimental setup

A. Data

In this work we use the CAL10k dataset [48]. This dataset contains $10,865$ full-length songs from over $4,500$ different artists, ranging over 18 musical genres. Throughout the paper we use the convenient term "song" to refer to a music item/piece (even though many of the items in CAL10k are pieces of classical music and would commonly not be called songs). It also contains semantic tags harvested from the Pandora website[1], including 475 acoustic tags and 153 genre (and sub-genre) tags. These tag annotations were assigned to the songs by human listeners, musical experts. The songs in CAL10k are weakly labeled in the sense that if a song doesn't have a certain tag, it doesn't necessarily mean that the tag is not relevant for the song, but for evaluation we assume that missing song-tag associations can be treated as negative labels. We filter the tags to include only the 581 tags that have at least 30 songs associated with them.

For the query-by-example task we work with the intersection of artists from CAL10k and the Last.FM[2] collaborative filtering data, collected by Celma ([49] chapter 3). As done in [29] we calculate the artist-artist similarity based on Jaccard index ([50]) and the binary song-song relevance metric, which is used as the target metric to be emulated by MLR.

For the dictionary training we use external data — 3560 audio files of songs/clips by ~ 700 artists that *do not appear* in CAL10k. These clips were harvested from various interfaces on the web and include both popular and classical music. This is unlike the sampling from *within* the experimental set, as was done in [21], which might cause over-fitting. The annotation files for the experimental 5-fold partition and the list of dictionary training songs are available on the author's website: http://acsweb.ucsd.edu/~yvaizman/metadata/cal10k_ground_truth.html.

B. Processing

Audio files are averaged to single channel (in case they are given in stereo) and re-sampled at $22,050$Hz. Feature extraction is done over half-overlapping short frames of $2,048$ samples (a feature vector once every $1,024$ samples, which is once every $\sim 46msec$). The power spectrum (squared magnitude of DFT) of each frame is summarized into 34 Mel-scaled frequency bins, and log value is saved to produce initial MFS features. To get the MFCC features a further step of discrete cosine transform (DCT) is done and 13 coefficients are saved. The 1^{st} and 2^{nd} instantaneous derivatives are augmented to produce MFCCΔ ($d = 39$) and MFSΔ ($d = 102$) feature vectors. The next step is to standardize the features so that each dimension would have zero mean and unit variance (according to estimated statistics). In order to have comparable audio features, we reduce the dimension of the MFSΔ to 39 dimensions using a PCA projection matrix (pre-estimated from the dictionary training data) to get MFSΔPC features.

In both low-level feature versions (MFCCΔ and MFSΔPC) we use linear transformations (DCT and PCA) to compress the energy, with two distinctions: First, in the DCT the projection vectors are predetermined (cosine functions with various periods) whereas in the PCA the projection vectors are learnt from examples of music, and therefore assumed to fit better to music data. Second, for the MFSΔPC we apply the decorrelating projection (PCA) *after* augmenting the instantaneous derivatives, in order to capture the correlations among consecutive frames.

The dictionary training set is used to both estimate statistics over the raw features (mean and standard deviations of MFCCΔ and MFSΔ and PCA matrix for the standardized MFSΔ), and to train the dictionary. From each training audio file a segment of $20\,\mathrm{sec}$ is randomly selected, processed and its feature vectors are added to a pool of vectors (resulting in

[1] http://www.pandora.com/mgp.shtml

[2] http://www.last.fm/

1.5 million vectors), which are scrambled to a random order and fed to the online dictionary training algorithm.

Since the online dictionary learning algorithm involves an encoding stage in each iteration, an encoding parameter should be selected for the dictionary training. For each codebook size k the LASSO codebook is trained with $\lambda = 1$ (this codebook is later used for the LASSO encoding with various values of λ) and the VQ codebook is trained with $\tau = 1$ (this codebook is later used for VQ encoding with various values of τ and for CS encoding with various values of θ).

For training the logistic regression model of a tag, an internal cross validation is done over different combinations of parameters (weight of regularization, weight of negative example, weight of positive example), each of which could take values of $[0.1, 1, 10, 100]$. This cross validation is done using only the training set, and the parameter set selected is the one that optimizes the AUC. After selecting the best parameter set for a tag, the entire training set is used to train the tag model with these parameters.

The query-by-tag evaluation is done with 5-fold cross validation. For each fold no artist appears in both the train set (songs by $\frac{4}{5}$ of the artists) and the test set (songs by the remaining $\frac{1}{5}$ of the artists). The performance scores that were averaged over tags in each fold are then averaged over the five folds. The query-by-example evaluation is done with 10 splits of the data in the same manner as done in [29]. We use the AUC rank measure to define the MLR loss between two rankings (marked as $\Delta(y*, y)$ in [47]). For each split we train W over the train set with multiple values of the slack trade off parameter C $(10^{-2}, 10^{-1}, \ldots, 10^{8})$ and for each value test the trained metric on the validation set. The metric that results in highest AUC measure on the validation set is then chosen and tested on the test set. We report the AUC results on the test set, averaged over the 10 splits.

For QbE PCA decorrelation and dimensionality reduction is performed on the data: in each split the PCA matrix is estimated from the train set and the song representation vectors (of train, validation and test set) are projected to a predetermined lower dimension (so the trained matrices W are in fact not $(k \times k)$ but smaller). In [29] the heuristic was to reduce to the estimated effective dimensionality — meaning to project to the first PCs covering 0.95 of the covariance (as estimated from the train set). However, in our experiments we noticed that reducing to the effective dimensionality caused deterioration of performance when the effective dimensionality decreased, while keeping a fixed reduction-dimension kept stable or improving performance. So keeping 0.95 of the covariance is not the best practice. Instead, for every k we fix the dimension of reduction (across different encoders and encoding parameters).

When testing each of the 10 splits, each song in the query set (either the validation set or the test set) is used as a query to retrieve relevant songs from the train set — the train songs are ranked according to the trained metric and the ranking for the query song is evaluated (AUC score). The average over query songs is then taken.

C. Experiments

Each experiment regards to a different type of audio-content representation. We experiment with different combinations of the following parameters:

- low-level features: MFCCΔ or MFSΔPC,
- codebook size $k \in \{128, 256, 512, 1024\}$,
- encoding method: the LASSO, VQ or CS,
- encoding parameters:
  - the LASSO: $\lambda \in \{0.01, 0.1, 0.5, 1, 2, 10, 100\}$,
  - VQ: $\tau \in \{1, 2, 4, 8, 16, 32\}$,
  - *CS*: $\theta \in \{0, 0.1, 0.2 \ldots, 0.9\}$,
- pooling function: either mean or max-abs,
- PPK-transformation: with or without.

V. Results

First, for comparison, we present query-by-tag baseline results: chance level scores are calculated here by using the representations with MFSΔPC, $k = 1024$ and VQ encoding with $\tau = 8$ (one of the best performing settings), scrambling the order of songs and performing the query-by-tag experiment. Then, to control for the necessity of the encoding stage in our scheme, we perform the experiments without the encoding (instead of encoding the feature vectors with a codebook, leaving them as low-level features and pooling them) for both the MFCCΔ and MFSΔPC low-level features. Finally, as an alternative to the codebook based systems, we evaluate the HEM-GMM system ([15], [20]), which is the suitable candidate from the generative models framework, being computationally efficient and assuming independent time frames (like our current codebook systems). We process the data as was done in [20] for HEM-GMM, using our current 5-fold partition. Table I presents these baselines.

			P@10	MAP	AUC
chance level			0.02	0.02	0.5
no encoding	audio feature	pooling			
	MFCCΔ	mean	0.09	0.07	0.76
	MFCCΔ	max-abs	0.09	0.07	0.75
	MFSΔPC	mean	0.10	0.08	0.77
	MFSΔPC	max-abs	0.09	0.07	0.75
	HEM-GMM		0.21	0.16	0.84

TABLE I: Query-by-tag — baseline results

For query-by-tag we show plots (figs. 1 and 2) of the P@10 rank measure (this measure is the more practical objective, since in real recommendation systems, the user typically only looks at the top of the ranked results). Graphical results for the other performance measures are provided in the supplementary material. Figure 2 and fig. 3 show the query-by-tag and query-by-example (respectively) performance of each encoder separately as a function of codebook size k (different subplots) and of the encoding parameter (x-axis). For query-by-example the PCA dimension chosen for each k is written in parenthesis in the title of each subplot. We also experimented with higher PCA dimensions and got similar results (the performance values were slightly higher, but the comparison among encoders or encoding parameters was the same. See supplementary material). In some plots error bars

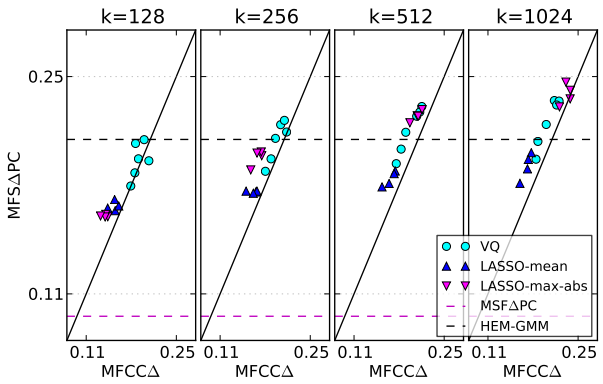


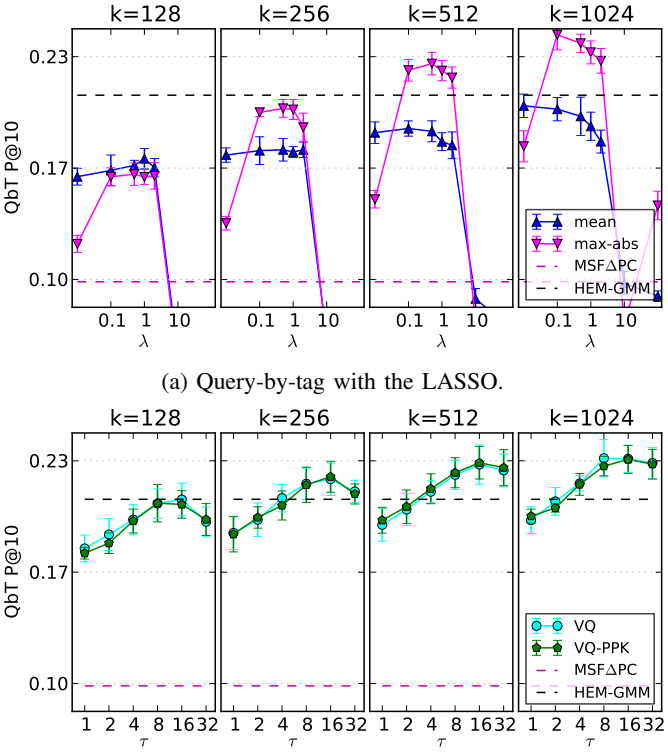
Fig. 1: Comparison of the two low-level audio features. Each point regards to a specific combination of encoder, encoding parameter and pooling, and displays the performance score (QbT P@10) when using MFCC∆ (x-axis) and MFS∆PC (y-axis) as low-level features.

are added: the error bars represent the standard deviation of the score (over the five folds for query-by-tag, and over the 10 splits for query-by-example).

Low-level features: Figure 1 shows the query-by-tag results for comparison between the two low-level features: MFCC∆ and MFS∆PC. Each point in the graphs compares the performance (P@10) using MFCC∆ (x-axis) to the performance using MFS∆PC (y-axis), when all the other parameters (k, encoding method, encoding parameter, pooling method) are the same. Multiple points with the same shape represent experiments with the same encoder and pooling, but different encoding parameter. The main diagonal line ($y = x$) is added to emphasize the fact that in the majority of the experiments performance with MFS∆PC was better than MFCC∆. Statistical tests (paired two-tailed t-test between two arrays of ~ 2900 per-fold-per-tag scores) support the advantage of MFS∆PC: most comparisons show statistically significant advantage of MFS∆PC (all except six points on the plots. P-value well below 0.05), and only one point (for $k = 128$ with VQ and $\tau = 32$) has significant advantage of MFCC∆.

While it is expected that the data-driven decorrelation (PCA) performs better than the predetermined projection (DCT), it is interesting to see that the difference is not so dramatic (the points are close to the main diagonal) — MFCC manages to achieve performance close to the data-trained method. Other than the advantage of training on music data, another explanation to the higher performance of MFS∆PC can be the effect of first taking a local dynamic structure (concatenating the "deltas" to the features) and only then decorrelating the features-∆ version (as we did here for MFS∆PC).

These results also demonstrate the advantage of using *some* encoding over low-level features before pooling them: all these performances (for both MFCC∆ and MFS∆PC) are better than the baseline results with no encoding (Table I. The highest of the "no encoding" baselines is also added as reference line in the plots). We can also notice the improvement with increasing codebook sizes (the different subplots). Similar results are seen for the other performance measures (AUC, MAP) — graphs shown in the supplementary material. The remainder of the results focuses on the MFS∆PC low-level features.



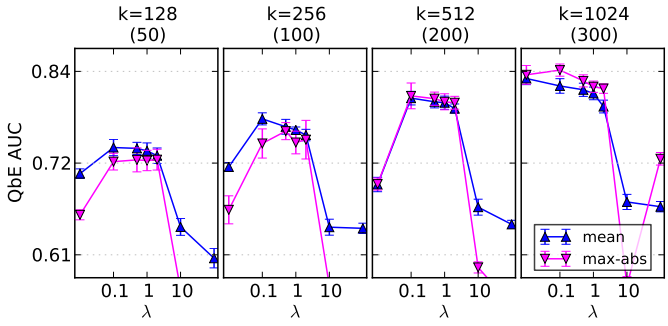
(a) Query-by-tag with the LASSO.

(b) Query-by-tag with VQ.

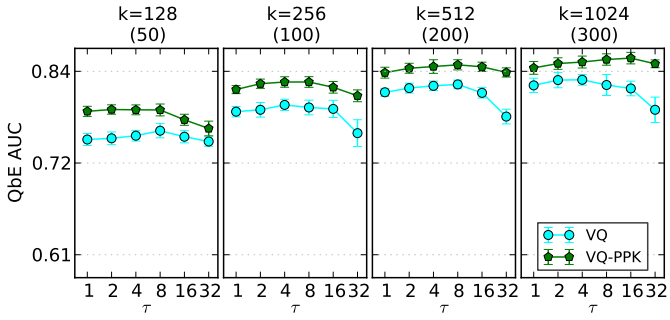
(c) Query-by-tag with CS.

Fig. 2: Query-by-tag with different encoders. Effect of pooling or PPK-transformation (shape) and encoding parameter (x-axis): λ (log-scale) for the LASSO (a), τ (log-scale) for VQ (b) and θ for CS (c). Error bars indicate one standard deviation over the five folds.

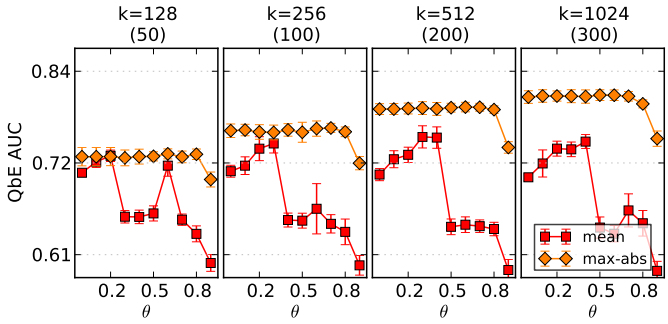
The LASSO encoding: Figures 2a and 3a show the QbT (P@10) and QbE (AUC) performance for the LASSO encoding. The LASSO is sensitive to the value of its parameter λ, and for this particular setting, the sweet spot seems to be around $\lambda \in [0.1, 2]$. When λ is too high ($\lambda = 10, 100$), the approximation error becomes too high and the resulted code loses important information, causing deteriorated performance. When λ is too small (0.01) the balance is also harmed and performance is deteriorated. Similar results are seen for



(a) QbE with the LASSO



(b) QbE with VQ



(c) QbE with CS

Fig. 3: Query-by-example with different encoders. Effect of pooling or PPK-transformation (shape) and encoding parameter (x-axis): λ (log-scale) for the LASSO (a), τ (log-scale) for VQ (b) and θ for CS (c). Error bars indicate one standard deviation over the 10 splits. For each subplot the number beneath the codebook size k is the reduced dimension used for QbE.

AUC and MAP measures (supplementary material). There is inconsistency regarding a preferred pooling function: max-abs sometimes has clear advantage over mean pooling (*e.g.* QbT with $k \geq 256$) and sometimes has disadvantage (*e.g.* QbE with $k \leq 256$), and AUC and P@10 are also inconsistent in that matter. The effect of λ on how "thin" the code vectors are is not direct and can change from song to song (examples are provided in supplementary material).

VQ encoding: Figures 2b and 3b show the QbT and QbE performance for VQ encoding. These results depict a clear effect of the VQ density parameter τ: "softening" the VQ by quantizing each frame to more than one codeword significantly improves the performance (more dramatically for QbT). There is an optimal peak for τ, typically at 8 or 16 — increasing τ further causes performance to deteriorate, especially with a small codebook. Since VQ encoding requires low computational load (compared to the LASSO), we also evaluate QbT experiments with larger sizes of codebook $k \in \{2048, 4096\}$ and an adjusted range of density parameter $\tau \in \{4, 8, 16, 32, 64, 128, 256\}$. QbT performance continues to increase with k. Although these large codebooks provide potential for much richer codes, the peak is still found at $\tau = 16$ and not higher. For $k = 2048, \tau = 16$ P@10 is 0.248 and for $k = 4096, \tau = 16$ P@10 is 0.251, both exceeding the performance achieved with the LASSO and $k \leq 1024$. Full results are presented graphically in the supplementary material.

Cosine similarity encoding: The QbT and QbE results for CS encoding (Figures 2c and 3c) demonstrate the effect of adjusting the sparsity parameter θ (the "knee" of the shrinkage function): the optimal value is not too small and not too large. This is more dramatically seen for QbT with mean pooling: there is a significant advantage in adding *some* non-linearity (having $\theta > 0$), and at the other end having the code too sparse (θ too large) causes a drastic reduction in performance. Same as for the LASSO, there is inconsistency regarding a preferred pooling function (mean better for QbT, max-abs better for QbE). The effect of θ on the sparsity of the frames' code is monotonous but not linear and can change from song to song (examples are provided in supplementary material). QbT experiments with larger codebooks ($k \in \{2048, 4096\}$) show improvement in the AUC and MAP measures (mostly for using the max-abs pooling), but P@10 remains the same as for smaller codebooks (see supplementary material for details).

PPK transformation: For the three encoding methods applying PPK transformation to the vectorial representations has no effect on QbT performance (for the LASSO and CS experiments with $k = 1024$ confirm this). This might be a result of the specific logistic regression system we use here for QbT. On QbE performance PPK has little effect for the LASSO and some improvement for CS with mean pooling (see supplementary material). PPK causes a significant, large improvement of QbE for the VQ representations (clearly seen in fig. 3b). Our QbE results for VQ partly replicate the trends found by McFee *et al.* in [29], with a main distinction: since in [29] the representations were reduced to the estimated effective dimensionality, which was a decreasing function of τ, there was a different effect of τ than what we find here (where we fix the reduced dimension for a given k). In [29], for $k = 512, 1024$ with PPK, increasing τ seemed to hurt the performance, whereas here we show that when PCA is done to a fixed dimension, increasing τ can maintain a stable performance, and even slightly improve the performance (for both with/without PPK), peaking at around $\tau = 8$.

Performance summary: Table II presents the three QbT measures for selected representations, and the generative model alternative (HEM-GMM) as baseline. For each measure, the leading system is marked in bold, and the other systems are compared to it by 2-tailed paired t-test between the two arrays of per-fold-per-tag scores ($N = 2905$). The p-values of the t-tests are written in parenthesis. Figure 4 shows the performance of the same representations in both query-by-tag and query-by-example. The best parameter values from each

representation				QbT		
k	encoding	parameter	pooling	P@10	MAP	AUC
1024	VQ (with PPK)	$\tau = 8$	mean	0.230 (9e − 08)	0.185 (1e − 09)	0.867 (9e − 06)
1024	VQ (no PPK)	$\tau = 8$	mean	0.235 (1e − 04)	0.188 (4e − 06)	0.868 (2e − 04)
1024	the LASSO	$\lambda = 0.1$	max-abs	**0.246**	**0.195**	**0.874**
1024	cosine similarity	$\theta = 0.4$	mean	0.212 (8e − 27)	0.175 (1e − 34)	0.863 (2e − 19)
1024	cosine similarity	$\theta = 0.8$	max-abs	0.190 (9e − 62)	0.156 (2e − 106)	0.852 (2e − 82)
512	VQ (with PPK)	$\tau = 8$	mean	0.226 (6e − 11)	0.181 (2e − 17)	0.867 (3e − 07)
512	the LASSO	$\lambda = 0.1$	max-abs	0.225 (2e − 13)	0.176 (6e − 39)	0.862 (3e − 46)
256	VQ (with PPK)	$\tau = 8$	mean	0.218 (4e − 19)	0.176 (5e − 31)	0.863 (2e − 14)
256	the LASSO	$\lambda = 0.1$	max-abs	0.199 (1e − 53)	0.153 (2e − 129)	0.840 (7e − 230)
128	VQ (with PPK)	$\tau = 8$	mean	0.207 (8e − 36)	0.165 (3e − 65)	0.857 (2e − 32)
128	the LASSO	$\lambda = 0.1$	max-abs	0.160 (6e − 142)	0.122 (9e − 261)	0.811 (0e + 00)
HEM-GMM				0.210 (3e − 30)	0.160 (1e − 78)	0.838 (3e − 107)

TABLE II: QbT results for selected experiments. The bottom line has results from the HEM-GMM system. Numbers in brackets are p-values of t-test comparing to the leading representation in the measure, whose score is marked in bold.

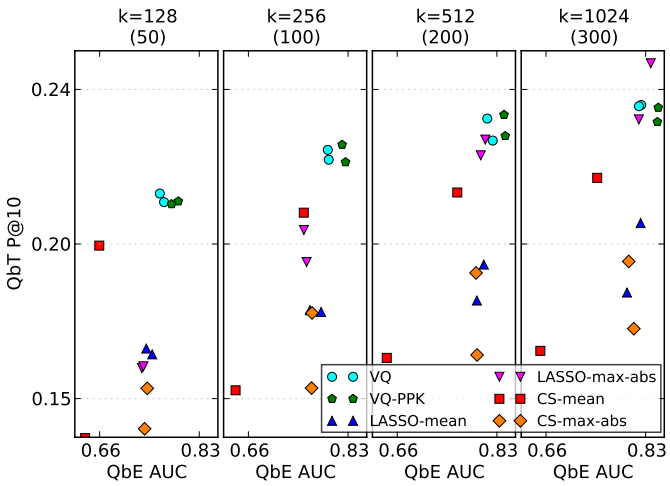


Fig. 4: Comparing both MIR tasks: Each point represents a different audio-representation (encoder, parameter, pooling, PPK) and describes its performance in query-by-tag (y-axis) and query-by-example (x-axis). From each encoder-pooling combination the two best performing parameter values are displayed (with same shape). For each subplot the number beneath the codebook size k is the reduced dimension used for QbE.

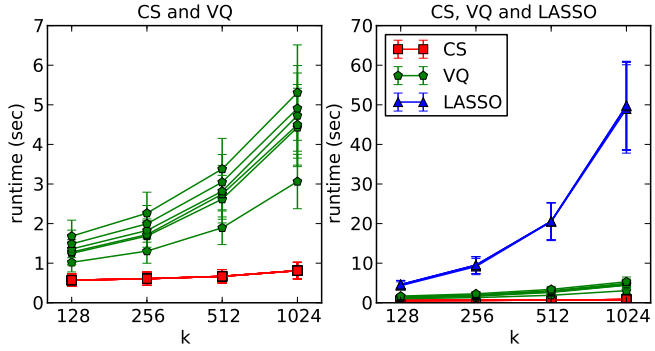


Fig. 5: Empirical runtime test. Average runtime for encoding a song as a function of k (log-scale), and standard deviation in error-bars. The left plot is a "zoom in" on CS and VQ only. Notice the right plot (containing also the LASSO) has a wider range for y-axis. Multiple points of the same shape represent encodings with different encoding parameter value.

encoder are presented.

The best QbT performance (for $k \leq 1024$) is registered for the LASSO with $k = 1024$, where VQ is slightly behind. However, this shouldn't be interpreted as an ultimate advantage of the LASSO, since it is not consistent with other codebook sizes and with QbE. Both CS and the LASSO are sensitive to the selection of their encoding parameter: selecting an inappropriate value results in poor performance of the representation. In practical systems such methods require cross validation to select the appropriate parameter value. VQ, on the other hand, is less sensitive to its density parameter τ. This is perhaps due to the fact that τ directly controls the level of sparsity in the VQ code, whereas for CS and the LASSO the level of sparsity is regularized indirectly. VQ is a stable representation method that can be easily controlled and adjusted with little risk of harming its informative power. VQ consistently achieves high QbT performance and highest QbE performance (this is also consistent when reducing to a higher PCA dimension. See supplementary material).

Examples of ranked songs results for selected tags are presented in supplementary material. These examples demonstrate how a good system (with top-16 VQ) manages to produce non trivial song rankings, and place relevant songs at the top, even when they are lacking ground truth annotation.

Encoding runtime

Since we are searching for practical representations for large scale systems, we should also consider computation resources when selecting a preferred representation method. We compare the runtime complexity of the three encoding methods, from feature vector $x_t \in \mathbb{R}^d$ to code vector $c_t \in \mathbb{R}^k$:

- CS involves multiplying x_t by matrix D ($O(dk)$), computing $\|x_t\|_2$ ($O(d)$) and applying shrinkage to the cosine similarities ($O(k)$), resulting in total complexity of $\boldsymbol{T_{\text{CS}} = O(dk)}$.
- VQ involves the same matrix-vector multiplication and norm calculation to compute the Euclidean distances. Then $O(c_{\tau,k}k)$ is required to find the τ closest codewords ($c_{\tau,k}$ is a small number that depends logarithmically on either τ or k, depending on the algorithm used), resulting in total of $\boldsymbol{T_{\text{VQ}} = O((d + c_{\tau,k})k)}$.
- The ADMM solution for the LASSO is an iterative

procedure. Each iterations includes a multiplication of a $(k \times k)$ matrix by a k dimensional vector ($O(k^2)$), a shrinkage function ($O(k)$) and vector additions ($O(k)$), resulting in complexity of $O(k^2)$ per iteration. On top of that, there is $O(dk)$ for once multiplying the dictionary matrix by the feature vector, and there are M_ϵ iterations, until the procedure converges to ϵ-tolerance, so the complexity for the LASSO encoding becomes $\boldsymbol{T_{\text{LASSO}} = O(M_\epsilon k^2 + dk)}$.

CS is the lightest encoding method and VQ adds a bit more computation. Recently linear convergence rate was shown for solving the LASSO with ADMM [51], implying that $M_\epsilon = O(\log \frac{1}{\epsilon})$, but even with fast convergence ADMM is still heavier than VQ. This theoretical analysis is verified in empirical runtime measurements, presented in Figure 5. We average over the same 50 songs, and use the same computer (PC laptop) with single CPU core. The runtime tests fit a linear dependency on k for CS and for VQ (with slope depending on τ) and a super-linear dependency on k for the LASSO.

Using the LASSO has an expensive runtime price. With small computational effort one can use VQ with a larger codebook (*e.g.* $k = 2048$) and get better performance in both MIR tasks.

VI. Conclusion

We show an advantage to using PCA decorrelation of MFSΔ features over MFCC. The difference is statistically significant, but small, showing that also the data-agnostic DCT manages to compress music data well. Increasing the codebook size results in improved performance for all the encoding methods. The LASSO and CS are inconsistent with regard to the preferred pooling method (mean or max-abs). For all the encoding methods the performance deteriorates when the encoding parameter has too high or too low values. While the LASSO and CS can suffer sharp decrease in performance when adjusting their parameters, VQ is more robust, having smooth and controlled change in performance when adjusting its density parameter τ.

We find that a simple, efficient encoding method (VQ) can successfully compete with the more sophisticated method (the LASSO), achieving better performance, with much less computing resources. Using *top-τ VQ with PPK transformation* consistently achieves high performance (almost always beating other methods) in both query-by-tag and query-by-example. It is fast and easy to compute, and it is easily adjustable with its parameter τ. We recommend this representation method as a recipe to be applied to other low-level features, to represent various aspects of musical audio. The resulting representations are concise, easy to work with and powerful for music recommendation in large repositories.

References

[1] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on speech and audio processing*, vol. 10, no. 5, pp. 293–302, 2002.

[2] A. Meng and J. Shawe-Taylor, "An investigation of feature models for music genre classification using the support vector classifier," in *Proc. International Society for Music Information Retrieval conference (ISMIR)*, 2005, pp. 604–609.

[3] J. Reed and C. Lee, "A study on music genre classification based on universal acoustic models," in *Proc. International Society for Music Information Retrieval conference (ISMIR)*, 2006, pp. 89–94.

[4] D. P. Ellis, "Classifying music audio with timbral and chroma features," in *ISMIR 2007: Proceedings of the 8th International Conference on Music Information Retrieval: September 23-27, 2007, Vienna, Austria*. Austrian Computer Society, 2007, pp. 339–340.

[5] R. Grosse, R. Raina, H. Kwong, and Y. Ng, A., "Shift-invariant sparse coding for audio classification." Conference on Uncertainty in AI, 2007.

[6] A. Manzagol, P., T. Bertin-Mahieux, and D. Eck, "on the use of sparse time-relative auditory codes for music." International Society for Music Information Retrieval conference (ISMIR), 2008.

[7] M. Mandel and D. Ellis, "Multiple-instance learning for music information retrieval," in *Proc. International Society for Music Information Retrieval conference (ISMIR)*, 2008, pp. 577–582.

[8] C. J. S. Essid and G. Richard, "Temporal integration for audio classification with application to musical instrument classification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 17, no. 1, pp. 174–186, 2009.

[9] P. Hamel and D. Eck, "Learning features from music audio with deep belief networks." International Society for Music Information Retrieval conference (ISMIR), 2010.

[10] M. Henaff, K. Jarrett, K. Kavukcuoglu, and Y. LeCun, "Unsupervised learning of sparse features for scalable audio classification," in *International Society for Music Information Retrieval conference (ISMIR)*, 2011, pp. 681–686.

[11] J. Wulfing and M. Riedmiller, "Unsupervised learning of local features for music classification," in *International Society for Music Information Retrieval conference (ISMIR)*, 2012, pp. 139–144.

[12] C. Yeh, M. C., and H. Yang, Y., "Supervised dictionary learning for music genre classification," in *ICMR*, 2012.

[13] C.-C. M. Yeh, L. Su, and Y.-H. Yang, "Dual-layer bag-of-frames model for music genre classification," in *Proc. ICASSP*, 2013.

[14] M. Mandel, G. Poliner, and D. Ellis, "Support vector machine active learning for music retrieval," *Multimedia systems*, vol. 12, no. 1, pp. 3–13, 2006.

[15] D. Turnbull, L. Barrington, D. Torres, and Lanckriet, "Semantic annotation and retrieval of music and sound effects," *IEEE Transactions on Audio, Speech, and Language Processing*, 2008.

[16] D. Eck, P. Lamere, T. Bertin-Mahieux, and S. Green, "Automatic generation of social tags for music recommendation," in *Advances in Neural Information Processing Systems*, 2007.

[17] 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*, vol. 37, no. 2, pp. 115–135, June 2008.

[18] L. Barrington, M. Yazdani, D. Turnbull, and G. Lanckriet, "Combining feature kernels for semantic music retrieval," 2008, pp. 723–728.

[19] B. Tomasik, J. Kim, M. Ladlow, M. Augat, D. Tingle, R. Wicentowski, and D. Turnbull, "Using regression to combine data sources for semantic music discovery," in *Proc. International Society for Music Information Retrieval conference (ISMIR)*, 2009, pp. 405–410.

[20] E. Coviello, A. Chan, and G. Lanckriet, "Time Series Models for Semantic Music Annotation," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 19, no. 5, pp. 1343–1359, July 2011.

[21] J. Nam, J. Herrera, M. Slaney, and J. Smith, "Learning sparse feature representations for music annotation and retrieval," in *International Society for Music Information Retrieval conference (ISMIR)*, 2012, pp. 565–570.

[22] K. Ellis, E. Coviello, A. Chan, and G. Lanckriet, "A bag of systems representation for music auto-tagging," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21-19, pp. 2554–2569, 2013.

[23] J. T. Foote, "Content-based retrieval of music and audio," in *Voice, Video, and Data Communications*. International Society for Optics and Photonics, 1997, pp. 138–147.

[24] B. Logan and A. Salomon, "A music similarity function based on signal analysis," in *IEEE International Conference on Multimedia and Expo*, 2001, pp. 745–748.

[25] J. Aucouturier and F. Pachet, "Music similarity measures: What's the use?" in *Proc. International Society for Music Information Retrieval conference (ISMIR)*, 2002, pp. 157–163.

[26] M. Slaney, K. Weinberger, and W. White, "Learning a metric for music similarity," in *Proc. International Society for Music Information Retrieval conference (ISMIR)*, 2008, pp. 313–318.

[27] M. Hoffman, D. Blei, and P. Cook, "Content-based musical similarity computation using the hierarchical Dirichlet process," in *Proc. Inter-*

national Society for Music Information Retrieval conference (ISMIR), 2008, pp. 349–354.
[28] K. Yoshii, M. Goto, K. Komatani, T. Ogata, and H. G. Okuno, "An efficient hybrid music recommender system using an incrementally trainable probabilistic generative model," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 16, no. 2, pp. 435–447, 2008.
[29] B. McFee, L. Barrington, and Lanckriet, "Learning content similarity for music recommendation," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 8, pp. 2207–2218, October 2012.
[30] B. Logan, "Mel frequency cepstral coefficients for music modeling," in *Proc. International Society for Music Information Retrieval conference (ISMIR)*, vol. 28, 2000.
[31] T. Bertin-Mahieux and D. P. Ellis, "Large-scale cover song recognition using the 2d fourier transform magnitude," in *Proceedings of the 13th International Conference on Music Information Retrieval (ISMIR 2012)*, 2012.
[32] P. Hamel, S. Lemieux, Y. Bengio, and D. Eck, "Temporal pooling and multiscale learning for automatic annotation and ranking of music audio." International Society for Music Information Retrieval conference (ISMIR), 2011.
[33] M. McKinney and J. Breebaart, "Features for audio and music classification," in *Proc. International Society for Music Information Retrieval conference (ISMIR)*, 2003, pp. 151 –158.
[34] A. Flexer, F. Gouyon, S. Dixon, and G. Widmer, "Probabilistic combination of features for music classification," in *Proc. International Society for Music Information Retrieval conference (ISMIR)*, 2006, pp. 111–114.
[35] A. Berenzweig, B. Logan, P. W. Ellis, D., and B. Whitman, "A large-scale evaluation of acoustic and subjective music-similarity measures," *Computer Music Journal*, vol. 28, no. 2, pp. 63–76, 2004.
[36] E. Coviello, Y. Vaizman, B. Chan, A., and G. Lanckriet, "Multivariate Autoregressive Mixture Models for Music Autotagging," in *13th International Society for Music Information Retrieval Conference (ISMIR 2012)*, 2012.
[37] E. Coviello, B. Chan, A., and G. Lanckriet, "The variational hierarchical EM algorithm for clustering hidden Markov models," in *Neural Information Processing Systems (NIPS 2012)*, 2012.
[38] T. Jebara, R. Kondor, and A. Howard, "Probability product kernels," *The Journal of Machine Learning Research*, vol. 5, pp. 819–844, 2004.
[39] R. Lyon, M. Rehn, S. Bengio, C. Walters, T., and G. Chechik, "Sound retrieval and ranking using sparse auditory representations," *Neural Computation*, vol. 22, no. 9, pp. 2390–2416, 2010.
[40] C. Smith, E. and S. Lewicki, M., "Efficient auditory coding," *Nature*, vol. 439, pp. 978–982, 2006.
[41] Y. Yang and M. Shah, "Complex events detection using data-driven concepts," in *ECCV*, 2012, pp. 722–735.
[42] A. Coates and A. Y. Ng, "The importance of encoding versus training with sparse coding and vector quantization," in *International Conference on Machine Learning (ICML)*, 2011.
[43] A. Coates, H. Lee, and A. Y. Ng, "An analysis of single-layer networks in unsupervised feature learning," *Journal of Machine Learning (JMLR)*, vol. 15, p. 48109, 2010.
[44] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.
[45] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2010.
[46] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online learning for matrix factorization and sparse coding," *The Journal of Machine Learning Research*, vol. 11, pp. 19–60, 2010.
[47] B. McFee and G. Lanckriet, "Metric learning to rank," in *Proceedings of the 27th International Conference on Machine Learning (ICML'10)*, June 2010.
[48] D. Tingle, Y. E. Kim, and D. Turnbull, "Exploring automatic music annotation with "acoustically-objectiv" tags," in *Proc. MIR*, New York, NY, USA, 2010.
[49] O. Celma, "Music recommendation and discovery in the long tail," 2010.
[50] P. Jaccard, "Etude comparative de la distribution florale dans une portion des alpes et des jura," *Bulletin del la Societe Vaudoise des Sciences Naturelles*, vol. 37, pp. 547–579, 1901.
[51] M. Hong and Z.-Q. Luo, "On the linear convergence of the alternating direction method of multipliers," *arXiv preprint arXiv:1208.3922*, 2012.

Yonatan Vaizman received B.Sc. in Computer Science and Computational Biology from the Hebrew University in Jerusalem, Israel (HUJI) in 2007 and M.Sc. in Electrical and Computer Engineering from University of California, San Diego (UCSD) in 2014. He is currently working towards his Ph.D. in Electrical and Computer Engineering at UCSD. His research focuses on signal processing and machine learning methods for computer audition and music information retrieval.

Brian McFee received the B.S. degree in Computer Science from the University of California, Santa Cruz in 2003, and M.S. and Ph.D. degrees in Computer Science and Engineering from the University of California, San Diego in 2008 and 2012. In 2012, he joined the Center for Jazz Studies at Columbia University as a postdoctoral research scholar. His research interests include applications of machine learning to music recommendation and audio analysis. In 2010, he was a recipient of the Qualcomm Innovation Fellowship. His favorite genre is chiptune.

Gert Lanckriet received the MS degree in electrical engineering from the Katholieke Universiteit Leuven, Belgium, in 2000 and the M.S. and Ph.D. degrees in electrical engineering and computer science from the University of California, Berkeley, in 2001 and 2005, respectively. In 2005, he joined the Department of Electrical and Computer Engineering, University of California, San Diego, where he heads the Computer Audition Laboratory. His research focuses on the interplay of optimization, machine learning, and signal processing, with applications in computer audition, and music and multimedia information retrieval. He was awarded the SIAM Optimization Prize in 2008 and is the recipient of a Hellman Fellowship, an IBM Faculty Award, an NSF CAREER Award, and an Alfred P. Sloan Foundation Research Fellowship. In 2011, MIT Technology Review named him one of the 35 top young technology innovators in the world (TR35). In 2014, he received the most influential 10-year paper award at the International Conference for Machine Learning (ICML). He is a senior member of the IEEE.