# Singer Recognition and Modeling Singer Error

Johan Ismael
Stanford University
jismael@stanford.edu

Nicholas McGee
Stanford University
ndmcgee@stanford.edu

## 1. Abstract

We propose a system for recognizing a singer based on past observations of the singer's voice qualities and their ability to sing a given song. Such a system could be useful to improve predictions in Query by Humming systems, or as biometric identity data in consumer applications. As we'll describe in more details in the following sections, we managed to obtain a fairly high accuracy on individual voice recognition by training a multi-class SVM using Mel-frequency Cepstral Coefficients (MFCCs) as our main features. We then tried to improve our accuracy by using insight from QbH systems, that is, characterizing singers also on how well they can perform specific notes. In the process we developed a note segmentation system which shows promising results using an unsupervised learning technique. Finally, we made the problem harder by attempting to recognize singers with background music.

## 2. Introduction

Our original idea was to implement a Query by humming (QbH) system which goal would be to recognize the songs people are humming, using a dataset of .wav files of people humming songs corresponding to some ground truth monophonic midi files[9]. Ideally, we would try to match hums with mp3 files instead of midi but we quickly realized that the melody extraction tools required to work with real music simply do not exist yet.

To implement this system, our first goal was to extract features, most precisely pitch and rythmic intervals (PIs and LIRs), as suggested by B. Pardo et al.'s paper[7]. The idea was to learn, for each person, what we call the singer's error. Knowing this error we could, from a hum, reconstruct the ground truth and then try to find a best match with our database of midi files.

As we started to implement our first feature extractor, we ran into the first roadblock: how to segment the original hum into different notes, which is required to perform a note by note comparison to the ground truth and build our model? We tried to use two tools freely available [4][2] but those did not give very good results. Our system is particularly sensitive to this step because with a few notes wrongly recognized, the whole comparison could be misaligned and therefore unusable. In addition, thanks to further research we realized that the most successful QbH systems [6] today are not trying to directly match hums with ground truth songs. They are instead matching hums with labeled hums, which seems to make a lot of sense since a hum and a ground truth are so far apart in terms of spectrum that it's almost like comparing carrots and tomatoes. And here is our second roadblock: it seems necessary to have a fairly big database of labeled hums. After unsuccessfully asking the Interactive Audio Lab [1] to use their data, we finally decided to solve a different but related problem since our QbH system seemed to be built on too many weak links.

We realized that with our dataset, we could solve another problem: singer recognition. Would it be possible to recognize someone singing a song, after training on them singing other songs? How accurate can we be? Would this be possible with famous singers supported by instruments?

## 3. Early Prototypes

### 3.1. First QbH Prototype

For our initial, exploratory prototype, we built a quick system in python which used the vamp melodia plugin [10], to process input and target songs, and calculated the edit distance between the input and each target song. For determining the cost of mismatches during edit distance calculation, we used a cost function which deemphasized common errors such as octave and half-step misses note misses, but without a more complex singer error model. On a set of 48 target songs and 8 user inputs, this prototype achieved 1 correct match, for an unimpressive 12.5% accuracy. At this point we began to realize that there were many more unsolved problems in processing the input for the system than in the learning algorithm itself.

### 3.2. EigenSongs

Eigenfaces uses PCA to reduce the feature space of an image by calculating a set of eigenvectors which capture the systematic variances in a set of facial images (essen-

tially, determining the most important patterns in the image)[12]. Because songs can similarly be interpreted as a linear feature vector of amplitudes, we wanted to see if the patterns in a song could be identified by performing PCA on several aligned recording of the same song. We ran into two roadblocks which caused us to abandon the idea. First, we would need a very large set of samples per song to do the initial PCA, and this was not practical given our dataset. Second, and more importantly, it is very hard to evaluate whether PCA is truly extracting useful information about the song, or simply extracting noise. Our initial prototype performed poorly, leading us to believe the technique was probably not reliable enough to develop further.

Given these unsuccessful attempts and what we learned from them, we decided to reformulate our problem and focus on singer recognition.

## 4. Singer Recognition: Description and Results

### 4.1. Overview of the Problem Solved

Our final system performs matching between unique individuals and audio clips of their voices. Given a recording of a person singing (who is known to have appeared in our training set), the system identifies which of the training singers is singing the input song. This recognition is useful for training and applying an individually tuned singer-error model for QbH, as well as for general voice fingerprinting.

### 4.2. Dataset

We drew 69 singers from the MIR-QBSH database [9], each of which had sung between 15 and 30 songs from a pool of 48 possible songs. We used 10 of these songs as training data to identify each singer, and tested our classifier using the remaining data.

### 4.3. Features and Training

Our algorithm trains a multi-class SVM (using liblinear[3]) based on the feature set extracted from each song in our training data. We tried several different feature sets during evaluation, but found that the most effective feature set was a simple one. Figure 1 shows the first 20 MFCC's for two different singers side by side. As is apparent in the figure, the cepstral breakdown of different singers' voices have perceivably different fingerprints. We found that the first 40 MFCC's provide the most useful information, with the addition of more coefficients providing little benefit after that. We discuss other possible good features in Section 4.1.

### 4.4. Results

Our results using this model are surprisingly good, considering the small size of the feature set. As can be seen

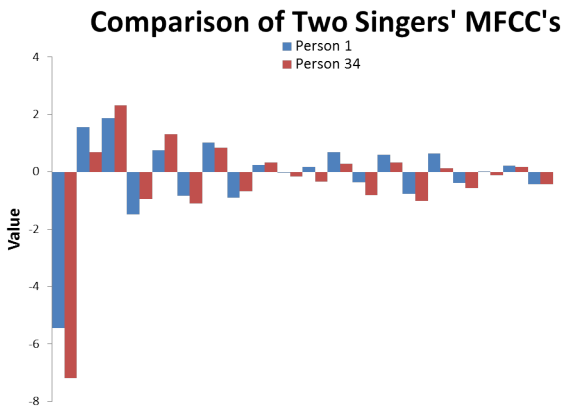


Figure 1. Noticeable differences appear in the MFCC's of different singers

| Features | Accuracy over 35 Singers |
|---|---|
| Spectrogram | 5.62% |
| 20 MFCC's | 74.94 % |
| **40 MFCC's** | **78.57%** |
| 40 MFCC's + spectrogram | 78.22% |
| 80 MFCC's | 78.22% |

Figure 2. Accuracy of our system using various feature sets

in Figure 3, on small numbers of unique singers, we were able to identify around 90% of test recordings as the correct singer. As the singer space increases, accuracy naturally decreases (some singers have very similar voices, so matching voice characteristics alone is not sufficient to distinguish them), but this system could be useful in many cases. For example, any device with multiple user accounts could use audio input to detect users and log into the correct account with a high degree of accuracy.

With these satisfying results, we decided to move forward and experiment in two directions. First, what features could we add to improve our accuracy? Second, how would our system perform on harder problems such as singer recognition with background music?

## 5. First Experiment: Improving Accuracy

Our first experiment aimed at improving the accuracy of our system. To do so, the idea has been to add new features

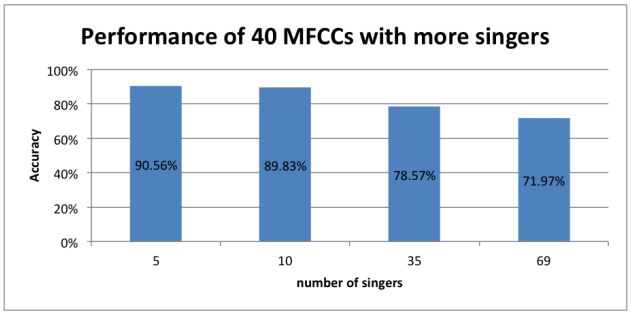


Figure 3. Accuracy of our system in different singer spaces

in our model that are able to capture something that is characteristic of a given singer and is not captured by MFCCs.

## 5.1. Using a Singer Error Model

### 5.1.1 The Idea

MFCCs are great at capturing the essence of someone's voice, whether they are singing or even just talking. So if we could add a feature that leverages an attribute specific to singing, it would probably be something that is not captured by MFCCs. Moreover, as we were reading about QbH systems earlier in our project, we were excited about one approach: modeling singer error. But this time, we wouldn't model singer error in order to reconstruct the ground truth of a song, but to characterize a particular singer. So how can we do that?

We can see this error as some Gaussian noise. So we'll represent it by a normal distribution, centered on the ground truth note, and we'll consider that the variance represents this error. Because we all have difficulties singing different notes, it seems relevant to have one Gaussian per type of note. Indeed, maybe singer A is really good a singing the middle C, but is terrible when it comes to high A's. Since the original files that we'll do the comparison with are using the midi format, we'll use midi notations to represent notes everywhere. They range from 21 to 108. Let's formalize these ideas:

$$X_i \sim \mathcal{N}(i, \sigma_i^2) \; \forall i \; \epsilon \; [21, 108]$$

where the $X_i$'s represent the note sung by a singer when trying to render the actual note numbered $i$.

To estimate these variances, we use the maximum likelihood estimate on our training set. For each singer and each note, we gather all the instances of what that singer actually sang when trying to sing the given note. The variance is then given by:

$$\frac{1}{m} \sum_{i=1}^{m} (x_i - i)^2$$

where the $x_i$'s represent the notes sung by our singers.

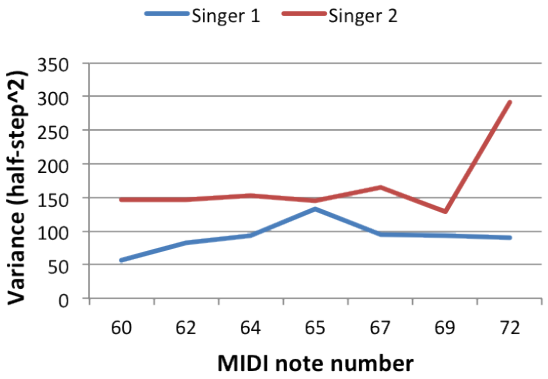


Figure 4. Variances in note accuracy for different singers

At the end of this process, we have a vector of variances per singer. The big issue is that to perform this estimation, we need to be able to segment the hums into separate notes. As stated in the introduction, note segmentation tools are not quite perfect. However, to get an idea of how discriminative our model was, we manually performed the note segmentation for two singers, five songs each. After this (long) process, we obtained the results shown in Figure 4

Though it is hard to draw solid conclusions out of two singers but it seems the two we used would be fairly separable with this characterization.

One observation with this model is that we are sensitive to a change of octave. If a singer transcribes perfectly a song one octave down, we'll understand this as a pretty large error. But this is what we want: if you're always singing A4 instead of A5, we want to capture it even though the rendition might be nice to listen to.

### 5.1.2 Plugging into our current algorithm

Our learning algorithm uses MFCCs classifies singer with one data point per singer,song pair whereas the model we've just mentioned outputs one data point per singer. To solve this problem we could calculate the maximum likelihood estimate for our gaussians once per song. But that means we would probably have very few samples per note for each data point. To alleviate, we could use 'binning' and divide our range of notes into three separate bins: low range, mid-range and high range. We would then end up three additional features for each data point: the error on the low range notes, the error on the mid range notes and the error on the high range notes.

Given the issues we had with note segmentation, we haven't had the opportunity to test the accuracy of our system with these additional features. However, we tried to improve note segmentation tools with unsupervised learning techniques.

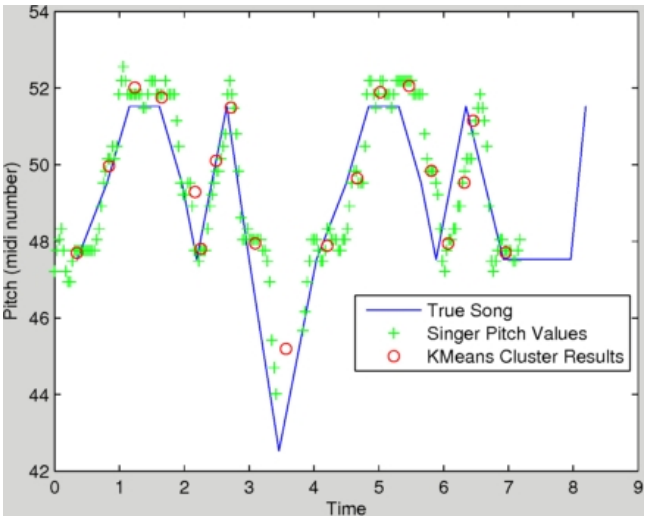


Figure 5. K-means clusters denoting the notes sung by the singer

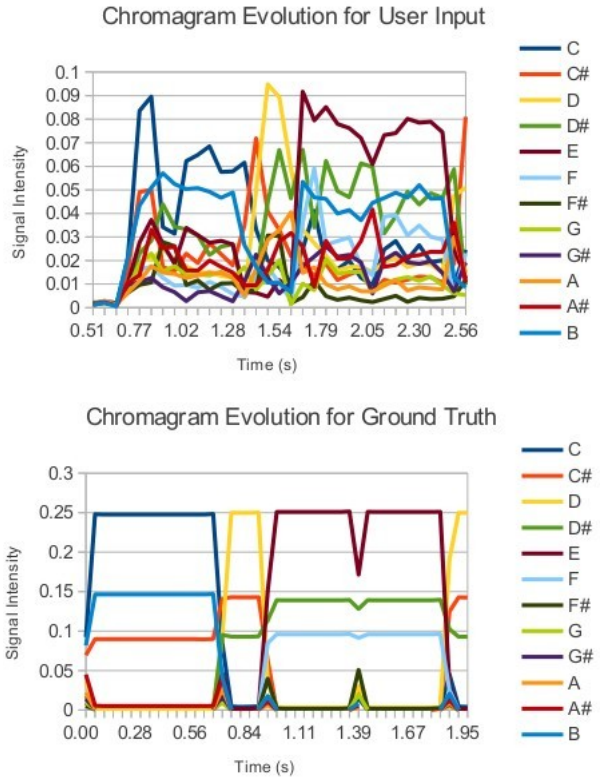


Figure 6. A comparison of chromatic composition

## 5.2. Segmenting Notes for Use in the Model

In developing the singer error model, one of the largest problems we ran into is that available audio tools cannot accurately identify notes being sung or hummed, thus making it difficult to train based on singer error. While we performed some manual labelling, this is not practical for large datasets. K-means clustering provides a practical method for labelling large datasets where the target song is known for each sample to be labelled. While it is often of primary concern in clustering algorithms to avoid locally maximal clusters, in our case we want the algorithm to settle into a local maximum given by the closest fit to the actual, target song. To achieve this, we seed the algorithm by choosing k to be the number of notes in the target song, and setting the means at the center of each target note. Notes which do not occur in the user's input naturally drop out because they do not become associated with any sample points. Our algorithm uses $(time, frequency)$ tuples as coordinates for the clustering algorithm, and the results on a sample song are shown in Figure 5. Results are very good for songs where the ground-truth song approximately matches the input in tempo and key, but not when the two are misaligned. To solve key alignment, we simply align the medians of the 2 inputs. Choosing the optimal tempo alignment is a much more complicated problem which we believe leaves room for future work.

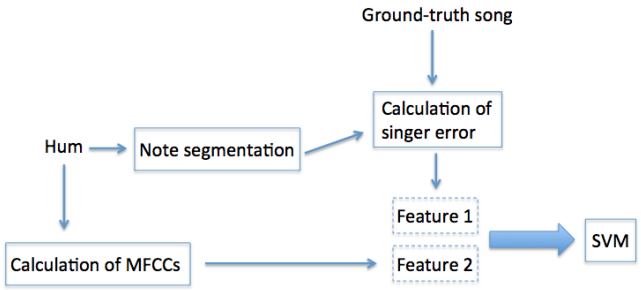


Figure 7. Our ideal singer recognition system

Another problem we observed which causes clustering to go awry is that clusters are often too close together, and there are not enough samples to differentiate them. This can be addressed by extracting more points to cluster on, but how? Figure 6 shows the evolution of a chromagram over time for the ground-truth song, as well as the noisier input from a user. Notice that despite the noise, dominant frequencies remain roughly consistent with the ground-truth song. Including non-dominant chroma as points and discounting their distance based on intensity would provide more sample points and hopefully improve the clustering algorithm.

## 5.3. The ideal system

An effective note segmentation system would allow us to automate the singer error calculation. Figure 7 illustrates what our ideal system would look like.

| | Total accuracy | Accuracy on acapella test data | Accuracy on test data with background music |
|---|---|---|---|
| **Training on interviews** | 56.3% | 72% | 0% |
| **Training on interviews and songs** | 65.8% | 100% | 0% |

Figure 8. Performance of our system recognizing famous singers

## 6. Second Experiment: Tackling more difficult problems

In addition to attempting to improve the accuracy of our system with more features, we were curious to see how well our SVM would perform on harder problems. So we decided to see what would happen if, after training our SVM on two famous singers talking in interviews, we tested on actual songs which featured both singing and instruments in the background. We picked Thom Yorke, singer of the British rock band Radiohead, and Barry Manilow, a famous American singer, for our experiment. We gathered interviews of them on Youtube. For test data, we picked some of their songs.

We obtained the results shown in Figure 8. Our first observation is that MFCCs don't work anymore where there is noise in the background. It would probably be helpful to separate voice from background instruments, but that's another hard problem. Interestingly, it seems that it is possible to train on spoken words as in interviews and obtain a decent accuracy on acapella singing. The accuracy reaching 100% on acapella singing has one caveat: we are training on different chunks of the same recording that we are using for testing so the results are probably good because of an overfitting problem.

## 7. Conclusions and Next Steps

We end this project with a better understanding of what machine learning can do for music today. On the one hand, spectrum analysis makes available very strong features like MFCCs that are very useful for identifying specific audio signals. On the other hand, when it comes to actually understanding the content, current solutions are ineffective or inconsistent. This makes it very hard to automatically extract features such as melody, or even notes, for use with learning algorithms. The Music Information Retrieval (MIR) community has been very active and working on these different problems. Hopefully, challenges such as source separation, note segmentation and melody extraction will be overcome in the near future and enable a lot of very exciting music-related applications.

When it comes to future work, we are particularly excited to use unsupervised learning for note segmentation, as opposed to pure signal processing approaches. Getting this problem solved would hopefully improve our singer recognition system, and even make QbH more addressable.

## References


[1] The Interactive Audio Lab. http://music.cs.northwestern.edu/.

[2] MIRToolbox. https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mirtoolbox.

[3] Liblinear – A Library for Large Linear Classification http://www.csie.ntu.edu.tw/ cjlin/liblinear/

[4] The vamp audio analysis plugin system. http://vamp-plugins.org/.

[5] Sonic annotator. http://omras2.org/SonicAnnotator.

[6] B. P. Arefin Huq, Mark Cartwright. Crowdsourcing a real-world on-line humming system. Sound and Music Computing, 2010. http://smcnetwork.org/files/proceedings/2010/64.pdf.

[7] D. L. et al. A query by humming system that learns from experience. 2007.

[8] L. Myers. An exploration of voice biometrics. 2004.

[9] J.-S. R. Jang. Mir-qbsh corpus. Available at the "MIR-QBSH Corpus" link at http://www.cs.nthu.edu.tw/ jang.

[10] J. Salamon. Melodia - Melody extraction vamp plug-in. http://mtg.upf.edu/technologies/melodia.

[11] J. Salamon et al. Tonal Representations for Music Retrieval: From Version Identication to Query-by-Humming http://www.justinsalamon.com/uploads/4/3/9/4/4394963/salamon-mmir.pdf

[12] P. B. et al. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. 1997.

[13] R. Kline et al. Approximate Matching Algorithms for Music Information Retrieval Using Vocal Input http://lyle.smu.edu/ mhd/8337sp07/kline.pdf