The bad and the good singer: query tuning analysis for audio to audio Query by Humming

Filippo Morelli

MASTER THESIS UPF 2013

Supervisors

Emilia Gómez and Justin Salamon

Department of Information and Communication Technologies

Universitat Pompeu Fabra, Barcelona

Acknowledgements

Thanks to Anna Laura Carrarini, Simonetta Carrarini, Ramon Ferrer Auzmendi, Daniela Marini, José Javier Valero Mas, Fabio Morelli and Luca Piersanti for their precious collaboration and to all the friends who dedicated their time and their talent to help this project go one step further.

Abstract

A tuning quality evaluation algorithm meant to be used as a user advising subsystem for Query-by-Humming applications is designed and presented.

A queries dataset from a previous study is extended and analysed in order to define some relevant tuning quality descriptors. The automatic procedures needed to calculate those descriptors are also devised and optimized.

A preexisting tuning quality expert annotation is expanded by collecting new evaluations in a specific experiment.

All the available data and methods are extensively tested using machine learning software.

Resum

Un algorisme per l'avaluació de la qualitat de l'entonació i destinat a ser utilitzat com un subsistema d'assesorament de l'usuari en aplicacions de Query-by-Humming es va dissenyar i presentar.

Un dataset de queries d'un estudi presedent es va estendre i estudiar per tal de definir alguns descriptors de la qualitat de l'entonació. Els procediments automàtics necessaris per calcular els descriptors també es van concebre i optimitzar.

Una preexistent anotació de la cualitat de l'entonació fet per un expert es va expandir mitjançant la recopilació de noves avaluacions en un experiment especific.

Totes les dades i metodes disponibles es van provar exhaustivament utilitzant un software de machine learning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Query-by-Humming systems are negatively affected by input queries with an overall poor quality of the tuning[2]. Therefore, the whole QbH task would benefit from the insertion of a module that automatically evaluates the tuning quality and warns the user if it is not sufficient in order for the target song to be successfully retrieved. In the present project we present a straightforward tuning evaluation algorithm on which a QbH user assisting module could be based.

We inherited a queries dataset from a previous study[2] together with an expert annotation about tuning quality. This preexisting evaluation was formulated, for simplicity, according to a basic classification system that consisted of two classes only: *good* tuning or a *bad* tuning. Since we wanted to exploit this data and benefit from the reduced complexity of the binary classification, we adopted it too. The queries dataset and its annotation were extended by adding new queries in order to even out the number of instances of each class.

The tuning evaluation system was designed by means of an iterative process consisting of the following stages:

- selecting a set of relevant descriptors for tuning quality evaluation,
- designing and optimizing the parameters of an algorithm to automatically calculate the descriptors,
- calculate the descriptors for each query in the dataset,
- evaluate the performance of the overall system using automatic classification.

Here we present the best performing and most complete set of descriptor according to our results and we describe the methods and the parameters with which they are calculated.

Finally, we report and comment a series of automatic classification experiments performed with Weka, one of the most common machine learning software. Our results demonstrated that the system performs well considering the limitations that the adoption of a binary classification involves.

In order to further test our system and provide extended results we made an annotation experiment with a group of volunteers and gathered a collective annotation that was used for some additional experiments.

The whole system was written in Python[1] and designed with the intention of developing a consistent set of tools or even a very basic framework for tuning estimation and evaluation.

1.1 Document outline

The present document is organized as follows:

- Chapter 2 presents the scientific background and the basic motivations of the present work;
- Chapter 3 contains the detailed description of the method followed to develop our system and the different stages that constitute it;
- Chapter 4 describes the results of the tuning evaluation performed with our system and tested using automatic classification, in addition it reports the collective annotation experiment;
- Chapter 5 resumes what was done in the project and explores some possible future perspectives, developments and applications.

[1] http://www.python.org/

Chapter 2

Background

In the last two decades the number and size of music and audio databases has increased, thanks to the growing interest in digital music services. At the same time, the need for new and effective ways of querying these databases has grown. The most immediate and spontaneous method chosen by humans to express such queries is using their voice, in particular singing or humming [15]. The task of interpreting the user vocal query and retrieving a corresponding target song is called Query-by-Humming (QbH) or Query-by-Singing/Humming (QbSH) [3].

Understandably, the user ability at precisely recalling and reproducing part of the target song strongly affects the retrieval performance [16]. Consequently, dealing with the many vocal inaccuracies produced by the average user is a challenging task, not only for any automatic QbH system designed so far, but also for humans themselves [17].

The concept of "Query-by-Humming" as a novel way of querying a database containing audio information and the first prototypical system were introduced by Ghias et al. in 1995 [3]. Since then, Query-by-Humming has established as a well-known task in the Music Information Retrieval (MIR) community[1], different systems has been proposed and tested (see 2.2.2) and even some commercial products implement and exploit this functionality (e.g. the SoundHound mobile app[2]).

Query-by-Humming represents both a theoretical context and an applicative framework for the present project. In addition, this work addresses the more general issue of tuning frequency instability in the vocal performances of poorly skilled singers.

[1] http://www.music-ir.org/mirex/wiki/2013:Query_by_Singing/Humming
[2] http://www.soundhound.com/

2.1 Query-by-Humming task

Pardo and Birmingham [17] compared QbH to posing a "musical question" to a clerk in a record shop, that is to say asking for a song whose only information the customer can recall is part of the melody. Consequently, the authors tested the performances of some common algorithms against those of humans in a typical QbH matching task. The latter prevailed with an average accuracy of 66%. This surprisingly mediocre result gives an idea, as well as an approximate measure, of the complexity of such a task. An automatic system not only have to face all the complications usually involved in MIR but also those associated with the peculiarities of human voice and human capacity to reproduce tunes.

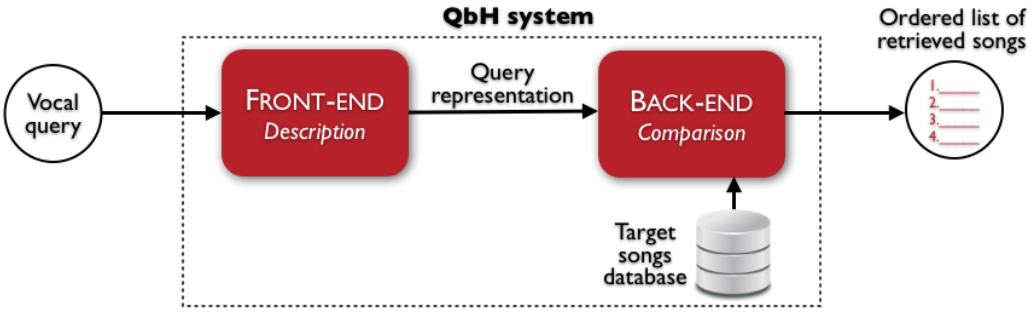


Figure 2.1: Schematic representation of a typical QbH system.

2.2 Query-by-Humming systems

2.2.1 Outline

As shown in Figure 2.1, a typical QbH system consists of two separate subsystems that are often studied and implemented separately.

The *front-end* subsystem processes the input vocal query to obtain a specific representation of it, usually a series of fundamental frequencies or *pitch contour*. This block may also perform some additional processing to refine the query representation (e.g. segmentation, quantization, normalization, transcription).

The *back-end* subsystem matches the query representation, as output by the *front-end*, against the songs in the database (also called *targets*). This normally requires the whole database to be pre-processed too in order to obtain a representation consistent with the one used for the input query and

perform the comparison in a coherent domain. Depending on the size of the database, the *back-end* may take advantage of some indexing technique to make the search faster and more effective. The output of the system is typically a list of retrieved songs ordered according to their similarity to the input query.

In one of the most common scenarios, the *front-end* extracts the pitch contour of the query and then transcribes it to obtain a symbolic representation (e.g. any kind of musical notation) then the *back-end* tries and matches this information with analogous transcriptions of the target songs.

To increase the performances, this very essential scheme is often expanded by adding processing modules to both the main blocks. For instance, the *front-end* could benefit from the insertion of a query analysis and/or query correction/reformulation modules. The first could alert the user if the quality of the singing is inadequate and ask for a new input while the second could automatically fix the query.

2.2.2 Examples

The first QbH system proposed by Ghias et al. [3] exploited the earliest algorithms for pitch estimation and used a Parsons code (a three symbols alphabet that indicates if a note has equal, upper or lower pitch with respect to the previous one) to represent both the pitch contour of the query and the target songs, which were then matched by string comparison.

Little et al. [7] created a system that adapted its internal parameters, either in the *front-end* and in the *back-end*, with a mechanism of *relevant feedback*. It returned a list of songs from the database, ranked by similarity, among which the user indicated the desired one. A genetic algorithm learnt from the user indications about the position of the correct song in the output list and automatically tweaked the system. The optimal values were customized for individual users as well as groups of users. The chosen representation for the queries was a sequence of pairs indicating the pitch interval between adjacent note segments and the ratio between the length of a note segment and the length of the following segment. This kind of representation has the advantage of being both transposition and tempo invariant, compared to an absolute melodic contour.

Duda et al. [18] did an experimental study testing a series of descriptors for the vocal input and distance measures for the comparison. To represent the query they created a method inspired by the "karaoke effect" that exploited information about the spatial arrangement of voices and instruments in the stereo mix. The resulting time series were aggregated into symbolic strings. This allowed to apply distance measures for string pattern matching when

Singers	MRR	First hit rate (%)
Good tuning	0.67	61.25
Bad tuning	0.33	28.95
All	0.56	50.85

Table 2.1: QbH accuracy on a collection of 2,125 songs for two singer profiles [2].

performing the comparison.

The system devised by Ryynämen and Klapuri [9] was based on an index of melodic excerpts built by extracting pitch vectors from a database of songs. The input vocal query got transcribed into notes and then converted into a pitch vector. Then the system searched for melodic fragments similar to the query pitch vector inside the database. The search was made more efficient by using locality sensitive hashing (LSH). The retrieved melodies were ranked according to their distance from query and returned to the user.

2.3 The *tuning problem* in vocal queries

The vocal ability of the singer is one of the main factors affecting the retrieval accuracy of a QbH system [19]. Meek and Birmingham were the first to catalogue and statistically model the most common errors of the average singer [16]. The present work specifically focuses on one kind of error, the tuning frequency instability throughout the query. Here, we refer to this issue as the *tuning problem*.

In a recent study [2], Salamon et al. made an approximate evaluation of the effects of the tuning problem using a baseline QbH system. The authors manually annotated their queries and divided them in two classes: “good tuning" and “bad tuning", intentionally ignoring any other mistake of the singer other than the tuning inconsistency. The results show a significant gap between the accuracies of the two classes, as reported in Table 2.1. Unal et al. got an analogous result dividing their singers into musically trained and untrained ones [20] while performing a statistical analysis of their query collection.

2.4 Tuning evaluation

Nakano et al. [21] proposed a method for evaluating singing ability not requiring any information about the sung melody but relying only on pitch

interval accuracy and vibrato. The dataset was composed of 600 melodies, hummed by female and male singers, already classified as *good* or *poor* by human subjects in a previous experiment assessing singing skills. For each sample in the dataset 5 descriptors were extracted: 2 related to tuning accuracy and 3 to vibrato. A 10-fold cross-validation experiment using the full dataset and a Support Vector Machine classifier got an accuracy of 83.3%.

Lech [12] studied, implemented and tested a QbH subsystem that detected out-of-tune notes and proposed corrections to the user, who then was able to accept or discard them and even to make other adjustments to the melodic pitch contour. This solution relied on existing algorithms for fundamental frequency detection, tuning frequency estimation and pitch shifting.

More recently, Nichols et al. [22] devised a system to discover talented musicians by analyzing online videos. Similarly to the present project, the authors assessed intonation by building a pitch histogram, then creating a grid in it according to a chosen reference tuning (the maximum bin in the histogram) and finally extracting statistical measures of the deviations from the grid.

In all the aforementioned studies, the tuning frequency is estimated directly from the input. Implementing one of those solutions in a QbH context would imply inserting a tuning evaluation module *before* the *front-end*. In the present project, the tuning evaluation module exploits the melodic pitch contour as output by the *front-end* and so it is thought to be inserted *after* it.

Chapter 3

Methodology

The goal of this project is to implement and study the accuracy of a tuning evaluation algorithm that takes as input pitch contours generated by the MELODIA melody extraction system [1]. This algorithm is meant to be used as a module in a complete QbH system that exploits MELODIA to represent both the vocal queries and the target songs. As seen in Chapter 2.4, earlier works exist that focus on tuning evaluation but, to the best of our knowledge, none developed an algorithm specifically meant to exploit the output of the *front-end* of a QbH system.

3.1 Queries collection

The findings, the data and the tools of a recent study on version identification and QbH by Salamon et al. [2] constituted a starting point for our work. In particular, we initially used the same collection of vocal queries (also available online[1]). This is a set of 118 queries sung by 17 subjects (9 females and 8 males) who were asked to select songs in a *Canonical Collection* of 481 elements. The singers have mixed musical experience, from having none at all to being amateur musicians. In order to simulate a realistic QbH scenario, all the queries were recorded with a common laptop microphone and no post-processing was applied. The queries duration have a minimum of 11s, a maximum of 98s and an average of 26.8s.

Together with the queries, the *expert annotation* of the original experiment was available. According to it, of the 118 queries 88 had a *good tuning* and 30 had a *bad tuning*. This twofold classification was adopted also in the present study, which extends the original straightforward evaluation of the tuning quality. Since we planned to use machine learning tool for automatic

[1] http://mtg.upf.edu/download/datasets/MTG-QBH

classification, we wanted to even up the number of instances of the two classes in the test dataset. In order not to reduce the size of the dataset, we extended it by adding new queries.

3.1.1 New queries

The simplest way of having an equal number of *good* and *bad* queries was to add 58 new *bad* queries to the original dataset, ending up with 88 instances of each class.

We searched for the new *bad* queries on Midomi[2], an online QbH service that allows the users to save and share what they record. The clips on this website are of mixed audio quality and may contain an instrumental backing, thus we were only interested in those with an acceptable level of noise (comparable to the one of the original queries) and sung a cappella by a single person. Eventually we selected 58 clips showing a clear *bad* tuning and whose target song was in the aforementioned *Canonical Collection*. They are sung by 31 different subjects (14 females and 17 males), have a minimum length of 22s, a maximum length of 47s and an average length of 33s.

The resulting queries dataset contains 176 queries. The original *expert annotation* was straightforwardly integrated by judging *bad* all the new queries, according to the criterion with which they were selected.

Lastly, the whole dataset was shuffled in order to randomize the sequence of queries preventing undesired clustering.

3.2 Queries description

Once collected the queries, the next step was building an algorithm to perform the kind of elaboration they would undergo if input in the *front-end* of the previously described hypothetical QbH system based on MELODIA. This algorithm was designed to perform all the steps described below automatically, letting the user set the values of a few parameters.

3.2.1 Melody extraction

As anticipated, the use of this state-of-the-art system for melody extraction was a premise of our work. In facts, this tool is especially powerful and QbH represents one of its main potential applications [2]. Here we recall briefly the different stages of the algorithm as described by the authors. Further references can be found in the original work by Salamon et al. [1].

[2] www.midomi.com

> First the audio signal is analyzed and spectral peaks (sinusoids) are extracted. This process is comprised of three main steps: first, a time-domain equal loudness filter is applied, which has been shown to attenuate spectral components belonging primarily to non-melody sources. Next, the short-time Fourier transform is computed and the local maxima (peaks) of the spectrum are detected at each frame. In the third step, the estimation of the spectral peaks' frequency and amplitude is refined by calculating each peak's instantaneous frequency (IF) using the phase vocoder method and re-estimating its amplitude based on the IF. The detected spectral peaks are subsequently used to compute a representation of pitch salience over time: a salience function. The salience function is based on harmonic summation with magnitude weighting, and spans a range of almost five octaves from 55 to 1,760Hz. In the next stage, the peaks of the salience function are grouped over time using heuristics based on auditory streaming cues. This results in a set of pitch contours, out of which the contours belonging to the melody need to be selected. The contours are automatically analyzed and a set of contour characteristics is computed. In the final stage of the system, the contour characteristics and their distributions are used to filter out non-melody contours. The melody F0 at each frame is selected out of the remaining pitch contours based on their salience.

The queries of the original dataset already came with a melodic pitch contour that was extracted by applying MELODIA optimally tweaked for each sample. Consequently, we only had to repeat this same process with the 58 new queries, adjusting the algorithm to extract the best possible pitch contour from each of them. In particular MELODIA lets the user change the value of the following parameters:

- maximum detectable frequency (to limit octave errors),
- voicing (to manage the assignment of pitches to voices[3]),
- minimum peak salience (to deal with noise).

To make the optimization process easier, we synthesized the extracted pitch contours in order to to detect possible inaccuracies directly by listening.

[3] in principle, this parameter should be ineffective when dealing with monophonic audio but, in practice, it is not. This happens because MELODIA is not yet optimized for the monophonic case.

The output of the algorithm is a series of time instants and frequency values in Hz. Next, this values are converted to cent according to:

$$n = 1200 \log_2 \left(\frac{f}{f_0} \right) \tag{3.1}$$

where n is the frequency in cent, f is the frequency in Hz and $f_0 = 440$Hz is a reference frequency. Equation 3.1 relates frequencies in Hertz and number of semitone intervals in equal temperament, once a reference tuning is fixed. If an audio input is perfectly equal tempered, the conversion to cent yields values that are equally spaced by multiples of 100 cent (e.g. 254, 354, 454, etc.). Additionally, if the tuning frequency of the piece is identical to the reference frequency used in Equation 3.1 the values in cent are exact integer multiples of 100 (e.g. 0, 100, 200, etc.). This is a consequence of the logarithmic subdivision of the octave in 12 intervals of 100 cent that characterizes equal temperament. In most practical situations, though, the In Figure 3.1 an example of melodic pitch contour extracted with MELODIA and converted to cent is shown.

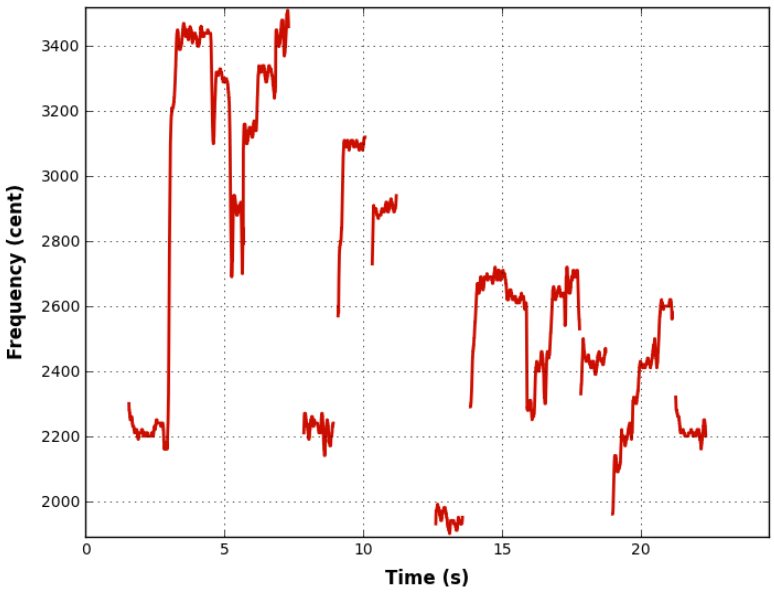


Figure 3.1: Melodic pitch contour extracted with MELODIA and converted to cent.

The output of the melody extraction tool can be considered equivalent to that of a very basic QbH *front-end*. It can serve as the input of the tuning

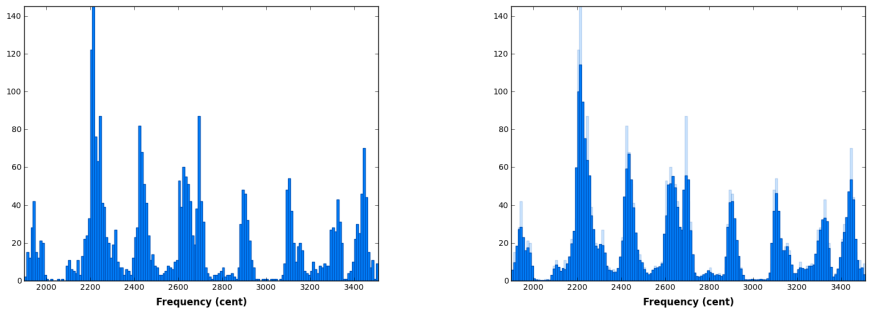
evaluation module we projected and whose processing stages are described below.

3.2.2 Pitch histogram

First of all, the melodic contour is processed to build a *pitch histogram*. Its frequency resolution is controlled by the `PH bps` parameter of the algorithm, that is set by default to 10 cent (MELODIA's frequency resolution).

In order to reduce its jaggedness and make the upcoming peak detection less error-prone, the histogram can be low-pass filtered. A parameter called `PHsmoothing` enables the corresponding processing, which consists in substituting each histogram value with the average over a rectangular window of size $\lfloor$`PH bps`$/3\rfloor$ centered on it.

Figure 3.2 shows the *pitch histogram* of the melodic contour of Figure 3.1 before and after being low-pass filtered.



(a) `smoothing`=0 (b) `smoothing`=3

Figure 3.2: *Pitch histogram* before (a) and after (b) low-pass filtering.

As already mentioned, the histogram of an ideal input, perfectly tuned in equal-temperament, would be a series of bars equally spaced by multiples of 100 cent, each one corresponding to a semitone. A small deviation from this ideal case still preserving good tuning (e.g. the presence of a slight vibrato) would yield some lower bars at each side of the central one giving each semitone approximately a hill shape. This observation allows to make a rough visual evaluation of the tuning: as long as each semitone shows a well centered, symmetrical and clearly peaked hill-shape not larger than 100 cent, the analyzed pitch contour sounds in tune. Besides, detecting the position of the semitones centers, when possible, allows to estimate the tuning frequency of the melody. This is the calculation our algorithm implements in its next stage.

First, all the local maxima in the histogram are detected. Then the algorithm goes through each of them, in descending order of magnitude, to detect only those that can be considered actual *peaks*, in the sense of semitone centers. The first maximum is automatically taken as a peak in order to use it as initial reference. All the remaining maxima are considered peaks only if their distance from already detected peaks is greater than half the histogram frequency resolution. This condition helps taking as peaks only those maxima that have a high chance of being the actual tip of the aforementioned hill-shaped distribution of bars inside a semitone. Consequently, a reasonably low-pass filtered histogram makes this detection considerably more accurate. The search stops if there are no more peaks or if their number has reached a maximum, set by an optional `TE Npeaks` parameter. In Figure 3.3 the peaks detected in the histogram of Figure 3.2 are highlighted in red.

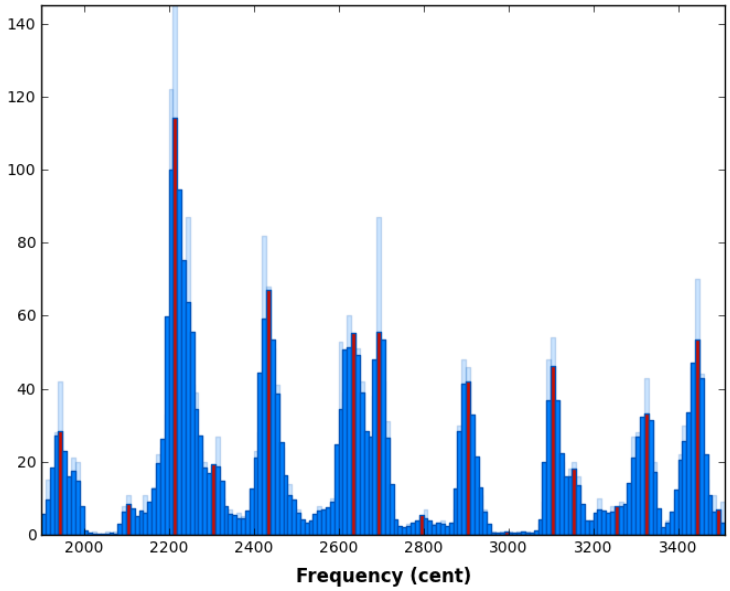


Figure 3.3: Pitch histogram with all the detectable peaks (possible semitone centers) highlighted in red.

One quick way of selecting a tuning frequency is straightforwardly adopting that of the highest peak. However, we preferred to implement a more refined procedure.

According to the chosen histogram resolution, there are `PH bps` bins every 100 cents (or 1 semitone). Consequently there are `PH bps` possible locations for the semitone centers and, correspondingly, `PH bps` choices for the tuning

frequency. Since, as we said before, we expect to have a local maxima at the center of each semitone, the detected peaks should help reveal a regularity pattern and suggest the most probable tuning frequency. Our algorithm minimizes the weighted sum of the distances between the detected peaks and the `PH bps` candidate semitone center locations, using the peaks magnitudes as weights. This is equivalent to fitting a grid on the histogram, as shown in Figure 3.4. Obviously, we obtain better results the more the melody follows equal-temperament. Unfortunately vocal queries rarely satisfy this condition exactly. Nonetheless, clearly *bad* queries generally show bigger deviations from ideality than the average, still allowing us to exploit the previous criterion to detect them.

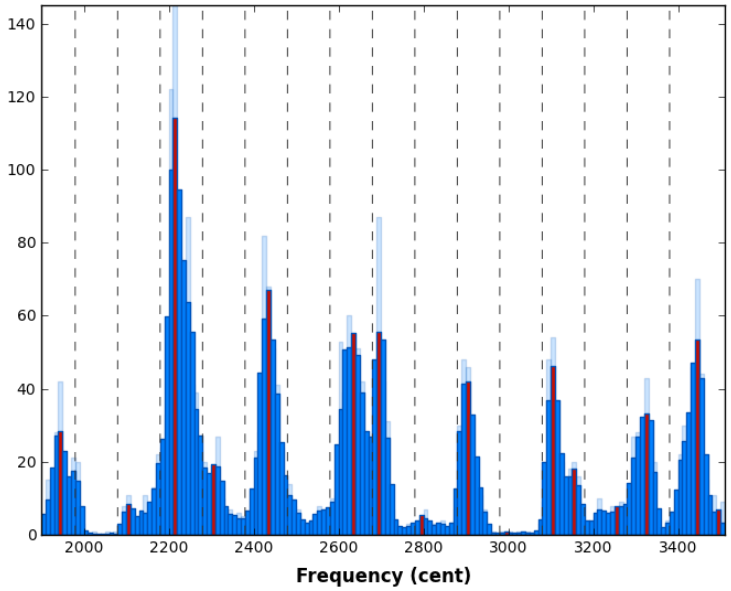
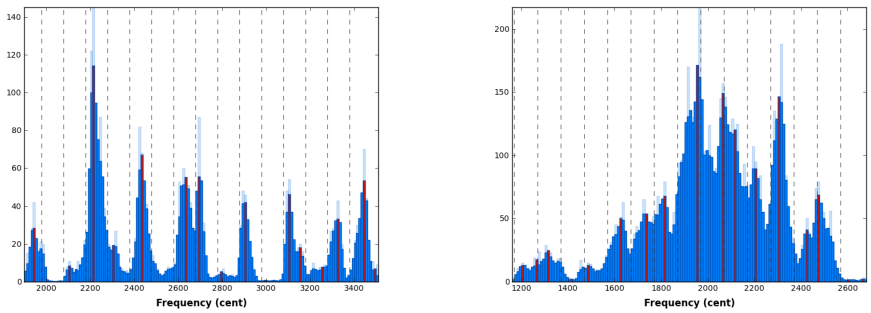


Figure 3.4: Pitch histogram with the peaks in red and the best fitting semitone grid highlighted by the gray dashed lines separating adjacent semitones.

Figure 3.5 provides an example of the difference between a query whose tuning frequency is approximately stable and easy to detect (*good* tuning) and one whose tuning frequency is much less evident (*bad* tuning). In the first histogram the semitone division is neater, even if the detected peaks does not always coincide with the semitone centers. In the second histogram there is a large group of tall bars spanning several semitones and giving little consistency to the detected grid.



(a) *Good* tuning (b) *Bad* tuning

Figure 3.5: Pitch histograms of vocal queries having (a) a clear or (b) an unclear tuning frequency.

3.2.3 Descriptors

In order to evaluate the quality of the tuning, we adopted a set of descriptors similar to that Nichols et al. [22] devised in a previous study to assess *intonation*.

First we extracted a feature vector of `PH bps` elements obtained by superimposing all the semitone intervals in which the histogram got divided (as shown in Figure 3.4), summing the bins in corresponding positions and normalizing so that the elements sum to 1. This descriptor is supposed to give an overall outline of the semitones, as shown in Figure 3.6).

Then we divided the frequency values of the pitch contour in a series of groups composed by all those samples included in a specific semitone interval. That allowed us to calculate the standard deviation, the skewness and the kurtosis of the distribution of frequency samples in each semitone and then average them. By default, the average operation is performed using the sums of all the samples in each group as weights. This option, though, can be disabled using the parameter `ST weights`.

We also added three novel descriptors: the first is the histogram peaks deviation average (the same implemented to estimate the tuning frequency), the second is identical to the first but considers *pits* (selected local minima) instead of peaks and the last is an analogous average deviation but calculated using the maximum values inside each histogram semitone. The maximum number of *peaks* and *pits* for this particular task are fixed by the `DEV Npeaks` and `DEV Npits` parameters. The three average calculations are weighted, respectively: with the peaks magnitudes, with the magnitudes of the samples on the left of the pits and with the magnitudes of the maxima.

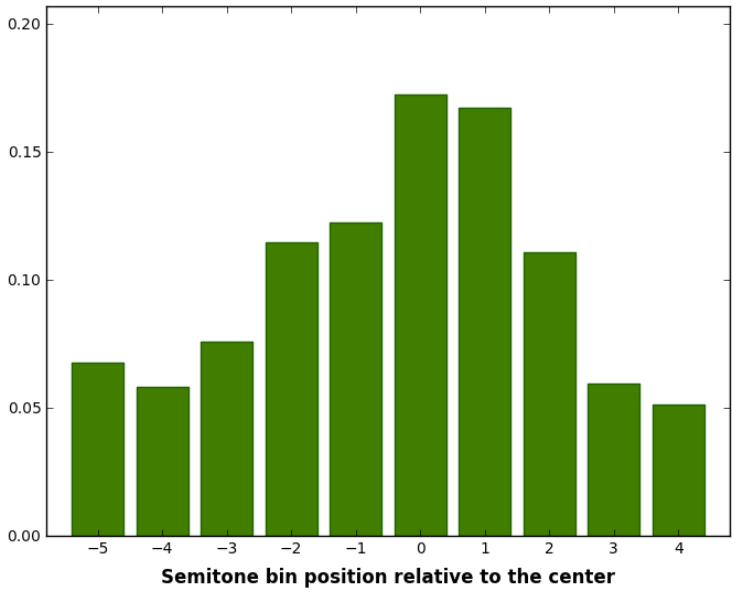


Figure 3.6: Sample plot of the overall semitones outline feature vector.

In total we collected `PHbps` + 6 values for each query:

- `PH bps` feature vector of the overall semitones outline,
- average standard deviation of the pitch contour samples divided in semitones,
- average skewness of the pitch contour samples divided in semitones,
- average kurtosis of the pitch contour samples divided in semitones,
- histogram *peaks* deviation average,
- histogram *pits* deviation average,
- histogram semitones maxima deviation average.

All this information was used to feed an automatic classification system whose performance was the main object of our experiments.

3.3 Queries classification

In order for our tuning evaluation system to be completely automatic, we had to integrate a machine learning algorithm that, by learning our dataset descriptors (the *attributes*) and tuning annotation (the *classes*), could classify vocal queries (the *instances*) with the best possible accuracy.

The software chosen to test automatic classification is Weka 3[4] by the University of Waikato. It is a powerful suite comprising different tools for machine learning. In particular we were interested in its algorithms for classification and for attributes evaluation.

Weka can open and manage different dataset file formats, the default one is a proprietary format having `arff` extension. It is a text file containing preliminary information about the attributes and the classes followed by the data itself structured as comma separated values.

We implemented an additional procedure to automatically set-up experiments. It takes the query audio file and a set of parameters in input, calculates the descriptors exploiting our tuning evaluation method and outputs a properly formatted `arff` file. Each parameter may be input as a list of values as well, in this case as many dataset files will be output as the possible settings combinations. Eventually Weka was fed with one or more of these resulting files, as it allows to operate on a single dataset (*Explorer* modality) as well as perform batch analysis with multiple datasets and multiple algorithms (*Experimenter* modality).

3.3.1 Validation technique

When a classification algorithm is trained, it optimizes its parameters building a model that fits the training data as well as possible. In general, though, testing the so-obtained classifier with an independent validation dataset, will result in a lower accuracy. This common phenomenon is called *overfitting* and happens when the model does not fit the validation data as well as it fits the training one. Cross-validation is a technique that helps to take into account this effect even when a proper validation set is not available. It consists in dividing the original dataset, creating a subset used to train the model and one used for validation. This procedure is generally repeated several times changing the subsets in order to reduce variability. Depending on the partitioning strategy, different cross-validation schemes can be defined.

[4] http://www.cs.waikato.ac.nz/ml/weka/

K-fold cross-validation

K-fold cross-validation consists in randomly partitioning the original dataset into K subsets of equal size. During one turn of the validation process, $K-1$ subsets are used to train the model while one subset is kept for testing. This procedure, called *fold*, is repeated K times ensuring that each subset is used exactly once as the validation data. The K results from the folds are combined (usually averaged) to produce a single estimation.

3.3.2 Accuracy measures

A common representation of the basic information describing the performance of a classifier is the so-called *confusion matrix*, a table which shows how the instances of each class have been categorized by the algorithm. In binary classification, confusion matrices are particularly easy to interpret as illustrated by Table 3.1 which refers to a generic *Good*/*Bad* classification task.

		Classified	
		Good	*Bad*
Actual	*Good*	True positive (TP)	False negative (FN)
	Bad	False positive (FP)	True negative (TN)

Table 3.1: Example of a two-classes confusion matrix.

From that data a series of measures can be calculated. We briefly describe the ones that we used in our tests.

Percent correct

The percent correct, also directly called accuracy, is the fraction of instances classified correctly out of the total number (see Table 3.1(a)). It represents the probability that a random sample is correctly classified and it is defined by:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \tag{3.2}$$

(a) Percent correct

Actual \ Classified	*Good*	*Bad*
Good	TP	FN
Bad	FP	TN

(b) Precision

Actual \ Classified	*Good*	*Bad*
Good	TP	FN
Bad	FP	TN

(c) Recall

Actual \ Classified	*Good*	*Bad*
Good	TP	FN
Bad	FP	TN

Table 3.2: Confusion matrices illustrating (a) percent correct, (b) precision and (c) recall.

Precision

Precision measures the fraction of instances assigned correctly to a specific class out of all those classified as such (see Table 3.1(b)). It represents the probability that a random sample assigned to a class actually belongs to that class and it is defined by:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.3}$$

Recall

Recall or sensitivity or true positive rate (TPR) measures the fraction of instances assigned correctly to a specific class out of all those actually belonging to that class (see Table 3.1(c)). It represents the probability that a random sample belonging to a class is correctly classified and it is defined by:

$$\text{recall} = \text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.4}$$

F-measure

F-measure combines precision and recall and it is defined as their harmonic mean:

$$\text{F-measure} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{3.5}$$

ROC curve

When assigning an instance to a discrete set of class, some algorithms use a fixed decision threshold that in unbiased binary classification is, generally, the central value (i.e. 0.5). A receiver operating characteristic or ROC curve, is a graphical tool representing how the performance of a binary classifier changes

as its threshold is varied. It is a plot having the true positive rate (TPR) on the vertical axis and the false positive rate (FPR) on the horizontal axis. The first is equivalent to recall and is the fraction of true positive instances out of the total positive ones (see 3.4 and Table 3.2(a)), the second is the fraction of false positive instances out of the negative ones (see Table 3.2(b)):

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{3.6}$$

(a) True Positive Rate

Actual \ Classified	*Good*	*Bad*
Good	TP	FN
Bad	FP	TN

(b) False Positive Rate

Actual \ Classified	*Good*	*Bad*
Good	TP	FN
Bad	FP	TN

Table 3.3: Confusion matrices illustrating (a) true positive rate (TPR) and (b) false positive rate (FPR).

A random classifier would always yield $TPR = FPR$, so it would generate a straight diagonal line going from the origin (no instance classified as positive: $TPR = 0$ and $FPR = 0$) to the upper right corner of the graph (all instances classified as positive: thus $TPR = 1$ and $FPR = 1$), as shown in Figure 3.7. This line divides the graph in two parts: all the points in the upper (lower) part represent classifiers performing better (worse) than the random one. All the curves start in the origin and end in the upper right corner but the more they bend towards the upper left corner (ideal classification: $TPR = 1$ and $FPR = 0$) the closer to ideality the associated algorithm performs. Consequently, the area under the curve can be used as an accuracy measure. In normalized units, its value goes from 0 to 1 (0.5 being the random classifier) and it is equal to the probability that the algorithm will classify as positive more than negative a randomly chosen instance of the positive class (i.e. *Good* in our examples). The ROC curve is one of a series of similar plots based on different accuracy measures. All the points on these curves are in a 1-to-1 correspondence because all the scalar accuracy measures are function of one free parameter. For instance, if once fixed TP (see Table 3.1), the algorithm function sets FP and then the total number of instances in each class allows to calculate FN and TN.

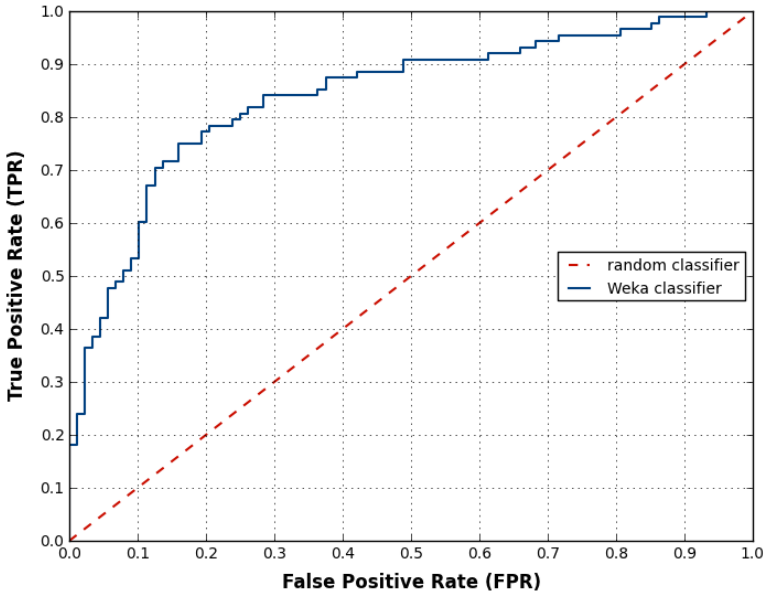


Figure 3.7: ROC curves of a random classifier and of a Weka classifier as generated by the software.

3.3.3 Attributes analysis

Each tuning quality descriptor we extracted from vocal queries (see Section 3.2) corresponds to an attribute in the classification task. Part of our study consisted in evaluating how much each of these attributes is meaningful and benefits the classification. In order to perform this analysis, we exploited the *select attributes* section of the Weka software that includes both attributes evaluation and attributes selection tools.

Attributes evaluation

This task consists in evaluating and ranking the attributes in a dataset according to a chosen measure of their worth. In particular, for our tests we used *information gain* which estimates how much the information entropy of the class (i.e. the amount of uncertainty about it) changes as a consequence of knowing the considered attribute:

$$\text{InfoGain}(class, attribute) = H(class) - H(class|attribute) \tag{3.7}$$

This value goes from 0, when the attribute yields no information about the class and knowing it does not modify the entropy of the class, to 1, when

the attribute provides all the information needed to determine the class and knowing it makes the entropy of the class go to zero.

Attributes selection

Attributes or features selection is a machine learning process that consists in selecting a subset of attributes that are particularly relevant in order to build a classification model. The objective of this procedure is to eliminate redundant or irrelevant attributes, when present. An attribute is considered *redundant* when it does not bring additional information to a preexisting subset, while it is *irrelevant* when it does not bring any useful information at all.

Usually the selection procedure consists of a searching task and an evaluating task. The first one decides how the algorithm moves in the space of possible subsets (which ones are considered and in what order) while the second compares the subsets according to a given measure. The main types of selection are the *wrapper*, the *filter* and the *embedded* ones. *Wrapper* methods compare subsets by training a classifier with them and considering the resulting accuracy whereas *filter* methods exploit specific information descriptors (e.g. correlation or mutual information). Lastly, the attribute selection procedures integrated and automatically performed as part of a classification algorithm are called *emdebbed* methods.

3.4 Additional annotation

As reported in 3.1, the original collection of queries included an *expert annotation* of tuning quality that was extended together with the dataset to include the new queries.

Most of our experiments used this expert annotation as a reference and the machine learning algorithms took the class value from it. Such an expert annotation, although supposedly reliable, is a human opinion not a ground truth, so it is intrinsically questionable and open to doubt. Therefore we were interested in collecting a wider opinion about our queries tuning quality and use it for comparison and further analysis.

We asked a small group of volunteers to listen to all the queries in our dataset and classify them as *good* or *bad* according to their overall tuning quality. This resulted in a *collective annotation* that was compared with the *expert* one both directly and by measuring the accuracy of classifiers trained with it.

Chapter 4

Results

The experiments we report and comment here, were performed to optimize our tuning evaluation algorithm and to test the effect of the annotation used to train automatic classification.

4.1 Weka set-up

We performed an extensive preliminary test using Weka's *Experimenter* tool in order to select a small number of classification algorithms to use in the experiments. It consisted in repeating 100 times a 10-fold cross-validation classification task on some test datasets generated with our system using all the available algorithms with default settings. Eventually, we chose the following algorithms:

- `NaiveBayes`: implements a straightforward Naive Bayes classifier;
- `BayesNet`: builds a Bayes Network model letting the user choose among various search algorithms and quality measures;
- `IBk`: implements an instance-based k-nearest neighbours classifier (with $k = 13$);
- `SGD`: implements Stochastic Gradient Descent to learn different linear models (e.g. Support Vector Machine and Logistic Regression);
- `SimpleLogistic`: builds a linear Logistic Regression model (in case of nominal classes it first converts them to binary);
- `SMO`: implements Sequential Minimal Optimization to train a Support Vector classifier with Polynomial or RBF kernel;

- `LMT`: builds a "Logistic Model Tree" which is a classification tree with Logistic Regression at the leaves;

4.2 Parameters optimization

According to what exposed in Chapter 3, our tuning evaluation system requires a few number of parameters to be set-up. In particular:

- `PH bps`: the pitch histogram resolution in bins[1] per semitone;
- `PH smoothing`: a boolean value that enables low-pass filtering of the pitch histogram;
- `TE Npeaks`: the maximum number of pitch histogram peaks to use to estimate tuning frequency (can be set to detect *all* the peaks);
- `DEV Npeaks`: the maximum number of pitch histogram peaks to use to calculate the average peaks deviation descriptor (can be set to detect *all* the peaks);
- `DEV Npits`: the maximum number of pitch histogram pits to use to calculate the average pits deviation descriptor (can be set to detect *all* the peaks);
- `ST weights`: a boolean value that enables the use of weights to average the semitones statistics (i.e. standard deviation, skewness and kurtosis);

As anticipated in Section 3.3, we designed an automatic procedure to automatically generate Weka datasets using our tuning evaluation algorithm. This was done mainly to test the corresponding configurations with the Weka *Experimenter* tool using the algorithms listed in Section 4.1. Therefore we set a variability range for all the parameters and we generated a dataset for each resulting combination. In particular, `PH bps` was varied from 7 to 10 (the maximum allowed by the melody extraction plugin), the numbers of peaks and pits were varied from 5 to 30 with a step of 5 plus the "all" case (when all the found peaks or pits are considered) while the remaining boolean parameters were alternatively True or False.

The combination that provided the best performance in the *Experimenter* testing environment and that was selected as optimal is reported in Table 4.1. The dataset obtained with this configuration was used for all the subsequent experiments. We will refer to it as the *optimal parameters dataset*. The results it yielded with all the considered classifiers are shown in Table 4.2.

Parameter	Value
PH bps	10
PH smoothing	Enabled
TE Npeaks	All
DEV Npeaks	All
DEV Npits	All
ST weights	Enabled

Table 4.1: Optimal values of the tuning evaluation algorithm parameters selected using Weka *Eperimenter*.

Algorithm	% correct	Precision	Recall	ROC area
NaiveBayes	77.89 ± 9.19	0.75 ± 0.10	0.87 ± 0.11	0.85 ± 0.10
BayesNet	77.45 ± 10.01	0.77 ± 0.11	0.81 ± 0.14	0.86 ± 0.09
IBk	79.15 ± 8.87	0.79 ± 0.11	0.82 ± 0.12	0.86 ± 0.09
SGD	80.34 ± 9.48	0.78 ± 0.11	0.86 ± 0.13	0.80 ± 0.09
SimpleLogistic	80.15 ± 9.51	0.80 ± 0.11	0.82 ± 0.13	0.87 ± 0.09
SMO	80.20 ± 9.39	0.79 ± 0.10	0.85 ± 0.13	0.80 ± 0.09
LMT	80.12 ± 9.50	0.80 ± 0.11	0.82 ± 0.13	0.87 ± 0.09

Table 4.2: Automatic classification results obtained with the optimal parameters configuration of Table 4.1.

The experiment consisted in repeating 50 times a 10-fold cross-validation classification with each algorithm. All the classifiers got an accuracy around 80% with a considerably large standard deviation. Therefore, according to a corrected[2] paired t-test automatically performed by Weka *Experimenter*, there are no statistically significant differences among those results. Nonetheless we decided to adopt `SimpeLogistic`, the algorithm offering the best overall result according to Table 4.2, to extend the analysis.

4.3 Best result analysis

The results shown in Table 4.3 and in Table 4.4 were obtained with Weka *Explorer* performing a 10-fold cross-validation experiment using the `SimpleLogistic` classifier and the *optimal parameters dataset*.

The the inaccuracy is about 20% and there is a slight overall predominance of *bad* tuning predictions. This circumstance is common to all the the tested

[1] the frequency units of the pitch histogram

[2] accounts for the samples not being independent because of cross-validation

Measure	**Values**
Correctly classified	143 (81.25%)
Incorrectly classified	33 (18.75%)
ROC area	0.875

	bad tuning	*good* tuning	weighted average
True positive	0.841	0.784	0.813
False positive	0.216	0.159	0.188
Precision	0.796	0.831	0.814
Recall	0.841	0.784	0.813
F-measure	0.818	0.807	0.812

Table 4.3: Detailed accuracy measures of a 10-fold cross-validation experiment with the `SimpleLogistic` classifier.

		Predicted tuning	
		Good	*Bad*
Actual tuning	*Good*	69	19
	Bad	14	74
		83	**93**

Table 4.4: Confusion matrix of a 10-fold cross-validation experiment with the `SimpleLogistic` classifier.

classifiers and is, probably, a consequence of a series of factors:

- the chosen *descriptors* could not be sufficient or completely appropriate for the task (see Section 4.4 for descriptors analysis);
- human annotations are subjective (see the experiment in Section 4.5) and can benefit of contextual clues not available to machines, in particular humans are not forced to decide according to a fixed threshold between *good* and *bad* tuning therefore their evaluation criteria may vary;
- limiting the tuning quality evaluation problem to a binary one considerably simplified it at the expense of accuracy.

Applying some Weka pre-processing filters, we isolated 30 instances that

are frequently mistaken by most of the classifiers we chose (see Section 4.1). In particular, we exploited a meta-classifier that allows to combine the results of a series of algorithms by means of different voting strategies (in our case a simple majority vote was adopted). Listening to those queries we could not identify a unique and simple reason why they get often misclassified. Nonetheless we could make a few interesting observations. Some very good performances were classified as having a *bad* tuning probably because of their complex melody rich in deliberate tuning excursions, for instance: "Desafinado" a bossa nova song by Antonio Carlos Jobim with an intentionally challenging melody or "Take the A Train" a jazz standard by Billy Strayhorn. Another detectable group of queries is characterized by moderately elaborated melodies sung by males with a deep and unsteady voice so that, probably, even if generally they are in-tune, they could be easily mistaken in a strict binary classification. A remarkable evidence is the presence, among these queries, of several interpretations by different subjects of the same target song like "Let It Be" or "Across the Universe" by The Beatles. Lastly, several queries correctly annotated as having a *bad* tuning quality had an overall mediocre but still acceptable singing quality and some short heavily detuned passages. This circumstance could have a simple but interesting explanation: according to what we observed during our collective annotation experiment, an average human listener tends to be strongly impressed by sudden out-of-tune bursts and to judge negatively the corresponding interpretation, mostly if it is generally mediocre, while our algorithm, which considers the vocal query as a whole and therefore averages its quality, may not collect enough evidence and come to the same conclusion. A more advanced system, able to exploit the time information as well as a whole range of tuning quality evaluations, could probably deal better with these issues.

4.4 Descriptors analysis

As anticipated in Section 3.3.3 we performed some experiments to evaluate the worth of our tuning quality descriptors. Specifically, we applied some of the tools for attribute selection of Weka *Explorer* to the same dataset analyzed in Section 4.3). Running an information gain (see Section 3.3.3) ranking filter we obtained the results shown In Table 4.5, according to which no attribute is really good for tuning evaluation. The highest merit was scored by the average deviation of all peaks and by some elements of the vector representing the overall semitones outline. The worth of the average statistical moments calculated on the pitch samples of each semitone varies from the fair result obtained by standard deviation to the low ones of skewness and kurtosis

respectively. Finally, the average deviation of the semitones maxima and of the pits proved to be almost entirely useless.

Attribute	**Merit**
all peaks average deviation	0.283
semitones outline bin 1	0.2594
semitones outline bin 2	0.2296
average standard deviation	0.2199
semitones outline bin 7	0.2095
semitones outline bin 5	0.1978
semitones outline bin 6	0.178
semitones outline bin 10	0.1308
average skewness	0.1255
average kurtosis	0.1221
semitones outline bin 3	0.1068
semitones maxima deviation average	0.0808
semitones outline bin 9	0.0681
all pits average deviation	0
semitones outline bin 4	0
semitones outline bin 8	0

Table 4.5: Results of an information gain attributes ranking filter.

To confirm the previous result, we also performed attribute evaluation with a wrapper algorithm exploiting the `SimpleLogistic` classifier and an internal 10-fold cross validation. The search selected a descriptors subset composed by *all peaks average deviation*, *semitones outline bin 1*, *semitones outline bin 5* and *semitones outline bin 7* (the order is casual). This result is coherent with the information gain ranking of Table 4.5 considering that it does not account for redundancy. The *average standard deviation* and the *semitones outline bin 2* descriptors were not selected because they probably provide a smaller or equal amount of information compared with, respectively, *all peaks average deviation* and *semitones outline bin 1*. Finally, this subset was tested with the same set-up used to analyze the entire set (see Section 4.3). Specifically, we performed a 10-fold cross-validation experiment using the *SimpleLogistic* classifier on the *optimal parameters dataset* in which all the attributes except the 4 selected ones where deleted. Some resulting accuracy measures are reported in Table 4.6. It shows that the performance is almost identical to that outlined by Table 4.3 only with a slightly lower value of the area under the ROC curve.

Measure	Values
Correctly classified	143 (81.25%)
Incorrectly classified	33 (18.75%)
ROC area	0.860

Table 4.6: Accuracy measures obtained with 4 selected attributes and the same experimental set-up used in Section 4.3.

4.4.1 Performance of the *all peaks average deviation* descriptor alone

For completeness, we also briefly tested the *all peaks average deviation* descriptor alone using `DecisionStump`, a straightforward classifier consisting of a one-level decision tree. This algorithm selects only one attribute and sets a threshold on its values to discriminate between the two classes. We applied this algorithm to some of our datasets, corresponding to different configurations, and this algorithm always adopted the *all peaks average deviation* descriptor for decision, in agreement with Table 4.5. Some accuracy measures of this classifier obtained with a 10-fold cross-validation experiment on the *optimal parameters dataset* are reported in Table 4.7. The accuracy is only 10% lower than the one obtained with the full set (or the automatically selected subset).

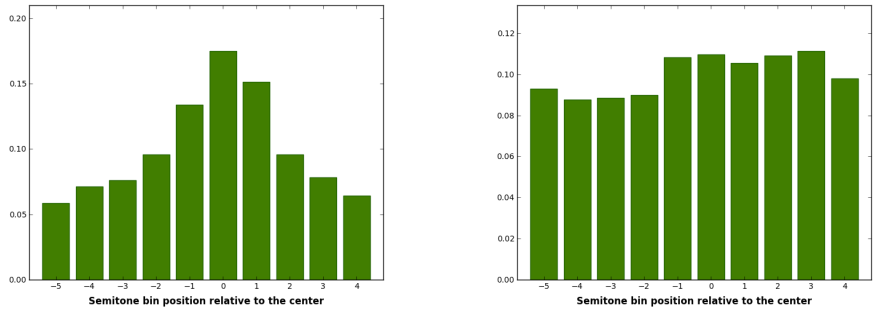
Measure	Values
Correctly classified	131 (74.43%)
Incorrectly classified	45 (25.57%)
ROC area	0.749

Table 4.7: Accuracy measures of a 10-fold cross-validation experiment with the `DecisionStump` classifier.

4.4.2 Performance of the *semitones outline vector* descriptor alone

In order to study the worth of the *semitones outline vector* as a descriptor, we isolated and tested the subset of attributes corresponding to its elements. This vector, as defined in Section 3.2, is meant to give a condensed representation of the general shape of the semitone intervals in which the pitch histogram got divided by the estimated tuning frequency. As shown by Figure 4.1, a query with a *good* tuning tends to have a clear, well-centered hill-shape while a *bad*

tuning do not present this regularity. Occasionally, though, this criterion fails and some queries that, according to a visual inspection of the plot should have a *bad* tuning, do sound in tune (see Figure 4.2). Some possible explanations of this apparent contradiction have been already discussed in Section 4.3.



(a) *Good* tuning (b) *Bad* tuning

Figure 4.1: Plot of the overall semitones outline of a (a) *good* tuning query and of a (b) *bad* tuning query.

We performed a classification experiment based only on the *semitones outline vector* attributes subset in the same measure set-up used to test the whole set and we obtained the results summarized in Table 4.8. This feature vector alone is able to provide a fair classification, likewise the peaks deviation descriptor (see Section 4.4.1),

Measure	**Values**
Correctly classified	135 (76.70%)
Incorrectly classified	41 (23.30%)
ROC area	0.846

Table 4.8: Accuracy measures obtained with the *semitones outline vector* attributes and the same experimental set-up used in Section 4.3.

4.5 Collective annotation

The objective of the next group of experiments was to test an alternative tuning quality annotation. The one that served as a reference for all our previous experiments was formulated by an expert so it is particularly reliable, nonetheless it is based on a human opinion and, as such, it can be questioned.

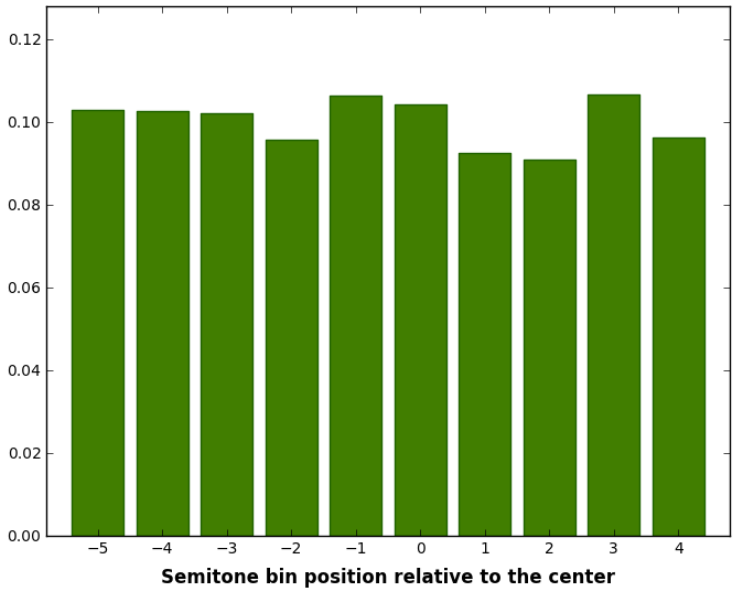


Figure 4.2: Sample plot of the overall semitones outline feature vector.

For this reason, we collected an additional annotation that could be used to validate some of our tests or directly to highlight possible controversial instances and compare them with those that are often mistaken by the classifiers.

A group of 8 people (3 females and 5 males) was asked to listen to all the queries and give a *good*/*bad* mark to their overall tuning quality. Each person was instructed to focus on the stability of the tuning frequency and to ignore other possible sources of error (e.g. rhythmic ones). Since the 8 members of the group had mixed musical experience, it seemed reasonable and convenient to add the original expert opinion to theirs. Eventually, we gathered a *collective annotation* formed by 9 complete sets of tuning quality assessments. The final binary classification of each query was obtained through majority voting (no ties possible thanks to the odd number of components).

Table 4.9 shows a contingency table, analogous to a confusion matrix, that summarizes the relations between the expert and the collective annotations. We can observe that there is a considerable overall agreement with 161 ($\tilde{0}.91\%$) concurrent classifications and that the disagreement regard almost entirely queries that have *bad* tuning according to the expert and are instead *good* tuning instances in the collective annotation. In general, the group seems less strict than the expert and this is consistent with their heterogeneous musical

		Collective opinion *Good*	*Bad*
Expert opinion	*Good*	86	2
	Bad	13	75
		99	**77**

Table 4.9: Contingency table of the expert and collective annotations.

experiences.

Comparing Table 4.9 to Table 4.4, we noticed the very different distribution of false negative and false positive instances. According to those values, the automatic classification is even more strict than the expert and tends to change the expert evaluation to *bad* tuning more than the contrary. These behaviour makes it evident that there could be few coincidences between the set of controversial instances of the two annotations and that of the queries frequently misclassified by the machine learning tools we tested (see Section 4.3). In particular, the 2 queries whose evaluation, according to the collective annotation, should be changed to *bad* tuning, are not among the recurring false negative instances of the classifiers. These queries are two different interpretations of the same song: "Mi vida así es morir de amor" by Camilo Sesto, which has a particularly challenging melodic contour so, apparently, the majority of our testers considered out-of-tune what both the expert and the classifier recognized as a fairly sung excerpt of a hard-to-sing song. Among the 13 queries that were moved to the *good* tuning class by the group, there are 3 that are also recurring false positive instances of the the classifiers. They are clear examples of those intermediate cases that a more advanced classification could treat more correctly than our binary system.

Measure	**Values**
Correctly classified	134 (76.14%)
Incorrectly classified	42 (23.86%)
ROC area	0.820

Table 4.10: Accuracy measures obtained using the collective annotation as class values and the same experiment set-up used in Section 4.3.

We repeated the same experiment done with the *optimal parameters*

dataset in Section 4.3 but using the collective annotation as the class values. The results are shown in Table 4.10 and are considerably lower than those obtained with the expert annotation (see Table 4.3). This unsurprising circumstance means that the criteria of the automatic evaluation have a higher agreement with those adopted by the expert than with those manifested by the collective annotation.

Chapter 5

Conclusions and future work

An automatic system for tuning quality evaluation was designed comprising a set of descriptors and automatic procedures to calculate them.

The system was preliminary tested using some machine learning tools for automatic classification and it obtained good accuracy values around 80%. Nonetheless, some limitations emerged that suggest possible future expansions of our work. They mainly consist in developing and testing more advanced descriptors and trying to overcome or control the consequences of the subjective elements of this problem.

In particular, the main possible improvements and extensions of the present study are:

- developing a proper ground truth: even if tuning evaluation is strongly affected by contextual and subjective elements, a simple experiment could be devised in which people listen and then sing melodic excerpts whose score is known allowing to detect and estimate the detuning;

- designing and testing new descriptors, possibly exploiting the temporal information: in the present work an overall atemporal analysis of the tuning was performed but exploiting the additional time dimension would probably result in a more complete representation of the evolution of the tuning quality;

- extending the range of the tuning quality measure: binary classification is a good straightforward approach, as proved by our result, but a less strict evaluation system would probably improve the performance substantially.

- testing in a baseline QbH system: the designed algorithm performed well enough to be used as an advising subsystem to inform the users

of the quality of their singing and it could be further improved and extended to implement an automatic correction strategy.

A remarkable result achieved by this project is the creation of a small but consistent set of tools, written in Python, to automatically perform all the main and the side tasks related to tuning evaluation exploited in the present work.

Bibliography

[1] J. Salamon and E. Gómez, "Melody extraction from polyphonic music signals using pitch contour characteristics," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, pp. 1759–1770, Aug. 2012.

[2] J. Salamon, J. Serrà, and E. Gómez, "Tonal representations for music retrieval: from version identification to query-by-humming," *International Journal of Multimedia Information Retrieval*, Dec. 2012.

[3] A. Ghias, J. Logan, D. Chamberlin, and B. C. Smith, "Query by humming: musical information retrieval in an audio database," in *Proceedings of the third ACM international conference on Multimedia*, pp. 231–236, ACM, 1995.

[4] R. J. McNab, L. A. Smith, D. Bainbridge, and I. H. Witten, "The new zealand digital library melody index," *D-Lib magazine*, vol. 3, no. 5, pp. 4–15, 1997.

[5] L. Prechelt and R. Typke, "An interface for melody input," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 8, no. 2, pp. 133–149, 2001.

[6] E. Unal, S. S. Narayanan, and E. Chew, "A statistical approach to retrieval under user-dependent uncertainty in query-by-humming systems," in *Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, pp. 113–118, ACM, 2004.

[7] D. Little, D. Raffensperger, and B. Pardo, "A query by humming system that learns from experience," in *Proceedings of the 8th International Conference on Music Information Retrieval, Vienna, Austria*, 2007.

[8] S. Rho, B.-j. Han, E. Hwang, and M. Kim, "Musemble: A music retrieval system based on learning environment," in *Multimedia and Expo, 2007 IEEE International Conference on*, pp. 1463–1466, IEEE, 2007.

[9] M. Ryynänen and A. Klapuri, "Query by humming of midi and audio using locality sensitive hashing," in *Proceedings of the 2008 IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 2249–2252, 2008.

[10] A. Ito, Y. Kosugi, S. Makino, and M. Ito, "A query-by-humming music information retrieval from audio signals based on multiple F0 candidates," in *2010 International Conference on Audio, Language and Image Processing*, pp. 1–5, Ieee, Nov. 2010.

[11] J. Qin, H. Lin, and X. Liu, "Query by Humming Systems Using Melody Matching Model Based on the Genetic Algorithm," *Journal of Software*, vol. 6, no. 12, pp. 2416–2420, 2011.

[12] M. Lech, "Automatic detection and correction of detuned singing system for use with query-by-humming applications," *Archives of Acoustics*, vol. 33, no. 4, pp. 99–104, 2008.

[13] M. Müller, P. Grosche, and F. Wiering, "Automated analysis of performance variations in folk song recordings," in *Proceedings of the international conference on Multimedia information retrieval*, pp. 247–256, ACM, 2010.

[14] G. Zhang, K. Lu, and B. Wang, *Query Reformulation Based on User Habits for Query-by-Humming Systems*, vol. 7675 of *Lecture Notes in Computer Science*, pp. 386–395. Springer Berlin Heidelberg, 2012.

[15] M. Lesaffre, D. Moelants, and M. Leman, *Spontaneous user behavior in "vocal" queries for music-information retrieval*, pp. 129–146. Music Query - Methods, Models, and User Studies, The MIT Press, 2004.

[16] C. Meek and W. Birmingham, "Johnny can't sing: A comprehensive error model for sung music queries," in *ISMIR 2002 Conference Proceedings*, pp. 124–132, 2002.

[17] B. Pardo and W. P. Birmingham, "Query by humming: how good can it get," in *Proceedings of the Workshop on Music Information Retrieval*, pp. 107–109, 2003.

[18] A. Duda, A. Nürnberger, and S. Stober, "Towards query by singing/humming on audio databases," in *Proceedings of the International Symposium/Conference on Music Information Retrieval*, 2007.

[19] T. De Mulder, J.-P. Martens, S. Pauws, F. Vignoli, M. Lesaffre, M. Leman, B. De Baets, and H. de Meyer, "Factors affecting music retrieval in query-by-melody," *Multimedia, IEEE Transactions on*, vol. 8, pp. 728 –739, aug. 2006.

[20] E. Unal, S. S. Narayanan, H.-H. Shih, E. Chew, and C.-C. J. Kuo, "Creating data resources for designing usercentric frontends for query-by-humming systems," *Multimedia systems*, vol. 10, no. 6, pp. 475–483, 2005.

[21] T. Nakano, M. Goto, and Y. Hiraga, "An automatic singing skill evaluation method for unknown melodies using pitch interval accuracy and vibrato features," *Rn*, vol. 12, p. 1, 2006.

[22] E. Nichols, C. DuHadway, H. Aradhye, and R. F. Lyon, "Automatically discovering talented musicians with acoustic analysis of youtube videos," in *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pp. 559–565, IEEE, 2012.